Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Conception d'interfaces standards pour une application destinée à des handicapés

Thirionet, Philippe

Award date:
1987

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 03. Jul. 2025

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

CONCEPTION D'INTERFACES STANDARDS POUR UNE APPLICATION DESTINEE A DES HANDICAPES

THIRIONET Philippe

Promoteur: Jean Ramaekers

Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique

Année académique 1986 –1987

Adresse : rue Grandgagnage, 21, B-5000 NAMUR (Belgium) - ☎ 081 - 22 90 65 - Télex 59 222 FAC NAM-B

L'objectif de ce mémoire consiste à développer un projet sur un ordinateur de type "IBM-PC compatible" permettant d'accroître l'autonomie de personnes qui ont perdu l'usage de leurs mains ou qui sont atteintes de tétraplégie. Ce projet nommé "AIDAMI" met à leur disposition un ensemble de services qui leur permettent d'allumer une lampe, de rédiger du courrier, d'effectuer une communication téléphonique... à l'aide de l'ordinateur, tout en sachant qu'elles n'ont pas la capacité physique de se servir du clavier de l'ordinateur. Il a donc été nécessaire de mettre à la disposition de ces personnes un interface composé par exemple d'un contact ou bouton poussoir qui leur permet d'entrer en communication avec l'ordinateur. Toutes les ressources offertes par l'ordinateur doivent être accessibles par le handicapé uniquement en manipulant ce contact, le clavier de la machine n'étant pas utilisé. Le handicapé doit également avoir la possibilité d'exécuter des logiciels "standards" vendus dans le commerce. Dans ce cas, nous avons été contraints de réaliser de la multiprogrammation. En effet, pendant que le logiciel "standard" s'exécute, il faut que le handicapé puisse toujours accéder aux services du projet "AIDAMI" car il peut par exemple éprouver le besoin d'allumer une lampe. Signalons enfin que le projet utilise la technique du multi-fenêtrage et qu'il peut être utilisé sur un micro-ordinateur équipé de deux écrans simultanément: un écran monochrome et un écran couleur, afin d'allouer un premier écran au projet "AIDAMI" et le second écran au logiciel "standard" susceptible d'être exécuté sur la machine.

Nous tenons à remercier:

Monsieur Jean Ramaekers, Directeur de l'Institut d'Informatique de Namur, pour l'aide et les conseils judicieux apportés au cours de la réalisation de ce travail.

Mademoiselle Cécile Mahiat,
de sa grande disponibilité et de son aide précieuse pour la rédaction de ce mémoire.

Monsieur Christian Meunier,
pour sa gentillesse et pour l'accueil chaleureux qu'il nous a réservé à son domicile.

Monsieur Philippe Andre, Madame Jocelyne Duterne,
Monsieur Bruno Bellofatto, Monsieur Jean-Marc Lassoie,
de l'aide administrative et technique apportée et de leur efficace dévouement au développement de ce projet.

Monsieur Jacques Pierson, Monsieur Philippe Tubello,
d'avoir mis à notre service leur compétence informatique.

Monsieur Marcel Clantin, Monsieur Gérard Paquet,
pour les remarques et propos pertinents énoncés lors de l'élaboration de ce projet.

Monsieur Ernest Pourignaux, Monsieur Paul Koeune, Ingénieurs de la société "Analis" à Namur,
d'avoir aimablement mis à notre disposition leur compétence technique.

Toutes les personnes de l'Institut d'Informatique, ou extérieures à la faculté, qui ont permis de mener à bien cette étude.

TABLE DES MATIERES
==========

INTRODUCTION 1

CHAPITRE 1 PRESENTATION DU SYSTEME

1.1 Introduction 3

1.2 Analyse des besoins 3

1.2.1 Le besoin fondamental : l'autonomie 3
1.2.2 Présentation et réponse aux besoins assurant l'autonomie 4

1.3 Représentation globale du système 12

1.4 Objectifs liés à la réalisation sur ordinateur 14

1.4.1 Maintenance du système 14
1.4.2 Utilisation de logiciels "standards" 15
1.4.3 Action sur l'environnement à tout moment..... 16
1.4.4 Portabilité du programme réalisé dans cette étude..... 16

1.5 Principes généraux de réalisation du système 17

1.5.1 L'interface utilisateur 17
1.5.2 Accès aux services offerts par l'ordinateur 17
1.5.3 Sélection d'un objet dessiné à l'écran 18
1.5.4 Les affichages sur écran 20

CHAPITRE 2 ARCHITECTURE LOGIQUE DU SYSTEME

2.1 Introduction 22

2.2 Découpe hiérarchique logique 22

2.2.1 Objectifs d'une découpe en niveaux 22
2.2.2 Présentation du schéma de la hiérarchie de l'application 25
2.2.3 Analyse des composants du graphe de la hiérarchie 27

CHAPITRE 3 ETUDE DU NIVEAU "SYSTEME D'EXPLOITATION" DE LA HIERARCHIE DE L'APPLICATION

3.1 Introduction 29

3.2 Présentation du système d'exploitation "idéal" relatif à cette application 29

3.3 Présentation de la structure hiérarchique du système d'exploitation 31

3.3.1 But d'une découpe hiérarchique du niveau "système d'exploitation" 31
3.3.2 Présentation et analyse des composants du graphe hiérarchique du système d'exploitation 32

3.4 Problèmes liés à la multiprogrammation 32

3.4.1 Définition d'outils pour implémenter la multiprogrammation ... 32
3.4.2 Perturbations provoquées par un logiciel "standard" sur la multiprogrammation 34
3.4.3 Le problème de l'accès au système d'exploitation 37

3.5 Définition d'un sous-système utile 39

CHAPITRE 4 PRINCIPES DE FONCTIONNEMENT DE L'IBM-PC

4.1 Introduction 41

4.2 Les registres de l'IBM-PC 41

4.3 Présentation sommaire de la mémoire centrale de l'IBM-PC 43

4.4 Segmentation d'un programme chargé en mémoire centrale 44

4.5 Le système d'exploitation DOS 45

4.5.1 Organisation du système d'exploitation DOS en mémoire centrale 46
4.5.2 Les interruptions du système d'exploitation 49

4.6 Méthodes d'accès aux périphériques 53

4.7 Le clavier 54

4.8 Les adaptateurs vidéo 55

4.8.1 L'adaptateur d'écran monochrome 56
4.8.2 L'adaptateur d'écran couleur 56
4.8.3 L'utilisation simultanée de deux adaptateurs 56

CHAPITRE 5 IMPLEMENTATION DU LOGICIEL

5.1 Introduction 58

5.2 Réalisation de la multiprogrammation avec le système d'exploitation DOS 58

5.2.1 Analyse et critique des diverses solutions pour implémenter la multiprogrammation à l'aide du DOS 59
5.2.2 Choix d'une solution pour réaliser la multiprogrammation 65

5.2.3 Technique d'implémentation de la solution assurant la multiprogrammation 67

5.2.3.1 Principe général d'implémentation de la solution assurant la multiprogrammation 67
5.2.3.2 Analyse détaillée des routines permettant d'implémenter la multiprogrammation 69
5.2.3.3 Principe de résolution de problèmes divers créés par le système d'exploitation DOS 76

5.3 Réalisation du gérant des processus 81

5.4 Mise en résidance du programme en mémoire centrale 85

5.5 Implémentation des primitives d'exclusion mutuelle 88

5.6 Installation de l'interface utilisateur 91

5.6.1 Développement du logiciel de l'interface utilisateur 91
5.6.2 Raccordement de l'interface utilisateur à l'ordinateur 93

5.7 Simulation du clavier par l'ordinateur 93

CONCLUSIONS 95

ANNEXE A Services et tâches disponibles dans le système "AIDAMI" 97

ANNEXE B Explicitation des primitives et des fichiers identifiés dans les composants de la hiérarchie de l'application 102

ANNEXE C Spécifications externes des primitives identifiées dans la découpe hiérarchique de l'application et implémentées dans notre programme 105

ANNEXE D Présentation du schéma hiérarchique du composant "système d'exploitation" de la hiérarchie de l'application 123

ANNEXE E Explicitation des primitives relatives à chaque composant du niveau "système d'exploitation" 126

ANNEXE F Spécifications externes des primitives identifiées dans le composant "système d'exploitation" et implémentées dans notre programme 128

ANNEXE G Présentation du contenu des variables d'état du clavier 133

ANNEXE H Les codes associés aux touches du clavier 134

ANNEXE I Mode d'emploi du programme 137

ANNEXE J Texte du programme 138

BIBLIOGRAPHIE 139

INTRODUCTION

INTRODUCTION

Toute personne recherche, durant son existence, une certaine autonomie et ressent le besoin d'accomplir un grand nombre d'actions et de travaux sans devoir dépendre d'un autre être humain. Citons par exemple la fermeture d'une porte, l'ouverture des rideaux, l'allumage d'une lampe... Or certains individus de notre entourage sont atteints d'un handicap physique et voient leur autonomie s'amoindrir. Bien évidemment, plusieurs types de handicaps physiques coexistent, mais l'étude réalisée dans le cadre de ce mémoire vise essentiellement à venir en aide aux personnes dont les possibilités de mouvements sont très limitées. Nous pensons plus particulièrement ici aux personnes qui ont perdu l'usage de leurs mains ou qui sont atteintes de tétraplégie.

Le meilleur moyen de savoir ce dont a besoin un tel handicapé consiste à dialoguer avec lui. Une équipe de chercheurs a travaillé dans ce domaine durant un an. Cette même équipe a développé à partir de cette analyse un système "prototype" permettant d'accroître l'autonomie du handicapé. Ce système nommé "AIDAMI" (AIde A partir de MIcroprocesseur) a déjà abouti à la réalisation d'un grand nombre d'objectifs. Nous nous baserons au cours de notre étude sur cette réalisation en y apportant des améliorations utiles.

La structure du système "prototype" réalisé par cette équipe est en effet parfois quelque peu confuse en raison de son développement progressif. Nous avons également remarqué que toutes les richesses offertes par le matériel supportant le système "AIDAMI" n'ont pas été totalement exploitées, ce qui réduit dans certains cas considérablement les services offerts au handicapé. En effet, ce "prototype" présente parfois des situations où un nombre très réduit de services peuvent être utilisés par la handicapé, les autres services n'étant plus accessibles pour des raisons de cohérence et de fiabilité du système. Enfin, le système "prototype" mis au point par l'équipe de chercheurs ne met à la disposition du handicapé qu'un seul des services offerts par le système à un moment donné; c'est-à-dire que lorsqu'un service est utilisé, celui-ci doit être exécuté jusqu'à son terme avant de pouvoir disposer d'un autre service.

Ainsi, l'objectif de ce mémoire consiste à structurer efficacement le projet "AIDAMI" en vue de faciliter des modifications futures éventuelles, de même qu'à étudier en détail le support matériel du système pour permettre au handicapé de disposer constamment de l'entièreté des services mis à sa disposition. Nous entendons par là le fait que l'utilisateur peut accéder à un service du système, interrompre momentanément ce service afin d'en exécuter un autre, et finallement revenir au premier service qui a été interrompu afin de poursuivre son utilisation. Le handicapé n'est donc pas obligé de terminer un service pour en disposer d'un autre. Nous disons dans ce cas que les services sont accessibles "simultanément". Tous les services sont ainsi disponibles quel que soit l'état du système, mises à part quelques situations de force majeure pour maintenir le système dans un état cohérent. Nous désirons également implémenter dans ce système des services supplémentaires utilisables par le handicapé. Citons notamment la possibilité de réaliser des dessins ou des graphiques.

Notons encore que le "prototype" du projet "AIDAMI" est un programme codé à l'aide des langages "assembleur" et "Pascal". Le langage "assembleur" est un langage de bas niveau qui ne se prête pas à la réalisation d'un programme structuré. C'est pourquoi nous souhaitons implémenter la nouvelle version de ce projet à l'aide d'un seul et unique langage de haut niveau tel que "Pascal" ou "C".

Nous présentons ci-dessous un résumé du contenu de chaque chapitre de ce mémoire.

Le début du chapitre 1 expose en détail le système "AIDAMI" réalisé par l'équipe de chercheurs. Nous y avons effectué quelques modifications afin de tenir compte des améliorations apportées par ce mémoire. Nous verrons que ce système est en fait implémenté sur un ordinateur: ceci nous permet d'analyser les objectifs visés par l'utilisation d'une telle machine. Quelques principes généraux de réalisation du projet sont ensuite présentés.

Le chapitre 2 contient la procédure suivie pour structurer efficacement le système. Cette structure repose sur une découpe hiérarchique de l'application "AIDAMI". Les objectifs et l'analyse de cette découpe font l'objet de ce chapitre.

La mise à la disposition du handicapé de plusieurs services simultanément pose un certain nombre de problèmes bien spécifiques. En effet, le système d'exploitation de l'ordinateur doit posséder certaines fonctionnalités indispensables pour mener à bien cette exigence. Le chapitre 3 présente le système d'exploitation dont le projet "AIDAMI" a besoin pour fonctionner correctement. Nous analysons brièvement ce système d'exploitation et en présentons une découpe hiérarchique afin d'identifier plus aisément les fonctionnalités exigées par le système "AIDAMI", qui ne sont pas offertes par le système d'exploitation dont nous disposons. Nous analysons ensuite les problèmes posés par l'implémentation des services exécutés de manière simultanée.

Le chapitre 4 expose quelques principes de fonctionnement de l'ordinateur utilisé pour implémenter le projet "AIDAMI". Ceux-ci facilitent la compréhension de l'étude faite au chapitre 5.

Le chapitre 5 présente l'implémentation du système "AIDAMI" sur l'ordinateur en tenant compte des améliorations apportées par ce mémoire. Nous développons en détail l'analyse effectuée pour mettre à la disposition du handicapé plusieurs services simultanément. Nous présentons les différentes solutions possibles pour réaliser cette exigence, ainsi que l'implémentation de la solution qui a été choisie. Divers aspects plus particuliers sont ensuite étudiés.

Enfin, nous signalons au lecteur que le texte du programme réalisé dans cette étude est exclusivement disponible auprès de Monsieur Jean Ramaekers, directeur de l'Institut d'Informatique, pour des raisons de confidence.

CHAPITRE 1

CHAPITRE 1 PRESENTATION DU SYSTEME

1.1 Introduction

Ce chapitre a pour but de familiariser le lecteur avec le système "AIDAMI". Nous présentons dans le paragraphe 1.2 les grands principes du système "AIDAMI". Ceux-ci reflètent les résultats obtenus par l'équipe de chercheurs qui a déjà travaillé sur le projet pendant un an. Des modifications ont cependant été apportées pour tenir compte des améliorations du système apportées par ce mémoire.

Nous exposons au paragraphe 1.4 quelques objectifs liés au mode de réalisation de ce projet. Enfin, une brève présentation de quelques principes de réalisation du système "AIDAMI" clôture ce premier chapitre.

1.2 Analyse des besoins

1.2.1. Le besoin fondamental: l'autonomie

Un des besoins fondamentaux de tout être humain et à fortiori d'un handicapé physique est l'autonomie. Accroître l'autonomie d'une personne ayant perdu l'usage de ses mains n'est pas chose facile. Ce problème pourrait être résolu par exemple en "bricolant" l'un ou l'autre objet facilitant la fermeture d'une porte, ou bien en utilisant un petit robot. Cependant, puisque l'informatique permet actuellement de résoudre beaucoup de problèmes dans notre société, pourquoi ne pas venir en aide à un handicapé physique en ayant recours à un micro-ordinateur. Ainsi, nous avons jugé bon d'assister le handicapé dans ses actions avec un micro-ordinateur, d'autant plus que le coût de ces machines devient très abordable.

Un problème majeur subsiste cependant: comment une personne ayant perdu l'usage de ses mains pourrait-elle se servir d'une telle machine, sachant que la manière classique de dialoguer avec un ordinateur est généralement réalisée via un clavier? Il a donc fallu imaginer un moyen pour que ces personnes puissent communiquer avec le micro-ordinateur. Une solution relativement facile à implémenter et peu coûteuse consiste à réaliser cette interface à l'aide d'un simple contact. Les personnes désireuses de se servir du système doivent avoir la capacité de manipuler ce contact avec un de leurs membres, assurant ainsi l'établissement ou la coupure d'un courant électrique. Ce contact constitue en fait l'interface la plus élémentaire qui soit. En effet, il existe actuellement plusieurs catégories d'interfaces qui ont chacune été étudiées pour faire face à un type de handicap donné. Citons par exemple une combinaison de plusieurs contacts, un joystick (manche à balai), etc...

Lorsqu'un tel interface entre le micro-ordinateur et le handicapé est choisi, il n'est pas toujours aisé de le connecter à la machine. Il est souvent nécessaire de réaliser un circuit électronique qui établit la connexion entre l'interface et le micro-ordinateur. La figure 1.1 présente une telle configuration.

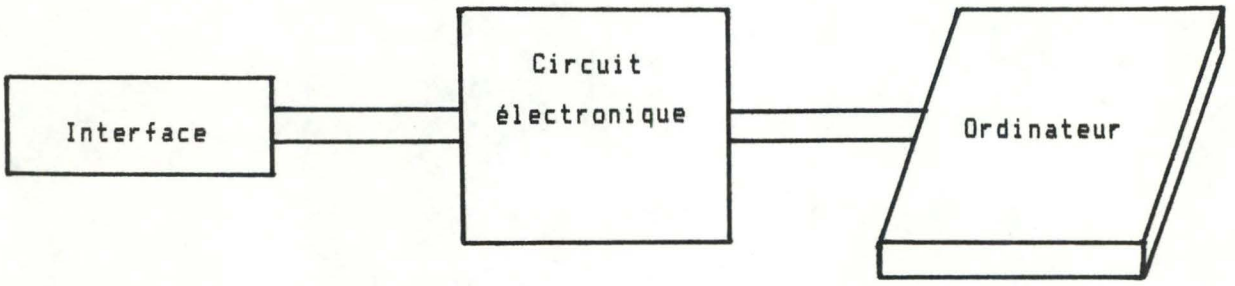

Figure 1.1 Connexion de l'interface à l'ordinateur.

1.2.2. Présentation et réponse aux besoins assurant l'autonomie

Le besoin d'autonomie ressenti par tout être humain n'est satisfait que si d'autres besoins plus spécifiques le sont également. En effet, l'autonomie n'est assurée que si la personne peut communiquer avec d'autres individus, prendre connaissance de certaines informations, rédiger son courrier, etc... sans l'aide d'une personne extérieure. L'ordinateur mis à la disposition du handicapé peut satisfaire dans une certaine mesure l'ensemble de ces besoins. Il faut donc avant toute chose assurer le dialogue entre le handicapé et l'ordinateur uniquement à l'aide de l'interface utilisateur. En effet, nous avons signalé que l'utilisateur n'a pas la capacité d'utiliser le clavier associé à la machine. Une manipulation adéquate de l'interface doit dans ce cas permettre à l'utilisateur de dialoguer avec la machine exactement comme s'il se servait du clavier. L'implémentation d'un programme simulant le clavier de la machine sur l'écran facilite ce dialogue. La figure 1.2 illustre un exemple de clavier simulé à l'écran.

Lorsque ce dessin est affiché sur l'écran, il est nécessaire de concevoir un mécanisme permettant de sélectionner une touche précise du clavier. Diverses méthodes sont envisageables: nous citons plus particulièrement: * le principe du balayage
* le principe du pointage direct

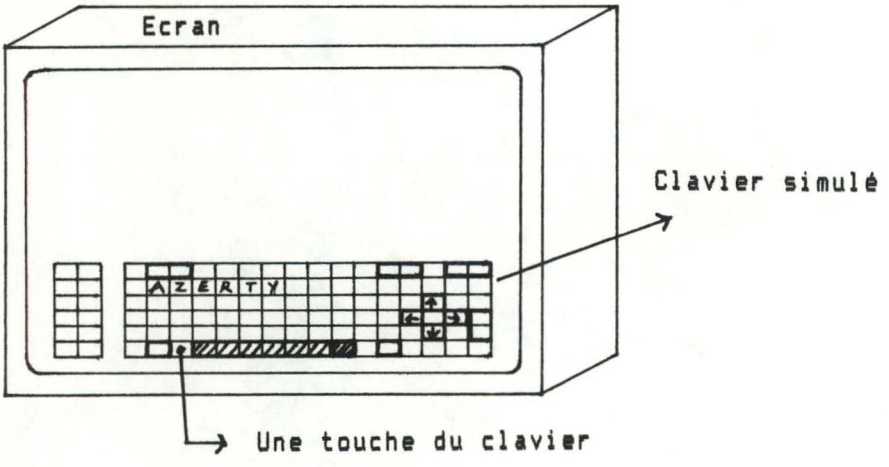

Figure 1.2 Simulation d'un clavier à l'écran.

a) Le principe du balayage

Ce principe repose sur le fait que chacune des touches du clavier affichées à l'écran est représentée à l'aide de deux couleurs différentes: une couleur est attribuée au caractère représentatif de la touche; par exemple le caractère "a" et une seconde couleur caractérise la "toile de fond" sur laquelle ce caractère est affiché. La figure 1.3 illustre ces concepts.

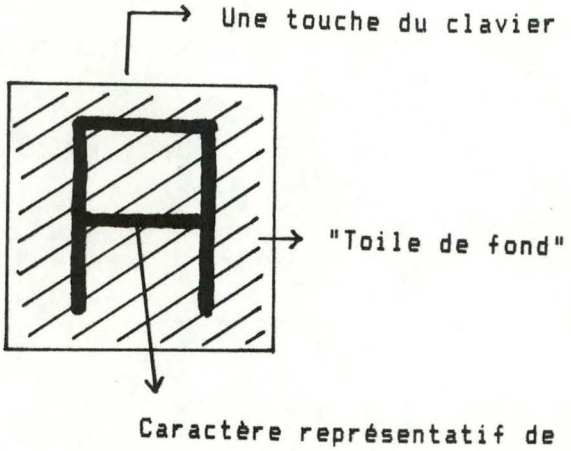

Figure 1.3 Représentation d'une touche du clavier à l'écran.

Le choix de ces deux coloris détermine le contraste entre le caractère affiché et la "toile de fond". Ces deux couleurs doivent bien évidemment être différentes car dans le cas contraire, le caractère serait confondu avec la toile de fond et ne serait plus visible.

La technique du balayage consiste à échanger la couleur du caractère avec la couleur de la toile de fond. Cet échange est réalisé pour une touche bien précise à un moment donné, pour une autre touche un instant plus tard, et le processus continue indéfiniment pour chacune des touches du clavier (figure 1.4). Nous appelons "contrastée" une touche dont les couleurs sont échangées.

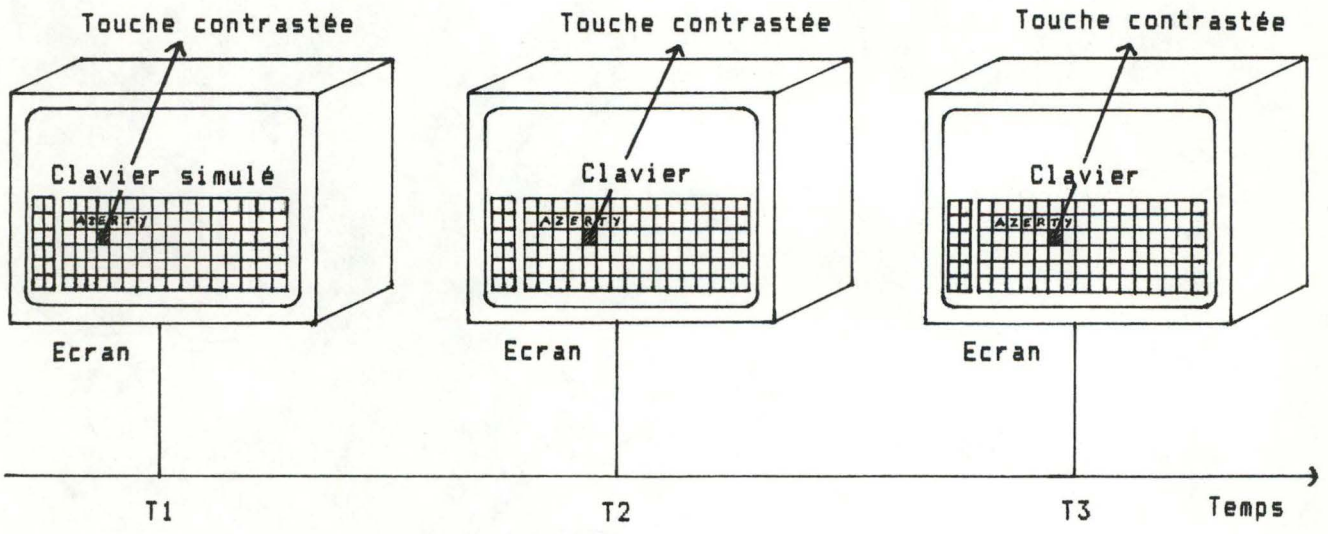

Figure 1.4 Balayage des touches du clavier.

Pour sélectionner une touche particulière du clavier simulé à l'écran, il suffit d'agir adéquatement sur l'interface mis à la disposition du handicapé lorsque la touche désirée est contrastée. Une telle action sur l'interface serait par exemple l'appui sur le contact. (voir figure 1.5)

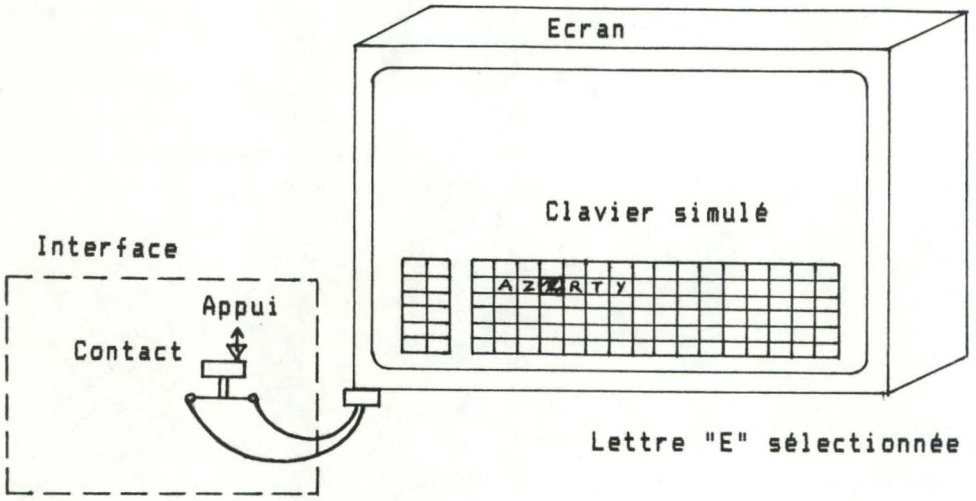

Figure 1.5 Sélection d'une touche du clavier par balayage.

b) Le principe du pointage direct

Dans le cas du pointage direct, il est nécessaire de disposer d'un curseur mobile à l'écran. Ce curseur est généralement représenté sous la forme d'une flèche qui peut pointer vers un endroit quelconque de l'écran. (voir figure 1.6)

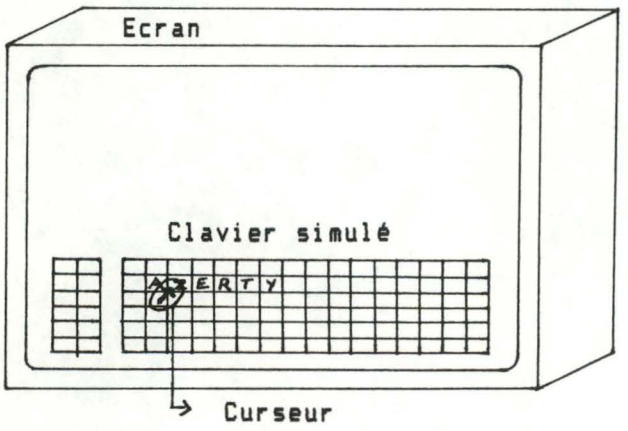

Figure 1.6 Représentation du curseur mobile à l'écran.

La sélection d'une touche du clavier consiste à pointer celle-ci avec le curseur. L'utilisation d'un interface de type "souris" ou "joystick" (manche à balai) se prête très bien à ce genre de manipulation. Il suffit enfin de signaler à la machine que la touche ainsi pointée est bien celle que l'utilisateur désire. Une seconde action sur l'interface telle qu'une pression sur un contact permet de réaliser cela. (figure 1.7)

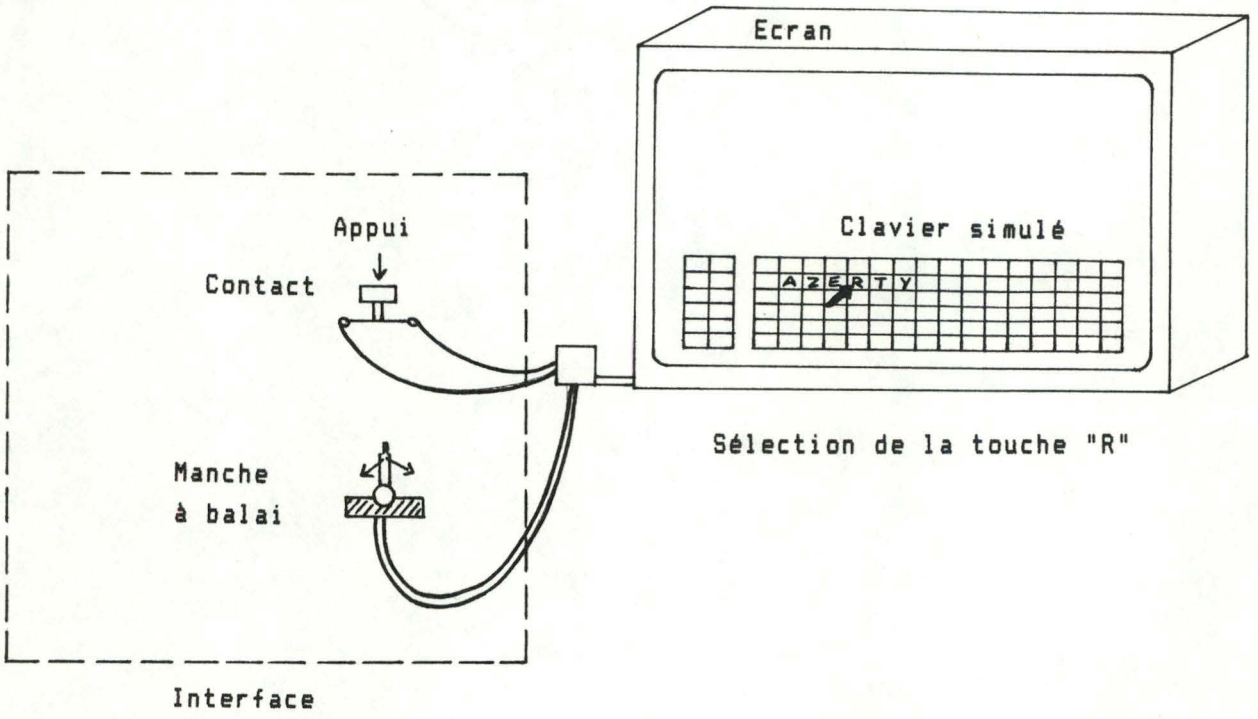

Figure 1.7 Sélection d'une touche au clavier par pointage direct.

On remarque que dans ce type de sélection des touches, l'interface entre le handicapé et l'ordinateur est composé de deux parties. Par exemple: une souris ou un joystick accompagné d'un contact. Cependant, la manipulation d'un tel interface est plus complexe et demande davantage de mobilité de la part du handicapé.

L'élaboration du système AIDAMI a déjà permis d'identifier les besoins assurant l'autonomie, en collaboration avec une personne atteinte de tétraplégie. Nous les exposons ci-après ainsi que les moyens mis en oeuvre pour les satisfaire. Dans le cadre de cette étude, ces moyens sont représentés par des "tâches".

Lorsque l'utilisateur entre en communication avec l'ordinateur à l'aide de son interface, il peut accéder aux ressources offertes par la machine. Celle-ci met notamment à sa disposition un ensemble d'outils qui permettent de satisfaire les besoins accroissant son autonomie. Ces outils constituent en réalité un programme exécuté par l'ordinateur et qui fait l'objet de cette étude.

Nous exposons à présent l'ensemble des tâches mises à la disposition du handicapé.

1) La tâche "courrier"

L'édition d'un texte ou la création d'un document à l'aide de la tâche "courrier" constitue un outil de grande valeur pour le handicapé. Cette tâche ne nécessite aucune ressource particulière si ce n'est une petite partie du programme faisant l'objet de cette étude. Cependant, les services offerts par cette tâche ne sont pas très élaborés comme nous le verrons dans la suite de ce paragraphe; il est en effet possible d'utiliser des logiciels ou programmes vendus dans le commerce qui mettent à la disposition de l'utilisateur des outils beaucoup plus puissants. Cette tâche permet de manipuler des documents ou des textes en utilisant le clavier simulé à l'écran. Une impression de ceux-ci à l'aide d'une imprimante peut également être effectuée.

2) La tâche "téléphone"

L'objectif poursuivi par la tâche "téléphone" est de permettre à une personne ne sachant plus maîtriser l'usage de ses mains, d'entrer en communication avec un autre être humain à l'aide d'un téléphone. Ceci semble relativement difficile si ce handicapé ne dispose que d'un appareil téléphonique classique, auquel cas il faut pouvoir manipuler les touches du clavier numérique ainsi que le cornet. Une solution consiste à mettre à la disposition de l'utilisateur un "mains-libres". Cet appareil fournit les mêmes services qu'un combiné téléphonique normal, si ce n'est que le cornet est remplacé par un haut-parleur et un micro qui permettent de dialoguer avec le correspondant tout en restant à une certaine distance de l'appareil.

Le "mains-libres" est un appareil disponible dans le commerce, mais celui-ci n'est pas spécialement conçu pour être sollicité par l'intermédiaire d'un micro-ordinateur. Sa connexion à la machine doit dans ce cas être réalisée grâce à un circuit électronique spécial. La figure 1.8 illustre le raccordement d'un "mains-libres" à l'ordinateur.

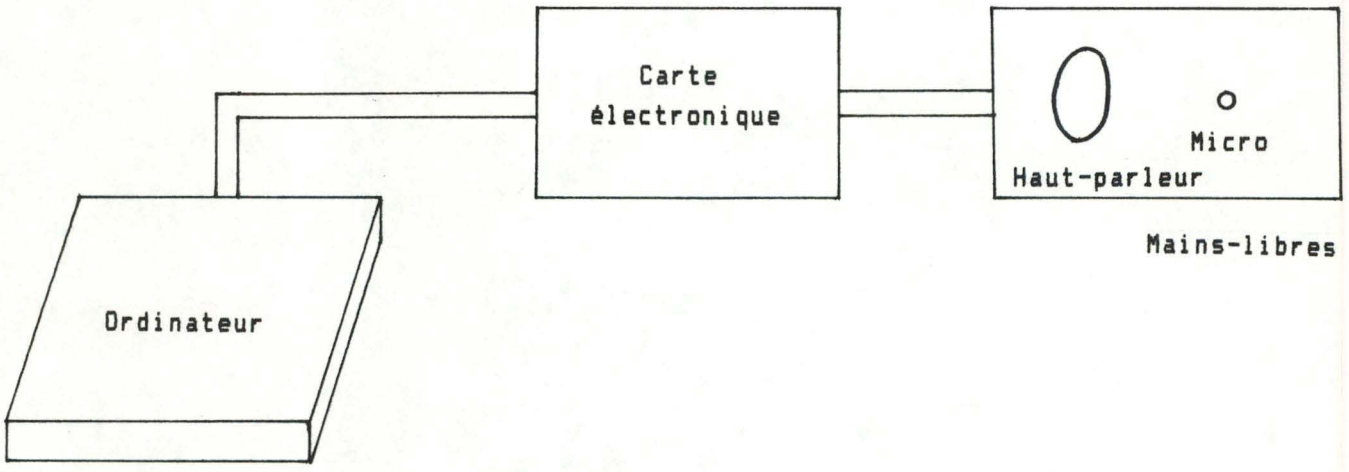

Figure 1.8 Raccordement d'un "mains-libres" à l'ordinateur.

Une telle configuration doit permettre au handicapé de commander le "mains-libres" à partir du programme prévu à cet effet afin de mettre à sa disposition les mêmes fonctionnalités qu'un combiné téléphonique classique. Les principales fonctionnalités offertes par cette tâche sont le "décrochage" et le "raccrochage" du téléphone par l'ordinateur, et la composition d'un numéro de téléphone. Nous constatons ici que seule l'implémentation d'un programme sur l'ordinateur ne peut satisfaire les besoins pressentis par l'utilisateur. D'autres appareils ou circuits électroniques sont également nécessaires pour satisfaire ces besoins.

3) La tâche "interphone"

La communication avec un autre être humain à partir d'un interphone est relativement fort semblable à celle réalisée par le téléphone. L'utilisateur doit disposer d'un interphone grâce auquel il peut converser avec cette personne. L'appareil dispose également d'un micro et d'un haut-parleur, cependant il ne permet d'entrer en communication qu'avec un individu se trouvant dans le voisinage immédiat du handicapé; par exemple dans une pièce voisine de l'habitation. Ceci permet de limiter les déplacements effectués par le handicapé.

4) La tâche "commande d'un modem"

La réalisation de la tâche "commande d'un modem" nécessite l'acquisition par le handicapé d'un modem (modulateur/démodulateur). Cet appareil permet de connecter l'ordinateur au réseau téléphonique dans le but d'échanger des informations avec un autre ordinateur; citons notamment l'accès aux bases de données publiques. Cette seconde machine doit elle aussi être raccordée à ce réseau. Un schéma global de cette configuration est présenté à la figure 1.9.

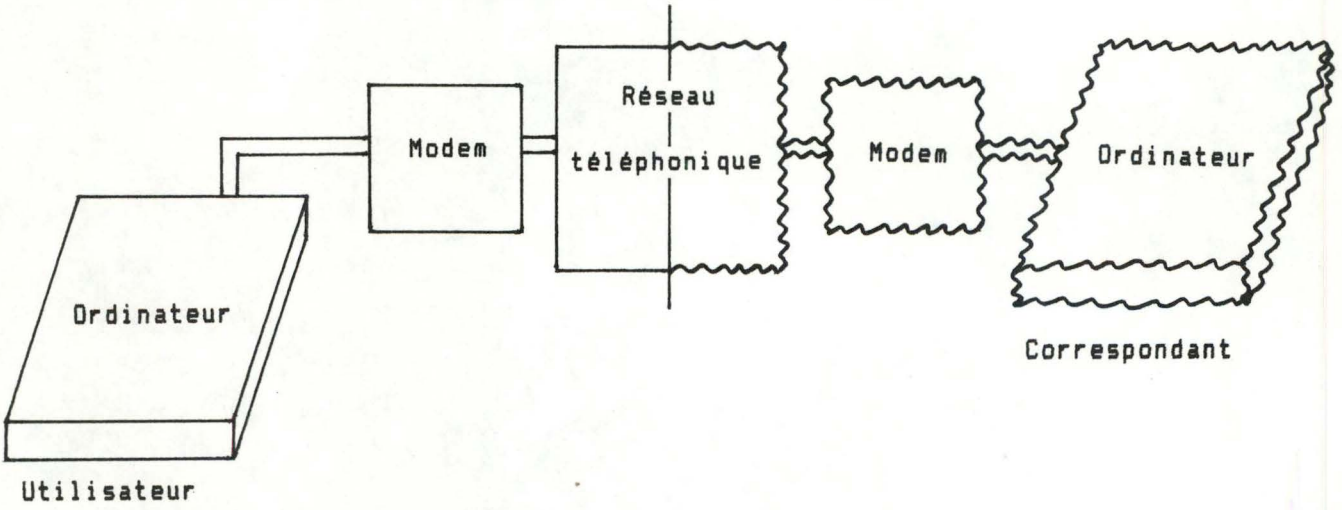

Figure 1.9 Raccordement d'un ordinateur au réseau téléphonique via un modem.

5) La tâche "commande"

La tâche "commande" doit fournir à l'utilisateur la possibilité de déconnecter ou de connecter des appareils électriques (lampes, ventilateurs...) au réseau électrique 220 volts. Chacun de ces appareils doit donc être relié à ce réseau via un interrupteur. Celui-ci est susceptible d'être actionné grâce à un signal issu de l'ordinateur. En pratique, nous disposons d'un circuit électronique sur lequel sont implémentés plusieurs interrupteurs assurant la commande des appareils électriques à partir de la machine. Cette tâche permet ainsi au handicapé d'allumer ou d'éteindre une lampe, ou d'autres appareils. (voir figure 1.10)

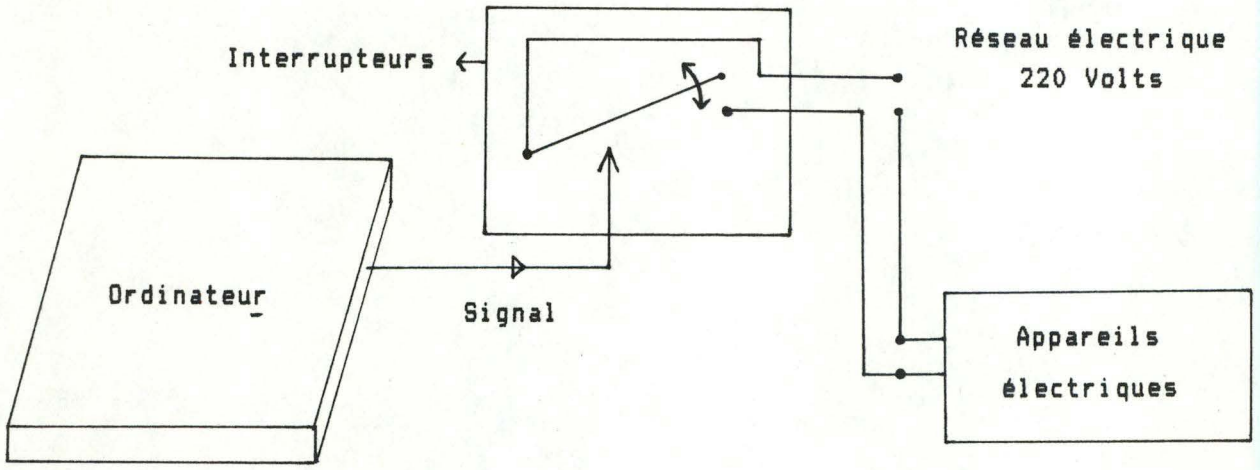

Figure 1.10 Commande d'un appareil électrique à l'aide d'un interrupteur.

6) La tâche "télécommande"

L'accès aux boutons de réglage d'une télévision ou d'un magnétoscope présente beaucoup de difficultés pour un handicapé tel que ceux envisagés dans cette étude. La possibilité de commander ces appareils à partir d'une télécommande à distance a déjà fortement facilité cet accès. Néanmoins, une solution plus élégante à ce problème consiste à simuler l'appui sur une touche de la télécommande à l'aide de l'ordinateur.

La première étape consiste à représenter à l'écran tous les boutons disponibles sur la télécommande. La sélection d'un de ces boutons peut être réalisée de la même manière que lorsque l'on effectuait la sélection d'une touche du clavier simulé: soit par le principe du balayage, soit par le principe du pointage direct (voir ci-dessus). Le raccordement de la télécommande à l'ordinateur constitue la seconde étape. La figure 1.11 présente une telle configuration.

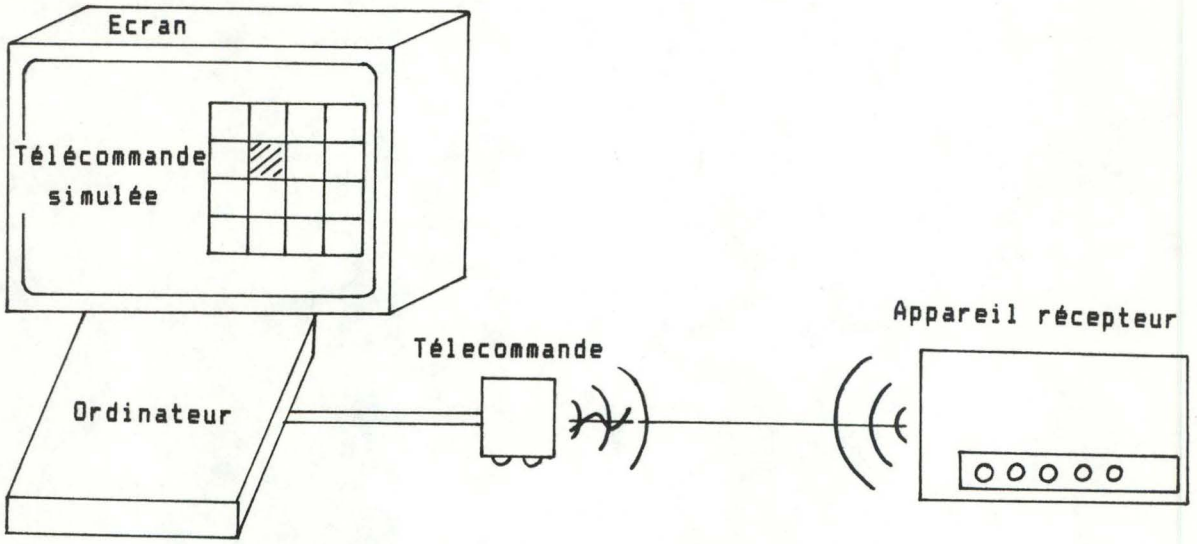

Figure 1.11 Raccordement d'une télécommande à l'ordinateur.

Le système présenté ici offre au handicapé l'accès à toutes les fonctions disponibles sur la télécommande.

7) La tâche "outils"

Le handicapé peut disposer d'accessoires de bureau et d'autres utilitaires par l'intermédiaire de la tâche "outils". Parmis ces accessoires figurent notamment une horloge, une calculatrice, un bloc-notes; ceux-ci dispensent l'utilisateur de déplacements et de manipulations d'objets épuisants.

8) La tâche "logiciel"

D'autres types de services sont offerts par un ordinateur. En effet, non seulement il peut permettre au handicapé d'agir sur son environnement, mais aussi rendre des services tels que exécuter des logiciels "standards" vendus sur le marché ou aider l'utilisateur à créer ses propres programmes. Il est en général utile d'équiper l'ordinateur de deux écrans pour assurer le bon déroulement de cette tâche (voir figure 1.12). Nous expliquons au paragraphe 3.4.2 du chapitre 3 les raisons d'une telle exigence.

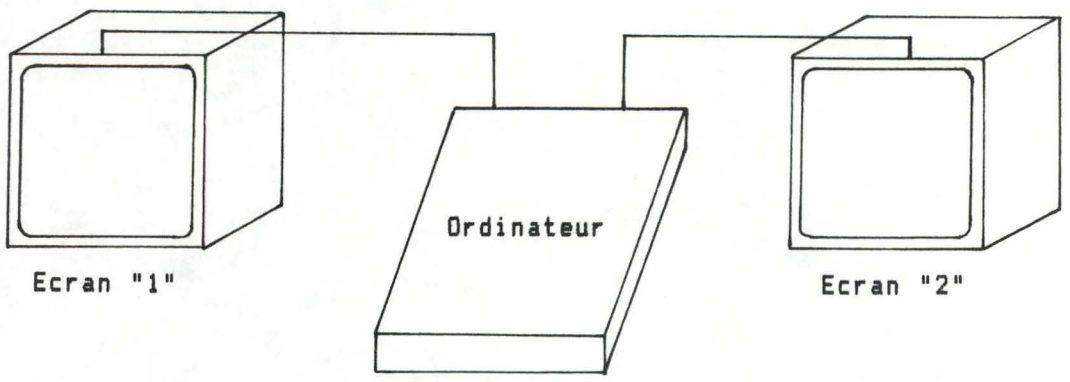

Figure 1.12 Raccordement de deux écrans à l'ordinateur.

9) La tâche "alarme"

La commande d'une alarme à partir du micro-ordinateur peut être réalisée en utilisant le principe de l'interrupteur déjà exposé dans la tâche "commande" (voir point "5" ci-dessus). Grâce à cette alarme, le handicapé peut déclencher un signal sonore de manière à demander l'intervention rapide d'un tiers dans les cas urgents.

En général, ces différentes tâches ne sont pas accessibles directement dans l'ordinateur car elles sont réparties dans des services. Un service est composé d'une ou de plusieurs tâches et est directement accessible par l'utilisateur. Nous avons identifié les services suivants: "courrier", "téléphone", "relais", "télécommande", "outils", "logiciels" et "sortie".

Une description des fonctions disponibles pour chaque tâche ainsi que des services est présentée en annexe "A".

1.3. Représentation globale du système

Une représentation globale du système est présentée à la figure 1.13. Nous avons repris dans ce schéma l'ensemble des interfaces et appareils électroniques dont nous avons parlé dans les paragraphes 1.2.1 et 1.2.2.

L'ordinateur, coeur du système, est représenté au centre de la figure. Sur cet élément central viennent s'interconnecter les appareils électroniques et interfaces suivants:

* les deux écrans permettant de visualiser les messages et autres objets visibles par l'utilisateur.
* les appareils électriques raccordés au réseau électrique 220 volts et alimentés via un interrupteur.
* le "mains-libres" muni de son circuit électronique spécial.
* les appareils électroniques commandables par une télécommande à distance.
* le clavier. Celui-ci est présent dans la configuration car il est toujours fonctionnel et peut permettre à une personne valide de l'utiliser afin d'aider le handicapé pour l'accomplissement d'une tâche quelconque.
* l'interface utilisateur (joystick, contact,...) et son circuit électronique spécial.
* le modem permettant une liaison avec le réseau téléphonique.

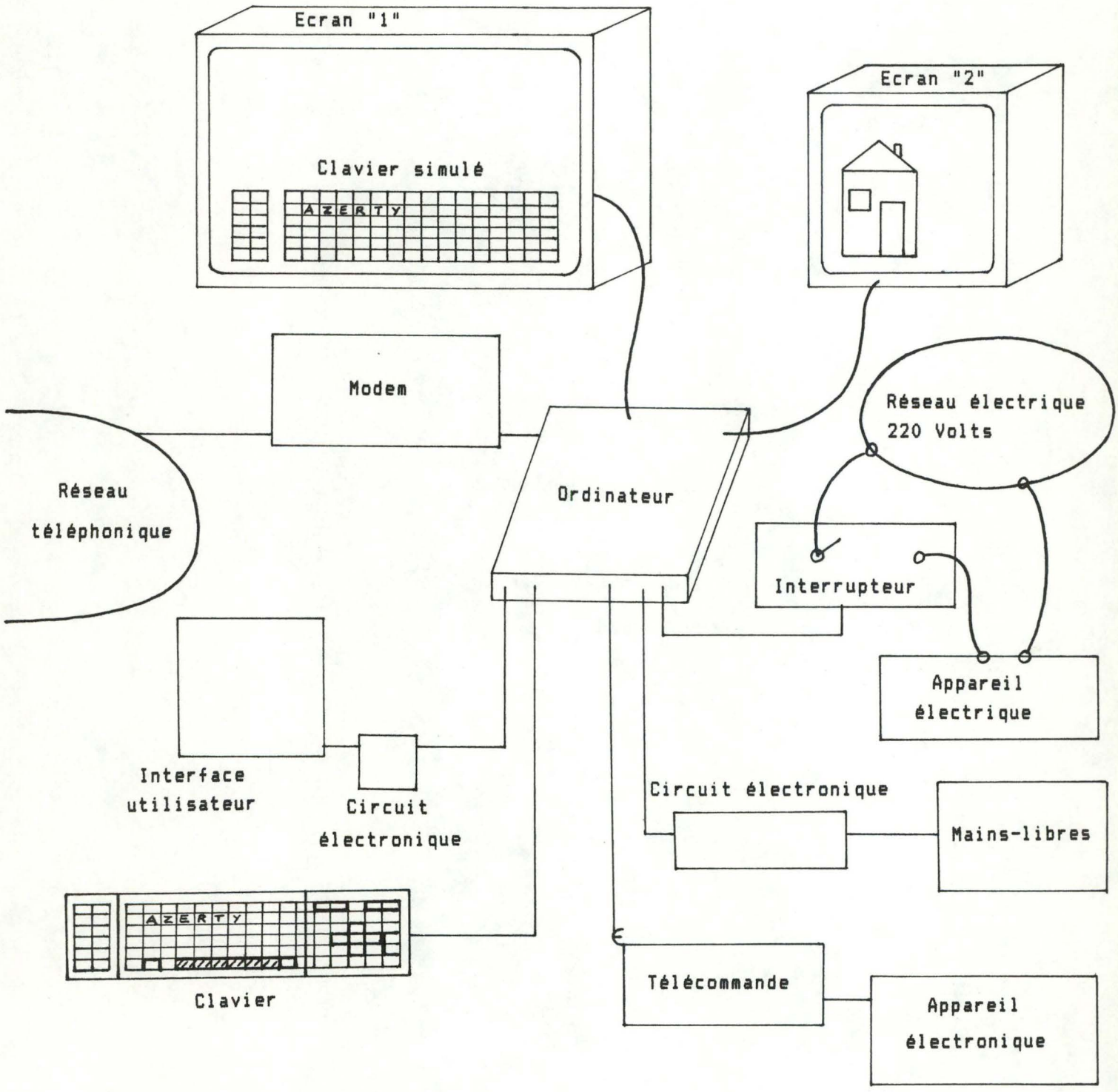

Figure 1.13 Représentation globale du système.

On consultera le manuel intitulé "Le Système Aidami Mons-Namur" [1] pour une description plus détaillée de l'architecture matérielle réalisée.

1.4. Objectifs liés à la réalisation sur ordinateur

1.4.1. Maintenance du système

L'architecture du système présentée au paragraphe 1.3 fournit au handicapé des services très divers. Cependant, chaque personne vit dans un environnement différent et il se peut que les services offerts par ce système soient insuffisants ou inadaptés pour une personne particulière. Citons par exemple la nécessité de changer l'interface entre le handicapé et la machine, ou le besoin d'adapter diverses commandes de télévision sur un même et unique système (car l'aspect d'une télécommande varie d'une marque d'appareil à l'autre), etc... On remarque donc que l'architecture du système ainsi que le programme assurant son fonctionnement sont susceptibles de subir de fortes modifications. Le but poursuivi est de pouvoir réaliser efficacement et rapidement ces changements au sein du système, c'est-à-dire d'assurer une maintenance facile du système. Pour effectuer cela, nous procédons de la manière suivante:

a) au niveau de l'architecture matérielle du système:

définir des catégories d'objets de l'environnement sur lequel l'utilisateur peut agir.

Par exemple:
- une première catégorie est composée de tous les appareils électriques qui peuvent être connectés ou déconnectés du réseau électrique 220 volts, à l'aide d'un interrupteur.
- une seconde catégorie comprend tous les appareils commandables à distance avec une télécommande sans fil.

Un circuit éléctronique distinct est réalisé pour chacune de ces catégories, ce qui permet de localiser les modifications électroniques à apporter lors du changement d'un appareil extérieur ou de tout autre composant relié à l'ordinateur tel qu'un interface, un modem, etc... La figure I.14 illustre ces concepts.

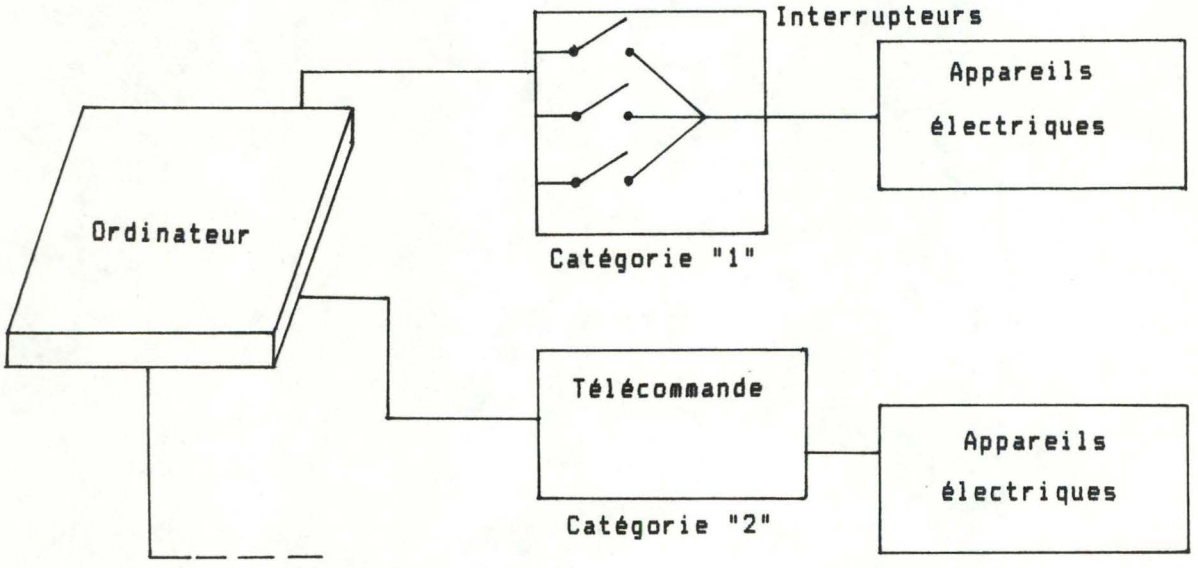

Figure 1.14 Catégories d'objets de l'environnement.

b) au niveau du programme qui offre les services relatifs à cette application:

décomposer le programme en plusieurs petits programmes, c'est-à-dire en plusieurs "modules". Chacun de ceux-ci doit effectuer un traitement spécifique. Le changement d'un ou d'une partie d'un service ne doit entrainer de modifications que dans un nombre minimum de modules. La figure 1.15 montre un programme composé de "n" modules. La décomposition d'un programme en modules est fortement facilitée lors de l'utilisation de langages de programmation de haut niveau tels que "Pascal", "C". C'est pourquoi nous nous efforçons de développer ce programme en utilisant au maximum un tel langage.

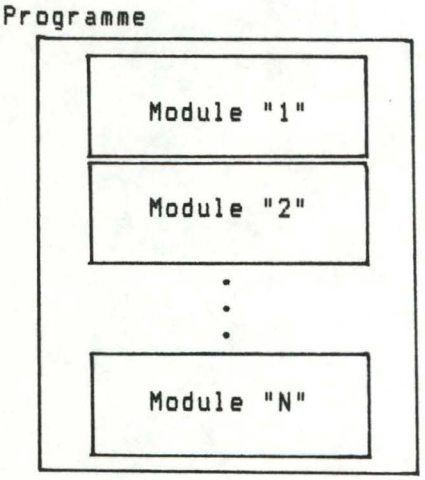

Figure 1.15 Décomposition d'un programme en modules.

1.4.2. Utilisation de logiciels "standards"

Etant donné l'importance croissante de l'informatique dans notre société, de nombreuses entreprises développent des logiciels "standards" qui offrent divers services aux utilisateurs. A titre d'exemple, citons:

- des logiciels de traitement de texte permettant de rédiger du courrier, de créer des documents, etc...
- des logiciels d'aide à la décision qui facilitent les tâches administratives et comptables dans une entreprise.
- des logiciels de graphisme permettant à l'utilisateur de créer ses propres dessins.
- etc...

Le programme développé dans cette étude doit permettre au handicapé d'utiliser de tels logiciels à l'aide de son seul interface.

Mais une restriction subsiste cependant. En effet, nous nous sommes plus particulièrement intéressés aux logiciels de graphisme grâce auxquels l'utilisateur peut laisser libre cours à son imagination pour créer ses propres graphiques et dessins. En général, ce type de logiciel fait

apparaître un curseur mobile en forme de flèche à l'écran. L'utilisateur doit être capable de déplacer ce curseur à un endroit quelconque de l'écran. On conçoit difficilement une manipulation de ce curseur à l'aide d'un interface utilisateur composé d'un seul et unique contact. Afin d'assurer un minimum d'efficacité pour déplacer le curseur, il est nécessaire de recourir à un interface plus élaboré tel qu'un joystick accompagné d'un contact (ou tout autre interface permettant de réaliser efficacement cette manipulation). Mais encore faut-il que le handicapé soit capable de manipuler un tel interface! Ce genre de service n'est par conséquent utilisable que par un nombre restreint de personnes handicapées.

1.4.3. Action sur l'environnement à tout moment

Lorsque le handicapé allume son ordinateur, il lance avant toute chose l'exécution d'un programme (faisant l'objet de ce mémoire) qui lui permettra de communiquer avec son environnement. (En pratique, le programme est lancé et exécuté automatiquement par l'ordinateur dès sa mise sous tension). A partir de cet instant, ce programme lui offre des services tels que "allumer une lampe", "éteindre la télévision". Or quel que soit le type d'utilisation de l'ordinateur, le handicapé doit pouvoir agir sur son environnement dans un bref délai car par exemple le téléphone peut sonner à tout moment, ou l'utilisateur peut éprouver le besoin d'allumer une lampe. Il est donc primordial que ce programme soit constamment accessible afin de rendre les services adéquats au handicapé, sans pour autant perdre ou arrêter le travail éventuellement en cours sur le micro-ordinateur.

1.4.4. Portabilité du programme réalisé dans cette étude

Le programme réalisé dans cette étude permet au handicapé d'agir sur son environnement, d'exécuter des logiciels standards, etc... Mais ce programme ne peut pas être exécuté sur n'importe quel ordinateur vendu dans le commerce. En effet, il existe un très grand nombre de types différents d'ordinateurs et tout programme ne peut être exécuté que sur un type unique d'ordinateurs (sauf dans le cas où l'entièreté du programme est réécrite sur un autre type d'ordinateur. Cette dernière solution nécessite en toute généralité une modification d'une partie assez importante du programme et représente une lourde tâche. Nous rejetons donc cette hypothèse). A notre avis, un des meilleurs choix qui peut être fait consiste à sélectionner un des types les plus répandus dans les entreprises et chez les utilisateurs particuliers. C'est pourquoi nous avons décidé de développer ce programme sur un micro-ordinateur du type "IBM-PC compatible". Néanmoins, même si deux ordinateurs sont de type identique, il est possible qu'un programme donné puisse s'exécuter sur le premier ordinateur et pas sur le second car ceux-ci peuvent être fabriqués par des constructeurs différents et présenter quelques petites différences. Toutefois, il est possible d'éviter cet inconvénient si on respecte la régle suivante: "contraindre le programme à n'utiliser que des objets communs à tous les ordinateurs du même type". On peut en conclure que le développement de ce programme ne pourra se faire qu'en prenant beaucoup de précautions afin d'assurer une portabilité maximale, c'est-à-dire qu'il puisse être exécuté sur le plus grand nombre possible d'ordinateurs du même type.

1.5. Principes généraux de réalisation du système

1.5.1. L'interface utilisateur

Nous avons signalé au paragraphe 1.4.2. qu'un des objectifs de cette étude consiste à fournir au handicapé les outils nécessaires lui permettant de réaliser du graphisme à l'aide de logiciels "standards". Nous en avons déduit que l'interface mise à la disposition de l'utilisateur devait être constituée d'un ensemble adéquat de composants afin de lui donner la possibilité de mouvoir de manière efficace le curseur visible à l'écran.

Après avoir demandé conseil à une personne atteinte de tétraplégie, l'interface utilisateur qui à notre avis semble le mieux adapté à ce genre d'application consiste en un joystick (manche à balais) accompagné d'un contact. Chacun de ces deux composants joue un rôle très spécifique:

* le joystick permet de déplacer le curseur à un endroit quelconque de l'écran.
* le contact permet de sélectionner l'objet désiré à l'écran, pointé par le curseur.

1.5.2. Accès aux services offerts par l'ordinateur

L'utilisateur doit pouvoir accéder rapidement et à tout instant à chacun des services offerts par l'ordinateur, à savoir les services "courrier", "téléphone", "commande", "télécommande", "outils", "logiciels" et "sortie".

Une manière de réaliser cela consiste à afficher ces services à l'écran sous la forme de menus. Un exemple de menu est présenté à la figure 1.16.

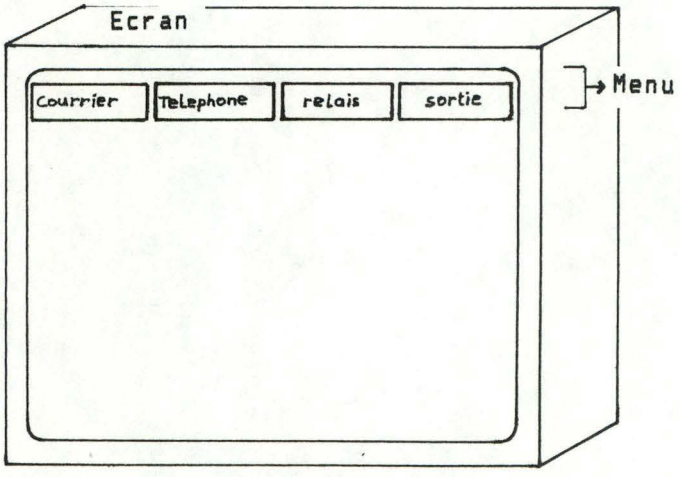

Figure 1.16 Exemple de menu affiché à l'écran.

Remarquons qu'en toute généralité, un service est décomposé en une série d'actions relativement élémentaires qui permettent à l'utilisateur d'accomplir des opérations plus spécifiques (voir annexe "A"). Ainsi, un système de menus hiérarchisés a été réalisé; le nombre de niveaux de cette hiérarchie étant fonction du degré de décomposition de chacun des services offerts. La figure 1.17 illustre le concept de menu hiérarchisé. Nous appelons "composants" les différents éléments figurant dans un menu.

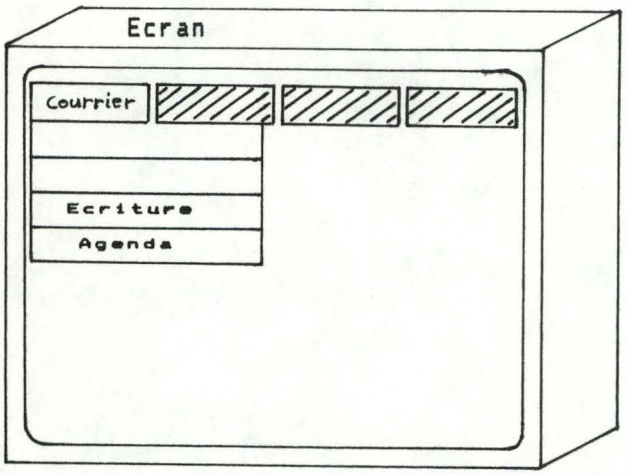

Figure 1.17 Exemple de menus hiérarchisés.

1.5.3. Sélection d'un objet dessiné à l'écran

La sélection d'un objet dessiné à l'écran peut être effectuée de deux manières différentes:

* par la technique du balayage.
* par le principe du pointage direct.

Ces deux techniques de sélection ont déjà été présentées au paragraphe 1.2.2. Ces objets peuvent être de nature très diverse et dans un souci de clareté, nous les avons classés dans 3 catégories:

- les composants des menus.
- les touches du clavier simulé.
- les boutons.

a) Les composants des menus

Puisque nous disposons d'un interface composé d'un joystick et d'un contact, nous avons décidé d'utiliser la technique du pointage direct pour sélectionner les composants des menus. La prise en considération d'un composant repose sur les faits suivants: l'utilisateur positionne le curseur sur le composant désiré à l'aide du joystick. Une pression sur le

contact contraste temporairement cet objet afin de montrer à l'utilisateur que celui-là vient d'être sélectionné. L'exécution du service ou de l'opération correspondante est ensuite prise en compte.

Cependant, pour des raisons de cohérence au sein de l'application, certains composants de menus peuvent être désactivés, c'est-à-dire que l'utilisateur se trouve dans l'impossibilité de les sélectionner. Citons par exemple le cas où l'utilisateur vient de sélectionner le service "téléphone": si on a sélectionné le composant "décrocher le combiné", ce dernier sera désactivé tant que le handicapé n'aura pas sélectionné le composant "raccrocher le combiné".

b) Les touches du clavier simulé

Le principe de sélection d'une touche du clavier simulé a déjà été exposé au paragraphe 1.2.2. Nous avons également jugé efficace de sélectionner les touches par la technique du pointage direct.

Cependant, le clavier est très souvent utilisé par le handicapé, surtout lorsqu'il s'agit de réaliser du traitement de texte. L'utilisateur doit donc manipuler constamment le bras de levier du joystick: ceci nécessite des efforts considérables de la part du handicapé. Nous avons décidé de laisser à l'utilisateur le choix du type de sélection des touches du clavier: soit le pointage direct, soit le balayage. En effet, la sélection d'une touche par la technique du balayage ne nécessite que l'appui sur le contact de l'interface utilisateur; ce mouvement est beaucoup moins éprouvant.

c) Les boutons

Les boutons sont représentés à l'écran sous forme de cercle ou de rectangle dont la taille n'est limitée que par la taille de l'écran disponible. A l'intérieur ou dans l'environnement immédiat de ce bouton figure généralement une chaîne de caractères signalant à l'utilisateur l'action déclenchée par la sélection de ce bouton. La figure 1.18 présente deux boutons affichés à l'écran: le sélection du premier provoque l'action "raccrocher le téléphone"; celle du second a pour effet d'allumer une lampe.

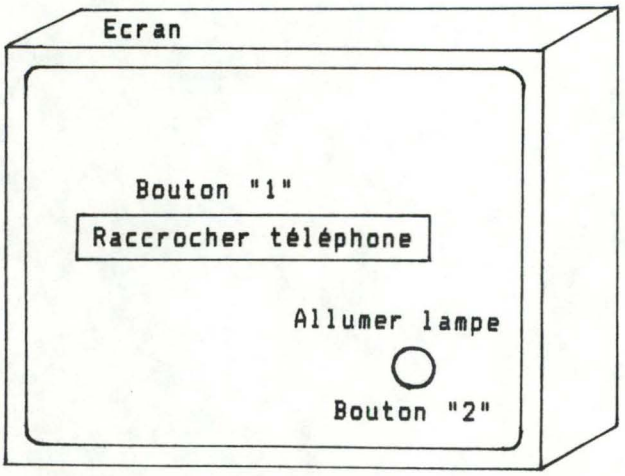

Figure 1.18 Représentation de deux boutons à l'écran.

En réalité, le clavier simulé et les menus dont nous avons parlé ci-dessus sont entièrement composés de boutons. Chaque touche du clavier simulé est un bouton ainsi que chaque composant d'un menu. La sélection d'un bouton peut donc également être réalisée par la technique du balayage ou par la technique du pointage direct.

1.5.4. Les affichages sur écran

Les affichages de messages et de graphiques sur l'écran sont réalisés en utilisant la technique du multi-fenêtrage. Cette technique est basée sur le principe suivant: l'écran mis à notre disposition peut être divisé en plusieurs zones de forme rectangulaire appelées "fenêtres". Une fenêtre représente un espace de l'écran dans lequel l'utilisateur peut afficher des caractères ou des graphiques. La figure 1.19 représente l'affichage de deux fenêtres à l'écran.

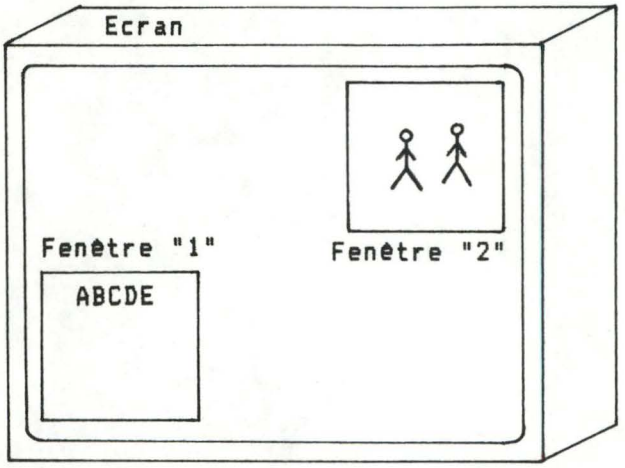

Figure 1.19 Représentation de deux fenêtres à l'écran.

Plusieurs fenêtres présentes à l'écran simultanément peuvent se recouvrir plus ou moins fortement sans pour autant perdre l'information contenue dans la fenêtre partiellement recouverte. Il est également possible de modifier la position d'une fenêtre affichée à l'écran. La figure 1.20 présente deux fenêtres au temps T1 dont l'une est partiellement recouverte. Une modification de la position de celles-ci permet de les visualiser entièrement. La représentation de l'écran au temps T2 différent de T1, illustre les deux fenêtres totalement visibles.

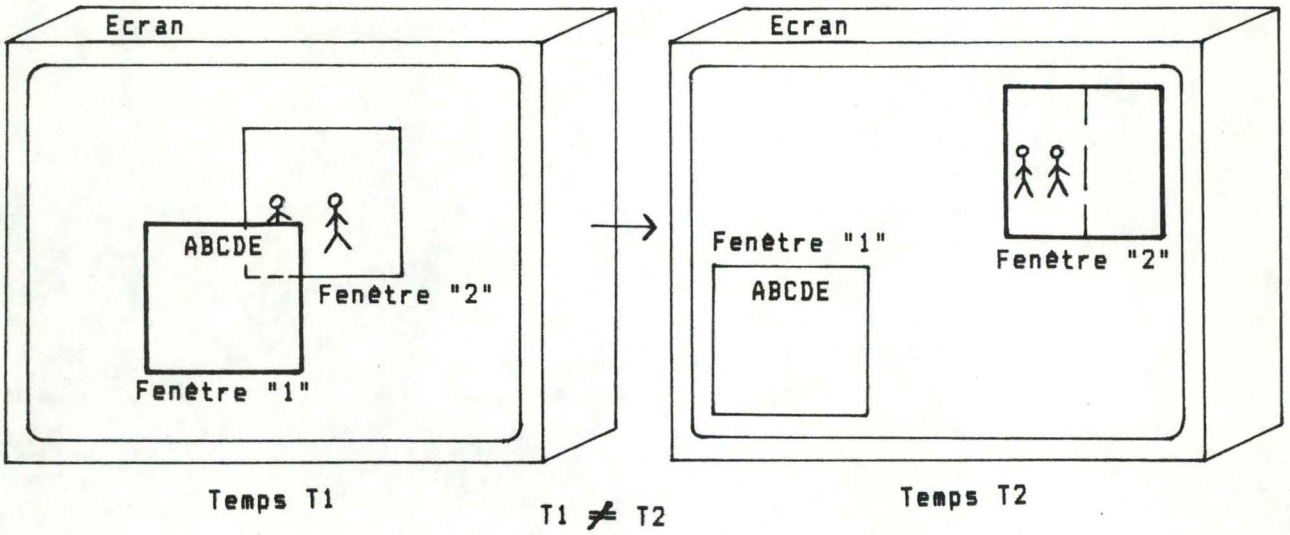

Figure 1.20 Modification de la position d'une fenêtre.

CHAPITRE 2

Chapitre 2 ARCHITECTURE LOGIQUE DU SYSTEME

2.1 Introduction

Les spécifications des besoins exposées au chapitre 1 nous permettent de réaliser une architecture logique de l'application d'aide aux handicapés. Effectuer une architecture logique consiste à décomposer l'application en plusieurs niveaux. Chacun de ces niveaux présente un certain degré d'abstraction de l'application, le niveau le plus élevé réalisant l'abstraction la plus forte. Nous commençons par exposer l'utilité d'une telle découpe dans le cadre de cette application, ensuite nous la présentons et expliquons en détail les raisons de ce choix de décomposition.

2.2 Découpe hiérarchique logique

2.2.1 Objectifs d'une découpe en niveaux

Les objectifs poursuivis par une découpe hiérarchique en niveaux dépendent de l'application à développer et des intentions du programmeur. En effet, le développement d'une application s'effectue selon un certain compromis: plus la découpe en niveaux de l'application est rigoureuse et efficace quant à la maintenance du programme, plus les performances en temps lors de l'exécution auront tendance à diminuer.

Il existe plusieurs types de hiérarchies présentant chacune des propriétés particulières et permettant d'atteindre un certain nombre d'objectifs. Une hiérarchie est représentée par un graphe composé de noeuds et d'arcs orientés qui relient ces noeuds.

L'application à développer peut être décomposée en un ensemble de composants qui réalisent chacun une partie des opérations que l'application doit effectuer. Par exemple, un composant offre les fonctionnalités: "afficher un objet quelconque à l'écran". Chaque noeud du graphe de la hiérarchie représente un composant et toutes les relations pouvant exister entre ces composants sont représentées par des arcs orientés. La figure 2.1 illustre une hiérarchie.

Sur cette figure, l'arc "1" part du composant "A" et aboutit au composant "B" nous disons dans ce cas qu'une relation existe entre le composant origine "A" et le composant destinataire "B".

Une hiérarchie est caractérisée par le fait que chaque noeud du graphe appartient à un certain niveau "i" et qu'il ne peut exister de relation qu'entre un noeud origine de niveau "i" et un noeud destinataire de niveau inférieur ou égal à "i". Par exemple, une relation du composant "B" vers le composant "A" sur la figure 2.1 est invalide.

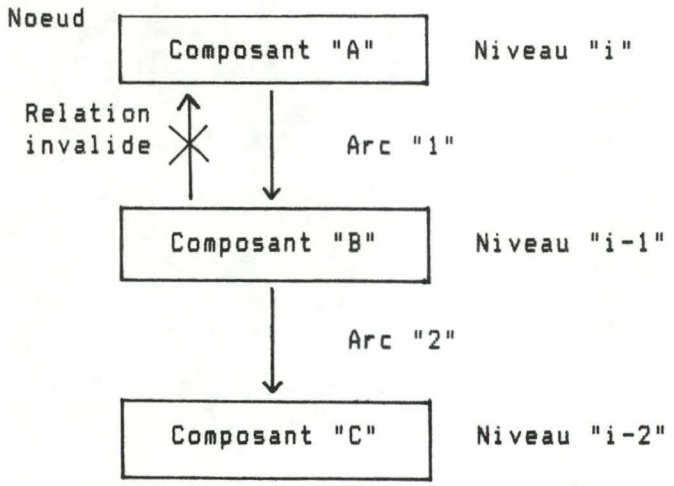

Figure 2.1 Représentation d'une hiérarchie.

Les composants identifiés dans une hiérarchie peuvent se décomposer en un ensemble de modules qui réalisent chacun un traitement plus spécifique. Nous avons défini précédemment un composant offrant les fonctionnalités: "afficher un objet quelconque à l'écran". Un module de ce composant assure par exemple la fonctionnalité "afficher un bouton à l'écran" ou "afficher le curseur à l'écran", etc...Nous disons également qu'un tel module constitue une primitive du programme. Chaque noeud du graphe de la hiérarchie peut donc être décomposé en un ensemble de "sous-noeuds" correspondant chacun à un module et entre lesquels peut également exister une relation.

Nous nous intéressons plus particulièrement ici à une hiérarchie de type "utilise" car celui-ci permet d'atteindre un grand nombre d'objectifs assurant une importante efficacité du produit logiciel développé. Un exemple de hiérarchie de type "utilise" est présenté à la figure 2.2. Sur cette figure, nous distinguons une relation du type "utilise" du composant "A" vers le composant "B". Nous pouvons dire que le composant "A" utilise le composant "B". Mais puisque chaque composant est constitué d'un ensemble de modules, nous pouvons dire qu'un module "a" du composant "A" utilise un ou plusieurs modules "b" du composant "B".

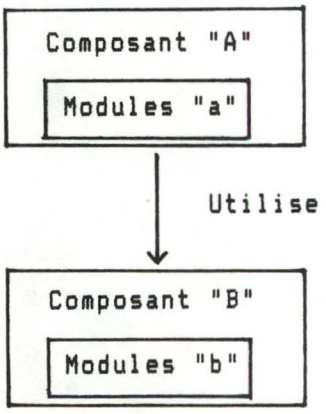

Figure 2.2 Exemple de hiérarchie de type "utilise".

Une relation de type "utilise" a pour signification: "un module "a" utilise un module "b" si et seulement si le fonctionnement correct du module "a" dépend de la disponibilité d'une version correcte du module "b". Nous entendons par "version correcte" d'un module, un module dont les spécifications et la conception sont corrects.

Nous présentons ici les principaux objectifs d'une structure hiérarchique de type "utilise":

* grande fiabilité du logiciel: les modules résultant de la découpe sont facilement testables individuellement.

* maintenabilité aisée: lorsque le logiciel est terminé, il est primordial de pouvoir le modifier rapidement, c'est-à-dire que le programmeur doit avoir la possibilité de contracter ou d'étendre facilement le logiciel sans trop de difficultés.

* élimination des redondances fonctionnelles: la structure hiérarchique permet d'éviter de créer deux modules distincts qui effectuent un traitement identique.

* réemploi de modules dans d'autres contextes fonctionnels: certaines parties du programme peuvent être utilisées par d'autres applications en pratiquant le moins de modifications possibles.

* portabilité: une indépendance maximale est assurée par rapport au hardware et au software (par exemple par rapport au système d'exploitation), en localisant cette dépendance dans des modules spécifiques du programme.

* factorisation du travail: l'application peut être décomposée en un ensemble de modules qui réalisent chacun une partie des opérations que l'application doit effectuer.

Pratiquer une découpe hiérarchique de type "utilise" de notre application est par conséquent indispensable pour les raisons suivantes:

** le projet AIDAMI est susceptible de subir de fortes modifications en raison des nombreux types différents d'interfaces utilisateur connectables sur l'ordinateur. Les appareils électriques à commander par l'intermédiaire de la machine peuvent également être très diversifiés.

** la portabilité est une qualité essentielle que ce programme doit posséder car celui-ci est destiné à être commercialisé et donc à être exécuté sur un grand nombre d'ordinateurs de type "IBM-PC compatible". Si des aspects particuliers dans le programme ne sont pas portables, il est indispensable de les localiser dans quelques modules du programme afin de pouvoir facilement apporter les modifications qui s'imposent.

** les autres avantages offerts par une hiérarchie de type "utilise" ne peuvent qu'améliorer les qualités du produit développé.

2.2.2 Présentation du schéma de la hiérarchie de l'application

Le schéma de la découpe en niveaux de l'application est présenté à la figure 2.3. Chaque niveau est composé d'un ou de plusieurs composants localisés sur une même latitude. Les flèches reliant les différents composants du graphe représentent les relations existant entre eux.

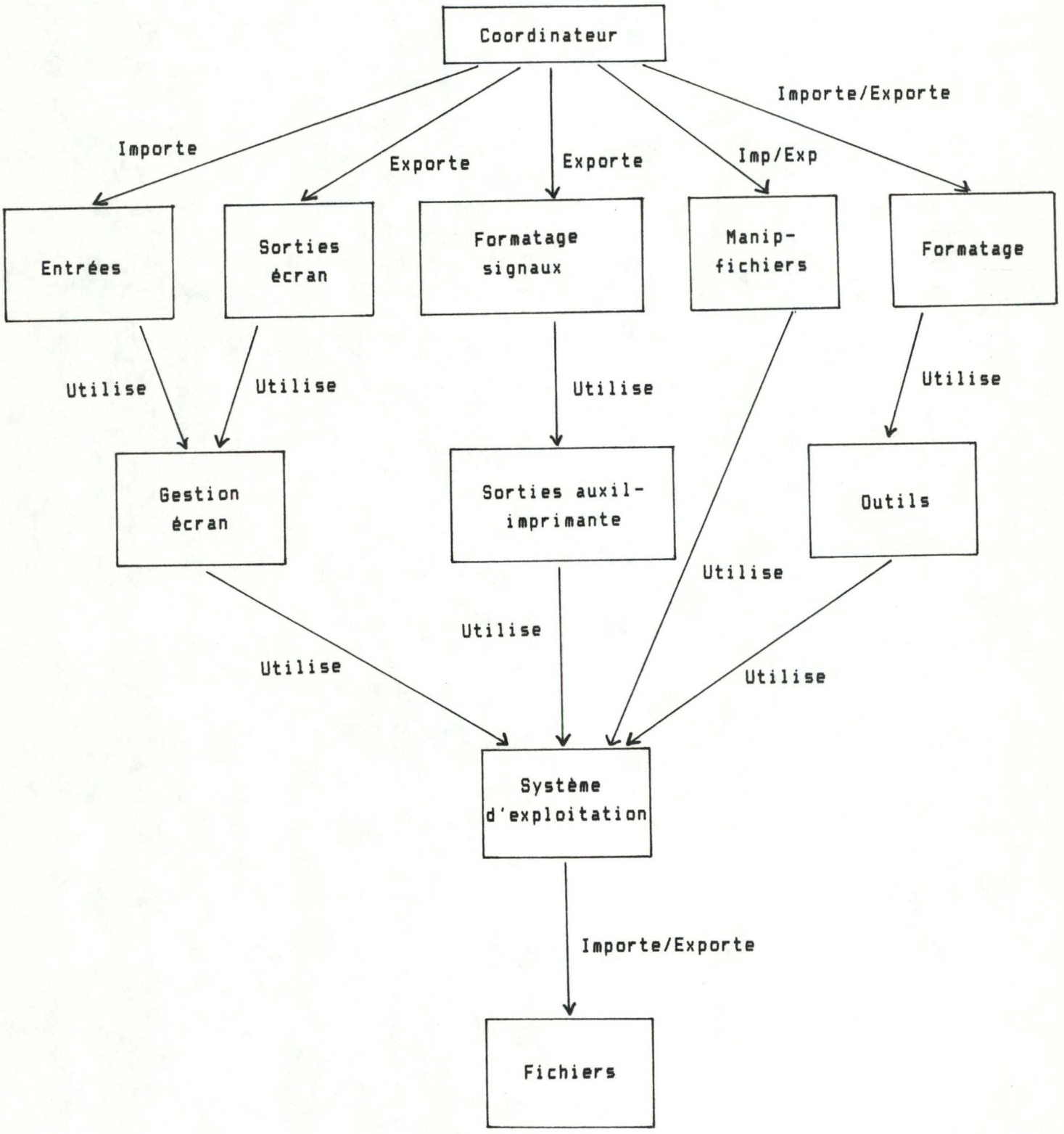

Figure 2.3 Découpe hiérarchique de l'application.

Nous distinguons trois types de relations sur cette figure:

a) la relation "importe": cette relation a pour signification:
un module "a" référence un module "b". Chaque module "b" ainsi référencé renvoie vers le module "a" des données et des informations qui sont utilisées par le module "a" (voir figure 2.4).

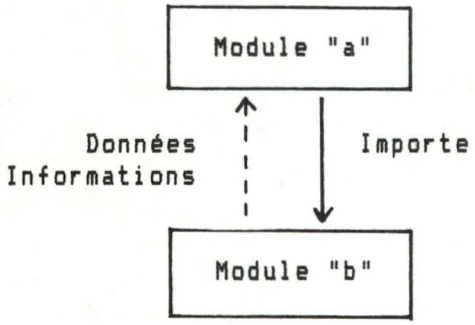

Figure 2.4 Signification de la relation "importe".

b) la relation "exporte": la signification de cette relation est:
un module "a" référence un module "b". Chaque fois qu'un module "a" effectue une telle référence, il émet en même temps des données et des informations qui sont utilisées par le module "b" référencé (voir figure 2.5).

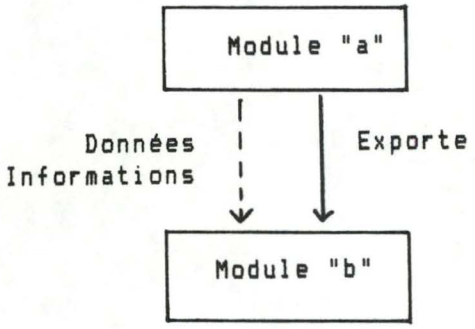

Figure 2.5 Signification de la relation "exporte".

c) la relation "utilise": la signification de cette relation a déjà été donnée au paragraphe 2.2.1.

Remarquons que le graphe de la figure 2.3 ne constitue pas entièrement une hiérarchie de type "utilise" en raison des relations "importe" et "exporte".

2.2.3 Analyse des composants du graphe de la hiérarchie

Nous décrivons brièvement les fonctionnalités offertes par chacun des composants du graphe de la hiérarchie présenté à la figure 2.3 et justifions leur emplacement et les relations qui y sont associées.

* Coordinateur : ce composant assure le séquencement des actions décrites dans le(s) processus en cours d'exécution. Il réalise ainsi dans un certain ordre des "appels" vers les modules des niveaux inférieurs et assure le transfert d'informations entre certains modules du graphe, lorsque cela est nécessaire. Le coordinateur pourra être aisément modifié et permettre la réalisation de multiples scénarios au sein du même programme.

* Les entrées : les entrées lisent des données ou des états relatifs aux différents périphériques de la machine tels que l'écran, le clavier, l'interface utilisateur. Ces états ou données sont mémorisés dans des variables du programme, et mis à la disposition des autres primitives du projet.

* Les sorties écran : ces modules permettent essentiellement de réaliser une présentation "agréable" de l'application à l'écran. Les services offerts par ce composant permettent de réaliser du multi-fenêtrage à l'écran.

* Formatage signaux : ces primitives formatent les codes binaires destinés à la commande des appareils électriques extérieurs. Un fichier contient les types d'appareils et leurs codes propres, de même que l'ordre et la vitesse dans lesquels ces codes doivent être émis. La modification des codes d'un appareil électrique est donc réalisée par un simple accès à ce fichier.

Ces trois composants "entrée/sortie" ont été réalisés pour assurer un maximum d'indépendance de l'application vis-à-vis des organes d'entrée/sortie de la machine.

* Gestion écran : ce composant contient un grand nombre de primitives indispensables au bon fonctionnement du multi-fenêtrage. La liste de ses actions élémentaires n'est pas exhaustive et il est possible de la compléter à souhait en fonction du degré de souplesse désiré pour la présentation de l'application à l'écran. Ces modules assurent une exécution correcte des primitives d'entrée/sortie relatives aux affichages à l'écran.

* Sorties auxil-imprimante : les primitives de ce composant envoient des codes binaires vers certains circuits électroniques réalisés par l'équipe "AIDAMI", en vue de commander les appareils électriques extérieurs. Le second rôle de ce composant est d'assurer l'envoi des signaux de contrôle vers une imprimante afin

d'effectuer une impression de fichier.

* Manip-fichiers : ce composant réalise des accès élaborés dans les fichiers de caractères et dans les fichiers d'enregistrements accessibles. Les primitives offertes utilisent directement les services du système d'exploitation car, comme nous le verrons par la suite, des opérations sur les fichiers y sont déjà implémentées.

* Formatage : les primitives de ce composant assurent une transformation du format de l'heure, des nombres arithmétiques et des dates en fonction de la nationalité de l'utilisateur. Un fichier préalablement créé permet de choisir ces différents formats. Pour fonctionner correctement, les primitives de ce composant doivent disposer d'une valeur à transformer générée par le composant "outils".

* Outils : ce composant offre des services généraux couramment utilisés tels que la recherche de l'heure, des calculs à effectuer, etc... Il est extensible selon les besoins de l'utilisateur. Les valeurs des dates, calculs et autres sont générées dans ce composant selon un format unique défini par le programmeur. Ces primitives utilisent directement les fonctions du système d'exploitation.

* Système d'exploitation : sa fonction principale consiste à offrir aux primitives des niveaux supérieurs un accès à toutes les ressources du système. Le système d'exploitation offre donc ses services à toutes les fonctions des niveaux supérieurs de l'application.

* Les fichiers : ceux-ci sont des ensembles de ressources mis à la disposition de l'application par l'intermédiaire du système d'exploitation. Parmis ces fichiers, on trouve notamment:

  * Des fichiers de programme source.
  * Des fichiers exécutables.
  * Des fichiers de données.

L'ensemble des primitives et des fichiers relatifs à chacun de ces composants est présenté à l'annexe "B".

CHAPITRE 3

CHAPITRE 3 ETUDE DU NIVEAU "SYSTEME D'EXPLOITATION" DE LA HIERARCHIE DE L'APPLICATION

3.1 Introduction

Pour commencer ce chapitre, nous déterminons un système d'exploitation "idéal" nécessaire pour assurer le bon fonctionnement de notre application. Pour implémenter aisément ce système d'exploitation, nous en effectuons une découpe hiérarchique. En effet, le composant "système d'exploitation" défini dans la découpe hiérarchique de l'application au chapitre 2 peut lui-même être décomposé en plusieurs niveaux comprenant chacun un certain nombre de primitives. Le système d'exploitation ainsi défini possède certaines particularités qui posent quelques problèmes pour implémenter ce projet. Une analyse détaillée de ceux-ci est présentée. Enfin, étant donné l'ampleur de l'application à mettre en oeuvre, nous définissons un sous-système utile qui sera implémenté sur micro-ordinateur.

3.2 Présentation du système d'exploitation "idéal" relatif à cette application

Nous présentons dans ce paragraphe un système d'exploitation "idéal", c'est-à-dire un système d'exploitation répondant aux besoins de notre projet afin d'assurer son bon fonctionement sur l'ordinateur. Rappelons que l'objectif primordial d'un système d'exploitation est de permettre à une application d'accéder à toutes les ressources de l'ordinateur tout en lui "cachant" les méthodes d'accès à ces données.

Nous avons vu au chapitre 1 que l'ordinateur met à la disposition du handicapé un ensemble de services qui permettent d'accroître son autonomie. Chacun de ceux-ci est composé d'une ou de plusieurs tâches réalisant un traitement bien particulier. Parmi ces tâches figure notamment la tâche "logiciel" qui offre à l'utilisateur la possibilité d'exécuter des logiciels "standards" vendus sur le marché, ainsi que ses propres programmes. Le bon déroulement de cette tâche sur l'ordinateur nécessite un système d'exploitation offrant un minimum de fonctionnalités.

L'exécution de la tâche "logiciel" sur la machine présente la particularité suivante: d'une part le handicapé communique avec l'ordinateur à l'aide de son interface; le programme développé dans cette étude doit assurer cette communication. Ce programme est donc à priori constamment exécuté par la machine car le handicapé doit pouvoir accéder aux services offerts par le système à tout moment. Si l'utilisateur désire lancer l'exécution d'un logiciel "standard" à l'aide de la tâche "logiciel" par une simple manipulation de son interface, il est primordial que cette exécution soit lancée par notre programme. D'autre part, lorsque le logiciel "standard" s'exécute, il est très probable qu'il demande à un certain moment à l'utilisateur d'entrer une donnée dans l'ordinateur. Dans le cas normal, cette donnée est introduite à l'aide du clavier. Mais puisque le handicapé ne sait pas se servir du clavier, il ne peut introduire une donnée dans la machine que par l'intermédiaire de son interface. Or nous savons que la communication entre le handicapé et l'ordinateur à l'aide de l'interface utilisateur ne peut être assurée que par notre programme. Il faut alors interrompre momentanément le logiciel

"standard" afin de permettre au handicapé d'introduire ses données. La figure 3.1 illustre ces concepts. Nous constatons que deux programmes sont susceptibles d'être exécutés par la machine chacun à leur tour. Ceci nous conduit à la notion de multiprogrammation.

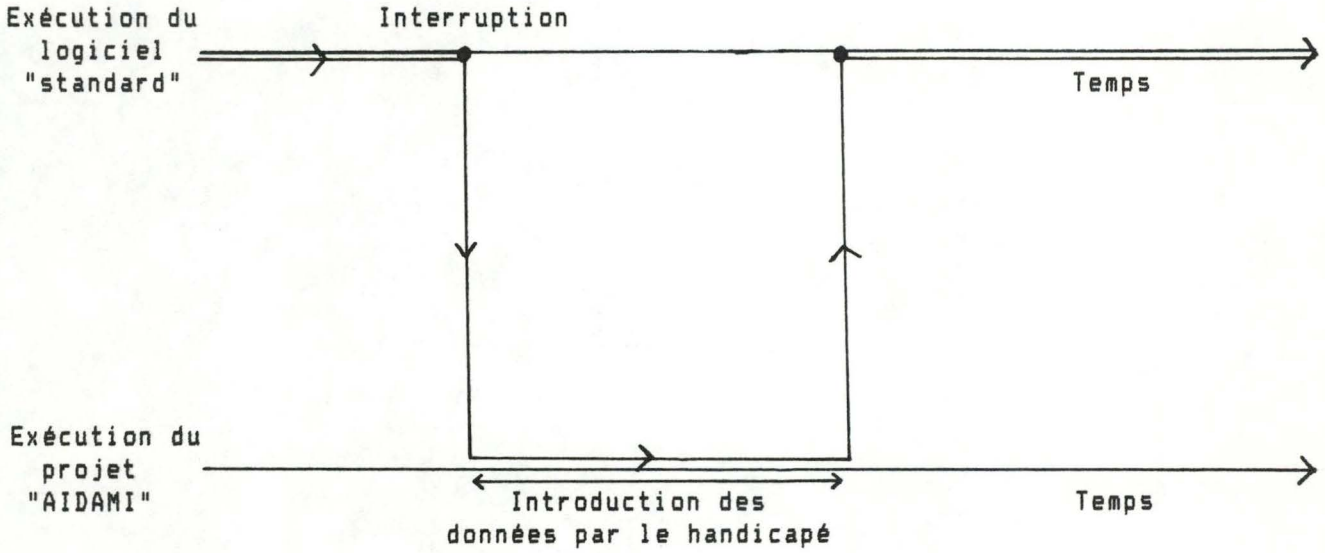

Figure 3.1 Interruption du logiciel "standard" lors de l'introduction d'une donnée par le handicapé.

On entend par multiprogrammation le fait que plusieurs programmes résidant en même temps en mémoire centrale sont exécutés de façon imbriquée dans le temps par un seul processeur (ou par plusieurs processeurs dans les gros ordinateurs).

Un cas particulier de multiprogrammation peut également être identifié dans cette application. En effet, le système doit mettre à la disposition de l'utilisateur diverses tâches accessibles à tout moment. Le développement d'un logiciel simulant un système multi-tâches semble une solution adéquate pour résoudre ce problème. Un système multi-tâches est caractérisé par le fait que plusieurs tâches peuvent être effectuées simultanément par l'utilisateur. A titre d'exemple, considérons la situation suivante: le handicapé est en train de rédiger une lettre à l'aide de la tâche "courrier". Subitement, le téléphone sonne: si l'on désire prendre la communication, l'exécution de la tâche "téléphone" doit être effectuée. Ceci aura pour effet d'interrompre momentanément la tâche "courrier", mais cette dernière pourra être reprise là où elle avait été interrompue dès que la communication téléphonique sera terminée.

Nous pouvons donc dire qu'un système multi-tâches est un cas particulier de la multiprogrammation en ce sens que plusieurs tâches peuvent être exécutées simultanément. La seule différence réside dans le fait que la multiprogrammation permet une exécution simultanée de plusieurs tâches appartenant chacune à des programmes différents alors que dans le cas d'un système multi-tâches, toutes les tâches font partie d'un seul et unique programme.

Chaque fois qu'une tâche est utilisée par le handicapé, un nouveau processus est créé dans le système et se charge de la mener à bien. En conséquence, puisque plusieurs tâches peuvent être utilisées en même temps, plusieurs processus peuvent exister à un moment donné et il est indispensable que le système d'exploitation en assure une gestion adéquate.

En conclusion, le système d'exploitation "idéal" doit permettre la multiprogrammation. D'autres fonctionnalités doivent également être disponibles dans ce système d'exploitation.

Nous présentons à présent les fonctionnalités du système d'exploitation "idéal" indispensables à notre application:

* gérer les processus
* permettre la multiprogrammation
* gérer les interruptions
* gérer les entrées/sorties
* gérer les fichiers
* gérer la mémoire centrale
* gérer les noms
* gérer les travaux

Remarque:
****

Une manière de rendre le système d'exploitation "idéal" tout à fait indépendant de l'application consiste à n'y autoriser des appels qu'en générant une interruption logicielle. Ceci présente l'avantage que le système d'exploitation forme ainsi un segment de code susceptible d'être utilisé par tout type d'application. Par contre, il est indispensable de se définir des zônes de communication communes à l'application et au système d'exploitation afin de pouvoir transmettre les paramètres nécessaires au bon déroulement des primitives utilisées. De plus, une gestion minimale de ces zônes doit être implémentée pour assurer la concordance entre les paramètres.

3.3 Présentation de la structure hiérarchique du système d'exploitation.

3.3.1 But d'une découpe hiérarchique du niveau "système d'exploitation"

Nous avons déjà signalé que nous désirons implémenter cette application sur un micro-ordinateur du type "IBM-PC compatible". Or le système d'exploitation d'une telle machine constitue en général un logiciel "standard" disponible sur le marché. Il existe donc plusieurs systèmes d'exploitation susceptibles d'être exécutés par l'ordinateur et présentant des fonctionnalités qui leur sont propres.

Le but de la découpe hiérarchique du système d'exploitation consiste à fournir au programmeur la possibilité d'identifier rapidement les modules définis dans cette découpe, qui doivent être modifiés si l'on change de système d'exploitation. Pour ce faire, nous nous basons sur le système d'exploitation "idéal" défini au paragraphe 3.2 qui offre toutes les fonctionnalités indispensables pour assurer le bon fonctionnement de notre

application. Il suffit ensuite d'identifier les composants et primitives déjà implémentés par le système d'exploitation choisi, et de développer dans notre application les primitives non définies dans ce système d'exploitation.

3.3.2 Présentation et analyse des composants du graphe hiérarchique du système d'exploitation

Afin de ne pas allourdir cet exposé, nous présentons et analysons les composants du graphe hiérarchique du système d'exploitation en annexe "D".

3.4 Problèmes liés à la multiprogrammation

3.4.1 Définition d'outils pour implémenter la multiprogrammation

Nous avons vu au paragraphe 3.2 que la tâche "logiciel" nécessite un système d'exploitation capable de supporter la multiprogrammation. Nous présentons dans ce paragraphe quelques outils permettant de l'implémenter.

Nous disposons d'un système dans lequel plusieurs processus peuvent coexister. Ceux-ci sont caractérisés par un des trois états suivants: actif, prêt ou bloqué. Un processus actif est un processus auquel un processeur est alloué; un processus prêt est en attente d'un processeur et dispose de toutes les ressources dont il a besoin pour s'exécuter; enfin, un processus bloqué est un processus en attente de ressources. Or puisque nous disposons d'une machine ne possédant qu'un seul processeur, il ne peut y avoir qu'un seul processus actif à un instant donné, les autres processus étant soit prêts, soit bloqués. (nous ne prenons pas en compte ici les éventuels processeurs de périphériques). Par conséquent, si plusieurs processus sont créés dans le système, il est indispensable de sauvegarder les données et résultats propres à chacun des processus interrompus; le gérant des interruptions s'en charge.

Par ailleurs, si un processus actif "A" est interrompu et si un nouveau processus "B" est activé et utilise le même segment de code que le processus "A", on dira que ce segment de code possède la propriété de réentrance. Un même et unique segment de code pourra donc à un instant donné assurer le bon déroulement de plusieurs processus. En pratique, il existe deux méthodes permettant d'allouer les instructions d'un segment de code à plusieurs processus:

a) interdire toute réentrance dans le segment de code en dupliquant ses instructions pour chacun des processus qui désire l'utiliser.

b) autoriser la réentrance, mais encore faut-il déterminer les segments de code du programme qui ne sont pas réentrants. En effet, certains segments de code du programme ne peuvent pas être réentrants afin d'éviter des incohérences dans le système. Ces segments sont dans ce cas appelés "section critique".

Il serait illusoire de choisir la solution interdisant toute réentrance car cela nous obligerait à dupliquer inutilement des segments de codes du programme. En effet, un segment de code devrait être recopié en mémoire centrale en autant d'exemplaires que de processus désireux de l'utiliser au même moment. La seconde solution autorisant la réentrance semble nettement mieux adaptée à condition de n'autoriser au maximum qu'un seul processus dans une section critique à un certain moment.

Une question se pose donc: comment limiter à un seul le nombre de processus dans une section critique?

* Soit en utilisant des primitives d'exclusion mutuelle. Ces primitives sont composées d'un ensemble d'instructions dont l'exécution ne peut être interrompue par un autre processus. On distingue deux types de primitives d'exclusion mutuelle qui réalisent essentiellement les opérations suivantes:

  - une primitive du premier type précède une section critique et a pour effet de positionner un verrou.
  - une primitive du deuxième type dépositionne un verrou et est localisée à la sortie d'une section critique.

  L'observation de la valeur du verrou permet de savoir si un processus utilise déjà une section critique particulière.

* Soit en groupant dans un ou plusieurs modules les différentes sections critiques. Ces modules sont appelés "moniteur" et ne sont accessibles que par un seul processus à la fois. L'exclusion mutuelle est ainsi assurée.

Avant de poursuivre l'exposé, nous donnons ici une brève définition: deux processus sont "concurrents" s'ils désirent accéder au même moment à une ressource commune du système.

Critique de ces deux méthodes:

* La méthode utilisant les moniteurs résoud les problèmes de la concurrence entre les processus qui désirent entrer dans une section critique. En effet, il faut assurer qu'un seul de ces processus entre dans une section critique à un moment donné et que les autres processus qui désirent également y entrer soient mis en attente jusqu'à ce que la section critique soit libérée. Les processus pénètrent donc chacun à leur tour dans la section critique. De plus, les moniteurs permettent de garder les avantages d'une construction de programme structurée.

* La méthode utilisant les primitives d'exclusion mutuelle est plus générale en ce sens que ces primitives peuvent être insérées partout dans le programme. Néanmoins, cette solution est plus dangereuse car l'oubli d'une primitive d'exclusion mutuelle dans le programme serait fatal.

Dans le cadre de ce projet, nous avons opté pour l'utilisation des primitives d'exclusion mutuelle. En effet, malgré que les moniteurs offrent l'énorme avantage de garder le programme structuré, nous constatons que, dans le cadre de cette application, les primitives d'exclusion mutuelle, plus simples à réaliser, permettent aussi de conserver cet avantage. L'explication de cette affirmation est présentée au paragraphe 5.5.

En résumé, les opérations à effectuer pour assurer la multi-programmation dans le cadre de notre application sont les suivantes:

* déterminer les sections critiques du programme.

* Insérer les primitives d'exclusion mutuelle dans le programme afin de garantir l'exclusion mutuelle.

* permettre la réentrance dans les modules susceptibles d'être utilisés par plusieurs processus au même moment.

3.4.2 Perturbations provoquées par un logiciel "standard" sur la multi-programmation

Rappelons que le programme développé dans cette étude doit permettre au handicapé d'utiliser les différents services qui lui sont offerts. Parmi ceux-ci figurent plus particulièrement l'utilisation d'un logiciel "standard". L'utilisateur est ainsi à même de disposer des outils du logiciel "standard" par une simple manipulation de son interface, sans introduire de commandes dans la machine via le clavier. Pour cela, toutes les actions provoquées par la pression d'une touche du clavier doivent être simulées par une manipulation adéquate de l'interface et par notre application de telle manière que le logiciel "standard" ne s'aperçoive pas de ce subterfuge.

Nous avons identifié trois perturbations majeures provoquées par le logiciel "standard" sur notre programme:

a) suspension de l'exécution de notre programme

b) utilisation simultanée de l'écran par notre programme et par le logiciel "standard"

c) accès simultané au système d'exploitation par notre programme et par le logiciel "standard"

Nous analysons ci-après ces différentes perturbations.

a) Suspension de l'exécution de notre programme

La première perturbation provoquée par le logiciel "standard" est caractérisée par le fait que son exécution suspend le déroulement de notre programme. Le retour à notre application ne peut se faire que lorsque le logiciel "standard" se termine, c'est-à-dire lorsqu'il a accompli toutes ses opérations. Or nous savons que ce logiciel ne peut généralement effectuer toutes ses opérations sans que l'utilisateur ne doive lui fournir certaines données. L'utilisateur étant ici le handicapé, l'introduction d'une telle donnée ne peut être réalisée que par l'intermédiaire de l'interface. Or la communication entre l'interface et la machine nécessite l'exécution de notre programme; il faut donc interrompre un bref instant le logiciel "standard" afin de permettre l'introduction de cette donnée dans la machine.

Le principe utilisé pour envoyer un caractère vers le logiciel "standard" est le suivant:

* interrompre le logiciel "standard".
* manipuler l'interface pour sélectionner le caractère à envoyer.
* exécuter notre application pour réaliser la simulation de la pression d'une touche au clavier et rendre ce caractère disponible pour le programme "standard".
* poursuivre l'exécution du logiciel "standard" après avoir restauré son environnement.

Mais si on interrompt le logiciel "standard" pour exécuter une partie de notre projet, pourquoi ne pas envisager par la même occasion de mettre à la disposition du handicapé tous les autres services implémentés dans dans notre programme?

Ainsi, lors d'une interruption du logiciel "standard", l'utilisateur aurait la possibilité de répondre à un appel téléphonique, d'allumer une lampe, etc... Nous allons dans le cadre de cette étude essayer de répondre à une telle exigence, mais cela nous crée un nouveau problème que nous décrivons ci-après.

b) Utilisation simultanée de l'écran par notre programme et par le logiciel "standard"

Le handicapé peut utiliser alternativement les services offerts par notre programme et les outils mis à sa disposition par le logiciel "standard". La sélection d'un service disponible dans notre projet est possible grâce à l'affichage à l'écran d'un menu. Mais le logiciel "standard" utilise lui aussi l'écran afin d'y afficher ses messages, graphiques ou résultats. Il est donc très probable que les affichages du logiciel "standard" effacent tout ou une partie des menus affichés par notre programme, et vice-versa, l'affichage d'un menu à l'écran peut effacer des données utiles générées par le logiciel "standard". Ceci constitue la seconde perturbation. Deux solutions peuvent être envisagées pour pallier à cet inconvénient:

* utiliser la technique du "multi-fenêtrage"
* utiliser deux écrans simultanément

1) La technique du multifenêtrage

La technique du multi-fenêtrage a déjà été exposée au paragraphe 1.5.4. Elle consiste à afficher à l'écran des fenêtres qui représentent un espace de l'écran dans lequel l'utilisateur peut afficher des caractères ou des graphiques. Ces fenêtres peuvent se recouvrir plus ou moins fortement sans perdre l'information contenue dans chacune d'elles. L'affichage des messages et dessins peut alors s'effectuer sans interférence si on attribue un certain nombre de fenêtres à notre application et d'autres fenêtres au logiciel "standard".

2) Utilisation simultanée de deux écrans

La seconde méthode permettant de séparer les objets affichés par notre programme de ceux du logiciel "standard" consiste à attribuer un écran à chacun d'eux. Cette solution est plus attrayante car elle permet d'attribuer l'entièreté d'un écran au programme et au logiciel "standard" (voir figure 3.2).

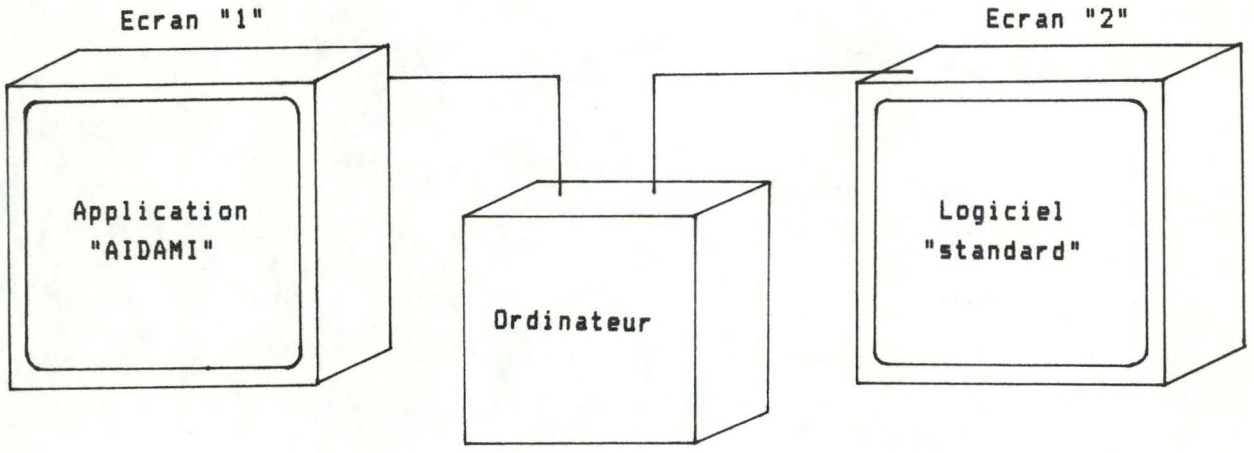

Figure 3.2 Utilisation simultanée de deux écrans.

Cependant, la solution idéale consiste à combiner ces deux méthodes: notre programme et le logiciel "standard" disposent chacun de leur écran respectif et peuvent le diviser en plusieurs zones ou fenêtres. Ceci se prête très bien pour un système de type multi-tâches car à chaque tâche est attribuée une fenêtre particulière.

c) Accès simultané au système d'exploitation par notre programme et par le logiciel "standard"

Nous souhaitons dans la mesure du possible implémenter cette application en utilisant un langage de programmation de haut niveau tel que "Pascal" ou "C". Or lorsque le compilateur d'un langage évolué traduit un code source en code objet, certaines instructions du langage évolué sont exécutées en effectuant des appels au système d'exploitation. Citons par exemple les instructions d'entrée/sortie du programme. Nous pouvons en

conclure que notre programme effectue de temps en temps des appels au système d'exploitation lorsqu'il en a besoin. Or les logiciels "standards" sont en principe eux aussi soumis à la même règle et utilisent également les services offerts par le système d'exploitation.

Le problème à résoudre est le suivant: lorsque l'exécution du logiciel "standard" est lancée, celui-ci effectue des appels au système d'exploitation dès qu'il en a besoin; nous pouvons dire que ces appels sont générés aléatoirement dans le temps. Or il se peut que le logiciel "standard" soit en train d'utiliser le système d'exploitation et que subitement, le téléphone se mette à sonner. Le handicapé doit, pour décrocher le téléphone, interrompre le logiciel "standard" et exécuter notre application. Mais la réalisation du service "décrocher le téléphone" a également pour effet de générer un appel au système d'exploitation pour assurer l'envoi d'un signal vers le "mains-libres". Nous pouvons conclure que dans ce cas, deux processus utilisent au même moment le système d'exploitation. Une telle situation serait tout à fait acceptable si tous les modules du système d'exploitation étaient réentrants. Or nous allons voir dans le paragraphe 3.4.3. que cette condition n'est pas satisfaite. Il est donc indispensable de trouver une solution permettant de limiter le nombre d'accès par des processus dans les sections critiques du système d'exploitation, tout en sachant que ce a dernier est également un logiciel "standard" et qu'il ne peut par conséquent pas être modifié.

Remarque:
****

Nous faisons l'hypothèse que des instructions telles que les instructions d'entrée/sortie utilisent les services offerts par le système d'exploitation. Par conséquent, on suppose qu'elles ne font aucun accès direct aux périphériques ou à d'autres ressources gérées en temps normal par le système d'exploitation. En effet, les programmes compilés et les logiciels "standards" doivent posséder la propriété de portabilité car ils sont susceptibles d'être exécutés sur un grand nombre d'ordinateurs du même type. Il serait donc peu probable que de tels produits accèdent directement aux organes d'entrée/sortie ou à d'autres ressources particulières car ceux-ci peuvent nécessiter une gestion différente au sein d'ordinateurs du même type.

3.4.3 Le problème de l'accés au système d'exploitation

L'accès par un processus aux services du système d'exploitation s'effectue par l'intermédiaire des interruptions. Lors d'un tel accès, ce processus exécute divers modules du système d'exploitation. Ceux-ci peuvent être séparés en deux classes: les modules formant des sections critiques et les modules réentrants. Nous donnons ci-après un bref aperçu des modules figurant dans l'une ou l'autre classe. Mais pour simplifier notre étude, nous nous sommes davantage orientés vers le cas plus particulier du système d'exploitation d'un micro-ordinateur d'autant plus que notre application est destinée à être implémentée sur une telle machine.

a) Les modules du type "section critique"

Le système d'exploitation permet d'accéder plus facilement à des ressources telles que l'écran, les disques, le clavier, etc... et n'oblige pas le programmeur à réécrire dans chacun de ses programmes ces différents modules d'accès. Mais une telle ressource n'est accessible que par un seul processus à un moment donné. En effet, le non respect de cette règle créerait des incohérences dans le système qui, en conséquence, réagirait de manière imprévisible. Considérons le cas d'une écriture sur disque: si le lecteur de disques écrit un caractère sur une piste du disque, il faut empêcher qu'un autre programme désireux lui aussi d'écrire des données sur le disque ne vienne interrompre brutalement cette écriture. Si cela se produisait, il serait très difficile, voire impossible de déterminer exactement l'état dans lequel était le lecteur de disques avant l'interruption, de même que la partie du caractère qui a déjà été mémorisée sur le disque avant l'interruption. Il est donc primordial dans ce cas d'attendre que le caractère soit entièrement mémorisé sur le disque et que le lecteur de disques soit revenu à un état cohérent et connu. Nous pourrions encore donner de tels exemples pour chacun des périphériques ou pour des ressources partagées accessibles par plusieurs processus au même moment en écriture et en lecture. Nous pouvons en conclure que les modules permettant l'accès à ces ressources constituent des sections critiques et qu'ils ne peuvent dans ce cas être accessibles que par un et un seul processus à un certain moment; de tels processus sont donc concurrents. Une étude du système d'exploitation mis à notre disposition est capitale afin de déterminer les modules soumis à de telles exigences.

b) Les modules réentrants

Il parait évident que tous les modules du système d'exploitation ne constituent pas des sections critiques. Dans ce cas, ces segments de code peuvent être élaborés de telle manière que plusieurs processus puissent simultanément exécuter leurs instructions. Comme nous l'avons déjà signalé, nous pouvons de cette manière éviter de devoir dupliquer le code de ces modules pour chacun des processus qui désirent les utiliser. Ces modules sont qualifiés de "réentrants".

Cependant, en toute généralité, un tel module effectue des opérations sur des données et fournit des résultats spécifiques à chaque processus qui l'utilise. Il faut donc s'assurer que tel résultat appartient bien à tel processus et qu'aucun résultat ainsi obtenu ne sera effacé avant que le processus auquel il est destiné en ait pris connaissance. Ceci peut être réalisé en associant à chacun des processus une zone mémoire dans laquelle tous les résultats relatifs à ce processus sont mémorisés. L'obtention des valeurs désirées nécessite un simple accès à cette zone.

En bref, on distingue deux types d'accès au système d'exploitation:

* un accès "concurrent" aux modules qui gèrent les périphériques et les ressources partageables accessibles en écriture et en lecture.

* un accès par plusieurs processus simultanément aux autres modules du système d'exploitation s'ils sont réentrants ou s'ils peuvent être rendus réentrants.

3.5 Définition d'un sous-système utile

Le programme que nous désirons implémenter dans le cadre de cette étude offre au handicapé un certain nombre de services. Mais le développement de ce programme dans son intégralité nécessite une quantité importante de travail. C'est pourquoi nous avons décidé de définir un sous-système utile. Nous entendons par "sous-système utile" une réalisation partielle de l'application, en ce sens que nous ne développons et n'implémentons qu'un sous-ensemble des services et tâches décrits dans l'annexe "A". Nous énumérons ci-après les services réalisés dans cette étude.

* Le service "téléphone" dont les actions suivantes sont développées:

  - sélectionner un numéro.
  - décrocher le téléphone.
  - raccrocher le téléphone.

* Le service "logiciel" avec les actions:

  - afficher le clavier.
  - afficher les logiciels exécutables sur IBM-PC.

* Le service "outils" avec les actions:

  - afficher l'horloge.

* Le service "sortie".

La sélection d'un de ces services est réalisée par l'intermédiaire d'un menu principal dans lequel figure chaque nom de service.

Pour des raisons de facilité de développement de l'application, tous les déplacements du curseur à l'écran sont générés par les touches fléchées du clavier et le bouton du joystick ou de la "souris" est remplacé par les touches de fonction "F9" et "F10". En effet, les micro-ordinateurs des facultés ne sont pas équipés en permanence d'une "souris" ou d'un joystick. Ceci ne présente pas d'inconvénients majeurs car il suffit de modifier quelques modules du programme pour assurer le fonctionnement correct de la la "souris" ou du joystick qui constituent l'interface utilisateur réel du handicapé.

De plus, nous avons décidé de ne pas utiliser le second écran pour la simple raison que cela demande énormément de travail de programmation (la justification de cela est exposée dans le chapitre 5), ce qui n'est pas l'objectif poursuivi par cette étude. La gestion des fenêtres est minimalisée pour la même raison.

Le système d'exploitation que nous avons décidé d'utiliser dans le cadre de cette étude s'intitule "DOS" (Disk Operating System). Le choix de ce système d'exploitation se justifie par le fait qu'il est utilisé à grande échelle dans les micro-ordinateurs du type "IBM-PC compatible".

Enfin, l'implémentation de ce programme est réalisée en utilisant le langage "Turbo-Pascal" accompagné de l'utilitaire "Turbo-Graphix" qui facilite la gestion des fenêtres de l'application.

CHAPITRE 4

CHAPITRE 4 PRINCIPES DE FONCTIONNEMENT DE L'IBM-PC

4.1 Introduction

Nous présentons dans ce chapitre quelques principes de fonctionnement d'un ordinateur du type "IBM-PC compatible". Ces explications permettront de comprendre plus aisément les décisions prises pour réaliser ce projet. Nous supposons que le lecteur a déjà acquis quelques notions de base sur les micro-ordinateurs. Il se peut d'ailleurs qu'il connaisse déjà les informations que nous donnons ici; dans ce cas, nous lui proposons de passer directement au chapitre 5.

4.2 Les registres de l'IBM-PC

Les registres de l'IBM-PC sont des zones mémoires auxquelles le processeur peut accéder très rapidement. Ceux-ci peuvent être répartis en quatre groupes:

1) les registres généraux: AX, BX, CX et DX.

Ces quatre registres sont utilisés pour des stockages temporaires de résultats intermédiaires fréquemment utilisés.

2) Les principaux registres d'index et de pointeur: SP, BP et IP.

Fonctionnalité de ces registres:

SP: pointeur de pile: registre contenant un pointeur vers l'élément figurant au sommet de la pile du système. Une pile est une zone de la mémoire qui permet de mémoriser les valeurs contenues dans les registres de l'IBM-PC et les variables déclarées dans les procédures d'un programme.

BP: pointeur de base: registre permettant d'accéder aux paramètres de la pile.

IP: pointeur d'instruction: registre contenant l'adresse relative de la prochaine instruction à exécuter.

3) Les registres de segment: CS, DS, SS et ES.

Un segment est une zone de la mémoire centrale attribuée à un programme pour y stocker des informations.

Fonctionnalité de ces registres:

CS: segment de code: registre contenant l'adresse du début du segment dans lequel figure le code des instructions du programme à exécuter.

DS: segment de données: registre contenant l'adresse du début du segment dans lequel figurent les variables, données et tampons du programme à exécuter.

SS: segment de pile: registre mémorisant l'adresse de début de la pile utilisée par le programme.

ES: extra segment: registre contenant l'adresse du début du segment utilisé pour des opérations spéciales.

La taille de chacun des segments identifiés de la sorte ne peut excéder 64 Kbytes.

Dans un tel segment, une cellule mémoire particulière peut être adressée de la manière suivante: connaissant la valeur du registre de segment dans lequel se trouve la cellule de la mémoire désirée et la valeur du déplacement (adresse relative) de cette cellule par rapport au début du segment, l'adresse absolue de la cellule est donnée par l'expression suivante:

adresse absolue = "(valeur du registre de segment * 16) + valeur du déplacement de la cellule dans ce segment"

La figure 4.1 présente quatre segments localisés dans la mémoire centrale. Le segment numéro 4 est pointé par un registre de segment et contient la cellule mémoire désirée.

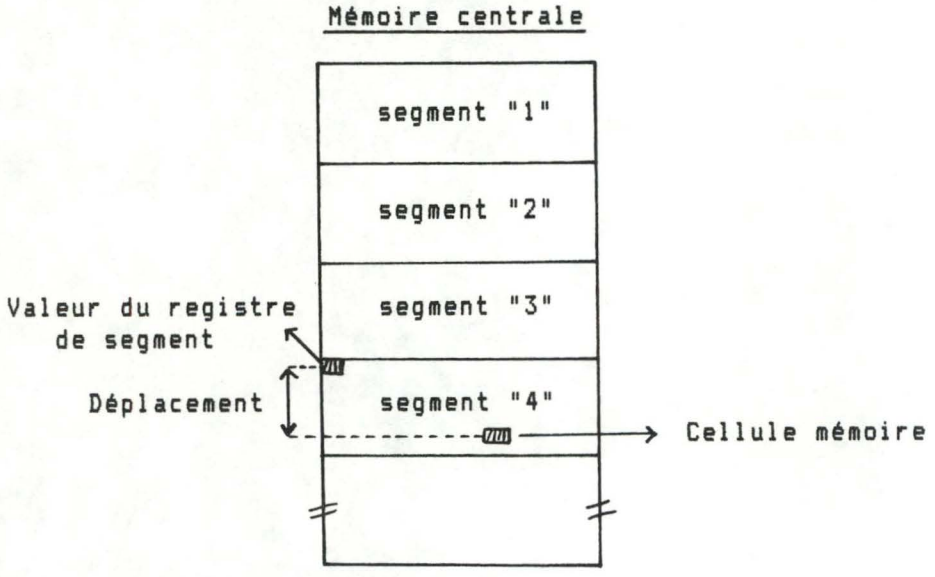

Figure 4.1 Adressage absolu d'une cellule de mémoire.

Considérons l'exemple suivant:

soit: * CS = 0C38h ("h" signifie que le nombre est représenté en base 16)
* déplacement = 2411h

Le calcul suivant permet d'obtenir l'adresse absolue en mémoire centrale de cette cellule:

l'adresse absolue = (0C38h * 10h) + 2411h
= C380h + 2411h
= E791h

4) Le registre des indicateurs: FR

Ce registre est quelque peu spécial en ce sens que chaque bit qui le constitue est interprété individuellement. Un bit représente l'état d'un indicateur à deux positions ("1" ou "0") utilisé par le système.

Nous citons plus particulièrement:

- le bit d'autorisation d'interruption IF qui indique l'autorisation ou l'interdiction d'un type particulier d'interruptions.

4.3 Présentation sommaire de la mémoire centrale de l'IBM-PC

La mémoire centrale d'un IBM-PC est divisée en plusieurs zones, chacune jouant un rôle bien spécifique dans l'ordinateur. Celles-ci sont présentées à la figure 4.2(a).

L'espace d'adressage de la mémoire s'étend de l'adresse 00000h à l'adresse FFFFFh; ceci permet de disposer d'une mémoire centrale dont la taille maximale atteind 1 M bytes. En pratique, celle-ci est inférieure à ce maximum et est souvent limitée à 256 Kbytes ou à 640 Kbytes.

La première zone de la mémoire centrale est dénommée "mémoire standard". Celle-ci comprend plusieurs parties: (voir figure 4.2(b))

* les vecteurs d'interruption du système d'exploitation.
* les vecteurs d'interruption utilisables par le programmeur.
* les zones de communication du système d'exploitation.
* les fichiers comprenant le code du système d'exploitation.
* des zones réservées.

La zone "user memory" est disponible pour l'utilisateur. Celui-ci peut y mémoriser ses fichiers, ses programmes, etc...

Une zone contient le buffer des écrans vidéo permettant de mémoriser les caractères affichés à l'écran. Cette zone est communément appelée "RAM vidéo" et débute à l'adresse B0000h.

La zone "ROM" située entre les adresses F4000h et FFFFFh n'est accessible qu'en mode lecture. Cette portion de mémoire comprend en particulier le segment non effaçable du système d'exploitation débutant à l'adresse FE000h.

Enfin, d'autres zones de la mémoire sont réservées pour assurer le bon fonctionnement de l'ordinateur.

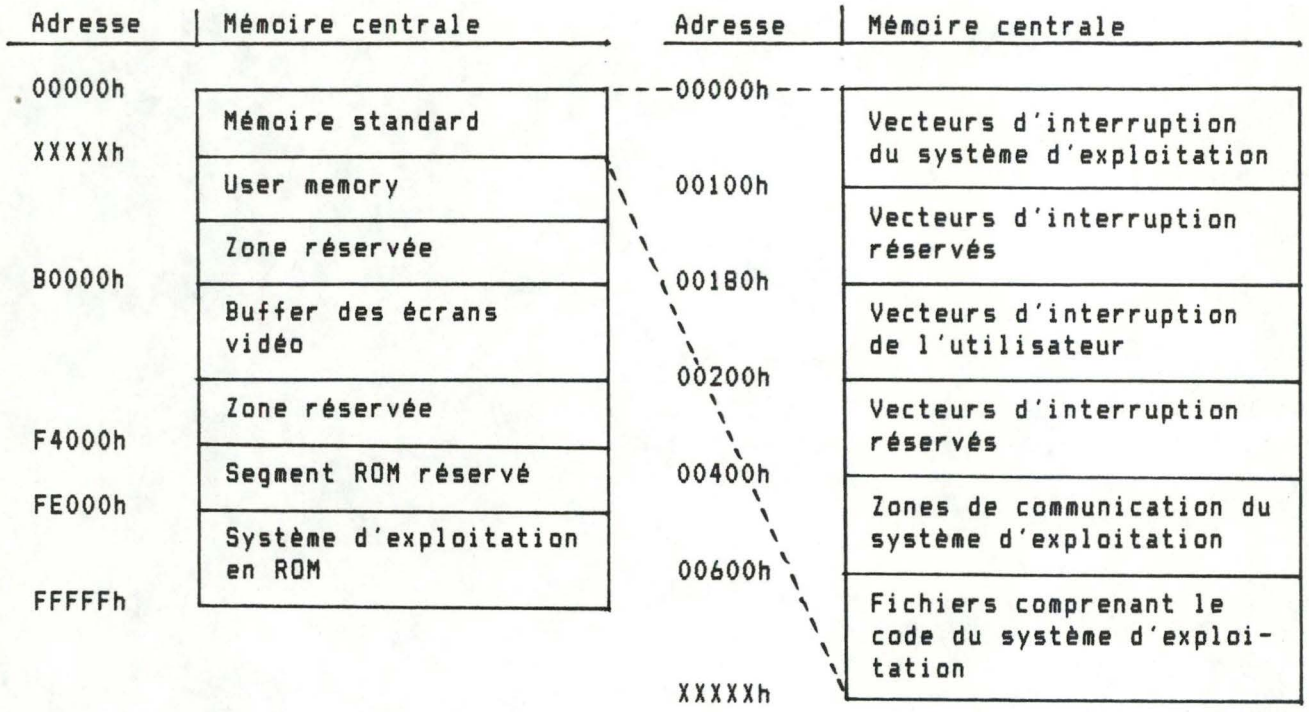

Figure 4.2 (a)
Zones de la mémoire centrale

Figure 4.2 (b)
Contenu de la "mémoire standard".

4.4 Segmentation d'un programme chargé en mémoire centrale

Lorsque l'utilisateur désire exécuter un programme, un certain nombre d'opérations doivent être effectuées. Nous supposons que ce programme exécutable est initialement mémorisé sur un support magnétique et qu'il n'est pas déjà chargé en mémoire centrale. Dans ce cas, les opérations suivantes sont effectuées:

* réservation d'un espace libre de la mémoire centrale destiné à contenir le programme à exécuter.

* chargement du programme dans cet espace libre.

* initialisation de paramètres de la machine et plus particulièrement des registres de segment CS,DS,ES et SS dont nous avons parlé au paragraphe 4.2. Chaque registre de segment identifie le début d'un segment particulier de la mémoire et les segments pointés par les registres CS, DS et SS sont inclus dans l'espace réservé avant le chargement du programme (voir figure 4.3).

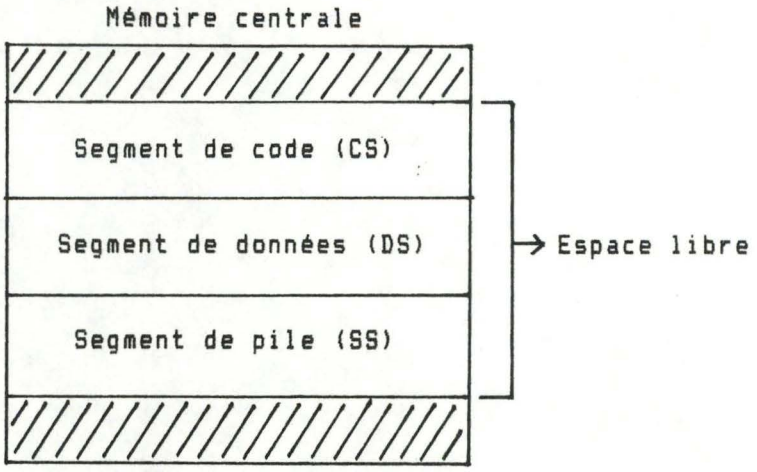

Figure 4.3 Segmentation d'un espace libre de la mémoire centrale.

Rappelons que les segments définis de la sorte ne peuvent excéder 64 Kbytes. Lorsque la zone occupée par le code ou par les données est inférieure à ce maximum, ils peuvent se chevaucher partiellement sans pour autant nuire au bon fonctionnement du système (voir figure 4.4).

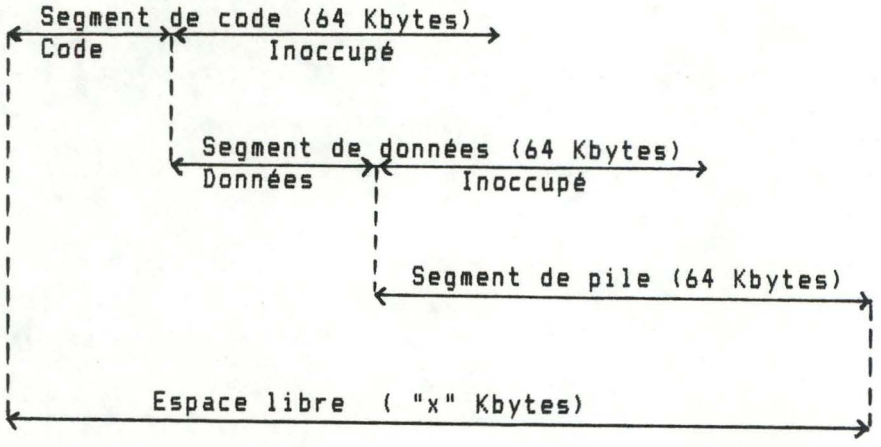

Figure 4.4 Recouvrement partiel des segments.

Enfin, signalons que les 256 premiers bytes du segment de code du programme chargé en mémoire centrale sont occupés par le préfixe de segment de programme (PSP) qui est créé lors du chargement du programme. Ce préfixe contient des informations nécessaires au système d'exploitation DOS.

4.5 Le système d'exploitation DOS

Nous avons décidé d'utiliser le système d'exploitation DOS pour implémenter le projet "AIDAMI". Nous présentons dans ce paragraphe quelques informations utiles le concernant.

4.5.1 Organisation du système d'exploitation DOS en mémoire centrale

Plusieurs segments de code constituent le système d'exploitation DOS:

- le segment IBMBIO.COM.
- le segment IBMDOS.COM.
- le segment COMMAND.COM.
- le segment ROM BIOS.

Chacun de ceux-ci occupe une zone particulière en mémoire centrale; leurs emplacements sont représentés à la figure 4.5. Chaque zone est caractérisée par une adresse de début et une adresse de fin. Lorsqu'une adresse est représentée par "xxxxxh", cela signifie que sa valeur est susceptible de changer en fonction de la version du DOS utilisée. Notons que le segment ROM BIOS n'est pas à proprement parler un segment du système d'exploitation DOS car il réside en permanence dans la mémoire morte de l'ordinateur et peut par conséquent être utilisé par d'autres systèmes d'exploitation. Cependant, nous le considérons comme tel dans cette étude puisqu'il est utilisé par DOS. La figure présente la répartition suivante:

* les vecteurs d'interruption du système d'exploitation s'étendent de l'adresse 00000h à l'adresse 00100h.
* les zones de communication du système d'exploitation figurent entre les adresses 00400h et 00600h.
* les fichiers du système d'exploitation dont l'adresse initiale est 00600h, ont une adresse terminale indéterminée car elle varie en fonction de la version du DOS utilisée.

D'autres emplacements en mémoire sont également alloués au système d'exploitation afin d'assurer son bon fonctionnement. Nous explicitons ci-après ces zones.

* vecteurs d'interruption du BIOS: cette partie de la mémoire contient des vecteurs d'interruption dont la plupart permettent d'exécuter les primitives et les fonctions contenues dans le segment ROM BIOS.

* vecteurs d'interruption du DOS: cette zone comprend les vecteurs d'interruption permettant d'exécuter les primitives et les fonctions contenues dans les modules IBMBIO.COM, IBMDOS.COM et COMMAND.COM.

* Zone de communication BIOS: ce segment de mémoire permet de mémoriser les valeurs des variables d'état des différents périphériques du système et certaines données nécessaires pour assurer le bon fonctionnement du système. Cette zone est utilisée par le segment ROM BIOS.

* Zone de communication DOS: cette zone est utilisée par les fichiers IBMBIO.COM, IBMDOS.COM et COMMAND.COM pour mémoriser les valeurs de leurs variables intermédiaires.

* Les fichiers IBMBIO.COM, IBMDOS.COM et COMMAND.COM (1ère et deuxième partie) constituent le système d'exploitation DOS proprement dit. Ceux-ci sont chargés en mémoire centrale lors de la mise sous tension de l'ordinateur et restent résidants en mémoire centrale, sauf la seconde partie du fichier COMMAND.COM qui peut être rechargée pendant le fonctionnement de la machine. Ces modules offrent des services tels que:

- gestion des fichiers
- accès simplifiés aux périphériques
- exécution de programmes exécutables

En général, les instructions d'entrée/sortie des programmes utilisent les primitives offertes par ces modules.

* Buffers et zones de contrôle du DOS: cette zone de la mémoire contient les buffers d'entrée/sortie vers les périphériques et des variables de contrôle nécessaires au DOS.

* Le segment ROM BIOS: cette partie de la mémoire contient les modules les plus primitifs du système d'exploitation et est localisée dans la mémoire morte de l'ordinateur.

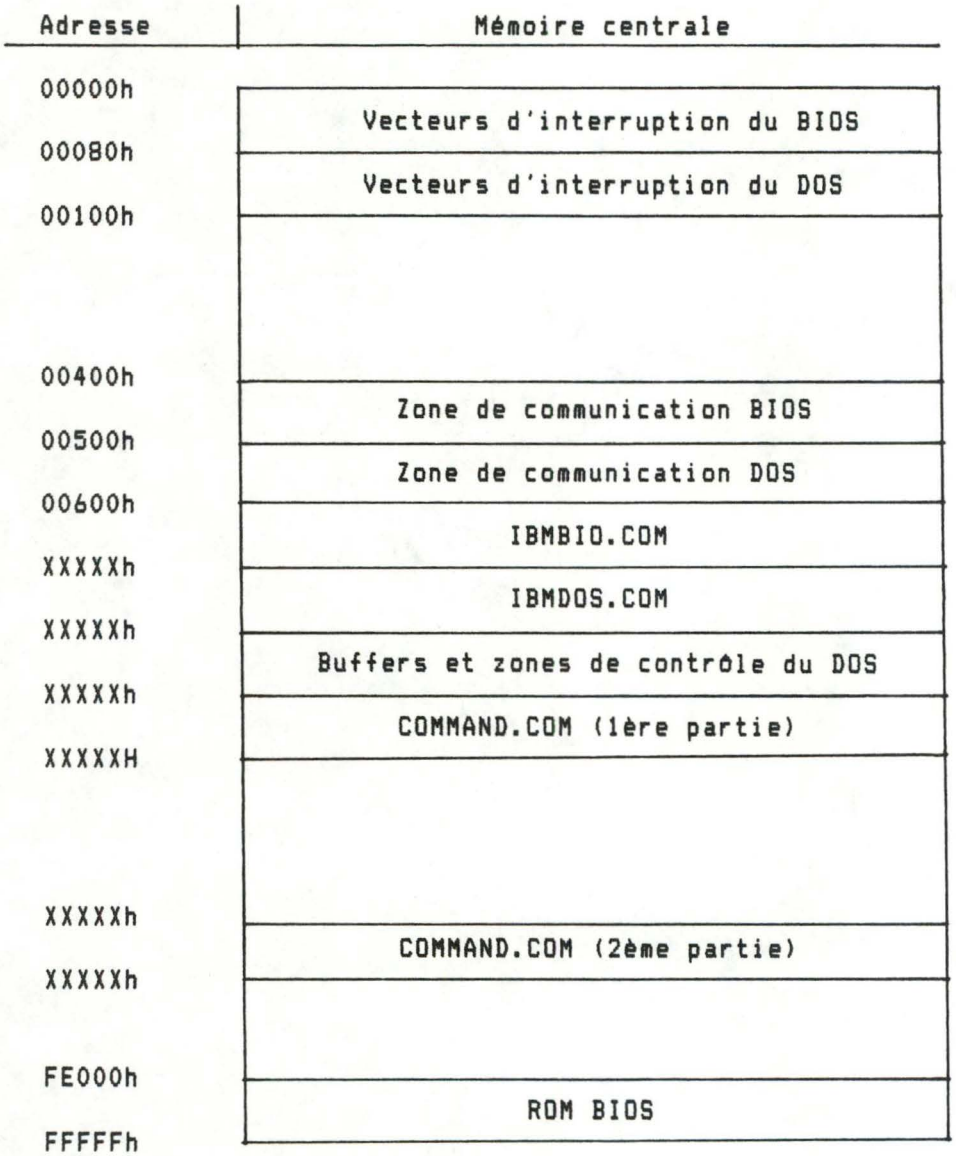

Figure 4.5 Emplacement des modules du système d'exploitation en mémoire centrale.

Fonctionnalité des segments de code du système d'exploitation

Nous avons identifié quatre segments de code dans le système d'exploitation DOS. Nous présentons brièvement les fonctionnalités de chacun d'eux.

* le segment ROM BIOS: ce segment assure l'initialisation de la machine lors de son démarrage ainsi que l'accès et la gestion des périphériques d'entrée/sortie.

* le segment IBMBIO.COM constitue une interface de bas niveau avec le segment ROM BIOS. Il gére les entrées/sorties des périphériques.

* le segment IBMDOS.COM constitue une interface de haut niveau vis-a-vis des programmes d'application. Il s'occupe notamment de la gestion des fichiers. Ce segment utilise les fonctions offertes par le module IBMBIO.COM.

* le segment COMMAND.COM traite les différentes commandes introduites dans l'ordinateur par l'utilisateur.

On peut remarquer que les segments du système d'exploitation ainsi définis forment une hiérarchie de type "utilise". La figure 4.6 présente cette hiérarchie.

Le programme d'application utilise les primitives du segment IBMDOS.COM lorsque par exemple il désire faire des entrées/sorties. Le segment IBMDOS.COM utilise ensuite le composant IBMBIO.COM qui lui-même utilise le segment ROM BIOS. Notons cependant qu'un programme d'application peut également faire appel aux composants du système d'exploitation autres que le segment IBMDOS.COM selon ses besoins.

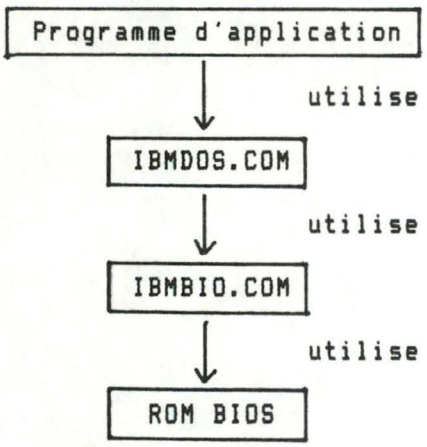

Figure 4.6 Hiérarchisation du système d'exploitation DOS.

4.5.2 Les interruptions du système d'exploitation

Le système d'exploitation est composé d'un ensemble de primitives qu'un programme peut utiliser au cours de son exécution. Il met à la disposition du programmeur des vecteurs d'interruption qui contiennent les adresses de début de ses routines de gestion des interruptions. La référence à l'un de ces vecteurs par un programme exécutable permet donc d'accéder à la routine correspondante du système d'exploitation. Nous présentons dans ce paragraphe le principe de fonctionnement des interruptions, ainsi que les fonctionnalités de quelques routines de gestion d'interruptions du système d'exploitation auxquelles nous faisons appel dans ce projet.

a) Principe de fonctionnement des interruptions

Tous les vecteurs d'interruption de l'ordinateur sont localisés dans la partie la plus basse de la mémoire centrale. On dénombre ainsi 256 vecteurs numérotés de 0 à 255 occupant chacun deux mots de 16 bits dans la mémoire. La figure 4.7 présente la table des vecteurs d'interruption de l'ordinateur. Les deux mots de chaque vecteur contiennent l'adresse de la routine de gestion d'interruptions à exécuter. Le premier mot représente le déplacement de la routine dans son segment de code (IP) et le second mot mémorise l'adresse de début du segment de code de cette routine (CS). L'entrée de la procédure gérant l'interruption est donc entièrement localisée par ces deux valeurs et se situe à l'adresse "(CS * 16) + IP". Puisque chaque vecteur d'interruption occupe deux mots de 16 bits, le vecteur numéro "i" est localisé à l'adresse 00000h + (4 * ih) de la mémoire centrale.

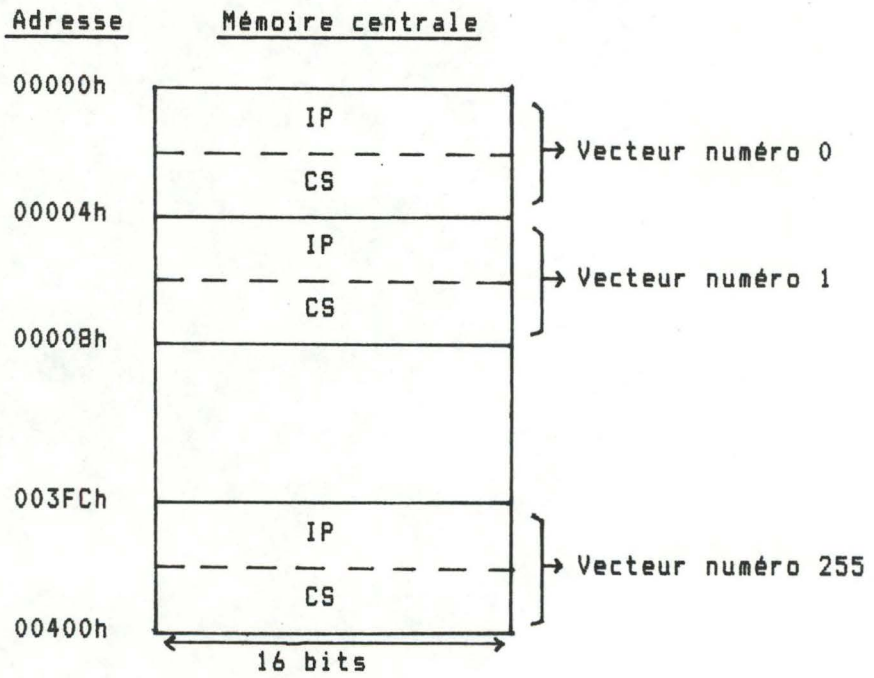

Figure 4.7 Table des vecteurs d'interruption.

Lors d'une interruption, l'adresse de retour et l'environnement du programme interrompu doivent être sauvegardés. La création d'une interruption par un événement quelconque a pour effet de sauver automatiquement dans la pile les registres d'indicateurs FR, de code CS et d'instructions IP, ainsi que de positionner à zéro le bit d'autorisation d'interruptions. La sauvegarde d'autres informations relatives au programme interrompu doit être prise en charge par la routine de gestion d'interruptions. L'état de la pile avant et après la création d'une interruption est représenté aux figures 4.8(a) et 4.8(b) respectivement.

Avant la prise en charge d'une interruption, la pile peut déjà contenir un certain nombre d'éléments. Le registre SP (Stack Pointer) pointe vers le dernier élément figurant au sommet de la pile. Après la création d'une interruption, les registres FR, CS et IP sont sauvés dans la pile dans cet ordre bien précis. Remarquons que le remplissage de la pile s'effectue par ordre décroissant des adresses en mémoire. Le registre SP pointe alors vers l'élément au sommet de la pile qui est le registre IP.

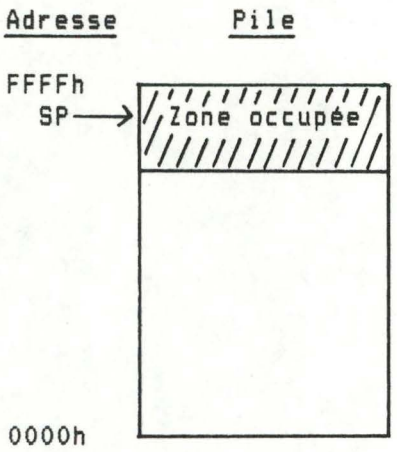

Figure 4.8 (a) Etat de la pile avant la prise en charge d'une interruption.

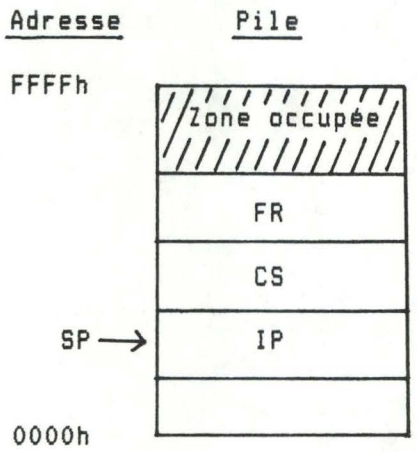

Figure 4.8 (b) Etat de la pile après la prise en charge d'une interruption.

Deux types d'interruptions sont présents dans cet ordinateur.

1) Les interruptions externes

Nous distinguons trois catégories d'interruptions externes:

- l'interruption NMI.
- les interruptions INTR.
- l'interruption RESET.

* L'interruption NMI est toujours prise en compte par le processeur dès la fin de l'instruction en cours. Il n'est donc pas possible de l'inhiber temporairement. Une telle interruption est activée par exemple lors d'une chute de tension dans la machine, c'est-à-dire lors de la survenance de problèmes très graves.

* Les interruptions INTR (INterrupt Request) sont activées par divers circuits et périphériques tels que le clavier, le lecteur de disquettes, l'horloge. La prise en compte de telles interruptions est gérée par un circuit interne à la machine (circuit électronique numéro 8259) qui dispose de 8 entrées numérotées de 0 à 7. La figure 4.9 représente schématiquement le circuit électronique (8259) équipé de ses 8 entrées. A chacune de ces entrées est associé un périphérique ou un circuit qui émet un signal sur son entrée respective lorsqu'il désire interrompre le processeur.

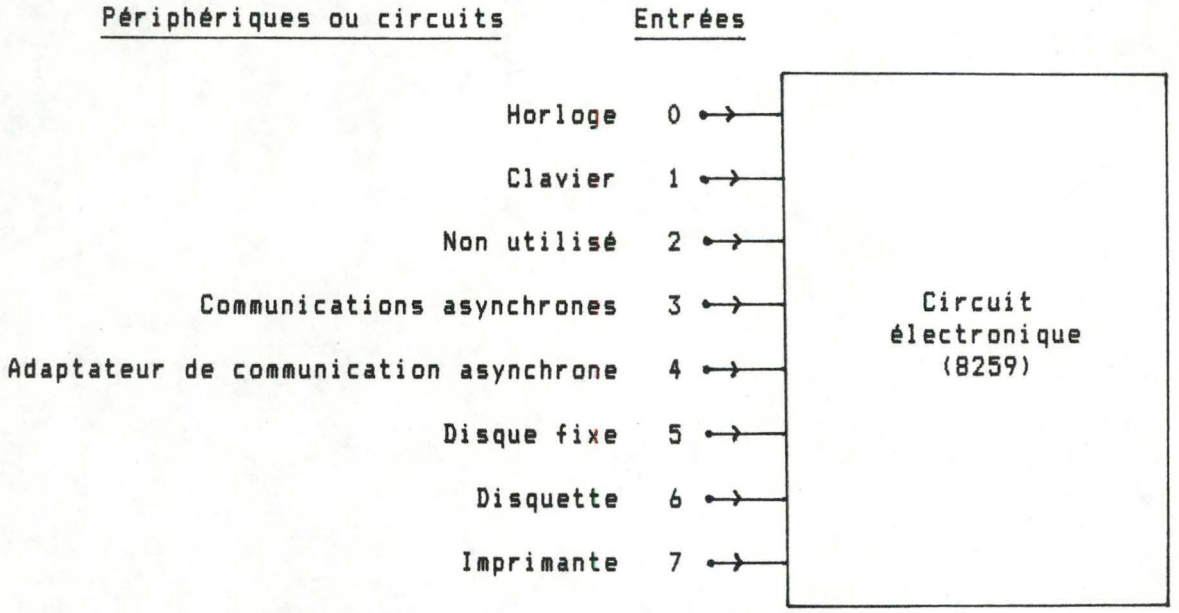

Figure 4.9 Présentation du circuit gérant les interruptions de type "INTR".

Par ailleurs, on peut se demander ce qu'il se passe si deux signaux arrivent en même temps sur deux entrées différentes du circuit. Lorsque cela se produit, il est nécessaire de ne prendre en considération qu'un des deux événements, le second étant mis "en attente". Pour résoudre ce problème, une priorité est associée à chaque entrée du circuit: l'entrée 0 correspond à la priorité la plus élevée et l'entrée 7 à la priorité la plus faible. Ainsi par exemple, lorsque deux signaux arrivent simultanément sur les entrées 1 et 3, la demande d'interruption 1 est prise en charge tandis que la demande d'interruption 3 est mise en attente jusqu'à ce que le traitement de l'interruption 1 soit terminé. (Nous avons ici fait l'hypothèse qu'il n'y avait pas d'autres interruptions de type INTR en cours de traitement). Si, lorsqu'un signal est envoyé sur une entrée, le traitement d'une interruption "A" de type INTR est en cours, le circuit électronique doit analyser si la nouvelle demande d'interruption "B" est plus prioritaire par rapport à l'interruption "A". Si l'interruption "B" est plus prioritaire, elle est prise en compte immédiatement; dans le cas contraire, elle est mise en attente.

Lorsqu'une interruption de type INTR est prise en compte, le processeur exécute les instructions de la routine de gestion des interruptions correspondante. La fin de l'exécution de cette routine doit être signalée au circuit controleur des interruptions par un signal particulier appelé "End Of Interrupt (EOI)". L'émission de ce signal permet au circuit (8259) de prendre en compte une interruption de type INTR de niveau inférieur. Ainsi par exemple, l'oubli du signal EOI à la fin de la routine de gestion de l'horloge bloquerait tout le système puisque l'interruption d'horloge a la priorité la plus élevée.

Les interruptions de type INTR peuvent être autorisées ou interdites en fonction de l'état du bit indicateur d'interruptions "IF". Si IF est positionné à zéro, toute interruption de type INTR est interdite; s'il est positionné à un, ces interruptions sont autorisées. Deux instructions en assembleur permettent de modifier l'état de ce bit:

- l'instruction STI positionne IF à 1.
- l'instruction CLI positionne IF à 0.

* L'interruption RESET permet de relancer le système après avoir abandonné toute opération matérielle et logicielle. Ce n'est donc pas une interruption au sens propre du terme car il n'y a pas de retour au programme interrompu.

2) Les interruptions internes

Il existe deux catégories d'interruptions internes:

- les interruptions logiques.
- les interruptions activées par une instruction.

* Les interruptions logiques sont activées par le micro-processeur lors de situations telles que la division par zéro, le dépassement de capacité.
* Les interruptions activées par une instruction produisent le même effet que l'appel d'une procédure ou d'un sous-programme. L'adresse de la routine de gestion d'interruptions doit être mémorisée dans le vecteur d'interruption correspondant. L'instruction assembleur "INT" suivie du numéro de vecteur d'interruption permet d'accéder à cette routine. Par exemple, si le vecteur numéro 60h contient l'adresse de la routine, l'instruction INT 60h permet d'y accéder. Les registres FR, CS et IP sont toujours sauvés automatiquement dans la pile comme pour toutes les autres interruptions. Ce type d'interruption étant généré par une instruction en assembleur micro-programmée dans le processeur, il n'est pas possible de l'inhiber (sauf en cas d'erreur grêve dans la machine).

b) Présentation des principales routines de gestion d'interruptions du système d'exploitation

Le système d'exploitation DOS met à la disposition du programmeur de nombreuses primitives accessibles à l'aide des interruptions. Nous présentons brièvement dans ce paragraphe les interruptions quelque peu spéciales auxquelles nous faisons appel dans cette application. Le lecteur trouvera une description détaillée de toutes ces interruptions dans le manuel de référence du DOS [6].

* L'interruption 1Ch: cette interruption est activée en faisant référence au vecteur d'interruption numéro 1Ch de l'ordinateur. La particularité de cette interruption réside dans le fait qu'elle est automatiquement appelée environ 20 fois par seconde par l'ordinateur. Le programmeur n'est donc pas obligé d'y faire référence explicitement dans son programme. En pratique, ce vecteur est appelé par la routine de gestion des interruptions associée au vecteur numéro 08h de l'ordinateur. Ce vecteur numéro 08h est appelé 20 fois par seconde par l'horloge du système, par l'intermédiaire du

circuit controleur des interruptions et constitue une interruption externe type "INTR". Le vecteur numéro 1Ch est donc lui aussi appelé 20 fois par seconde. Il est possible d'interdire temporairement l'interruption 1Ch en positionnant le bit indicateur d'interruptions IF à zéro.

* L'interruption 21h: une référence à l'interruption 21h est réalisée par l'intermédiaire du vecteur numéro 21h. Cette interruption présente une particularité en ce sens que plusieurs primitives du système d'exploitation sont accessibles à partir de cette unique interruption. La sélection d'une primitive particulière est réalisée en mémorisant dans un registre du processeur le numéro de la fonction désirée. Parmi ces fonctions, nous avons retenu principalement:

  - La fonction 25h: cette fonction permet d'affecter l'adresse d'une routine de gestion d'interruptions à un vecteur d'interruption.

  - La fonction 31h: la fonction 31h permet de terminer un programme et de rendre le contrôle au système d'exploitation DOS tout en gardant le programme en mode résidant. Notons cependant que cette primitive ne peut être utilisée que pour mettre en mode résidant un type particulier de programmes exécutables. En effet, deux types de fichiers exécutables par le système d'exploitation DOS existent: les fichiers dont l'extension est donnée par ".COM" et ceux ayant ".EXE" comme extension. Seuls les fichiers du type ".COM" peuvent être mis en mode résidant.

  - La fonction 35h: cette fonction permet de lire l'adresse contenue dans un vecteur d'interruption donné.

4.6 Méthodes d'accès aux périphériques

Les périphériques peuvent échanger des informations avec l'ordinateur. Mais cet échange de données ne peut se faire directement: un circuit situé entre le périphérique et la machine permet l'établissement d'une telle liaison. Ce circuit comporte un ou plusieurs ports d'entrée/sortie. Chaque port constitue un espace mémoire dans lequel des données peuvent être lues ou écrites. Cet espace mémoire ne fait pas partie de la mémoire centrale et est accessible à l'aide de l'adresse qui lui est associée. Nous pouvons donc dire qu'un port constitue un tampon entre l'ordinateur et le périphérique et permet l'échange de données entre ces deux entités. La figure 4.10 donne les adresses de quelques ports d'entrée/sortie de l'IBM-PC.

Adresse des ports	Périphériques ou circuits
020h ⟶ 021h	Controleur d'interruptions (8259).
060h ⟶ 063h	L'interface de périphériques.
200h ⟶ 20Fh	Les manettes de jeu.
3B0h ⟶ 3BFh	L'écran monochrome.
3D0h ⟶ 3DFh	Le moniteur couleur et graphique.

Figure 4.10 Adresses des principaux ports d'entrée/sortie.

4.7 Le clavier

Le clavier est un périphérique permettant à l'utilisateur d'entrer en communication interactive avec la machine. Dans le cadre de ce projet, on pourrait se demander pourquoi il est nécessaire de savoir comment le clavier fonctionne puisque le handicapé ne l'utilise pas. En fait, nous avons déjà signalé qu'il doit être simulé à l'écran afin que le handicapé dispose des fonctionnalités offertes par le clavier réel, tout en s'assurant que le logiciel "standard" susceptible de s'exécuter ne s'aperçoive pas de cette simulation. Il est donc nécessaire de connaître le principe de fonctionnement du clavier afin de réaliser correctement cette simulation. Nous présentons ci-après son principe de fonctionnement.

Le clavier de l'IBM-PC est géré de la manière suivante: chaque touche du clavier est identifiée par un "code de recherche". Lorsqu'une touche est enfoncée, le "code de recherche" qui la représente est envoyé vers l'ordinateur. De plus, l'appui sur une touche génère un signal à l'entrée numéro 1 du circuit controleur des interruptions (8259). Ce signal constitue donc une demande d'interruption. Lorsque cette interruption est prise en charge, le processeur exécute la routine de gestion d'interruption du clavier en référençant le vecteur d'interruption numéro 9.

Les opérations réalisées par la routine de gestion du clavier sont:

1) Lecture dans le port 60h du "code de recherche" correspondant à la touche enfoncée.
2) Libération de l'interruption du clavier pour permettre de prendre en compte la touche suivante enfoncée ou relâchée.
3) Détermination du code ASCII correspondant au "code de recherche".
4) Mémorisation du code ASCII et du "code de recherche" de la touche dans le buffer du clavier. En effet, le clavier est caractérisé par deux types de touches: les touches correspondant à un caractère affichable à l'écran et les touches de fonction telles que les touches "Ctrl", "Alternate", "home",... Le programme d'application utilise les deux codes simultanément pour déterminer s'il s'agit d'une touche de fonction ou d'une touche correspondant à un caractère, qui a été enfoncée au clavier.
5) Signalisation de fin de la routine de gestion du clavier au circuit controleur des interruptions (signal EOI).

Nous pouvons déduire de cette analyse que seules les opérations 3 et 4 consistant à mémoriser les codes ASCII et de recherche de la touche dans le buffer du clavier sont importantes dans le cadre de cette étude. En effet, d'une part le handicapé n'utilise pas le clavier et par conséquent l'interruption du clavier n'est pas activée. D'autre part, la prise en considération d'une touche du clavier par un programme d'application consiste uniquement en une lecture du contenu du buffer du clavier. Si le code d'une touche y est mémorisé, celui-ci est analysé par le programme en cours d'exécution. Nous allons par conséquent analyser plus en détail la mémorisation des codes d'une touche dans le buffer du clavier.

Mémorisation des codes d'une touche dans le buffer du clavier:

Lorsqu'une touche du clavier est enfoncée, son "code de recherche" est disponible dans le port 60h de l'ordinateur. La routine de gestion du clavier lit ce code, calcule le code ASCII correspondant et mémorise cette paire de codes dans le buffer du clavier. Ce buffer s'étend de l'adresse 0041Eh à l'adresse 0043Dh de la mémoire centrale et peut mémoriser jusqu'à 15 paires de codes. Deux pointeurs gèrent ce buffer: le premier appelé BUFFER_HEAD est localisé à l'adresse 0041Ah et contient l'adresse du premier caractère disponible dans le buffer. Le second pointeur BUFFER_TAIL se situe à l'adresse 0041Ch et mémorise l'adresse du dernier caractère introduit. Si ces deux pointeurs ont même valeur, cela signifie qu'il n'y a pas de caractère disponible dans le buffer. Le tampon du clavier est géré de manière circulaire et chaque paire de codes y occupe deux bytes.

D'autres variables déterminent l'état du clavier:

* Les octets KB_FLAG et KB_FLAG1, situés aux adresses 00417h et 00418h respectivement, dont les bits représentent l'état des touches de changement de mode et de verrouillage du clavier. La signification des bits de ces deux octets est présentée en annexe "G".

Les codes de recherche et ASCII représentatifs des touches du clavier sont donnés à l'annexe "H".

4.8 Les adaptateurs vidéo

L'affichage de messages et de résultats produits par un programme est réalisé grâce à un adaptateur vidéo qui constitue le circuit d'interfaçage entre l'ordinateur et l'écran proprement dit. Un ordinateur du type "IBM-PC compatible" peut supporter deux adaptateurs d'écran différents: un adaptateur d'écran monochrome et un adaptateur de moniteur couleur et graphique. Deux adaptateurs peuvent fonctionner simultanément moyennant cependant quelques restrictions que nous verrons au paragraphe 4.8.3. Nous exposons ci-après le principe de fonctionnement des deux adaptateurs.

4.8.1 L'adaptateur d'écran monochrome

L'adaptateur d'écran monochrome ne permet qu'un affichage monochrome à l'écran et ne peut travailler qu'en mode texte. L'affichage de caractères à l'écran est effectué en divisant l'écran en 25 lignes de 80 colonnes.

Chaque caractère affiché à l'écran occupe deux octets en mémoire centrale. La zone mémoire nécessaire pour représenter tous les caractères à l'écran est appelée RAM vidéo et est située à l'adresse B0000h dans la mémoire centrale.

4.8.2 L'adaptateur d'écran couleur

L'adaptateur d'écran couleur peut fonctionner en plusieurs modes: le mode texte et le mode graphique.

a) Le mode texte:

Deux configurations sont possibles en mode texte:

- un affichage de caractères à l'écran sur 25 lignes de 80 colonnes.
- un affichage de caractères sur 25 lignes de 40 colonnes.

b) Le mode graphique:

En mode graphique, deux types de résolution sont également permis:

- le mode graphique couleur, moyenne résolution, qui fournit une décomposition de l'écran en 200 lignes de 320 points.

- le mode graphique haute résolution décompose l'écran en 200 lignes de 640 points.

Pour chacun de ces modes, la RAM vidéo est localisée à l'adresse B8000h.

4.8.3 L'utilisation simultanée de deux adaptateurs

Un programme peut, au cours de son exécution, utiliser deux adaptateurs afin de réaliser des affichages sur deux écrans différents. Mais il est indispensable que ces deux adaptateurs ne soient pas identiques: le premier doit être un adaptateur monochrome et le second un adaptateur couleur. En effet, nous savons que la RAM vidéo de l'adaptateur monochrome se situe à l'adresse B0000h de la mémoire centrale, tandis que celle de l'adaptateur couleur est localisée à l'adresse B8000h. Il est alors possible de modifier chacune de ces zones de mémoire séparément selon que l'on désire effectuer un affichage sur l'un ou l'autre écran. D'autre part, il n'est possible à un moment donné de travailler que sur un seul des

deux écrans à la fois. Un octet de configuration du système appelé EQUIP_FLAG et situé à l'adresse 00410h permet de sélectionner l'un ou l'autre écran vers lequel le programme envoie ses données. Les bits 4 et 5 de cet octet indiquent le type de moniteur utilisé. La figure 4.11 illustre les configurations d'écran possibles en fonction de l'état de ces deux bits.

(EQUIP_FLAG) Bit 5	Bit 4	Type de moniteur utilisé
0	0	Pas de moniteur
1	0	Moniteur couleur graphique moyenne résolution
0	1	Moniteur couleur graphique haute résolution
1	1	Moniteur monochrome

Figure 4.11 Indicateur du type de moniteur utilisé.

CHAPITRE 5

CHAPITRE 5 IMPLEMENTATION DU LOGICIEL

5.1 Introduction

Nous présentons dans ce chapitre les principales techniques utilisées pour implémenter l'application d'aide aux handicapés sur un ordinateur de type "IBM-PC compatible". Nous commençons par exposer diverses techniques d'implémentation de la multiprogrammation en tenant compte du système d'exploitation choisi. Une critique de ces techniques nous conduit à choisir et implémenter l'une d'entre elles. Nous explicitons ensuite les méthodes utilisées pour rendre le programme résidant en mémoire centrale et pour exécuter plusieurs programmes de façon imbriquée dans le temps. Enfin, deux aspects fondamentaux sont analysés: l'installation de l'interface utilisateur et la simulation du clavier par l'ordinateur.

Les techniques d'implémentation du logiciel exposées dans ce chapitre ne sont compréhensibles que si le lecteur a déjà acquis quelques notions sur les ordinateurs de type "IBM-PC compatible". Le lecteur non averti trouvera au chapitre 4 les notions indispensables.

5.2 Réalisation de la multiprogrammation avec le système d'exploitation DOS

Comme nous l'avons déjà mentionné au chapitre 3, le système d'exploitation utilisé dans le cadre de cette étude est le système d'exploitation DOS (Disk Operating System).

Nous savons que l'implantation de notre projet nécessite un système d'exploitation capable de supporter la multiprogrammation. Or le DOS n'est à priori pas conçu pour répondre à cette exigence. C'est pourquoi nous analysons et critiquons dans ce paragraphe diverses techniques de réalisation de la multiprogrammation à l'aide du système d'exploitation DOS.

Les critères de choix d'une solution permettant de réaliser la multi-programmation à l'aide du système d'exploitation DOS sont les suivants:

- assurer une portabilité maximale du logiciel développé.
- garantir une fiabilité élevée du logiciel en minimisant le risque d'erreurs.
- permettre une maintenance facile et rapide.
- minimiser le travail pour assurer la multiprogrammation.

5.2.1 Analyse et critique des diverses solutions pour implémenter la multiprogrammation à l'aide du DOS

A. Identifier une variable commune à toutes les primitives du système d'exploitation, dont la valeur est modifiée lorsqu'un processus utilise l'une d'elles

L'accès au système d'exploitation DOS est réalisé par l'intermédiaire des interruptions. Nous pouvons donc, en scrutant les adresses contenues dans les vecteurs d'interruption, connaître le point d'entrée de chaque routine de gestion d'interruptions du système d'exploitation. Nous nous sommes posés la question suivante: n'y a t-il pas, au début et à la fin de ces routines, une ou plusieurs instructions modifiant la valeur d'une variable du système qui serait alors utilisée de la même manière qu'un verrou? Si une telle variable existe, nous pouvons examiner sa valeur et savoir si un processus utilise les primitives du système d'exploitation.

Cette solution est très simple à implémenter car il suffit d'identifier cette variable et de vérifier sa valeur pour savoir si un processus utilise déjà le système d'exploitation.

Cependant, cette solution n'est pas acceptable pour deux raisons:

1) Le logiciel constituant le système d'exploitation DOS est un logiciel "standard" vendu dans le commerce. Mais ce programme n'a pas été conçu une fois pour toutes. En effet, il subit des modifications au cours des années dans le but de corriger certaines erreurs et d'y apporter des améliorations. Plusieurs versions sont donc disponibles et rien ne dit que si aujourd'hui les routines de gestion d'interruptions modifient la valeur d'une variable, les versions futures du DOS effectueront encore cette modification.

2) Il faut s'assurer que toutes les primitives non réentrantes du système d'exploitation modifient la valeur d'une ou de plusieurs variables communes, mais que celles-ci ne sont en aucun cas modifiées à un moment inopportun par le système d'exploitation. Ceci nécessite une vérification de toutes les routines dans toutes les versions disponibles du DOS.

Nous pouvons conclure que cette solution ne respecte pas du tout le critère de portabilité. De plus, l'implémentation de la multiprogrammation à l'aide de cette méthode recquiert un travail énorme car il faut examiner toutes les routines du système d'exploitation DOS. Cette solution n'est par conséquent pas acceptable.

B. Assurer la réentrance dans le système d'exploitation DOS

Nous avons vu au chapitre 3 que tout système d'exploitation comprend un ensemble de modules qui sont répartis dans deux classes:

- les modules constituant des sections critiques.
- les modules permettant la réentrance (moyennant une adaptation éventuelle).

Pour réaliser la multiprogrammation à l'aide du système d'exploitation DOS, nous devons identifier l'ensemble des modules qui constituent des sections critiques ainsi que ceux qui acceptent la réentrance.

Cette seconde méthode pour implémenter le multiprogrammation consiste à effectuer les opérations nécessaires pour assurer la réentrance dans les modules qui ne sont pas des sections critiques.

** La première étape consiste donc à identifier et répartir les modules du système d'exploitation DOS dans les deux classes de modules citées ci-dessus.

1) Les sections critiques du système d'exploitation DOS

L'identification des sections critiques dans le système d'exploitation DOS repose sur l'étude faite au paragraphe 3.4 concernant les problèmes liés à la multiprogrammation. En effet, nous savons que tout périphérique tel qu'un disque, une bande magnétique ainsi que toute ressource partagée ne peut être accédé simultanément par plusieurs processus. Nous pouvons en déduire que toutes les primitives du système d'exploitation assurant l'accès direct et la gestion de ces périphériques, ainsi que l'accès à certaines ressources partagées constituent des sections critiques. Dans le cas du DOS, ces primitives sont réparties dans tous les segments du système d'exploitation de la manière suivante:

* le segment ROM BIOS: - primitives d'accès aux périphériques.
  - primitives d'accès à certaines ressources partagées (heure, initialisation des périphériques).

* les segments IBMBIO.COM, IBMDOS.COM et COMMAND.COM contiennent des primitives d'accès en écriture et en lecture à des ressources partagées (date, tables, variables de contrôle...).

Il faut rechercher dans chacun de ces modules les primitives constituant des sections critiques afin de créer un mécanisme n'y autorisant l'accès que par un seul processus à la fois.

2) Les modules réentrants du système d'exploitation DOS

Le système d'exploitation DOS ne possède à priori aucun module réentrant. Comment identifier des modules réentrants dans un tel logiciel? Nous savons, en toute généralité, qu'un module qui ne fait pas d'accès direct à un périphérique et qui ne manipule pas de ressources partagées accessibles en écriture peut, s'il n'est pas déjà réentrant, être transformé en un module réentrant. Cela peut être réalisé en sauvant les résultats obtenus pour chaque processus qui utilise le module. Nous pouvons en conclure que toutes les primitives du système d'exploitation qui ne sont pas des sections critiques peuvent être rendues réentrantes à la condition que la sauvegarde des résultats propres à chaque processus puisse être assurée. Dans ce cas, de telles primitives font partie de la classe des modules réentrants.

** Assurer la réentrance dans les modules qui ne constituent pas une section critique et l'exclusion mutuelle dans les autres modules représente la deuxième étape.

Nous pouvons en conclure que dès que la classe de chaque module est identifiée, la limitation du nombre de processus dans les sections critiques et la réalisation de quelques opérations sur les modules réentrants assure la multiprogrammation avec le système d'exploitation DOS.

Mais ces objectifs ne sont pas simples à réaliser pour les raisons suivantes.

D'une part, les sections critiques sont réparties dans tous les segments du système d'exploitation. Certaines d'entr'elles sont donc directement accessibles à l'aide d'une interruption. Par exemple, les sections critiques du segment IBMDOS.COM présentent cette caractéristique. Nous ne pouvons assurer l'exclusion mutuelle de ces modules sans avoir recours à un traitement spécial que nous présentons au point "c" ci-après.

D'autre part, les modules du DOS qui ne constituent pas des sections critiques ne sont à priori pas réentrants. Il serait possible de les rendre réentrants si les hypothèses qui suivent étaient toutes deux satisfaites:

* il ne peut y avoir plus d'un processus actif dans le système à un moment donné. En effet, si deux processus étaient actifs en même temps dans un module réentrant du DOS, certains résultats intermédiaires pourraient être perdus. Cette hypothèse est satisfaite dans un ordinateur du type "IBM-PC compatible" car celui-ci ne possède qu'une seule unité centrale et par conséquent il ne peut y avoir qu'un seul processus actif à un certain moment. (Nous ne considérons pas ici les processeurs périphériques car ils assurent seulement les échanges de données entre les périphériques et la mémoire centrale, ni le coprocesseur (8087) qui permet d'accélérer les opérations arithmétiques et n'effectue aucun appel au système d'exploitation).

* il est indispensable de connaître exactement les zones de la mémoire dans lesquelles les modules mémorisent les résultats de leurs opérations. Or le DOS étant un logiciel "standard" susceptible de subir des modifications perpétuelles dans les années à venir, ces zones de mémoire peuvent très bien changer d'emplacement d'une version à l'autre du logiciel. De plus, le DOS est un logiciel "standard" pour lequel nous n'avons trouvé aucune description suffisamment détaillée des zones de communication. Cela signifie qu'il faut analyser dans les moindres détails les instructions des primitives du DOS afin de localiser les zones de mémoire dans lesquelles les résultats intermédiaires sont mémorisés; cette recherche doit être faite pour toutes les versions du DOS. La portabilité du logiciel que l'on développerait de cette manière serait compromise.

Réaliser la multiprogrammation par cette méthode demande également un travail colossal pour identifier les zones de communication de chaque primitive du DOS. Nous rejetons aussi cette méthode.

C. Modifier les adresses contenues dans les vecteurs d'interruption du système d'exploitation DOS

Les primitives du système d'exploitation sont accessibles par un programme à l'aide des interruptions. Une autre solution envisageable pour assurer l'exclusion mutuelle consiste à modifier tous les vecteurs d'interruption du système d'exploitation en y mémorisant l'adresse d'une routine unique. Nous pouvons insérer une primitive d'exclusion mutuelle dans cette routine, qui modifie la valeur d'un verrou. Ainsi, lorsqu'un processus entre dans le système d'exploitation, il exécute cette routine qui affecte au verrou une certaine valeur pour signaler que le système d'exploitation est utilisé. La valeur de ce verrou est réaffectée à sa valeur initiale dès que le processus sort du système d'exploitation. Une simple vérification de la valeur du verrou permet de savoir si un processus utilise les primitives du système d'exploitation. Toute requête au système d'exploitation par un programme quelconque a donc pour effet d'exécuter cette routine qui doit assurer les fonctions suivantes:

- exécuter une primitive d'exclusion mutuelle pour signaler qu'un processus entre dans le système d'exploitation qui constitue une section critique.
- générer une interruption afin que le processus puisse accéder à la primitive du système d'exploitation dont il a besoin.
- exécuter une primitive d'exclusion mutuelle pour signaler que le processus qui utilisait le système d'exploitation vient de sortir de la section critique.

Cette solution est attrayante car l'exclusion mutuelle est assurée relativement aisément: il suffit de modifier les adresses de tous les vecteurs d'interruption du DOS.

Cependant, la modification des vecteurs d'interruption du système d'exploitation réduit considérablement la portabilité du logiciel obtenu. En effet, rien ne dit que dans les années à venir, d'autres vecteurs ne seront pas eux aussi réservés pour le système d'exploitation. Néanmoins, cette hypothèse est peu probable.

Par ailleurs, nous sommes confrontés à un problème qui porte davantage atteinte à la portabilité. En effet, considérons le cas suivant: soit un logiciel "standard" en train de s'exécuter sur l'ordinateur. Supposons que ce logiciel ait besoin d'une information en provenance de l'utilisateur, c'est-à-dire du clavier, pour pouvoir poursuivre son exécution. Dans une telle situation, le logiciel effectue une entrée/sortie; ceci a pour conséquence de faire appel à une primitive du système d'exploitation. Cette primitive effectue les opérations suivantes:

* attendre qu'un caractère soit introduit au clavier
* dès qu'un caractère est disponible, le mettre à la disposition du programme d'application

Nous remarquons que l'attente de l'introduction d'un caractère au clavier par l'utilisateur est réalisée à l'aide d'une boucle localisée dans le segment ROM BIOS du système d'exploitation. Nous pouvons en déduire l'affirmation suivante: un processus réside dans la section critique constituée par le système d'exploitation chaque fois que le logiciel "standard" attend l'introduction d'un caractère au clavier de la part de l'utilisateur. Or dans le cadre de cette application, l'utilisateur est en fait le handicapé qui ne peut introduire de caractère dans la machine qu'à

l'aide de son interface. De plus, nous savons que la communication entre l'interface et l'ordinateur n'est possible que par l'intermédiaire de notre programme.

Les opérations à effectuer pour introduire à l'aide de l'interface utilisateur, un caractère dans le buffer du clavier sont:

- faire apparaître le clavier simulé à l'écran s'il n'est pas visible.
- sélectionner, à l'aide du pointeur de l'écran, le caractère désiré dans le clavier simulé à l'écran.
- lire le caractère sélectionné.
- mémoriser ce caractère dans le buffer du clavier.

Il est évident que certaines de ces opérations nécessitent des appels aux primitives du système d'exploitation. En effet, l'introduction d'un caractère à l'aide de l'interface utilisateur nécessite plusieurs accès à l'écran. L'utilisation d'un langage de programmation de haut niveau réalise ces accès en utilisant les routines d'entrée/sortie du système d'exploitation. Nous sommes donc ici en présence d'un interblocage car un processus réside déjà dans le système d'exploitation en attente de l'introduction d'un caractère, et un processus de notre programme doit lui aussi y accéder pour introduire un caractère dans le buffer du clavier.

Pour résoudre ce problème, nous envisageons les solutions suivantes:

1) La première solution consiste à identifier les vecteurs d'interruptions qui permettent d'accéder aux routines du système d'exploitation susceptibles d'attendre un message en provenance de l'utilisateur. Dès qu'un processus du logiciel "standard" désire accéder à l'une de ces primitives, nous devons l'empêcher de pénétrer dans la section critique formée par le système d'exploitation et exécuter notre programme. En effet, nous savons que dans ce cas, le logiciel "standard" attendra un message de la part de l'utilisateur. Nous constatons que la portabilité n'est plus aussi étendue en raison de l'identification de certains vecteurs d'interruption. Il faut également être certain d'identifier toutes les routines susceptibles de présenter cette particularité sous peine de créer un interblocage. Les versions futures du DOS peuvent aussi comprendre de nouvelles routines semblables.

2) Empêcher toute utilisation du système d'exploitation lorsque l'utilisateur doit introduire une donnée nécessaire au logiciel "standard". Cette seconde solution nécessite la réécriture de nombreuses primitives du système d'exploitation et très probablement l'utilisation d'un langage de programmation de bas niveau, ce qui contredit les objectifs poursuivis par cette étude. De plus, la portabilité du programme ainsi développé est relativement difficile à assurer.

Nous constatons que cette troisième solution permettant d'implémenter la multiprogrammation pose quelques problèmes de portabilité car il est indispensable d'identifier les routines susceptibles de créer un interblocage dans toutes les versions du DOS. Cependant, cette "non-portabilité" peut être localisée dans un module de notre programme. Nous ne rejetons par conséquent pas cette solution car les modifications à apporter au programme pour assurer son bon fonctionnement sur divers ordinateurs de type "IBM-PC compatible" peuvent être rapidement effectuées.

D. Vérifier la valeur du compteur d'instructions donnée par l'expression "(CS * 10h) + IP"

Les segments de code du système d'exploitation DOS sont localisés dans des zones particulières de la mémoire centrale et contiennent les primitives mises à la disposition du programme de l'utilisateur. Ainsi, lorsque ce programme génère une interruption, celui-ci est interrompu et le contrôle est passé à la routine de gestion des interruptions appelée. A partir de cet instant, le processeur exécute des instructions constituant cette routine. Or celle-ci est une primitive du système d'exploitation et figure certainement dans un des segments du DOS. Le compteur d'instructions référence donc l'instruction en train d'être exécutée qui figure dans un de ces segments. Pour assurer l'exclusion mutuelle du système d'exploitation, il faut que le logiciel développé dans cette étude examine la valeur du compteur d'instructions. Pour examiner cette valeur, notre programme doit interrompre l'exécution de la primitive du système d'exploitation. Nous savons en effet que les registres FR, CS et IP sont automatiquement sauvés dans la pile lors d'une interruption. Il suffit donc de vérifier la valeur des registres CS et IP sauvés dans la pile pour savoir si un processus utilise déjà le système d'exploitation. Si cette valeur référence une cellule figurant dans une des zones de la mémoire contenant un segment de code du DOS, cela signifie qu'un processus utilise déjà une des primitives du système d'exploitation. Le processus désireux d'accéder au système d'exploitation est alors mis en attente. La réalisation de cette solution nécessite la connaissance exacte de chaque zone mémoire occupée par les segments de code du système d'exploitation.

Cette solution est également facile à implémenter et assure l'exclusion mutuelle du système d'exploitation DOS.

Cependant, nous sommes toujours confrontés au problème des versions multiples du logiciel DOS. En effet, nous avons déjà signalé au chapitre 4 que les zones de la mémoire occupées par le système d'exploitation varient en fonction de la version du DOS utilisée. De plus, les zones de la mémoire occupées par les segments de code du système d'exploitation ne sont pas fixes; la localisation des segments de code varie également en fonction de la taille de la mémoire centrale. Bref, il est très difficile de déterminer exactement les emplacements de tous les segments de code du système d'exploitation. Remarquons également que la méthode du compteur d'instructions présente le même inconvénient que celui mentionné au point "c" ci-dessus: la demande d'introduction d'une donnée en provenance du handicapé par le logiciel "standard" crée un interblocage si l'on ne prend pas de précautions spéciales.

En conclusion, la portabilité et la fiabilité du système ne sont pas assurées et nous rejetons cette solution.

E. Ecrire des routines personnelles en assembleur

Le système d'exploitation est composé d'un ensemble de primitives que tout programme peut utiliser. Il parait cependant évident qu'un programme susceptible d'être exécuté sur un ordinateur tel que ceux envisagés dans cette étude n'est pas obligé d'utiliser ces primitives. Dans ce cas, si ce programme désire effectuer des entrées/sorties ou d'autres opérations

spécifiques, le programmeur doit y inclure un ensemble de procédures écrites en langage assembleur afin d'éviter toute requête au système d'exploitation. L'exclusion mutuelle de ce dernier est assurée puisqu'aucune de ses primitives n'est utilisée.

Cependant, il ne faut pas perdre de vue que les périphériques constituent des sections critiques. Il est donc indispensable de savoir si un autre processus utilise déjà le périphérique auquel notre programme désire accéder. Or cette condition n'est pas toujours vérifiée. De plus, nous pourrions créer des incohérences dans le système car en général, des variables définissent l'état des périphériques connectés à l'ordinateur. Celles-ci constituent des ressources partagées accessibles à la fois par les processus du logiciel "standard" et par les processus de notre programme d'application. Mais il n'est pas toujours possible d'isoler dans une section critique les groupes d'instructions accédant à ces variables. En effet, un examen rapide du code assembleur mémorisé dans la mémoire ROM de la machine montre clairement que la plupart de ces groupes d'instructions sont interruptibles. Des incohérences sont donc à craindre. Par ailleurs, l'utilisation d'un langage de bas niveau contredit nos hypothèses concernant cette application.

Nous ne pouvons pas non plus accepter cette solution.

Remarque:
****

Il est toutefois possible de limiter le nombre de périphériques accessibles par un programme développé à l'aide de cette dernière solution. En effet, l'équipe AIDAMI a réalisé des circuits électroniques supplémentaires permettant de commander le téléphone, les appareils électriques extérieurs, etc... à partir de l'ordinateur (voir chapitre 1). Ceux-ci constituent en réalité des périphériques d'entrée/sortie. Les logiciels "standards" ignorent la présence de ces circuits et n'y accèdent pas. Les routines écrites dans un langage de bas niveau peuvent alors accéder en toute sécurité à ces périphériques. Malheureusement, cette méthode réduit énormément les possibilités offertes par le programme. (Cette solution a d'ailleurs déjà été exploitée par l'équipe AIDAMI lors de la réalisation du prototype de l'application d'aide aux handicapés).

5.2.2 Choix d'une solution pour réaliser la multiprogrammation

Nous constatons qu'aucune des solutions ébauchées au paragraphe 5.2.1 ne satisfait les quatre critères énoncés au paragraphe 5.2 simultanément. Nous choisissons dans ce cas la solution qui à notre avis les respecte de manière optimale. Nous avons décidé de réaliser la multiprogrammation en modifiant les vecteurs d'interruption du système d'exploitation. Nous donnons ci-après une justification de ce choix:

- le critère de portabilité n'est en toute évidence pas respecté. Cependant, nous pouvons localiser cette défaillance dans un nombre limité de modules de notre programme. De cette manière, toute modification à réaliser pour assurer le bon fonctionnement du logiciel sur une machine du type "IBM-PC compatible" pourra être rapide.

- Le critère de fiabilité est lui aussi quelque peu défaillant en raison des interblocages susceptibles de survenir dans le système. Mais l'identification des routines du système d'exploitation pouvant créer un interblocage est relativement aisée. En effet, puisque le système se bloque lorsque le logiciel "standard" attend l'introduction d'un caractère au clavier par le handicapé, nous pouvons en déduire que toutes les primitives d'accès au clavier peuvent créer un interblocage. Une observation des fonctionnalités des routines du DOS permet d'identifier facilement et rapidement de telles primitives.

- le critère de maintenance est respecté car les modifications éventuelles sont localisées dans un nombre limité de modules.

- Enfin, le travail nécessaire à la réalisation de cette méthode n'est pas excessif.

Le choix de cette technique nous semble donc adéquat dans le cadre de cette application.

Remarque:
****

Deux solutions se présentent encore pour résoudre notre problème:

a) Ecrire notre propre système d'exploitation.

Cette solution consiste à écrire notre propre système d'exploitation qui serait accessible par le logiciel "standard" et par notre programme d'application. Nous pouvons dans ce cas assurer l'exclusion mutuelle vis-à-vis des sections critiques à l'aide de primitives d'exclusion mutuelle. Cependant, le travail recquis serait énorme. De plus, notre hypothèse consistant à utiliser un minimum d'instructions en assembleur n'est pas respectée.

b) Utiliser un système d'exploitation réalisant la multiprogrammation

L'utilisation d'un système d'exploitation autorisant la concurrence entre les processus permet de résoudre notre problème sans trop de difficultés. L'unique objectif non réalisé dans ce cas réside dans le fait que nous devons abandonner le système d'exploitation DOS au profit d'un autre système d'exploitation beaucoup moins répandu tel que "concurrent PC-DOS". Cette solution nous semble être la meilleure dans le cadre de cette application, mais encore faut-il s'assurer que tous les logiciels "standards" pouvant être exécutés sur le système d'exploitation DOS peuvent l'être également sur le système d'exploitation "concurrent" choisi. Nous n'avons pas choisi cette solution pour implémenter notre programme car il n'existe pas de système d'exploitation "concurrent" aux facultés suffisamment abordable vis-à-vis du temps imparti pour la réalisation de ce mémoire.

5.2.3 Technique d'implémentation de la solution assurant la multiprogrammation

La solution choisie pour implémenter la multiprogrammation dans notre programme consiste à modifier tous les vecteurs d'interruption du système d'exploitation. Nous pouvons de cette manière exécuter des primitives d'exclusion mutuelle afin de limiter le nombre de processus dans la section critique formée par le système d'exploitation. Nous présentons dans ce paragraphe la technique choisie pour implémenter cette solution.

5.2.3.1 Principe général d'implémentation de la solution assurant la multiprogrammation

Le principe général d'implémentation de la solution assurant la multiprogrammation consiste à effectuer les opérations qui suivent.

1) Mémoriser dans des variables du programme les adresses contenues dans tous les vecteurs d'interruption du système d'exploitation avant leur modification.

2) Modifier les adresses des vecteurs d'interruption du système d'exploitation en y mémorisant l'adresse d'une routine de notre programme. Chaque vecteur d'interruption référence sa propre routine. Nous devons donc créer autant de routines que de vecteurs modifiés. Nous appelons "inter i" la routine référencée par le vecteur numéro "i". De cette manière, lorsqu'un programme utilisateur référence un vecteur d'interruption du système d'exploitation, un nouveau processus est créé et exécute la routine "inter i" pointée par ce vecteur.

3) Chaque routine "inter i" effectue les traitements suivants:

   ** sauver les registres du processeur dans la pile du système.

   ** mémoriser dans une variable de notre programme le numéro du vecteur d'interruption appelé.

   ** appeler une procédure commune pour tous les vecteurs d'interruption modifiés. Soit "entrée" le nom de cette procédure.

   La figure 5.1 schématise l'ensemble des opérations à réaliser avant l'exécution de la routine "entrée".

4) Lorsqu'un processus ayant exécuté la procédure "inter i", entre dans la procédure "entrée" commune à tous les vecteurs modifiés, les opérations suivantes sont exécutées.

   ** Rechercher dans la pile la valeur de l'adresse de retour au programme utilisateur interrompu et mémoriser cette adresse dans une variable de notre programme. L'adresse de retour au programme interrompu a en effet été mémorisée dans la pile lors de l'appel de l'interruption par ce programme.

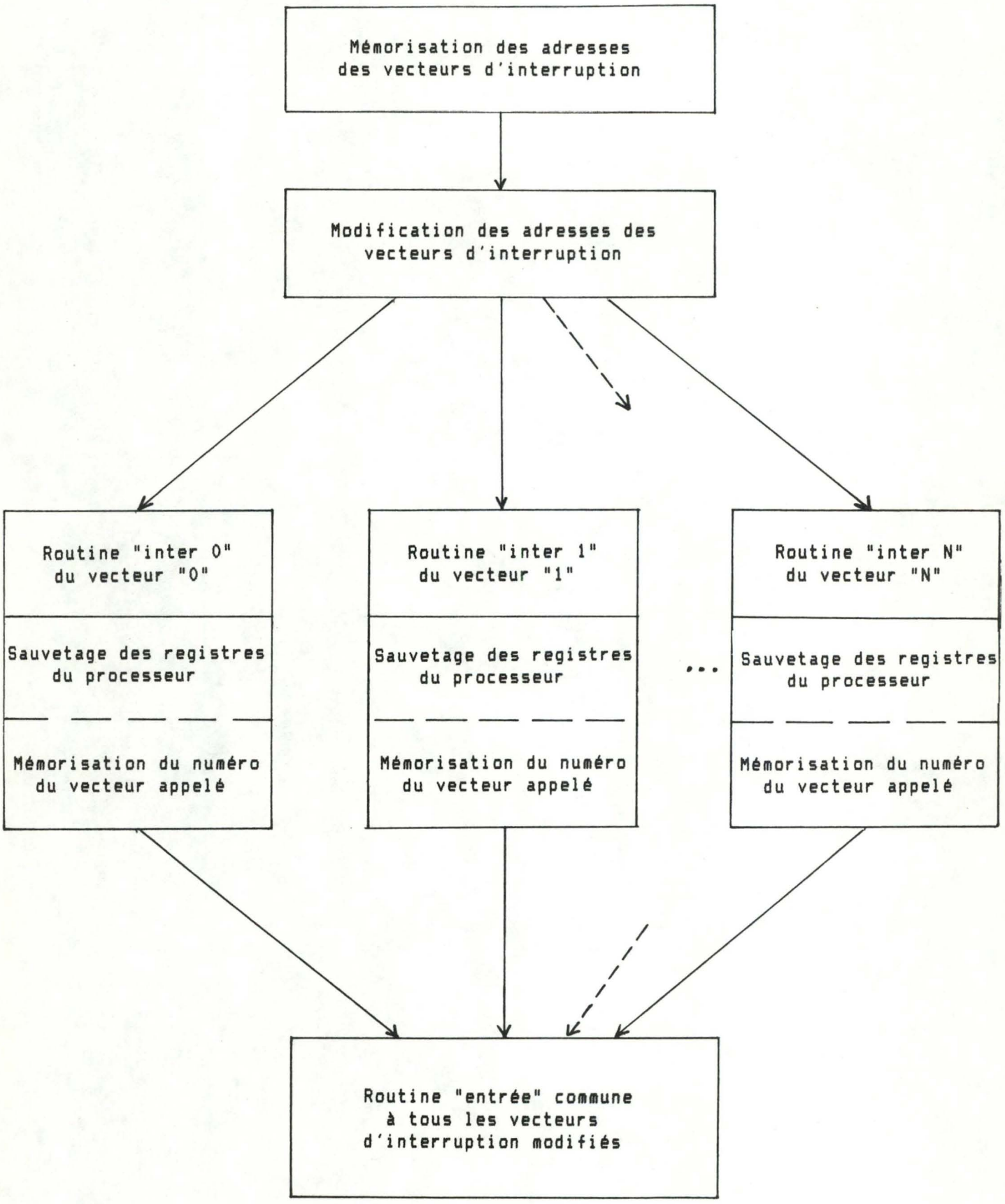

Figure 5.1 Opérations à réaliser avant l'exécution de la routine "entrée" pour implémenter la multiprogrammation.

** Positionner un verrou à l'aide d'une primitive d'exclusion mutuelle pour signaler qu'un processus entre dans une section critique.

** Modifier dans la pile l'adresse de retour au programme interrompu. La valeur de cette adresse est remplacée par l'adresse d'entrée dans une procédure définie dans notre programme. Soit "sortie" le nom de cette procédure. Celle-ci est également commune à tous les vecteurs d'interruption modifiés par le programme.

** Restaurer les registres du processeur qui ont été préalablement sauvés dans la pile par la routine "inter i".

** Effectuer l'appel de la routine de gestion des interruptions du système d'exploitation.

5) Dès que l'exécution de la routine du système d'exploitation est terminée, celle-ci rend le contrôle du système à la procédure "sortie" qui réalise les traitements suivants:

** sauver les registres du processeur dans la pile.

** dépositionner un verrou en utilisant une primitive d'exclusion mutuelle pour signaler qu'un processus sort d'une section critique.

** restaurer les registres du processeur sauvés dans la pile au début de cette routine.

** retourner au programme utilisateur interrompu.

La figure 5.2 donne une représentation globale de la technique d'implémentation de la multiprogrammation.

Nous analysons à présent en détail le contenu de ces différentes routines car le système d'exploitation DOS présente quelques particularités qui nous obligent à exécuter certaines opérations supplémentaires.

5.2.3.2 Analyse détaillée des routines permettant d'implémenter la multiprogrammation

Nous passons en revue chaque routine permettant d'implémenter la multiprogrammation. Une analyse détaillée de leur contenu permet de mieux percevoir l'utilité de certaines opérations effectuées.

++ Routine de mémorisation des adresses contenues dans les vecteurs d'interruption à modifier:

cette routine mémorise les adresses des vecteurs dans un tableau de nombres entiers. La lecture de ces adresses est réalisée en utilisant la fonction 35h de l'interruption 21h du système d'exploitation, ce qui accroit la portabilité du logiciel.

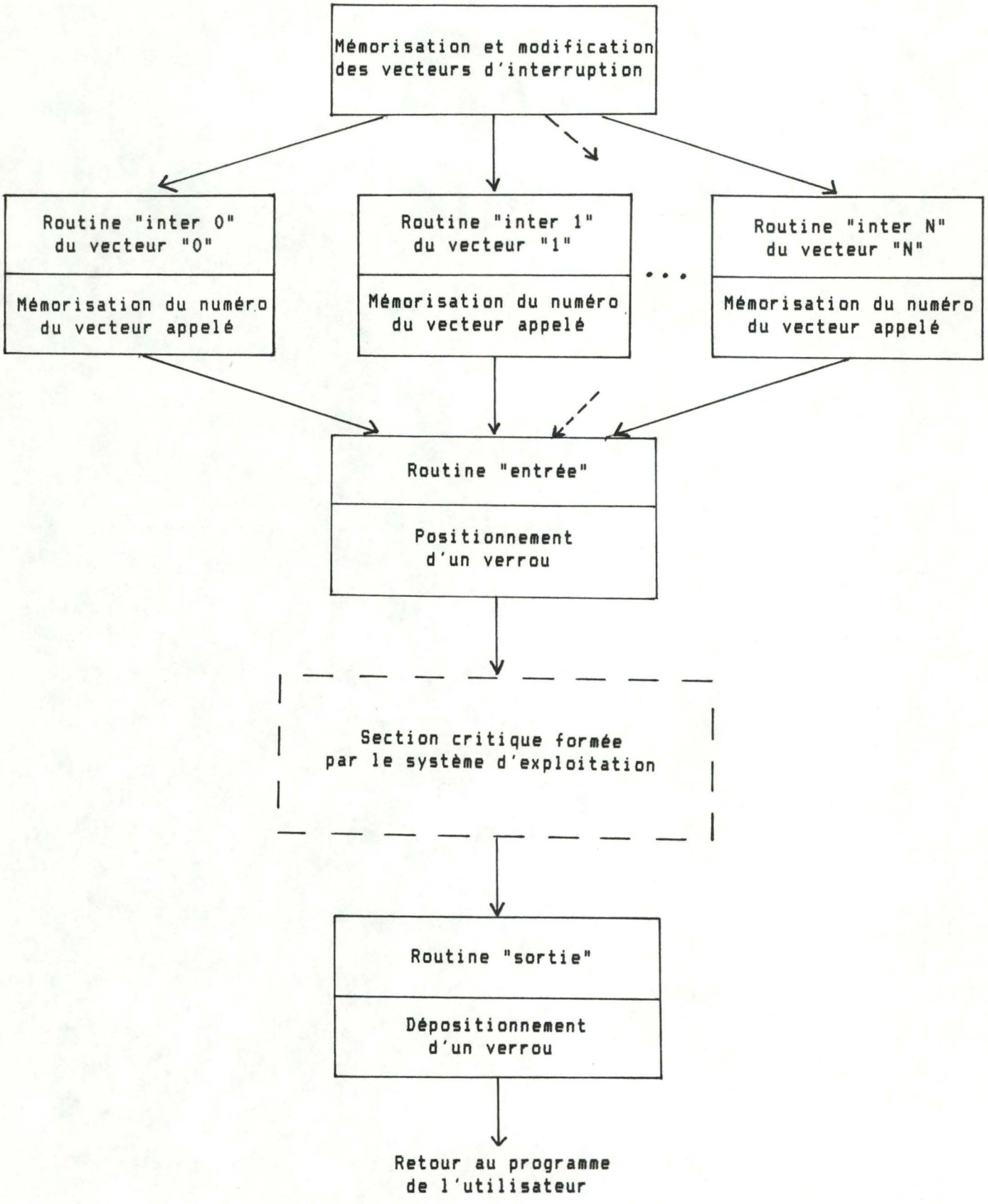

Figure 5.2 Représentation globale de la technique d'implémentation de la multiprogrammation.

++ Routine de modification des adresses des vecteurs d'interruption:

cette routine utilise la fonction 25h de l'interruption 21h du système d'exploitation pour modifier les valeurs des vecteurs d'interruption. La portabilité est dans ce cas également meilleure. Les nouvelles adresses à mémoriser dans les vecteurs d'interruption sont les adresses de début des routines "inter i".

++ Routines "inter i" associées à leur vecteur d'interruption respectif:

ces routines commencent par sauver dans la pile du système tous les registres du processeur excepté le registre de segment de pile SS qui pointe vers le segment contenant la pile.

La seconde opération consiste à mémoriser dans une variable de notre programme le numéro du vecteur d'interruption appelé par le programme de l'utilisateur. Mais nous avons mentionné au chapitre 4 que les variables déclarées dans un programme sont en toute généralité localisées dans le segment de données associé à ce programme. Ces variables sont donc accessibles à l'aide du registre de segment de données DS. Or ce registre peut contenir une valeur quelconque lorsque l'interruption est générée par le programme de l'utilisateur. Il faut dans ce cas restaurer la valeur de ce registre avant de réaliser une opération quelconque sur une variable de notre programme. Cette valeur doit être mémorisée pendant la phase d'initialisation de notre programme.

La dernière opération réalisée par les routines "inter i" consiste à appeler la procédure "entrée" commune à tous les vecteurs d'interruption modifiés. Nous devons ici respecter une particularité du système d'exploitation DOS. En effet, lors de la génération d'une interruption, les registres d'indicateurs FR, de segment de code CS et d'instruction IP sont sauvés automatiquement dans la pile. Or certaines primitives du système d'exploitation modifient des bits du registre d'indicateurs FR mémorisé dans la pile. Lors du retour au programme interrompu, ce dernier analyse l'état des bits de ce registre et réagit en fonction des nouvelles valeurs qui y sont mémorisées. Nous pouvons en conclure que la pile du système est utilisée pour passer des paramètres entre le programme de l'utilisateur et les primitives du système d'exploitation. Il est donc primordial de restaurer ces paramètres dans la pile s'ils sont modifiés par nos routines, avant d'effectuer l'appel au système d'exploitation et avant de retourner au programme utilisateur interrompu. Mais nous avons remarqué que l'appel d'une procédure en "Pascal" est toujours précédé d'un appel à une routine figurant dans les librairies associées à ce langage. Cet appel est généré automatiquement par le compilateur du "Turbo-Pascal". Malheureusement, cette routine modifie le contenu de la pile. En effet, l'appel d'une procédure en "Pascal" implique la mémorisation dans la pile des variables locales de la procédure appelante et des paramètres de la procédure appelée. Nous avons par conséquent été contraints d'utiliser une instruction en assembleur permettant d'effectuer un saut inconditionnel indirect vers la procédure "entrée".

Les opérations effectuées par les procédures "inter i" sont schématisées à la figure 5.3.

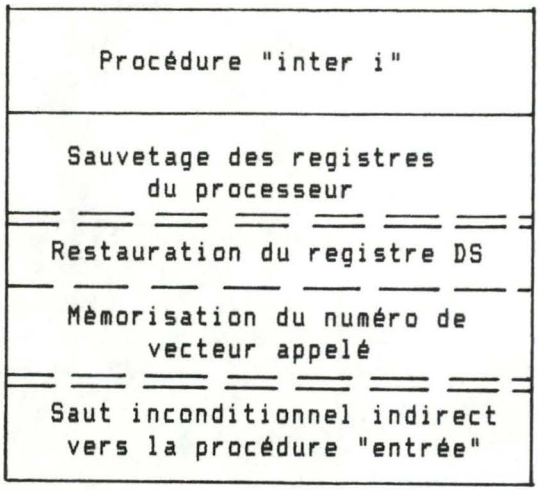

Figure 5.3 Opérations effectuées par les procédures "inter i".

++ Procédure "entrée":

la première opération effectuée par cette routine consiste à rechercher dans la pile les registres CS et IP mémorisés lors de la génération de l'interruption par le programme de l'utilisateur. Les valeurs de ces deux registres sont sauvées dans des variables de notre programme. Le contenu de ces registres représente l'adresse de retour au programme utilisateur interrompu.

L'exécution d'une primitive d'exclusion mutuelle constitue la seconde opération. Nous sommes cependant de nouveau sollicités par les restrictions imposées par le système d'exploitation. En effet, celui-ci possède sa propre pile; celle-ci est utilisée par la majorité de ses primitives. Ainsi, lorsqu'un programme utilisateur appelle une primitive du système d'exploitation, le registre de segment de pile SS est très souvent modifié et pointe dans ce cas vers le segment de pile propre au système d'exploitation. (Nous pouvons remarquer que plusieurs piles peuvent coexister dans l'ordinateur). Mais la taille de la pile associée au système d'exploitation est très réduite et ne permet en principe aucune utilisation spéciale de la part de l'utilisateur si ce n'est le sauvetage des registres du processeur lors d'une interruption. Considérons l'exemple suivant:

le programme utilisateur génère une interruption dans le but d'utiliser une primitive du système d'exploitation. Le registre de segment de pile SS pointe vers la pile associée à ce programme. Un vecteur d'interruption réservé au système d'exploitation est alors référencé, ce qui a pour conséquence de passer le contrôle à la routine "inter i" associée à ce vecteur. Le contrôle est ensuite passé à la procédure "entrée" qui positionne un verrou à l'aide d'une primitive d'exclusion mutuelle. En effet, le processus figurant dans cette procédure désire pénétrer dans la section critique formée par le système d'exploitation. Ces deux routines "inter i" et "entrée" précédant la section critique ne modifient pas le registre SS; elles utilisent donc également la pile associée au programme de l'utilisateur.

Supposons à présent que l'interruption générée par le programme de l'utilisateur désire accéder à une primitive du système d'exploitation localisée dans le segment IBMDOS.COM du système

d'exploitation. Nous savons que ce segment utilise généralement les primitives contenues dans le segment IBMBIO.COM du système d'exploitation, accessibles elles aussi à l'aide des interruptions. Dans ce cas, la primitive située dans le segment IBMDOS.COM modifie le registre de segment de pile SS car la pile associée au système d'exploitation est utilisée. Ainsi lorsque cette primitive génère une interruption pour donner le contrôle à une routine située dans le segment IBMBIO.COM, le registre SS référence la pile du système d'exploitation. Or nous avons modifié tous les vecteurs d'interruption du système d'exploitation afin d'exécuter une primitive d'exclusion mutuelle. Nous en concluons que cette fois, la primitive d'exclusion mutuelle située dans la routine "entrée" sera exécutée en utilisant la pile associée au système d'exploitation. Un dépassement de capacité de cette pile est à craindre.

Il faut donc modifier la valeur du registre SS chaque fois qu'une instruction figurant dans les procédures "inter i", "entrée" ou "sortie" est susceptible de créer un dépassement de capacité de la pile associée au système d'exploitation. C'est pourquoi nous avons modifié le registre de segment de pile SS, le pointeur de base BP et le pointeur de pile SP avant d'exécuter la primitive d'exclusion mutuelle. Celle-ci dispose de cette manière de sa propre pile pour s'exécuter. La restauration des anciennes valeurs de ces trois registres est réalisée aussitôt après l'exécution de la primitive d'exclusion mutuelle.

La troisième opération consiste à mémoriser dans la pile l'adresse d'entrée de la procédure "sortie". Cette adresse est mémorisée dans les registres de segment de code CS et d'instructions IP qui ont été sauvés dans la pile lors de la génération de l'interruption par le programme de l'utilisateur. Ces deux registres contenaient initialement l'adresse de retour au programme utilisateur interrompu. Si l'on y mémorise l'adresse de début de la routine "sortie", le contrôle pourra être passé à cette routine dès la fin de l'exécution de la primitive du système d'exploitation. Ceci nous permet de dépositionner le verrou associé à cette section critique avant de rendre le contrôle au programme utilisateur interrompu. Rappelons qu'il est indispensable de veiller à ne pas modifier le contenu du registre des indicateurs FR figurant dans la pile.

Il s'agit ensuite de restaurer les registres du processeur sauvés préalablement dans la pile par la routine "inter i". Ceci représente la quatrième opération.

Enfin, la cinquième opération a pour effet d'appeler la primitive du système d'exploitation désirée par le programme de l'utilisateur. Cette opération est tout à fait particulière car il faut de nouveau répondre aux exigences du système d'exploitation DOS. En effet, nous devons ici respecter deux contraintes:

* d'une part, nous ne pouvons modifier le contenu du registre des indicateurs FR mémorisé dans la pile puisqu'il est utilisé comme paramètre entre le programme de l'utilisateur et le système d'exploitation.

* d'autre part, quelques primitives du système d'exploitation permettant de terminer l'exécution d'un programme et de rendre le contrôle au système d'exploitation présentent la particularité suivante: avant d'effectuer un appel à l'une de ces primitives,

le registre de segment de code CS doit pointer vers le segment de code dans lequel est mémorisé le programme qui désire se terminer. En effet, ces primitives restaurent les adresses de certains vecteurs d'interruption mémorisés dans le préfixe de segment de programme (PSP) associé au programme qui se termine. Or nous savons que ce PSP occupe les 256 premiers octets du segment de code d'un programme.

Nous en déduisons que nous ne pouvons pas effectuer un appel à une primitive du système d'exploitation en utilisant une instruction en assembleur "INT" permettant de référencer un vecteur d'interruption. En effet, cette instruction serait référencée par les registres CS et IP lorsqu'elle serait exécutée. Or les registres FR, CS et IP sont sauvés automatiquement dans la pile lors de l'exécution de cette instruction. Le registre CS ainsi sauvé ne pointe certainement pas vers le segment de code du programme qui désire se terminer.

Nous ne pouvons pas non plus utiliser une instruction de saut inconditionnel indirect car une telle instruction nécessite que l'adresse vers laquelle le saut doit être réalisé soit mémorisée dans une constante ou dans un registre du processeur.

** La mémorisation de cette adresse dans une constante est absurde car cela ne permettrait de référencer qu'une seule primitive du système d'exploitation. De plus, il est fort probable que cette adresse change en fonction de la version du DOS utilisée si cette adresse référence une routine du DOS.

** Par ailleurs, l'utilisation d'un registre pour mémoriser l'adresse vers laquelle doit s'effectuer le saut est inadmissible car nous perdrions la valeur contenue dans ce registre avant de faire le saut vers la routine du système d'exploitation.

Il ne nous reste plus qu'une seule solution: effectuer de la programmation dynamique.

Dans ce cas, l'appel d'une primitive du système d'exploitation est réalisé grâce à une instruction de saut inconditionnel absolu. Cette instruction a pour effet de modifier les registres de segment de code CS et d'instructions IP. Ces deux registres ainsi modifiés représentent l'adresse de la prochaine instruction à exécuter. Or puisque l'adresse associée à l'instruction de saut doit être connue lors de la compilation, nous devons lui donner une valeur initiale. Celle-ci ne peut être qu'insignifiante avant l'exécution du programme, c'est pourquoi nous lui attribuons la valeur nulle. Au cours de l'exécution du programme, nous devons accéder directement aux cellules de la mémoire centrale dans lesquelles l'adresse associée à l'instruction de saut est mémorisée. En effet, lors de la compilation du programme, celles-ci contiennent la valeur nulle tout à fait insignifiante. Lorsque nous avons déterminé l'adresse vers laquelle le saut doit être effectué, nous pouvons la mémoriser dans les cellules de la mémoire centrale associées à l'instruction de saut. Dès que cette adresse est mémorisée, l'instruction de saut peut être exécutée.

Cependant, cette solution présente le désavantage suivant: il est nécessaire de connaître l'adresse de la cellule mémoire contenant l'instruction de saut inconditionnel. Pour connaître cette adresse, nous avons créé une procédure en "Pascal" dont le corps est constitué uniquement par l'instruction de saut absolu. Puisque le "Turbo-Pascal"

met à notre disposition une primitive permettant de connaître l'adresse d'entrée d'une procédure, il est aisé de déterminer l'adresse de l'instruction de saut absolu.

La figure 5.4 présente les opérations effectuées par la procédure "entrée".

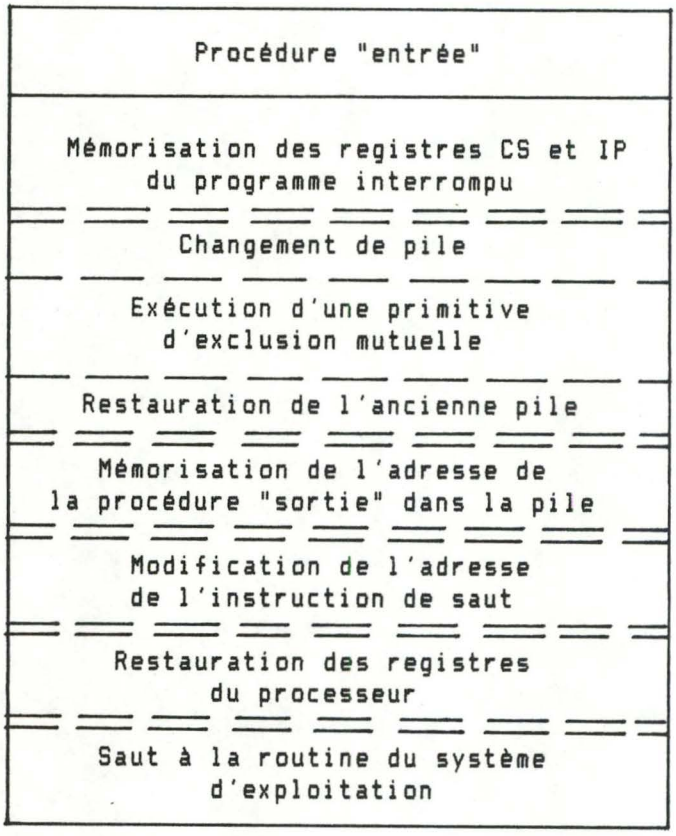

Figure 5.4 Opérations effectuées par la procédure "entrée".

++ Routine "sortie":

cette routine effectue également des opérations particulières pour tenir compte des exigences imposées par le système d'exploitation DOS. Ces contraintes sont identiques à celles mentionnées ci-dessus pour la procédure "entrée". Nous énumérons ci-après les opérations réalisées par la routine "sortie".

** Sauvetage des registres du processeur dans la pile.

** Modification du registre de segment de données DS.

** Changement de pile.

** Exécution d'une primitive d'exclusion mutuelle.

** Restauration de l'ancienne pile.

** Mémorisation de l'adresse de retour au programme de l'utilisateur dans les registres CS et IP localisés dans la pile.

** Restauration des registres du processeur.

** Exécution de l'instruction assembleur "IRET" permettant de rendre le contrôle au programme de l'utilisateur à l'issue de la procédure "sortie".

Signalons que l'implémentation de ces routines est réalisée en "Pascal" de très bas niveau. Nous entendons par là le fait que le "Turbo-Pascal" permet d'incorporer des instructions en assembleur dans les instructions du "Pascal". Nous sommes forcés d'utiliser de tels procédés pour effectuer des manipulations fiables sur les registres du processeur pendant le traitement des interruptions. Néanmoins, la structure du programme n'est pas perturbée car ces instructions de bas niveau peuvent être insérées dans des procédures du langage de haut niveau.

5.2.3.3 Principe de résolution de problèmes divers créés par le système d'exploitation DOS

Nous exposons dans ce paragraphe d'autres particularités et contraintes du système d'exploitation DOS. Celles-ci créent divers problèmes pour implémenter la multiprogrammation. Nous présentons également les techniques mises en oeuvre pour résoudre ces problèmes.

La première contrainte imposée par le système d'exploitation DOS a déjà été exposée au paragraphe 5.2.3.2. Nous avons signalé que certaines primitives du DOS permettent de terminer un programme et de rendre le contrôle au système d'exploitation. Ces primitives nécessitent que le registre de segment de code CS pointe vers le segment de code du programme qui se termine. Or nous avons signalé que la procédure "entrée" modifie la valeur de l'adresse de retour au programme utilisateur interrompu avant d'effectuer l'appel d'une primitive du système d'exploitation. Ceci permet de donner le contrôle à la procédure "sortie" à l'issue de l'exécution de cette primitive. Cette modification d'adresse ne peut plus être réalisée avant l'exécution d'une routine de terminaison de programme car le registre de segment de code CS ne pointrait plus vers le segment de code du programme qui désire se terminer. Nous devons par conséquent identifier les interruptions générées par le programme utilisateur afin d'effectuer un traitement spécial lorsque de telles routines doivent être exécutées. Nous exposons ci-dessous le traitement spécial à effectuer lorsqu'une routine permettant de terminer un programme doit être exécutée. Nous avons associé aux routines de terminaison de programme du système d'exploitation un verrou particulier qui leur est réservé. Ce verrou est positionné dans la routine "entrée" puisque le processus désireux d'accéder au système d'exploitation exécute toujours cette routine. Mais puisque la routine "sortie" n'est plus exécutée, la seconde primitive d'exclusion mutuelle qui s'y trouve ne l'est plus non plus. Nous devons par conséquent placer cette primitive d'exclusion mutuelle dans une autre procédure de notre programme. Nous avons fait l'hypothèse suivante: nous avons supposé que les routines du système d'exploitation qui permettent de terminer un programme et de rendre le contrôle au système d'exploitation, n'utilisent pas les routines du système d'exploitation permettant d'afficher des caractères à l'écran. Dans ce cas, nous pouvons dire que dès qu'une routine du système d'exploitation permettant d'afficher un caractère à l'écran est appelée, la routine du système d'exploitation permettant de terminer un programme est certainement terminée. Nous avons alors placé la seconde primitive d'exclusion mutuelle associée aux routines de terminaison de programme dans la procédure "inter 10h" de notre programme, qui est pointée par le vecteur d'interruption numéro 10h. Ce dernier est en effet référencé

lorsqu'un affichage à l'écran doit être effectué. Nous sommes ainsi certains que cette seconde primitive d'exclusion mutuelle sera exécutée car toute terminaison de programme est suivie de l'affichage du "prompt" à l'écran, qui nécessite l'utilisation de l'interruption numéro 10h du système d'exploitation. (Nous supposons également que l'utilisateur n'a pas modifié des paramètres dans le système d'exploitation afin d'inhiber l'affichage de ce "prompt"). Nous avons effectué la même opération pour les routines du système d'exploitation qui permettent de charger un programme en mémoire centrale et de lancer son exécution aussitôt après. La figure 5.5 illustre la méthode permettant d'assurer l'exclusion mutuelle pour les primitives qui permettent de terminer ou de lancer un programme.

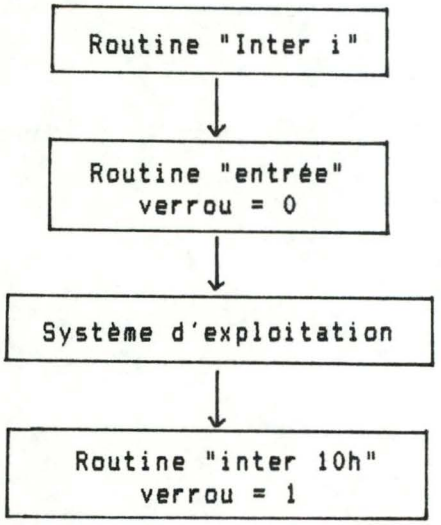

Figure 5.5 Technique d'implémentation de l'exclusion mutuelle pour les primitives de terminaison et de lancement de programme.

Remarque:
****

Lorsqu'un programme se termine, il rend le contrôle au système d'exploitation que nous avons défini comme étant une section critique. Nous faisons ici exception à cette règle pour un segment particulier du système d'exploitation; il s'agit du segment "COMMAND.COM". En effet, lorsqu'un programme se termine, le contrôle de la machine est rendu à l'interpréteur des commandes du système d'exploitation, qui est localisé dans le segment "COMMAND.COM". Par conséquent, il ne s'agit pas d'une routine d'interruption du système d'exploitation, mais d'un segment de code qui permet d'introduire des commandes dans l'ordinateur. Nous ne pouvons dès lors pas détecter si un processus réside dans ce segment de code avec les primitives d'exclusion mutuelle que nous avons créées, puisque celles-ci détectent la présence d'un processus dans une routine de gestion des interruptions du système d'exploitation. Ceci ne présente pas d'inconvénient pour la raison suivante: ce segment constitue en réalité la deuxième partie du segment de code "COMMAND.COM", qui est une partie non résidante du système d'exploitation. Nous pouvons donc dire que les routines d'interruption du système d'exploitation n'utilisent pas ce segment puisqu'il peut être effacé de la mémoire centrale pendant l'exécution d'un programme. Nous pouvons ignorer la présence d'un processus dans ce segment à condition de ne pas le soumettre à de la réentrance.

Cette exigence est en fait toujours réalisée dans notre application. En effet, si le programme "AIDAMI" désire utiliser des commandes de l'interpréteur des commandes, nous devons dupliquer la deuxième partie du segment "COMMAND.COM" et mettre cette seconde copie du segment à la disposition de notre programme. En effet, ce segment n'est pas résidant en mémoire centrale et nous devons alors le dupliquer et mettre la seconde copie en mode résidant afin de la rendre constamment disponible pour notre programme. Il ne peut donc y avoir qu'un seul processus dans le segment "COMMAND.COM" du système d'exploitation proprement dit.

La seconde contrainte que nous analysons ici a déjà été mentionnée au paragraphe 3.4.2. Lorsque le handicapé utilise un logiciel "standard", ce dernier demande en général des informations ou des données à l'utilisateur. Le logiciel analyse ces données introduites et réagit en fonction de leur contenu. Nous avons déjà signalé que dans pareille situation, un processus associé au logiciel "standard" figure dans la section critique du système d'exploitation et boucle en attente d'une donnée en provenance de l'utilisateur. Le handicapé ne peut plus utiliser notre application pour introduire ses données puisque la section critique est déjà occupée par un processus. Un interblocage est de nouveau à craindre. Nous expliquons ci-dessous la solution utilisée pour résoudre ce problème.

La première étape consiste à identifier toutes les primitives du système d'exploitation qui permettent de lire un caractère introduit au clavier. En effet, ces primitives attendent qu'un caractère soit introduit au clavier avant de retourner au programme utilisateur interrompu. Nous devons ensuite insérer un test supplémentaire dans la procédure "entrée", qui permet de vérifier si l'interruption générée par le programme de l'utilisateur référence une des primitives d'accès au clavier. Si c'est le cas, nous devons interrompre momentanément le processus figurant dans la procédure "entrée" et exécuter notre application. En effet, nous savons dans ce cas, que des caractères doivent être introduits par le handicapé dans le buffer du clavier, à l'aide de son interface. Lorsque le handicapé a introduit les données nécessaires à l'aide de notre programme, nous pouvons poursuivre l'exécution du processus qui a été interrompu dans la procédure "entrée". La primitive du système d'exploitation permettant de lire un caractère au clavier ne boucle plus puisque des caractères sont disponibles dans le buffer du clavier.

Mais cette solution nous crée un autre problème. En effet, pendant que le processus du programme de l'utilisateur est interrompu dans la procédure "entrée", le handicapé introduit des données dans le buffer du clavier en utilisant les services offerts par notre application. Or notre programme utilise lui aussi les primitives du système d'exploitation. Par conséquent, dès la génération d'une interruption par notre programme, un nouveau processus est créé et exécute les procédures "inter i" et "entrée" précédant la section critique du système d'exploitation. La procédure "entrée" est dans ce cas soumise à de la réentrance car un processus du programme utilisateur y figure déjà en attente de la fin de l'introduction des données par le handicapé. Nous avons préféré éviter toute réentrance dans cette procédure pour ne pas devoir sauver les résulats intermédiaires qu'elle produit. Cela peut être réalisé en modifiant temporairement l'adresse associée aux instructions de saut inconditionnel indirect figurant dans les procédures "inter i". L'exécution de la procédure "entrée" n'est dans ce cas plus

effectuée. La nouvelle adresse référence une quatrième procédure nommée "sortie-directe" qui effectue les opérations suivantes:

** mémoriser dans les cellules de la mémoire centrale, associées à une instruction de saut inconditionnel absolu, l'adresse de la primitive du système d'exploitation à exécuter.

** restaurer les registres du processeur initialement sauvés dans la pile par une procédure "inter i".

** appeler la primitive du système d'exploitation.

Une illustration de cette méthode est donnée à la figure 5.6. Celle-ci présente schématiquement les opérations effectuées par les procédures "inter i" et "sortie-directe". Ces procédures ne réalisent plus que des opérations permettant d'appeler la primitive du système d'exploitation nécessaire. Le nombre d'opérations réalisées est beaucoup moins élevé que lorsqu'on passe par les procédures "inter i", "entrée" et "sortie"; les performances du système sont d'autant meilleures.

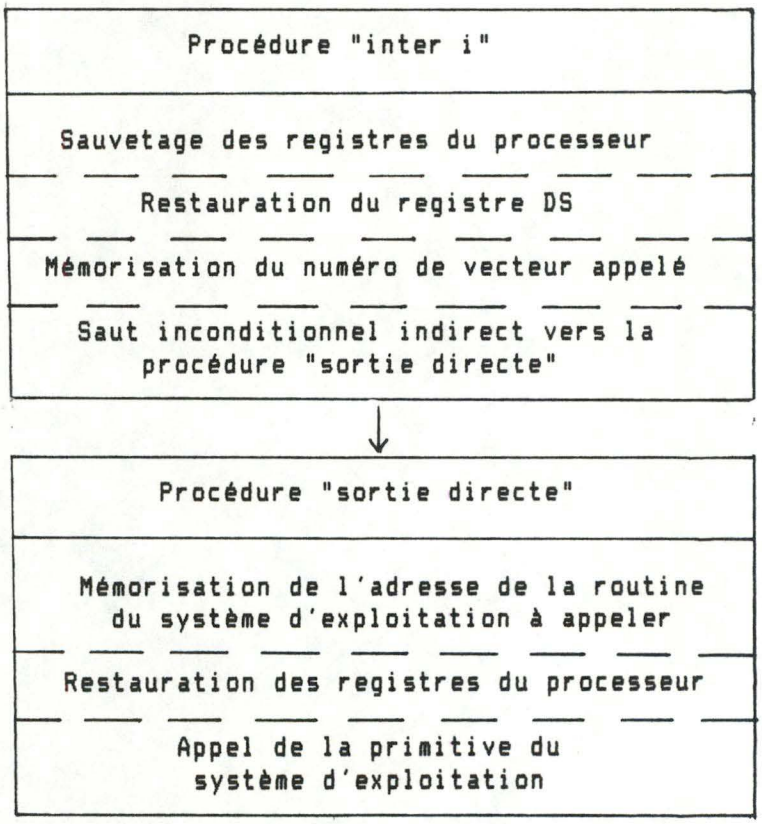

Figure 5.6 Méthode permettant d'éliminer la récursivité dans la procédure "entrée".

Remarquons que les primitives d'exclusion mutuelle localisées dans les procédures "entrée" et "sortie" ne sont plus exécutées. Ceci ne présente pas d'inconvénient car le logiciel "standard" est interrompu et n'utilise pas la section critique formée par le système d'exploitation. Notre programme peut donc y accéder librement.

Signalons également que pour éviter la réentrance, les procédures "inter i", "entrée", "sortie" et "sortie directe" doivent être ininterruptibles. En effet, toute interruption provoque l'exécution d'un certain nombre de ces procédures. Il est donc nécessaire d'inhiber les interruptions externes de type "INTR" pendant l'exécution de chacune de ces procédures.

Nous analysons à présent la troisième contrainte imposée par le système d'exploitation DOS. Nous avons mentionné au chapitre 4 que le DOS est structuré de manière hiérarchique. Les composants de cette hiérarchie sont appelés "IBMDOS.COM", "IBMBIO.COM" et "ROM BIOS". Nous savons que par exemple les primitives du composant "IBMDOS.COM" utilisent les primitives des composants "IBMBIO.COM" et "ROM BIOS". Considérons le cas suivant: le programme de l'utilisateur désire exécuter une primitive du système d'exploitation localisée dans le composant "IBMDOS.COM". Ce programme effectue dans ce cas une interruption. Les routines "inter i" et "entrée" sont en toute généralité exécutées avant de donner le contrôle à la primitive du système d'exploitation. Or cette primitive appelée par la procédure "entrée" exécutera très probablement une seconde primitive du système d'exploitation située dans les composants "IBMBIO.COM" ou "ROM BIOS". Par conséquent, cette seconde primitive sera appelée à l'aide d'une interruption qui aura également pour effet d'exécuter les procédures "inter i" et "entrée". Nous sommes à nouveau en présence de réentrance dans la procédure "entrée". Nous avons appliqué la technique suivante pour éviter cette réentrance: dès qu'un processus pénètre dans la procédure "entrée", nous modifions temporairement l'adresse associée aux instructions de saut inconditionnel indirect figurant dans les procédures "inter i". La nouvelle adresse associée à ces instructions de saut indirect référence alors la procédure "sortie directe". De cette manière, toutes les primitives du système d'exploitation appelées par la primitive qui se situe dans le composant "IBMDOS.COM" du système d'exploitation, n'exécutent pas la procédure "entrée". L'adresse associée aux instructions de saut inconditionnel indirect des procédures "inter i" est restaurée à sa valeur initiale dans la procédure "sortie", juste avant de rendre le contrôle au programme de l'utilisateur qui a été interrompu. La prochaine interruption générée par le programme de l'utilisateur provoquera alors à nouveau l'exécution de la procédure "entrée" afin d'exécuter la primitive d'exclusion mutuelle qui s'y trouve.

Nous présentons ici la quatrième particularité du système d'exploitation DOS. Les vecteurs d'interruption utilisés par le système d'exploitation contiennent en général l'adresse de la routine de gestion des interruptions correspondante. Nous avons cependant remarqué que trois de ces vecteurs (vecteurs numéro 1Dh, 1Eh et 1Fh) référencent non pas une routine, mais une table de données mémorisée en mémoire centrale. Ces tables contiennent des informations utilisées exclusivement par le système d'exploitation. Nous ne pouvons dès lors pas modifier l'adresse de ces vecteurs car le système d'exploitation accède directement aux données figurant dans ces tables à partir de leur adresse d'entrée en mémoire centrale. Ceci ne présente pas de danger pour notre application car seul le système d'exploitation a le droit d'accéder à ces tables. Nous supposons

par conséquent que le programme de l'utilisateur n'y accède pas.

Signalons également que nous avons jugé bon de ne pas modifier les adresses des vecteurs d'interruption numéro 22h, 23h et 24h. En effet, ceux-ci sont exclusivement référencés par le système d'exploitation. De plus, si les adresses mémorisées dans ces vecteurs sont modifiées pendant l'exécution d'un programme, celles-ci sont restaurées à leur valeur initiale dès que le programme se termine et rend le contrôle au système d'exploitation. Nous évitons ainsi de devoir constamment mettre à jour le contenu de ces vecteurs chaque fois que l'exécution d'un programme se termine.

Remarque: ******** Il est indispensable de vérifier si des adresses de vecteurs d'interruption du système d'exploitation sont modifiées soit par le système d'exploitation, soit par l'exécution d'un logiciel "standard". En effet, si c'était le cas, les routines "inter i", "entrée" et "sortie" ne seraient plus exécutées lors de l'appel à un de ces vecteurs et l'exclusion mutuelle du système d'exploitation ne serait plus assurée. Nous avons fait l'hypothèse que toute adresse de vecteur d'interruption ne peut être modifiée que par l'intermédiaire de la fonction 25h de l'interruption 21h du système d'exploitation. En effet, cette fonction est spécialement prévue à cet effet dans le but d'assurer une plus grande portabilité des logiciels qui effectuent des modifications d'adresse de vecteurs d'interruption. Dans ce cas, il suffit de vérifier si le vecteur modifié par cette fonction est un vecteur d'interruption du système d'exploitation. Lorsqu'il s'agit d'un vecteur du système d'exploitation, nous devons de nouveau mémoriser dans ce vecteur l'adresse de la procédure "inter i" de notre programme qui y est associée.

5.3 Réalisation du gérant des processus

La réalisation du gérant des processus dans le cadre de notre projet est très simplifiée. Nous profitons de quelques possibilités offertes par l'ordinateur de type "IBM-PC compatible" pour le développer.

Nous avons vu au chapitre 4 que l'ordinateur met à la libre disposition du programmeur quelques uns de ses vecteurs d'interruption. Parmi ceux-ci, nous avons retenu le vecteur d'interruption numéro 1Ch. Ce vecteur est référencé automatiquement par l'horloge de l'ordinateur environ 20 fois par seconde.

Rappelons que le handicapé doit pouvoir accéder à tout moment aux services et tâches offerts par notre programme. D'autre part, il doit avoir la possibilité d'exécuter un logiciel "standard" et d'échanger des informations avec ce logiciel en utilisant son interface utilisateur. Ce logiciel s'exécute jusqu'à ce qu'il ait besoin d'une information de la part de l'utilisateur, ou qu'il soit interrompu par le handicapé si ce dernier désire utiliser un service offert par le projet AIDAMI.

Nous avons imaginé la solution suivante: nous supposons que le logiciel "standard" est déjà chargé en mémoire centrale et qu'il est en train de s'exécuter. Pendant ce temps, le vecteur d'interruption numéro 1Ch est appelé 20 fois par seconde par l'horloge de l'ordinateur. L'appel de ce vecteur a pour conséquence d'interrompre brièvement le logiciel "standard". Nous concluons que si nous mémorisons dans le vecteur numéro 1Ch l'adresse de début d'une routine, celle-ci est appelée 20 fois par seconde. Cette routine peut par exemple lire l'état du contact de l'interface utilisateur et effectuer l'un ou l'autre traitement en fonction de cet état. Nous pouvons de cette manière interrompre quasi instantanément le logiciel "standard" et effectuer une opération spécifique.

Nous avons pleinement exploité cette solution car nous avons mémorisé dans le vecteur numéro 1Ch l'adresse d'une routine qui permet d'exécuter tous les services et toutes les tâches implémentées dans notre programme. Cette routine effectue globalement les opérations suivantes:

** tester l'état du contact de l'interface utilisateur.

** si ce contact est ou a été enfoncé, le contrôle est passé à notre programme.

** rendre le contrôle au logiciel "standard" interrompu.

De cette manière, dès que le handicapé enfonce le contact de son interface, il peut disposer des services offerts par le projet AIDAMI.

Analyse détaillée du gérant des processus

En pratique, l'implémentation du gérant des processus est plus complexe. D'autres opérations doivent être effectuées par la routine référencée par le vecteur d'interruption numéro 1Ch avant de pouvoir donner le contrôle à notre application. Nous passons ci-après en revue chacune de ces opérations et justifions la présence de celles qui sont les plus inattendues.

1) Sauver les registres du processeur dans la pile associée au logiciel "standard" interrompu.

2) Restaurer le registre de segment de données DS car nous utilisons dans cette routine des variables déclarées dans notre programme.

3) Envoyer au circuit controleur des interruptions (8259) un signal de fin d'interruption "EOI".
   En effet, nous savons que, en réalité, le vecteur numéro 1Ch est appelé 20 fois par seconde par la routine de gestion des interruptions correspondant au vecteur d'interruption numéro 08h de l'ordinateur. L'interruption numéro 08h constitue une interruption externe de type "INTR". Or toute routine de gestion d'interruptions externes de type "INTR" doit envoyer un signal de fin d'interruption EOI au circuit controleur des interruptions. Ceci permet de prendre en considération des interruptions de type "INTR" de priorité moins élevée. Le schéma global des traitements effectués par la routine de gestion d'interruptions associée au vecteur numéro 08h est donné à la figure 5.7. Nous remarquons sur cette figure que le premier traitement effectué par cette routine consiste à sauver les registres du

processeur. Ensuite, elle réalise ses propres traitements et lorsque ceux-ci sont terminés, elle référence le vecteur d'interruption numéro 1Ch. Au retour de l'interruption 1Ch, le signal EOI est enfin envoyé au circuit controleur des interruptions et les registres du processeur sont restaurés.

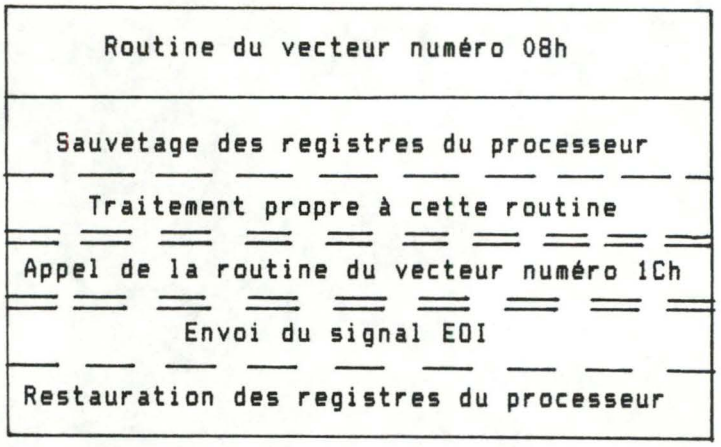

Figure 5.7 Représentation schématique des opérations réalisées par la routine associée au vecteur d'interruption 08h.

Cependant, notre programme est exécuté à partir d'une routine activée par l'interruption 1Ch. Par conséquent, le signal EOI n'a pas encore été envoyé et il est de notre devoir d'émettre ce signal dès le début de l'exécution de notre programme afin de réactiver les interruptions de type "INTR" de priorité inférieure. En effet, l'oubli de ce signal au début de notre application bloquerait tout le système car le vecteur numéro 08h correspond à l'interruption de type "INTR" de priorité la plus élevée.

4) Tester l'état du contact de l'interface utilisateur.

5) Tester si un processus figure déjà dans la section critique constituée par le système d'exploitation.
On pourrait dire que ce test est mal placé car le processus s'exécutant dans cette routine ne désire pas ou à tout le moins ne désire pas encore utiliser le système d'exploitation. Nous avons placé ce test au début de la routine car quelle que soit la tâche choisie par le handicapé, le processus en train de s'exécuter utilisera au moins une fois les primitives du système d'exploitation. En effet, l'entrée dans notre programme s'accompagne de l'affichage d'une fenêtre à l'écran dans laquelle les services disponibles sont présentés à l'utilisateur. De plus, pour sortir du programme, il faut au minimum effectuer une pression sur le contact de l'interface utilisateur. Or la prise en considération d'une pression sur ce contact est réalisée par une primitive figurant dans la section critique. On pourrait se demander pourquoi nous n'avons pas scindé le système d'exploitation en plusieurs sections critiques capables d'effectuer un type de traitement bien spécifique. Nous aurions par exemple une section critique comportant toutes les primitives d'accès à l'écran, une autre réalisant les accès sur disque ou sur disquette. Nous n'avons pas réalisé cette solution pour la raison suivante: nous

ne disposons pas d'informations suffisamment détaillées au sujet des zones de communication du système d'exploitation. Nous craignons dans ce cas par exemple que certains résultats intermédiaires découlant de l'exécution des primitives d'accès à l'écran occupent la même zone en mémoire centrale que les résultats issus de l'exécution des primitives d'accès sur disque. L'interruption des premières pour exécuter les secondes risque par conséquent d'effacer certains résultats intermédiaires associés aux primitives interrompues.

6) Tester si un processus est déjà en train d'utiliser notre programme.
Rappelons que le vecteur d'interruption numéro 1Ch est référencé automatiquement 20 fois par seconde et qu'il permet d'interrompre le logiciel "standard" afin d'exécuter notre projet. Supposons que le logiciel "standard" soit en train de s'exécuter sur l'ordinateur. Tout à coup, le handicapé presse le contact de son interface utilisateur. Si aucun processus ne figure dans la section critique formée par le système d'exploitation, le contrôle de la machine est passé à notre application. Mais pendant que le handicapé utilise les services offerts par le projet AIDAMI, le vecteur d'interruption numéro 1Ch est toujours référencé 20 fois par seconde. Or le handicapé doit appuyer sur le contact de son interface s'il désire sélectionner un objet à l'écran. Par conséquent, si aucun processus ne figure dans le système d'exploitation et si l'utilisateur presse le contact de l'interface, un nouveau processus entre dans notre programme et perturbe le déroulement normal des opérations. Pour remédier à ce problème, nous considérons que notre projet constitue une seconde section critique. Celui-ci est isolé par deux primitives d'exclusion mutuelle: l'une à l'entrée et l'autre à la sortie. Nous ne permettons de cette manière l'utilisation de notre application que par un seul processus à un instant donné.

En résumé, le système dispose de deux sections critiques: la première est formée par les primitives du système d'exploitation et la seconde est constituée par notre programme. Ces deux sections critiques possèdent leur verrou respectif; celui-ci est positionné ou dépositionné par les primitives d'exclusion mutuelle.

7) Changer la pile avant de lancer l'exécution de notre application.
Le logiciel "standard" s'exécute en utilisant la pile qui lui est associée. Lorsqu'il est interrompu en raison d'un appel au vecteur d'interruption numéro 1Ch, cette même pile est toujours utilisée. Si le contrôle est passé à notre programme et que celui-ci utilise également cette pile, nous risquons de créer un dépassement de capacité, c'est-à-dire que les 64 Kbytes disponibles dans la pile peuvent être insuffisants pour assurer le bon déroulement du logiciel "standard" et de notre application. Nous avons par conséquent jugé bon d'associer une nouvelle pile à notre programme. Cette solution présente néanmoins l'inconvénient de nécessiter 64 Kbytes supplémentaires en mémoire centrale.

8) Exécuter notre application.

9) Exécuter une primitive d'exclusion mutuelle pour signaler que l'on sort de la section critique constituée par notre projet.

10) Restaurer la pile du logiciel "standard".

11) Restaurer les registres du processeur.

12) Rendre le contrôle au logiciel "standard" interrompu.

Ce gérant des processus très simplifié s'exécute "en sens unique". Nous entendons par "sens unique" le fait que le logiciel "standard" peut être interrompu afin d'exécuter le projet "AIDAMI", mais notre programme ne peut pas être interrompu par le logiciel "standard". Cependant, un tel gérant des processus est suffisant. En effet, lorsque le handicapé utilise les services du projet "AIDAMI", il n'y a aucune raison que le logiciel "standard" vienne interrompre ces opérations. Le retour à ce logiciel est demandé par l'utilisateur dès qu'il n'utilise plus notre application.

D'autre part, comment assurer la réalisation d'un système multi-tâches au sein de notre programme à l'aide de ce gérant des processus? En réalité, nous n'avons pas besoin de ce gérant car le système multi-tâches peut être réalisé uniquement en structurant de manière particulière notre programme. Nous avons en effet présenté au chapitre 2 une découpe hiérarchique de l'application. Les composants de cette hiérarchie comprennent un certain nombre de primitives disponibles dans notre programme. Chaque primitive réalise une opération bien spécifique et peut être utilisée par toutes les tâches disponibles dans le projet "AIDAMI". Or puisque nous utilisons la technique du multi-fenêtrage, chaque fenêtre représentée à l'écran accompagnée de la structure de données qui la décrit, contient toutes les informations utiles au sujet de l'état d'avancement de la tâche qu'elle représente. Le dialogue entre l'écran et l'utilisateur est en effet réalisé uniquement à l'aide de messages et de graphiques affichés à l'écran. Citons par exemple les boutons qui sont contrastés s'ils sont activés, les caractères d'un document sont affichés dans la fenêtre respective. Chaque primitive du composant "gestion écran" de la hiérarchie permet de modifier un aspect très spécifique dans une fenêtre. Il suffit par conséquent de mémoriser la fenêtre représentée à l'écran et de garder intacte la structure de données qui lui est associée pour conserver toutes les informations nécessaires pour reprendre cette tâche interrompue.

5.4 Mise en résidance du programme en mémoire centrale

Nous savons que le système mis à la disposition du handicapé doit lui permettre d'accéder rapidement et à tout moment aux services qui lui sont offerts par la machine. De plus, l'utilisateur peut exécuter des logiciels "standards" tout en ayant encore accès aux services du projet "AIDAMI". Nous avons vu que la réalisation de ces concepts nécessite l'utilisation de la multiprogrammation; c'est-à-dire que plusieurs programmes résident en même temps en mémoire centrale et sont exécutés de manière imbriquée dans le temps par le processeur de l'ordinateur. Ainsi, dans le cadre de ce projet, notre programme et le logiciel "standard" doivent figurer dans la mémoire centrale en même temps. Analysons davantage les facteurs à considérer pour réaliser efficacement cette mise en résidance des programmes en mémoire centrale.

Le handicapé doit pouvoir accéder aux services offerts par la machine à tout moment et rapidement. Nous pouvons résoudre ce problème en mémorisant notre programme en mémoire centrale et en le plaçant en mode résidant. De cette manière, ce programme est constamment présent en mémoire centrale et est directement accessible dès que le handicapé presse le contact de son interface. La mise en mode résidant d'un programme présente

la particularité suivante: le programme est chargé en mémoire centrale et à la fin de ce chargement, une primitive spéciale du système d'exploitation est utilisée pour le mettre en mode résidant. Lorsqu'un programme est résidant, il est possible de charger un autre logiciel en mémoire centrale sans recouvrir l'espace de la mémoire occupé par le programme résidant. Les deux programmes ainsi chargés en mémoire centrale sont totalement indépendants l'un de l'autre. Nous devons donc ensuite charger le logiciel "standard" en mémoire. Les deux programmes sont ainsi disponibles dans la mémoire. Le gérant des processus peut donner le contrôle à l'un ou l'autre programme.

Signalons que lorsqu'un programme est en mode résidant, il n'est plus possible de récupérer la zone de la mémoire qu'il occupe tant que l'utilisateur n'a pas effectué un redémarrage complet de l'ordinateur. (Nous entendons par "redémarrage" un RESET ou une extinction suivie d'une remise sous tension de la machine). Il est par conséquent défavorable de mettre le logiciel "standard" en mode résidant car la taille de la mémoire centrale est très limitée. En effet, le handicapé pourrait envisager d'utiliser plusieurs logiciels "standards" successivement: ceux-ci seraient tous mémorisés en mémoire centrale qui serait remplie très rapidement.

Opérations à effectuer pour mettre un programme en mode résidant

Un programme chargé en mémoire centrale occupe plusieurs zones dans cette mémoire.

La première zone contient le code des instructions du programme et est appelée "segment de code". Le registre CS contient l'adresse de la première cellule mémoire de cette zone.

Les variables, données et tampons associés au programme figurent dans la seconde zone de la mémoire centrale. Celle-ci est appelée "segment de données" et est localisée par l'adresse contenue dans le registre DS.

Enfin, à tout programme est associée une pile qui permet notamment de mémoriser les valeurs contenues dans les registres du processeur et les variables déclarées dans les procédures du programme. Cette troisième zone est pointée par le registre de segment de pile SS.

Nous avons signalé au chapitre 4 que la taille de ces zones ne peut excéder 64 Kbytes et que celles-ci peuvent se chevaucher si la place occupée par le code ou par les données est inférieure à ce maximum. La figure 5.8 illustre les trois zones occupées par un programme. Ce dernier est chargé en mémoire centrale par le système d'exploitation dans le premier emplacement libre de la mémoire, en partant des adresses basses, capable de contenir tout le programme. Les segments de code et de données sont localisés au début de cet emplacement, tandis que la pile occupe les 64 Kbytes à la fin de l'emplacement. Signalons que ces zones sont localisées en mémoire centrale dans cet ordre bien précis: le segment de code suivi du segment de données, et enfin la pile.

Remarque: Le chargement d'un fichier exécutable dont l'extension est ".EXE"
******** n'est pas toujours réalisé de cette manière. Par contre, cette technique est valable pour tous les fichiers ".COM".

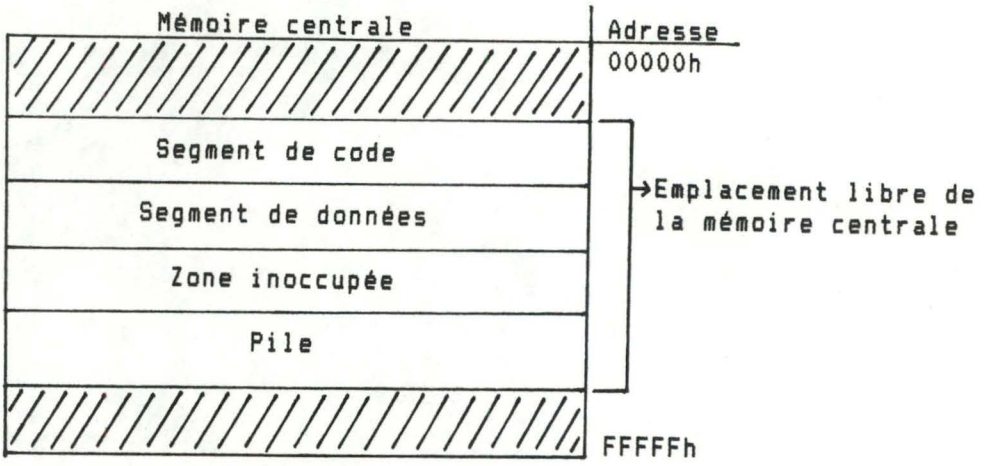

Figure 5.8 Zones occupées par un programme chargé en mémoire centrale.

Le système d'exploitation met à la disposition du programmeur plusieurs primitives permettant de mettre un programme en mode résidant. Parmi celles-ci, nous avons décidé d'utiliser la fonction 31h de l'interruption 21h du système d'exploitation car elle permet de réserver un emplacement libre de taille quelconque en mémoire centrale. Cette fonction nécessite un paramètre en entrée dans lequel doit être mémorisé le nombre de paragraphes à réserver en mémoire centrale pour y placer le futur programme résidant. Un paragraphe occupe 16 bytes en mémoire centrale. Le nombre de paragraphes à déterminer à l'entrée de la primitive du système d'exploitation représente la place mémoire à réserver pour contenir les trois zones associées au programme à rendre résidant. Ce nombre est déterminé de la manière suivante:

Nous pouvons déterminer le nombre de paragraphes nécessaires au segment de code en calculant la différence entre le registre de segment de code CS et le registre de segment de données DS. En effet, ceux-ci référencent respectivement le début du segment de code et le début du segment de données associés au programme. Le nombre de paragraphes occupés par le segment de code est donné par:

nbre-para-code = DS - CS

Le nombre de paragraphes occupés par le segment de données est déterminé de la manière suivante: les variables globales déclarées dans le programme sont mémorisées séquentiellement dans le segment de données, à la suite des tampons et données nécessaires aux librairies du langage "Pascal". Par conséquent, la dernière variable globale déclarée dans le programme est située au sommet de la zone occupée par les variables dans le segment de données. La valeur de l'adresse relative de la dernière variable globale déclarée dans le programme, augmentée de la place occupée par cette variable, représente l'adresse de la première cellule mémoire qui suit la zone occupée dans le segment de données. Cette adresse, divisée par la taille d'un paragraphe, donne

le nombre de paragraphes à réserver pour le segment de données. Ce nombre vaut donc:

$$\text{nbre-para-données} = \frac{\text{adresse relative de la 1ère cellule inoccupée}}{\text{taille d'un paragraphe}}$$

Remarque: il faut veiller à ce que la variable globale déclarée dans
******** le programme et utilisée pour déterminer la taille du segment de données à réserver, soit la dernière variable globale déclarée.

Il n'est pas facile de déterminer la place occupée par le segment de pile car ce dernier n'est utilisé que lors de l'exécution du programme. Nous avons par conséquent envisagé la solution suivante: réserver pour cette zone la taille maximale qu'elle peut occuper, soit 64 Kbytes ou 4096 paragraphes.

Le nombre total de paragraphes à réserver pour rendre le programme résidant est donné par l'expression:

nbre-total-para = nbre-para-code + nbre-para-données + 4096

Il suffit enfin d'appeler la primitive du système d'exploitation à la fin du programme à rendre résidant.

L'espace maximum que notre programme pourrait occuper en mémoire centrale vaut 3 * 64 Kbytes, soit 192 Kbytes. Les segments résidants du système d'exploitation nécessitent quant à eux environ 50 Kbytes de mémoire. D'où, la quantité maximale de mémoire nécessaire pour le DOS et le projet "AIDAMI" s'élève à 242 Kbytes. Si le handicapé désire en plus exécuter des logiciels "standards", il est indispensable de disposer d'une mémoire centrale plus importante. Dans ce cas, une mémoire centrale d'au moins 512 Kbytes s'impose.

5.5 Implémentation des primitives d'exclusion mutuelle

Les primitives d'exclusion mutuelle sont indispensables pour empêcher la réentrance dans certaines parties d'un programme. Nous présentons dans ce paragraphe ces primitives ainsi que la technique utilisée pour les implémenter dans notre programme.

Deux primitives sont nécessaires pour assurer l'exclusion mutuelle dans une section critique d'un programme:

** la première est située à l'entrée de la section critique: nous l'appelons "attendre(s)". La variable "s" constitue le verrou associé à ces primitives et est initialisée à "1" dans la phase d'initialisation de notre application. Les fonctionnalités de cette primitive sont données ci-dessous.

"attendre(s)" : * si s = 1 alors s := 0;
sinon placer le processus dans une file d'attente;

** la seconde primitive est localisée à la sortie de la section critique. Celle-ci est nommée "signaler(s)" et effectue les opérations suivantes:

"signaler(s)" : * s := 1;
* sélectionner un processus en attente dans la file et le faire entrer dans la section critique;

Nous avons choisi une solution simplifiée de ces primitives. Cette solution ne donne en effet aucun renseignement sur la manière dont la file d'attente est gérée. Nous pouvons cependant l'utiliser dans le cadre de notre projet pour la raison suivante: deux programmes au plus sont pris en charge par notre gérant des processus: le projet AIDAMI et le logiciel "standard". Si nous considérons une vue macroscopique du système, nous pouvons dire que deux processus seulement peuvent être actifs à un moment donné. D'autre part, nous faisons l'hypothèse que tout processus figurant dans une file d'attente jusque la libération de la section critique correspondante, effectue une attente active. Nous entendons par "attente active" le fait que le processus en attente boucle sur quelques instructions et reste à l'état "actif" pendant toute son attente. Par conséquent, il n'y aura jamais plus d'un seul processus dans les files associées aux sections critiques. Signalons enfin que l'exécution de ces deux primitives doit présenter un certain degré d'ininterruptibilité pour éviter que deux processus ne lisent ou ne modifient en même temps la valeur de la variable "s".

Réalisation pratique des primitives d'exclusion mutuelle

Les primitives d'exclusion mutuelle sont réalisées à l'aide de deux procédures: la première s'intitule "attendre(s)" et est appelée à l'entrée d'une section critique. La seconde, nommée "signaler(s)" est appelée dès qu'un processus sort d'une section critique. L'ininterruptibilité des instructions figurant dans ces procédures est réalisée en exécutant l'instruction en assembleur "CLI" au début de la procédure. En effet, seule l'interruption numéro 1Ch de l'ordinateur peut interrompre le logiciel "standard" pour exécuter notre application. Par contre, le logiciel "standard" ne dispose d'aucun moyen pour interrompre notre programme. Or nous savons que l'interruption numéro 1Ch est appelée par la routine de gestion des interruptions associée au vecteur numéro 08h. Ce dernier est référencé par l'horloge de l'ordinateur et constitue donc une interruption externe de type "INTR". Interdire les interruptions externes au début de la primitive d'exclusion mutuelle garantit donc bien qu'un processus peut exécuter cette primitive sans être interrompu. (Nous ne considérons pas ici les interruptions générées lors d'une chute de tension du réseau, etc... car les processus activés dans de telles circonstances n'exécutent pas les primitives d'exclusion mutuelle que nous avons créées).

Cependant, nous n'avons pas pu réaliser la primitive "attendre(s)" exactement comme elle a été définie ci-dessus. En effet, nous avons signalé au paragraphe 5.3 que le contrôle de la machine est donné au projet "AIDAMI" par l'intermédiaire de l'interruption numéro 1Ch. Cette dernière interrompt le logiciel "standard" environ 20 fois par seconde. L'entrée d'un processus dans le programme "AIDAMI" ne peut être autorisée que si aucun processus ne figure dans les sections critiques formées par le système d'exploitation ou par le programme "AIDAMI" lui-même. Les verrous associés à ces deux sections critiques sont positionnés et dépositionnés respectivement avant et après leur section critique respective. Mais la vérification de l'état de ces verrous est réalisée dans l'interruption numéro 1Ch pour voir si un processus peut entrer dans le programme "AIDAMI". (Nous pouvons dire que le logiciel "standard" possède une priorité d'accès aux sections critiques supérieure au programme "AIDAMI"). Nous savons en effet que ce dernier utilise le système d'exploitation dès que un processus y pénètre. Par conséquent, la primitive "attendre(s)" doit être scindée en deux parties. La première partie précède une section critique et effectue l'opération:

attendre(s): si s = 1 alors s := 0;

La seconde partie est localisée dans l'interruption 1Ch juste avant le point d'entrée dans le programme "AIDAMI", et réalise le traitement suivant:

si s = 1 alors donner le contrôle au programme "AIDAMI"
sinon ne rien faire et retourner au logiciel "standard"

En effet, si s = 0 avant le point d'entrée du programme "AIDAMI", cela signifie qu'un processus figure déjà dans une des sections critiques et le contrôle doit être rendu au logiciel "standard" puisque celui-ci a été interrompu par l'interruption numéro 1Ch. Ce dernier a en effet une priorité d'accès aux sections critiques plus grande que le programme "AIDAMI". Pour réaliser cela, il suffit de terminer au plus vite la routine de gestion de l'interruption numéro 1Ch. La figure 5.9 illustre la méthode utilisée pour implémenter les primitives d'exclusion mutuelle.

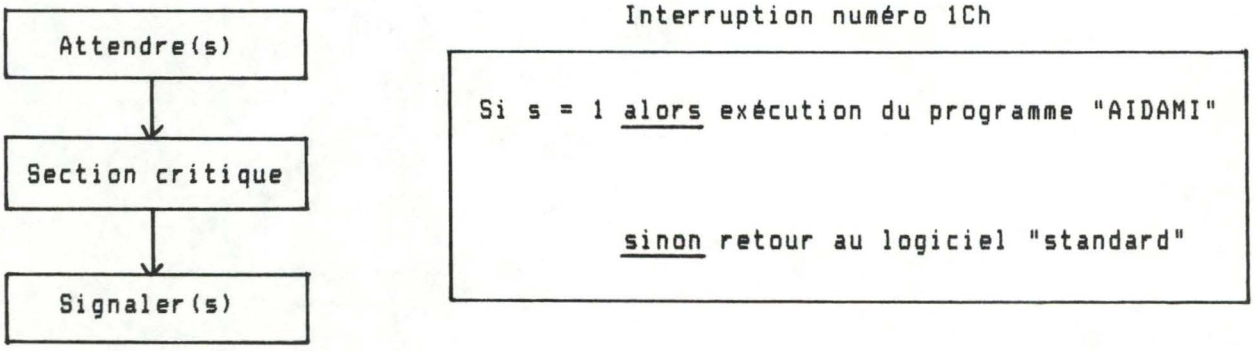

Figure 5.9 Méthode utilisée pour implémenter les primitives d'exclusion mutuelle.

Remarque:
****

Nous avons signalé au chapitre 1 que, dans le cadre de cette application, les primitives d'exclusion mutuelle permettent de garder un programme structuré tout comme les moniteurs. En effet, nous avons identifié deux sections critiques: la première est constituée par le système d'exploitation et la seconde est formée par le projet "AIDAMI". Les primitives d'exclusion mutuelle sont localisées à l'entrée et à la sortie de ces deux sections critiques. Par conséquent, aucune de ces primitives ne figure dans le segment de programme qui réalise les services du projet "AIDAMI". Seules les quelques routines assurant la multiprogrammation contiennent ces primitives.

5.6 Installation de l'interface utilisateur

Le handicapé peut entrer en communication avec l'ordinateur grâce à son interface utilisateur. Celui-ci est composé d'un contact, ou d'un joystick (manche à balai) accompagné d'un contact, ou de tout autre objet que le handicapé peut manipuler plus ou moins adroitement et qui peut être utilisé pour envoyer des informations à l'ordinateur. Quel que soit le type d'interface utilisé, il faut que les signaux qu'il émet soient compréhensibles par l'ordinateur. Des conventions doivent être établies pour assurer cette compréhension.

Nous avons mentionné au chapitre 1 que l'utilisation de logiciels de graphisme par le handicapé nécessite un interface plus élaboré qu'un simple contact. Nous avons décidé d'utiliser un interface composé d'un joystick accompagné d'un contact. Ce type d'interface permet de mouvoir un curseur et de sélectionner un objet quelconque représenté à l'écran par une simple pression sur le contact (voir paragraphe 1.2.2). Nous présentons dans ce paragraphe les principes utilisés pour installer cet interface utilisateur. Nous analysons le logiciel nécessaire pour gérer cet interface et discutons des possibilités de raccordement d'un tel type d'interface à l'ordinateur.

5.6.1 Développement du logiciel de l'interface utilisateur

L'interface utilisateur est dans le cadre de cette étude, composé d'un joystick et d'un contact. Il permet de mouvoir un curseur à l'écran et d'exécuter des actions spécifiques en poussant sur le contact. En fait, le fonctionnement d'un tel interface est tout à fait identique à celui d'une "souris" que l'on trouve sur un grand nombre de micro-ordinateurs. Nous nous sommes informés au sujet de ces "souris" et avons appris qu'il existe des logiciels spécifiques permettant d'assurer leur gestion. Nous avons alors jugé bon d'utiliser un de ces logiciels plutôt que de réécrire nos propres primitives pour gérer l'interface utilisateur. Le logiciel choisi s'appelle "MS-MOUSE" et met à la disposition de l'utilisateur un ensemble de primitives qui permettent de gérer efficacement la "souris" connectée à l'ordinateur. Ces primitives sont accessibles dans le cas du "Turbo-Pascal" en référençant le vecteur d'interruption numéro 33h de la machine. Nous disposons notamment des primitives suivantes:

** faire apparaître le curseur à l'écran.
** faire disparaître le curseur de l'écran.
** déplacer le curseur.
** lire la position du curseur et l'état du bouton de la "souris".

Nous suggérons au lecteur de consulter le manuel de référence de ce logiciel [7] pour disposer de plus amples détails à son sujet.

Une utilisation judicieuse de ces primitives permet d'établir la communication entre la "souris" et l'ordinateur. Puisque le principe de fonctionnement de l'interface utilisateur est identique à celui de la "souris", le handicapé peut également dialoguer avec la machine en utilisant ce logiciel.

Nous avons signalé au chapitre 3 que nous désirons remplacer temporairement l'interface utilisateur par des touches du clavier réel de la machine, pour des raisons de commodité. Dans ce cas, les touches fléchées du clavier permettent de modifier la position du curseur à l'écran; les touches de fonction F9 et F10 offrent les mêmes fonctionnalités que le bouton de la "souris". Le remplacement du bouton de la "souris" par les touches F9 et F10 du clavier a posé problème car le logiciel de la "souris" n'offre pas de primitive permettant de modifier le contenu de la variable qui représente l'état du bouton de la souris. En effet, lorsque la "souris" est utilisée avec un micro-ordinateur, celle-ci est raccordée à une carte électronique spéciale qui met à jour les variables du logiciel dès que le bouton de la "souris" est manipulé. Nous avons par conséquent été obligés de réaliser cette mise à jour dans notre propre programme. Nous devons en effet simuler parfaitement le fonctionnement de la "souris" en modifiant les variables de son logiciel car il ne faut pas oublier que notre application doit assurer le bon fonctionnement des logiciels "standards" et que ceux-ci réagissent en fonction du contenu de ces variables. Pour résoudre ce problème, nous avons déterminé à l'aide de l'utilitaire "DEBUG" les adresses des variables contenant les informations au sujet du bouton de la "souris". Notre programme modifie le contenu de ces variables chaque fois que les touches F9 et F10 sont manipulées. Cette solution porte atteinte à la portabilité de notre application car différentes versions du logiciel de la "souris" existent actuellement sur le marché. Les adresses de ces variables peuvent donc changer. Cependant, la simulation de l'interface utilisateur à l'aide du clavier n'est que temporaire.

Signalons également que le logiciel de la souris constitue une section critique et qu'il ne peut par conséquent être utilisé que par un seul processus à la fois. Or lorsque l'utilisateur appuie sur les touches fléchées ou sur les touches F9 ou F10 du clavier, une ou plusieurs primitives du logiciel de la souris sont exécutées. Si l'appui sur une de ces touches est réalisé lorsque le programme "AIDAMI" ou le logiciel "standard" utilise déjà le logiciel de la souris, le processus en train d'exécuter une primitive du logiciel de la souris est interrompu pour passer le contrôle à la routine de gestion du clavier. Dans ce cas, un second processus désire utiliser le logiciel de la souris. Nous avons donc été contraints de placer des primitives d'exclusion mutuelle à l'entrée et à la sortie des primitives du logiciel de la souris afin d'éviter toute réentrance. Nous identifions de cette manière une troisième section critique dans le système.

Remarque: nous avons intercepté l'interruption numéro 09h du système
******** d'exploitation, appelée lors de l'appui ou de la relâche d'une touche du clavier, afin de modifier l'effet produit par les touches fléchées et F9 et F10.

5.6.2 Raccordement de l'interface utilisateur à l'ordinateur

Rappelons que le raccordement de l'interface utilisateur à l'ordinateur nécessite un circuit électronique pour adapter les signaux émis par l'interface et les rendre compréhensibles par l'ordinateur. La "souris" raccordée sur la plupart des micro-ordinateurs présente la même particularité: elle dispose également d'un circuit électronique qui envoie des informations vers le logiciel de la "souris". Si nous désirons connecter un joystick muni d'un contact sur ce circuit à la place de la "souris", il faut s'assurer que les signaux émis par le joystick sont strictement équivalents à ceux émis par la "souris". Le lecteur trouvera dans le manuel "Inside Machintosh" (volume 3 page III-25) [2] le principe de fonctionnement hardware de la "souris".

Il est par ailleurs possible d'utiliser le joystick même si l'ordinateur n'est pas équipé du circuit électronique associé à la "souris". En effet, il existe des cartes électroniques vendues dans le commerce spécialement conçues pour assurer le raccordement d'un joystick à l'ordinateur. Notre projet devrait dans ce cas mettre en forme les signaux émis par le joystick et reçus sur le port 201h de la machine afin de les rendre compréhensibles par le logiciel de la "souris".

5.7 Simulation du clavier par l'ordinateur

Le handicapé communique avec l'ordinateur à l'aide de son interface utilisateur. Il doit ainsi avoir la possibilité de disposer de toutes les ressources de l'ordinateur sans utiliser le clavier réel de la machine. Pour cela, nous avons signalé au chapitre 1 que le clavier réel est simulé à l'écran; c'est-à-dire que les touches du clavier sont dessinées à l'écran. L'utilisateur a la possibilité de sélectionner ces différentes touches en manipulant adéquatement son interface. Lorsqu'une de ces touches simulées est sélectionnée, notre application doit réaliser les mêmes opérations que si cette touche avait effectivement été enfoncée sur le clavier réel. La réalisation de ces opérations est indispensable pour que le logiciel "standard" réagisse convenablement aux demandes faites par le handicapé. Nous allons à présent examiner un peu plus en détail les opérations à effectuer pour simuler le comportement du clavier réel de l'ordinateur.

Nous supposons que le handicapé vient de sélectionner une touche au clavier simulé à l'écran, à l'aide de son interface. Les opérations suivantes doivent ensuite être réalisées:

1) identifier la touche sélectionnée par l'utilisateur à l'écran. Cette touche appartient à l'un des deux types suivants: soit elle est de type "fonction" (par exemple les touches "Ctrl", "Alt", "Insert"), soit elle correspond à un caractère affichable à l'écran (par exemple les lettres de l'alphabet, les symboles spéciaux).

2) déterminer le "code de recherche" et le code ASCII associés à cette fonction ou à ce caractère. Ces codes sont identifiés à partir des tables situées en annexe "H". Par exemple, si le caractère "a" est sélectionné au clavier simulé, son "code de recherche" vaut 30 et son code ASCII a pour valeur le nombre 97.

Mais les touches correspondant à une fonction nécessitent en plus la mise à jour de quelques variables qui indiquent l'état actuel du clavier. Nous entendons par "état" par exemple le fait que la commande "Insert" soit active, que la touche "Ctrl" soit enfoncée. Ces variables d'état sont appelées "KB_FLAG" et "KB_FLAG_1" dans la zone de communication du système d'exploitation (paragraphe 4.7). Nous devons par conséquent également mettre à jour ces variables pour assurer le bon fonctionnement des logiciels "standards". Mais aucune primitive du système d'exploitation ne permet de modifier le contenu de ces variables. Nous devons dans ce cas y accéder directement dans la mémoire centrale, ce qui porte atteinte à la portabilité du programme. Cependant, les adresses de ces variables sont déclarées au début de notre programme: leur modification est donc très aisée. Bref, nous devons déterminer les codes permettant de modifier le contenu de ces variables lorsqu'une touche de fonction est sélectionnée.

3) mémoriser le "code de recherche" et le code ASCII dans le buffer du clavier. Pour les touches de fonction, il faut en plus envoyer le code permettant de modifier le contenu des variables d'état du clavier.
La mémorisation des codes ASCII et de recherche dans le buffer du clavier nécessite la réalisation préalable d'autres traitements plus spécifiques.

** Le premier traitement consiste à incrémenter le pointeur "BUFFER_TAIL" associé au buffer, de deux octets (ou à le faire pointer vers le début du buffer du clavier pour assurer sa gestion circulaire). Rappelons que ce pointeur mémorise le déplacement du dernier caractère introduit dans le clavier.

** Nous devons ensuite vérifier si la nouvelle valeur de ce pointeur est égale à la valeur du pointeur "BUFFER_HEAD" qui contient le déplacement du premier caractère disponible dans le buffer. Si les valeurs de ces deux pointeurs sont identiques, cela signifie que le buffer du clavier est déjà rempli. Les codes ne peuvent pas être mémorisés et sont perdus. Par contre, si leurs valeurs ne sont pas égales, les codes sont mémorisés dans le buffer dans l'ordre suivant: "code de recherche" suivi du code ASCII.

L'envoi du code de modification des variables d'état du clavier présente les caractéristiques suivantes: nous savons que deux octets caractérisent l'état du clavier. Chacun de leur bit indique l'état d'une touche particulière du clavier. Ces bits sont modifiables individuellement.

Deux types de modification sont possibles sur les variables d'état:

a) une mise à jour permettant d'activer une touche de fonction du clavier: un bit particulier des variables d'état doit être positionné à "1" à l'aide d'une fonction logique "OU".

b) une modification permettant de désactiver une touche de fonction du clavier: un bit doit être positionné à "0" en effectuant une opération logique "ET".

Nous sommes à nouveau obligés d'accéder aux variables et buffer du clavier par voie directe en mémoire centrale. Les exigences liées à la portabilité ne sont pas non plus respectées.

CONCLUSIONS

CONCLUSIONS

L'étude réalisée dans le cadre de ce mémoire a permis d'améliorer certains aspects du projet "prototype" réalisé par l'équipe "AIDAMI". Les grands objectifs poursuivis dans cette étude, tels que mettre à la disposition du handicapé l'entièreté des services de notre programme à tout moment, ainsi que lui permettre de réaliser des graphiques ou des dessins, ont été atteints. Cependant, de gros efforts sont encore à faire pour améliorer les performances du produit développé. En effet, nous avons essayé d'optimaliser la structure de l'application "AIDAMI" en vue de faciliter au maximum toute modification future. Mais ceci a pour conséquence de ralentir fortement le temps de réponse du système, qui en devient parfois intolérable. Ce temps de réponse peut cependant être considérablement réduit en utilisant judicieusement les outils logiciels (en particutier le gestionnaire de fenêtres "Turbo-Graphix") mis à notre disposition pour développer ce projet. En effet, tout affichage de fenêtre à l'écran s'accompagne d'un sauvetage préalable du contenu de l'écran sur le disque, de même que l'affichage de la fenêtre proprement dit nécessite beaucoup de temps car la description de celle-ci est disponible dans un fichier mémorisé sur le disque. Le logiciel permettant de gérer les fenêtres permet d'éviter un grand nombre de ces accès sur disque. Nous n'avons malheureusement pas eu le temps d'exploiter toutes les richesses de ce logiciel.

Le programme développé dans cette étude est entièrement écrit en "Turbo-Pascal". Ce langage de haut niveau permet de structurer adéquatement notre programme, malgré l'utilisation à certains endroits critiques d'instructions de très bas niveau, mais qui font partie intégrante du langage.

Signalons également qu'un des objectifs de ce mémoire consiste à étudier en détail un micro-ordinateur de type "IBM-PC compatible" en vue d'y implémenter efficacement le projet "AIDAMI". C'est la raison pour laquelle toute notre attention a été portée sur la manière de réaliser la multiprogrammation à l'aide d'un système d'exploitation qui ne répond pas à cette exigence. Nous avons recherché une solution optimale pour implémenter la multiprogrammation en vue d'assurer une portabilité maximale du projet "AIDAMI". Nous n'avons cependant pas trouvé de solution répondant parfaitement à cette exigence. Mais nous avons localisé cette déficience dans un nombre très limité de modules du programme afin d'effectuer rapidement les modifications qui s'imposent. D'autre part, nous avons consacré beaucoup de temps pour implémenter la multiprogrammation sur l'ordinateur car la liaison entre la théorie et la pratique n'est pas souvent bien établie. La programmation proprement dite de l'application est moins importante à nos yeux; c'est pourquoi nous n'avons réalisé et implémenté qu'une partie des services que le système "AIDAMI" est susceptible d'offrir au handicapé. La présentation des messages à l'écran est également très simple. La poursuite de ce projet devra notamment porter sur ces deux aspects.

Enfin, nous tenons à mentionner que le développement de ce projet dans le cadre de ce mémoire était initialement prévu sur un micro-ordinateur de type "Machintosh". Mais au cours de notre étude, nous avons abouti à la conclusion que ce type d'ordinateur n'est pas suffisamment "ouvert" au monde extérieur, c'est-à-dire qu'il n'est pas aisé d'y connecter des périphériques et interfaces tels que ceux nécessités par le projet "AIDAMI". Nous entendons par là le fait qu'il est en général nécessaire de

"bricoler" une carte électronique pour assurer le raccordement des interfaces et périphériques du système "AIDAMI" à l'ordinateur. Or nous désirons, dans le cadre de ce projet, utiliser au maximum des circuits électroniques standardisés et vendus dans le commerce pour effectuer ces raccordements, puisque le projet "AIDAMI" est destiné à être commercialisé. Ce n'est qu'au mois de janvier, après que nous ayons confirmé notre intuition, que nous avons décidé de changer de type de micro-ordinateur et d'implémenter ce projet sur un ordinateur de type "IBM-PC compatible".

ANNEXES

ANNEXE A : Services et tâches disponibles dans le système AIDAMI

Les besoins exposés au paragraphe 1.2 peuvent être satisfaits grâce à différents services qui se décomposent en actions ou tâches et fonctions diverses. Un exposé détaillé de ces services est donné ci-après.

Le service "courrier"

Le but du service "courrier" est de fournir à l'utilisateur la possibilité d'éditer un texte à l'aide du clavier simulé affiché sur l'écran de la machine.

Les diverses actions disponibles dans ce service sont:

- L'action "répertoire": cette action est composée elle-même de plusieurs fonctions:
  - * Liste des fichiers: affichage des fichiers présents sur le disque ou sur la disquette.
  - * Choix d'un fichier de travail: sélection d'un fichier qui fera l'objet d'une manipulation ultérieure.
  - * Création d'un fichier de travail: création d'un nouveau fichier.
  - * Destruction d'un fichier existant: supprime un fichier.
  - * Archivage d'un fichier existant: duplication d'un fichier.
  - * Vider le contenu d'un fichier: effacer les informations mémorisées dans un fichier.
- Action "clavier": cette action permet de sélectionner différents types de claviers en fonction de l'usage qu'en fait l'utilisateur.

  Les types de claviers sont:

  - Dactylographie: pour l'édition de texte.
  - Mathématique: pour les calculs scientifiques.
  - Turbo-Pascal: pour la programmation en "turbo-pascal".
  - Basic: pour la programmation en basic.
  - Fortran: pour la programmation en fortran.
  - Jeux: clavier spécial simplifié.

- Azerty: si l'utilisateur est habitué à travailler avec ce type de clavier.

La sémantique et la disposition des touches des différents types de claviers maximisent la vitesse d'édition d'un texte à l'écran. Par exemple, le clavier du type "turbo-Pascal" comporte des touches dont l'appui sur l'une d'elles contribue à afficher à l'écran tout un mot-clé utilisé fréquemment dans le langage de programmation "Pascal".

- Action "écriture": active le balayage du clavier affiché à l'écran.

- Action "relecture": relit un fichier page par page.

- Action "impression": imprime le contenu d'un fichier.

  Les fonctions disponibles sont:

  - Défiler le papier: fait avancer le papier à l'imprimante.

  - Imprimer le texte: imprime le fichier.

- Action "cahier": ouverture et affichage du fichier "cahier".

- Action "agenda": ouverture et affichage du fichier "agenda".

- Action "annuaire": ouverture et affichage du fichier "annuaire".

- Action "lecture notes": lit et mémorise une note dans un fichier.

- Action "lecture fichier": affichage d'un fichier sur le deuxième écran page par page.

Le service "téléphone"

- Action "sélection": cette action comprend les fonctions suivantes:

  * Composition d'un nouveau numéro: permet de créer un nouveau numéro de téléphone.

  * Sélection du dernier numéro composé: le dernier numéro de téléphone sélectionné devient le numéro d'appel courant.

  * Sélection d'un numéro dans l'annuaire: permet de choisir un numéro de téléphone dans l'annuaire.

- Action "décrocher": cette action réalise le décrochage du téléphone (mains-libres).

- Action "raccrocher": cette action réalise le raccrochage du téléphone.

- Action "discrétion": réduit l'intensité sonore du haut-parleur du "mains-libres" afin de rendre la communication plus intime.

- Action "bloc-notes": permet de prendre des notes pendant l'exécution d'une autre tâche.

- Action "annuaire": permet de relire le fichier "annuaire".

- Action "agenda": permet de relire le fichier "agenda".

Le service "relais"

Ce service est divisé en plusieurs fonctions élémentaires qui agissent comme des interrupteurs. Le choix d'une de ces fonctions ouvre ou ferme un des interrupteurs électroniques ou mécaniques auxquels sont connectés les appareils électriques extérieurs.

Le service "télécommande"

Ce service met en fonctionnement les appareils munis d'une télécommande à distance.

Les différentes fonctions disponibles sont:

- La fonction "télévision": les touches de la télécommande réelle sont simulées à l'écran. La sélection de l'une de ces touches simulées provoque la même réaction de la part du téléviseur que si une personne avait actionné la même touche sur la télécommande réelle.

- La fonction "magnétoscope": réalisation du même principe que pour la télévision.

- La fonction "chaîne hi-fi": réalisation du même principe que pour la télévision.

- La fonction "persiennes": cette fonction doit permettre l'ouverture et la fermeture des persiennes, mais elle n'est pas encore implémentée.

Le service "outils"

Une série d'accessoires de bureau sont disponibles dans le service "outils" afin d'aider l'utilisateur lors de la réalisation de ses tâches. Ce service comprend les utilitaires suivants:

- Horloge: affichage de l'heure générée par le système.
- Calculatrice: mise à la disposition de l'utilisateur d'une calculatrice.
- Bloc-notes: permet de prendre des notes pendant l'exécution d'une seule tâche.
- Mise à jour: manipulation des notes prises dans le bloc-notes à l'aide des fonctions suivantes:
  * "conserver la page de notes": sauver la page sur disque.
  * "jeter la page de notes": effacer la page.
  * "recopier la page dans le cahier": transférer la page dans le fichier "cahier".
  * "imprimer la page de notes".
  * "page suivante": afficher la page de notes suivante.
- Vitesse clavier: permet de choisir une vitesse de balayage du clavier.
- Vitesse menu: permet de choisir une vitesse de balayage des menus affichés à l'écran.
- Mode d'emploi: mode d'emploi de certaines fonctions du programme.
- Agenda: ouverture et affichage du fichier "agenda".

Le service "Logiciels"

Ce service permet d'utiliser un logiciel "standard" compatible avec le micro-ordinateur.

Les actions suivantes sont implémentées:

- Action "répertoire": Affiche à l'écran les différents logiciels exécutables présents dans le système.

- Action "clavier": affiche à l'écran un clavier simulé d'un certain type qui permet la manipulation des logiciels présents dans le système.

- Action "écran": permet d'échanger les contenus affichés sur les deux écrans de la machine.

- Action "IBM-PC": affiche la liste des fichiers exécutables sur un ordinateur de type "IBM-PC compatible".

- Action "Apple 2": affiche la liste des fichiers exécutables sur un ordinateur de type "Apple 2".

Le service "sortie"

Le service "sortie" permet de sortir du système "AIDAMI". Le système d'exploitation du micro-ordinateur reprend alors le contrôle des commandes.

ANNEXE B : Explicitation des primitives et des fichiers identifiés dans les composants de la hiérarchie de l'application

Nous énumérons ici les primitives et structures de données relatives à chaque composant du graphe hiérarchique de l'application. Cette liste n'est certes pas exhaustive, mais nous avons repris les fonctions qui nous paraissent à priori fondamentales.

* Entrées :
  - Lecture d'une chaîne de caractères dans une fenêtre.
  - Lecture de l'état d'un bouton multiple sélectif (*).
  - Lecture de l'état d'un réglage continu (**).
  - Vérifications syntaxiques.
  - Lecture de l'état du bouton de la souris.
  - Lecture de la position du curseur de la souris.

* Sorties :
  - Afficher les composants d'une fenêtre et la fenêtre localisée à un endroit précis à l'écran.
  - Effacer les composants d'une fenêtre et la fenêtre localisée à un endroit précis à l'écran.
  - Désactiver un composant d'un menu.
  - Activer un composant d'un menu.
  - Déplacer le curseur à un certain endroit à l'écran.
  - Faire apparaître le curseur.
  - Faire disparaître le curseur.
  - Changer l'aspect du curseur.
  - Activer le balayage du clavier.
  - Désactiver le balayage du clavier.

* Formatage signaux :
  - Imprimer un fichier.
  - Faire défiler le papier à l'imprimante.
  - Lire et préparer les codes de sortie des appareils électriques.

* Gestion écran :
  - Modifier l'état d'un bouton simple.
  - Afficher une fenêtre d'un certain type.
  - Rechercher un emplacement d'une chaîne de caractères dans une fenêtre affichée à l'écran.
  - Afficher un bouton simple.
  - Afficher une chaîne de caractères dans une fenêtre.
  - Sauver le contenu d'une fenêtre.

(*) Un bouton multiple sélectif est un bouton simple associé à d'autres boutons simples. Ces boutons associés dépendent l'un de l'autre en ce sens que seul un bouton de ce groupe peut être actif à un moment donné. Ce bouton est caractérisé par deux états: "actif" ou "inactif".

(**) Un réglage continu est un bouton caractérisé par un nombre d'états supérieur à deux. Ces états sont représentés par des nombres entiers. Par exemple, un bouton peut prendre les états "1" ou "2" ou "3"; l'état "2" étant un état intermédiaire.

- Contraster un composant.
- Désactiver le contraste d'un composant.
- Echanger le contenu des écrans.
- Dessiner une ligne à l'écran.
- Changer d'écran le curseur de la souris.
- Déplacer une fenêtre.
- Modifier la taille d'une fenêtre.
- Délimiter une zône maximale de mobilité du curseur.
- Transformer les coordonnées globales du curseur (écran total) en coordonnées locales (écran partiel).
- Mettre à jour la table de conversion "écran" / "action à prendre".
- Table de conversion "écran" / "action à prendre".
- Insérer un élément dans la table de conversion.
- Supprimer tous les éléments d'une fenêtre de la table de conversion.
- Déterminer l'action à effectuer en fonction de la position du curseur à l'écran.
- Table d'ordonnancement des fenêtres.
- Supprimer un élément de la table d'ordonnacement.
- Ajouter un élément dans la table d'ordonnacement.
- Table des fenêtres présentes en mémoire centrale.
- Mettre à jour la table des fenêtres.
- Rechercher le numéro de la dernière fenêtre insérée dans la table d'ordonnancement.

* Sortie auxil-imprimantes :
  - Imprimer une ligne d'un fichier.
  - Envoyer des signaux de sortie vers les appareils électriques extérieurs.

* Manip-fichiers :
  - Accès par clé dans un fichier.
  - Accès séquentiel en lecture dans un fichier.
  - Accès séquentiel en écriture dans un fichier de texte.

* Transformation formats:
  - Transformation du format de l'heure.
  - Transformation du format des calculs.
  - Transformation du format des dates.

* Outils :
  - Recherche de l'heure.
  - Lecture du répertoire des fichiers.
  - Changement de l'unité à disques.
  - Réglage de la vitesse de balayage du clavier.
  - Opérations arithmétiques.
  - Recherche de la date.

* Coordinateur:
  - Coordonner les processus.

* Système d'exploitation:

- Rendre un fichier de données disponible pour le processus qui désire l'utiliser.
- Exécuter un processus inactif en interrompant le processus en cours d'exécution.
- Ce module contient également des primitives relatives à chaque "sous-niveau" du système d'exploitation. Celles-ci sont exposées à l'annexe "E".

* Fichiers :

- Fichier contenant les codes de sortie pour différents types d'imprimantes.
- Fichier comprenant les formats des codes binaires à envoyer aux appareils électriques extérieurs.
- Fichier comprenant les formats des codes binaires à envoyer au "mains-libres".
- Ensemble de fichiers décrivant le contenu des fenêtres à afficher à l'écran au cours de l'exécution du programme.
- Fichier permettant de modifier la sémantique des touches du clavier réel.
- Fichier contenant la description des transformations de format à effectuer sur les dates, les calculs et l'heure.
- Fichier dans lequel sont mémorisées les erreurs pouvant survenir pendant l'exécution du programme.

Remarque: chacun de ces fichiers de données est créé grâce à un
******** programme spécial "creer_fich.COM" situé en annexe "J".

Les spécifications externes des primitives et fichiers implémentés dans notre programme peuvent être consultées dans l'annexe "C".

ANNEXE C: Spécifications externes des primitives identifiées dans la découpe hiérarchique de l'application et implémentées dans notre programme

Cette annexe présente les spécifications externes des primitives de la découpe hiérarchique de l'application, qui sont implémentées dans notre programme.

Composant "Entrées"

Lecture d'une chaîne de caractères dans une fenêtre

nom de la primitive: "lecture_string_fenetre".

Cette primitive permet de lire une chaîne de caractères dans un emplacement réservé dans une fenêtre.
Cette primitive lit un caractère introduit au clavier simulé, réalise l'écho de ce caractère dans l'emplacement de la fenêtre qui a été réservé et enfin lit la chaîne de caractères complète, affichée dans cet emplacement. Le caractère affiché en écho est placé derrière la chaîne de caractères éventuellement déjà présente dans l'emplacement.

arguments: - la chaîne de caractères à afficher à l'écran
- le caractère à ajouter en fin de cette chaîne et à afficher
- la couleur des caractères à afficher
- la hauteur des caractères
- le numéro de la fenêtre dans laquelle la chaîne de caractères est affichée
- x, y : les coordonnées de la chaîne de caractères

préconditions: - la couleur des caractères est un nombre entier tel que 0 <= couleur <= nb_couleurs
et - la hauteur des caractères est un nombre entier >= 1
et - le numéro de la fenêtre est un nombre entier tel que 1 <= numéro <= nb_max_fenetres
et - x et y sont des nombres entiers qui représentent les coordonnées du premier caractère gauche de la chaîne de caractères
et - 0 <= x <= 79
0 <= y <= 24

résultats: - affichage de la chaîne de caractères dans la fenêtre
et - affichage du caractère à ajouter en fin de cette chaîne
et - mise à jour de la chaîne de caractères: le caractère supplémentaire est ajouté en fin de cette chaîne

postconditions: - -----

Lecture de l'état du bouton de la souris
---
nom de la primitive: "lect_bout_souris"

Cette primitive permet de lire l'état du bouton de la souris ou du contact de l'interface utilisateur.

arguments: - -----

préconditions: - -----

résultats: - booléen : "actif_bouton"

postconditions: - "actif_bouton" = true si le bouton est enfoncé
= false sinon

Lecture de la position du curseur la souris
---
nom de la primitive: "lect_pos_souris"

Cette primitive permet de déterminer les coordonnées actuelles du curseur de l'écran.

arguments: - -----

préconditions: - -----

résultats: - coordonnées x et y de la position du curseur à l'écran

postconditions: - x et y sont des nombres entiers tels que
0 <= x <= 639
0 <= y <= 199

Composant "Sorties"
++++++++++++++++++

Afficher les composants d'une fenêtre et la fenêtre localisée à un endroit précis à l'écran
---
nom de la primitive: "affiche_fenetre"

Cette primitive permet d'afficher une fenêtre à l'écran ainsi que tous les objets qui lui sont associés.

arguments: - nom du fichier contenant la fenêtre sur le disque

préconditions: - -----

résultats: - affichage de la fenêtre et de ses composants à l'écran; la fenêtre est activée et les tables décrivant cette fenêtre sont mises à jour
ou - "erreur"

```
postconditions: - "erreur" inchangé et affichage de la fenêtre
            ou - "erreur" = 1 : "fichier non existant"
                          = 2 : "nom de fichier n'appartenant pas à la
                                 table des fichiers"
                          = 3 : "numéro d'enregistrement inexistant dans
                                 le fichier"
                          = 4 : "le fichier ne figure pas dans la table
                                 des fichiers"
                          = 6 : "le nombre maximum de fenêtres est déjà
                                 affiché à l'écran ou numéro de fenêtre
                                 inconnu dans la table d'ordonnancement"
                          = 7 : "données non mémorisées en mémoire
                                 centrale"
                          = 10: "tous les numéros de fenêtre sont déjà
                                 utilisés"
```

Effacer les composants d'une fenêtre et la fenêtre localisée à un endroit précis à l'écran

---

```
nom de la primitive: "efface_fenetre"

    Ce module permet d'effacer une fenêtre ainsi que tous les objets qui
lui sont associés.

arguments: - numéro de la fenêtre

préconditions: - le numéro de la fenêtre est un nombre entier

résultats: - [la fenêtre est effacée de l'écran
             et
             les tables d'ordonnancement, de fenêtre et de conversion sont
             mises à jour]
       ou - "erreur"

postconditions: - "erreur" inchangé et la fenêtre est effacée
            ou - "erreur" = 1 : "fichier inexistant"
                          = 11: "numéro de fenêtre invalide"
                          = 16: "aucune fenêtre affichée à l'écran"
                          = 18: "numéro de fenêtre ne référençant pas la
                                 dernière fenêtre affichée à l'écran"
```

Déplacer le curseur à un certain endroit à l'écran

```
nom de la primitive: "deplacer_curseur"

    Cette primitive permet de déplacer le curseur à un endroit donné de
l'écran.

arguments: - x, y : coordonnées où l'on désire déplacer le curseur

préconditions: - x et y sont des nombres entiers tels que
                 0 <= x <= 639
                 0 <= y <= 199
```

résultats: - déplacement du curseur à l'écran à la position déterminée par x et y

postconditions: - -----

Faire apparaître le curseur
---
nom de la primitive: "montrer_curseur"

Ce module permet de faire apparaître le curseur à l'écran.

arguments: - -----

préconditions: - le logiciel de la souris doit avoir été préalablement initialisé (fonction 0 de l'interruption 33h)
et - il faut s'assurer du même nombre d'appels aux fonctions du logiciel de la souris consistant à montrer et à cacher le curseur de l'écran

résultats: - le curseur de l'écran est visible

postconditions: - -----

Faire disparaître le curseur
---
nom de la primitive: "cacher_curseur"

Cette primitive permet de rendre le curseur de l'écran invisible.

arguments: - -----

préconditions: - -----

résultats: - le curseur n'est pas ou n'est plus visible à l'écran

postconditions: - s'assurer du même nombre d'appels aux fonctions du logiciel de la souris consistant à montrer et à cacher le curseur de l'écran

Composant "Formatage signaux"
++++++++++++++++++++++++++++

Lire et préparer les codes de sortie des appareils électriques
---
nom de la primitive: "preparer_code_appareils"

Ce module permet de déterminer la séquence des signaux à envoyer vers les appareils électriques.

arguments: - le code à envoyer
- le numéro du pointeur de la structure du fichier comprenant les caractéristiques d'envoi du code

préconditions: - le code est représenté par un nombre entier
et - le numéro du pointeur est un nombre entier

résultats: - envoi du code vers l'appareil électrique
ou - "erreur"

postconditions: - le code est bien envoyé et "erreur" est inchangé
ou - "erreur" = 4 : "numéro de pointeur inexistant dans la table des fichiers"
= 22: "numéro de téléphone invalide"

Composant "Gestion écran"
++++++++++++++++++++++++

Afficher une fenêtre d'un certain type
---

nom de la primitive: "affiche_fen"

Ce module permet d'afficher une fenêtre sans les autres objets qui y sont associés.

arguments: - le numéro de la fenêtre à afficher
- xh, yh, xl, yl : une paire de coordonnées
- le type de la fenêtre
- la couleur de fond
- la couleur des caractères
- l'entête de la fenêtre

préconditions: - le numéro de la fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres
et - xh, yh sont des nombres entiers représentant les coordonnées du coin supérieur gauche de la fenêtre et tels que
0 <= xh <= 79
0 <= yh <= 199
et - xl, yl sont des nombres entiers représentant les coordonnées du coin inférieur droit de la fenêtre et tels que
0 <= xl <= 79
0 <= yl <= 199
et - xh <= xl
yh <= yl
et - le type de la fenêtre est connu et défini dans le programme
et - la couleur de fond est un nombre entier tel que 1 <= couleur fond <= nb_couleurs
et - la couleur des caractères est un nombre entier tel que 1 <= couleur caractère <= nb_couleurs

résultats: - affichage de la fenêtre à l'écran

postconditions: - -----

Rechercher l'emplacement d'une chaîne de caractères dans une fenêtre affichée à l'écran

---

nom de la primitive: "rech_emplace_string"

Cette primitive permet de déterminer les coordonnées en x et y d'un emplacement réservé dans une fenêtre pour y afficher une chaîne de caractères.

arguments: - le numéro de l'emplacement de la chaîne de carastères
- le numéro de la fenêtre comprenant cet emplacement
- la table de conversion associée à cette fenêtre

préconditions: - le numéro de l'emplacement est un nombre entier >= 1
et - le numéro de la fenêtre est un nombre entier tel que 1 <= numéro <= nb_max_fenetres

résultats: - [x,y : coordonnées de l'emplacement.
et
la couleur des caractères à afficher dans cet emplacement
et
la hauteur des caractères à afficher]
ou - "erreur"

postconditions: - [l'emplacement est trouvé et "erreur" est inchangé
et
0 <= x <= 79
0 <= y <= 24
0 <= couleur <= nb_couleurs
et
les coordonnées x et y sont les coordonnées du premier caractère qui sera affiché dans cet emplacement
et
la hauteur des caractères est un nombre entier tel que 1 <= hauteur]
1 < hauteur
ou - "erreur" = 19 : " pas d'emplacement réservé pour une chaîne de caractères dans cette fenêtre"

Afficher un bouton simple

---

nom de la primitive: "affiche_bouton"

Cette primitive permet d'afficher un bouton simple à l'écran.

arguments: - le numéro du groupe du bouton
- l'état du bouton
- la couleur de fond
- la couleur du caractère
- les coordonnées xh, yh, xl, yl du bouton
- le numéro de la fenêtre dans laquelle on affiche le bouton
- le type de l'objet à afficher

préconditions: - le numéro du groupe est un nombre entier tel que 0 <= numéro du groupe <= N
et - l'état du bouton est un nombre entier tel que 0 <= état <= N
et - la couleur de fond est un nombre entier tel que 1 <= couleur de fond <= "nb_couleurs" qui est le nombre maximum de couleurs permises à l'écran
et - la couleur du caractère est un nombre entier tel que 1 <= couleur caractère <= nb_couleurs
et - "xh, yh" sont les coordonnées du coin supérieur gauche si le bouton est un rectangle
et - 0 <= "xh" <= "x_max_world" = coordonnée maximale dans une fenêtre selon l'axe des "X"
0 <= "yh" <= "y_max_world"
coordonnée maximale dans une fenêtre selon l'axe des "Y"
et - "xl, yl" sont les coordonnées du coin inférieur droit si le bouton est un rectangle
et - 0 <= xl <= x_max_world
0 <= yl <= y_max_world
et - le numéro de fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres: nombre maximum de fenêtres affichables en même temps à l'écran
et - le type d'objet est un nombre entier identifiant un type d'objet déclaré dans le programme

résultats: - affichage du bouton à l'écran dans la fenêtre

postconditions: - -----

Afficher une chaîne de caractères dans une fenêtre
---

nom de la primitive: "affiche_carac"

Cette primitive permet d'afficher une chaîne de caractères à un endroit précis de l'écran.

arguments: - x, y : coordonnées
- couleur des caractères à afficher
- facteur d'échelle de la grandeur des caractères
- numéro de la fenêtre
- valeur de la chaîne de caractères

préconditions: - x, y : entiers représentant les coordonnées du point au début de la chaîne de caractères
et - 0 <= x <= x_max_world
0 <= y <= y_max_world
et - la couleur du caractère est un nombre entier tel que 1 <= couleur caractère <= nb_couleurs
et - le facteur d'échelle est un nombre entier tel que 1 <= facteur <= N
et - le numéro de fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres

résultats: - affichage de la chaîne de caractères dans la fenêtre

postconditions: - -----

Transformer les coordonnées globales du curseur en coordonnées locales
---

nom de la primitive: "global_to_local"

Ce module permet de transformer les coordonnées globales du curseur, relatives à tout l'écran, en coordonnées relatives à l'espace adressable d'une fenêtre donnée.

arguments: - x et y : coordonnées globales du curseur à l'écran
- numéro de la fenêtre

préconditions: - x et y sont des nombres entiers tels que 0 <= x <= 79
0 <= y <= 199
et - les coordonnées xh, yh du coin supérieur gauche et les coordonnées xl, yl du coin inférieur droit de la fenêtre désignée sont telles que xh <= x <= xl
yh <= y <= yl
et - le numéro de la fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres

résultats: - les coordonnées globales du curseur de l'écran sont transformées en coordonnées X et Y définies dans l'espace adressable de la fenêtre désignée
ou - "erreur"

postconditions: - [X et Y sont des nombres entiers tels que
0 <= X <= x_max_world
0 <= Y <= y_max_world
et
"erreur" inchangé]
ou - "erreur" = 13 : "le numéro de fenêtre correspond à une fenêtre non définie dans la table des fenêtres"

Table de conversion "écran" / "action à prendre"

La table de conversion "écran" / "action à prendre" contient les éléments suivants pour chacune des fenêtres affichées à l'écran:

- un booléen "actif" permettant de savoir si la fenêtre est active. (Il ne peut y avoir qu'une seule fenêtre active à un moment donné).
- une liste chainée contenant la description des objets de type "rectangle",... affichés à l'écran dans la fenêtre. Chaque élément de la liste décrit un objet affiché et permet de vérifier si le curseur de l'écran est à l'intérieur d'un de ces objets. Si c'est le cas et si le bouton de la "souris" ou le contact de l'interface est enfoncé, le numéro de l'action à effectuer lors de la sélection d'un tel objet peut être lu dans l'élément de la liste qui décrit cet objet sélectionné.

Mettre à jour la table de conversion "écran" / "action à prendre"
---

nom de la primitive: "MAJ_table_conv"

Cette primitive permet de désactiver une fenêtre active et d'activer une fenêtre de numéro donné.

arguments: - numéro de la fenêtre

préconditions: - le numéro de la fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres

résultats: - activation de la fenêtre
et - désactivation de l'éventuelle autre fenêtre déjà active

postconditions: - -----

Insérer un élément dans la table de conversion
---

nom de la primitive: "inserer_table_conv"

Ce module permet d'ajouter un élément décrivant un objet d'une fenêtre, à la liste d'éléments qui décrit les objets associés à cette fenêtre. (NB: cette liste peut être initialement vide).

arguments: - type de l'objet à insérer dans la table
- coordonnées xh, yh, xl, yl de cet objet
- numéro de l'action associée à cet objet
- numéro du groupe du bouton
- numéro de la fenêtre comprenant cet élément

préconditions: - le type de l'objet est un nombre entier >= 0
et - xh, yh sont des nombres entiers correspondant aux coordonnées du coin supérieur gauche de l'objet si celui-ci est un rectangle
et - xl,yl sont des nombres entiers correspondant aux coordonnées du coin inférieur droit de l'objet si celui-ci est un rectangle
et - 0 <= xh, xl <= x_max_world
0 <= yh, yl <= y_max_world
xh <= xl
yh <= yl
et - le numéro de l'action est un nombre entier >= 0
et - le numéro de la fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres

résultats: - insertion de l'élément dans la table de conversion

postconditions: - -----

Supprimer tous les éléments d'une fenêtre de la table de conversion
---

nom de la primitive: "supprim_table_conv"

Cette primitive permet de supprimer la liste chainée des objets associés à une fenêtre.

arguments: - numéro de la fenêtre
- la table de conversion

préconditions: - le numéro de la fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres

résultats: - les éléments de la table de conversion sont supprimés et effacés de la mémoire centrale
et - le pointeur de la table qui pointe vers cette fenêtre est positionné à "nil"
et - désactivation de cette fenêtre

postconditions : - -----

Déterminer l'action à effectuer en fonction de la position du curseur à l'écran
---

nom de la primitive: "determiner_action"

Ce module permet de déterminer le numéro de l'action associé à un objet pointé par le curseur de l'écran, à partir des coordonnées actuelles du curseur.

arguments: - coordonnées x et y du curseur de l'écran

préconditions: - x et y sont des nombres entiers tels que 0 <= x <= 79
0 <= y <= 199

résultats: - [numéro de la fenêtre dans laquelle se trouve le curseur de l'écran
et
type de l'objet pointé par le curseur de l'écran
et
numéro du groupe de l'objet pointé
et
numéro de l'action associée à l'objet pointé]
ou - "erreur"

postconditions: - [le numéro de le fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres
et
"erreur" inchangé]
ou - "erreur" = 5 : "pas de fenêtre référencée à l'écran par le curseur"
= 8 : "le curseur pointe vers une fenêtre inactive"
= 12 : "le curseur pointe dans une fenêtre active, mais ne pointe vers aucun objet"

Table d'ordonnancement des fenêtres

La table d'ordonnancement des fenêtres est un tableau de nombres entiers dans lequel sont mémorisés les numéros des fenêtres affichées à l'écran. Le nombre d'éléments de ce tableau correspond au nombre maximum de fenêtres que l'on peut afficher en même temps à l'écran. Cette table permet de mémoriser l'ordre dans lequel les fenêtres ont été affichées à l'écran. Nous pouvons ainsi déterminer le numéro de la fenêtre qui a été affichée en dernier lieu et qui est la fenêtre active. (La première fenêtre affichée à l'écran n'est pas nécessairement identifiée par le numéro "1"). De plus, dès que la dernière fenêtre affichée est effacée de l'écran, nous pouvons connaître le numéro de la fenêtre qui a été affichée juste avant cette dernière fenêtre, et l'activer. La table est représentée à la figure C.1.

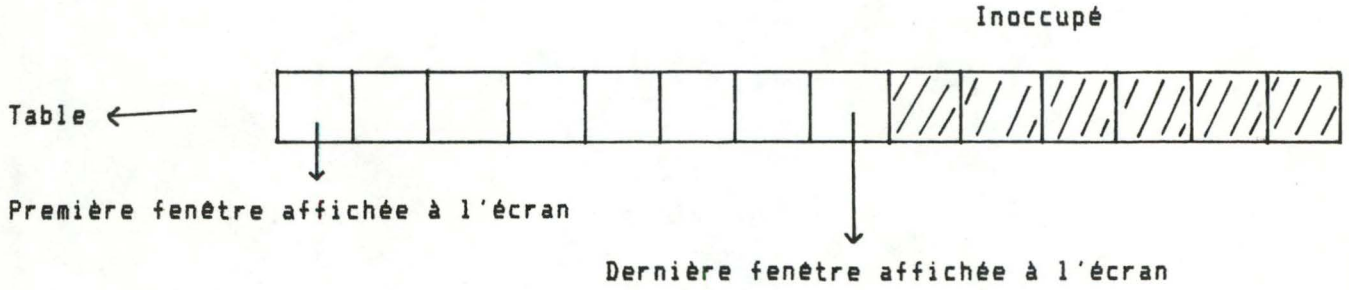

Figure C.1 Table d'ordonnancement des fenêtres.

Cette table est gérée selon une politique FIFO.

Supprimer un élément de la table d'ordonnancement

nom de la primitive: "supprim_fen_ordre"

Ce module permet de supprimer un numéro de fenêtre de la table d'ordonnancement. Ce numéro correspond à la fenêtre actuellement active.

arguments: - numéro de la fenêtre que l'on désire supprimer de la table

préconditions: - le numéro de la fenêtre est un nombre entier

résultats: - suppression du numéro de la fenêtre de la table d'ordonnancement
ou - "erreur"

postconditions: - "erreur" inchangé et suppression du numéro de fenêtre
ou - "erreur" = 6 : "numéro de fenêtre inconnu dans la table d'ordonnancement"

Ajouter un élément dans la table d'ordonnancement
---

nom de la primitive: "ajout_fen_ordre"

Ce module permet de mémoriser un numéro de fenêtre dans la table d'ordonnancement. La fenêtre identifiée par ce numéro doit être activée.

arguments: - numéro de la fenêtre à ajouter dans la table

préconditions: - le numéro de la fenêtre est un nombre entier tel que 1 <= N° fenêtre <= nb_max_fenetres

résultats: - mémorisation du numéro de la fenêtre dans la table d'ordonnancement
ou - "erreur"

postconditions: - [mémorisation du numéro de la fenêtre dans la première entrée de la table dont le contenu est nul, en scrutant ces entrées de la gauche vers la droite
et
"erreur" inchangé]
ou - "erreur" = 6 : "nombre maximum de fenêtres déjà affiché à l'écran"
= 9 : "numéro de fenêtre existant déjà dans la table d'ordonnancement"

Table des fenêtres présentes en mémoire centrale

La table des fenêtres présentes en mémoire centrale contient autant d'éléments que de fenêtres que l'on peut mémoriser en mémoire centrale en même temps. Pour chaque fenêtre mémorisée, cette table donne:

- un pointeur vers une liste chainée qui contient la description de tous les objets définis dans cette fenêtre ainsi que de la fenêtre elle-même.
- le nom du fichier sur disque contenant la description de tous les objets définis dans la fenêtre et de la fenêtre elle-même.

Rechercher le numéro de la dernière fenêtre insérée dans la table d'ordonnancement
---

nom de la primitive: "rech_der_table_ordre"

Cette primitive permet de lire dans la table d'ordonnancement le numéro de la fenêtre active affichée à l'écran;

arguments: - -----

préconditions: - -----

résultats: - le numéro de la fenêtre

postconditions: - [le numéro de la fenêtre correspond à la dernière fenêtre insérée dans la table d'ordonnancement et est un nombre entier tel que 1 <= numéro <= nb_max_fenetres
et
"erreur" est inchangé]
ou - "erreur" = 16 : "aucune fenêtre affichée à l'écran"

Composant "Sorties auxil-imprimantes"
++++++++++++++++++++++++++++++++++++

Envoyer des signaux de sortie vers les appareils électriques extérieurs
---

nom de la primitive: "envoyer_code_appareils"

Cette primitive permet d'envoyer un signal binaire vers un appareil électrique, via un port de sortie.

arguments: - numéro du bit sur lequel le signal doit être envoyé
- adresse du port de sortie
- délai d'attente pendant l'envoi du signal
- état du signal à transmettre

préconditions: - le numéro du bit est un nombre entier tel que 0 <= numéro <= 7
et - l'adresse du port référence un port existant et est représentée par un nombre entier >= 0
et - le délai est un nombre entier >= 0
et - l'état du signal vaut "1" ou "0" (binaire)

résultats: - envoi du signal sur le bit du port référencé

postconditions: - -----

Composant "Manip-fichiers"
++++++++++++++++++++++++++

Accès séquentiel en lecture dans un fichier
---

nom de la primitive: "acces_seq"

Cette primitive permet de lire un ou plusieurs éléments d'un fichier. Ces éléments sont mémorisés en mémoire centrale dans une liste chainée et sont accessibles séquentiellement à l'aide d'un pointeur.

arguments: - nom du fichier
- numéro de l'enregistrement à partir duquel on commence la lecture
- numéro de l'enregistrement final jusqu'où on effectue la lecture

```
préconditions: - les numéros d'enregistrement sont des nombres entiers
           et - [ N°  enregistrement initial <= N° enregistrement final ET
                  N° enregistrement initial >= 0 ]
                OU
                [ N°  enregistrement  initial >= 0   ET  N° enregistrement
                  final < 0 ]
           et - le fichier est fermé

résultats: - [mémorisation des enregistrements en mémoire centrale
             et
             un  pointeur  vers la liste des enregistrements  mémorisés  en
             mémoire centrale
             et
             le numéro de ce pointeur]
       ou - "erreur"


postconditions: - (mémorisation en mémoire centrale des enregistrements de:
                  [ N°  enregistrement  initial à N°  enregistrement  final
                    inclus si N° enregistrement final >= 0 ]
                  OU
                  [ N°  enregistrement  initial  à  fin du  fichier  si  N°
                    enregistrement final < 0 ]
                  et
                  le numéro du type de pointeur est un nom déclaré dans  le
                  programme
                  et
                  "erreur" inchangé)
            ou - "erreur" = 1 : "fichier non existant"
                          = 2 : "nom  de  fichier n'appartenant pas  à  la
                                 table des fichiers"
                          = 3 : "numéro  d'enregistrement inexistant  dans
                                 le fichier"
                          = 4 : "nom  de fichier ne correspondant pas à un
                                 nom  d'enregistrement  défini  dans  le
                                 programme / nom de fichier invalide"
                          = 7 : "les  données  ne sont pas  mémorisées  en
                                 mémoire centrale"
```

Accès séquentiel en écriture dans un fichier de texte
---

nom de la primitive: "ecrire_fich_seq_text"

Cette primitive permet d'écrire une ou plusieurs chaînes de caractères dans un fichier de texte. Ces chaînes doivent être préalablement mémorisées dans une variable du programme. Chaque ligne du fichier de texte est assimilée à un enregistrement et on mémorise une et une seule chaîne de caractères par ligne du fichier.

```
arguments: - le nom du fichier
           - le fichier
           - le numéro de l'enregistrement initial à écrire dans le fichier
           - le numéro de l'enregistrement final à écrire dans le fichier
           - la ou les chaînes de caractères à écrire dans le fichier
```

```
préconditions: - le numéro de l'enregistrement initial est un nombre entier
                 supérieur ou égal à 0
           et - le  numéro de l'enregistrement final est un nombre  entier
                >= au numéro de l'enregistrement initial
           et - les chaînes de caractères doivent être mémorisées dans  le
                tableau "tampon" du programme

résultats: - écriture des chaînes de caractères dans le fichier
       ou - "erreur"

postconditions: - les chaînes de caractères sont écrites dans le fichier et
                  "erreur" est inchangé
                  et
                  la différence entre le numéro d'enregistrement initial et
                  le numéro d'enregistrement final représente le nombre  de
                  chaînes  de  caractères  mémorisées  dans  le  fichier;
                  l'ancien contenu des enregistrements "numéro  initial"  à
                  "numéro final" est perdu
            ou - "erreur" = 1 : "fichier inexistant"
                          = 2 : "nom  de  fichier n'appartenant pas  à  la
                                 table des fichiers"
                          = 3 : "numéro      d'enregistrement      initial
                                 inexistant  dans le fichier ou  supérieur
                                 au numéro d'enregistrement final"
                          = 4 : "numéro  de  pointeur inexistant  dans  la
                                 table des fichiers"
                          = 7 : "données   non   mémorisées   en   mémoire
                                 centrale"
```

Composant "fichiers"
++++++++++++++++++++

Fichier comprenant les formats des codes binaires à envoyer au "mains-libres"

Le fichier comprenant les formats des codes binaires à envoyer au "mains-libres" est créé ou modifié à l'aide du fichier exécutable "créer_fich.com". Le nom de ce fichier de codes binaires doit commencer par "telep". Ce fichier contient les informations suivantes:

- le temps entre l'envoi de deux chiffres d'un même numéro de téléphone composé. Ce nombre est un entier >= 0.
- le temps entre deux impulsions relatives à un même chiffre, exprimé par un entier >= 0.
- la durée d'une impulsion qui est un nombre entier >= 0.
- l'adresse du port de sortie sur lequel est raccordé le "mains-libres". Cette adresse est représentée par un nombre entier >= 0.
- le numéro du bit du port sur lequel les signaux doivent être émis. Ce numéro est un nombre entier tel que 0 <= numéro <= 7.

Des détails supplémentaires au sujet de ces informations peuvent être consultés dans le manuel "Le Système AIDAMI Mons-Namur" [1].

Ensemble de fichiers décrivant le contenu des fenêtres à afficher à l'écran au cours de l'exécution du programme

---

Chaque fenêtre nécessaire dans le projet "AIDAMI" est entièrement décrite dans un fichier, ainsi que tous les objets qui doivent y être représentés. Les fenêtres sont numérotées et le nom des fichiers qui les décrivent doit commencer par "FENET". Ces fichiers peuvent être créés ou modifiés à l'aide du programme "créer_fich.com".

Nous passons à présent en revue tous les types d'objets qui peuvent être représentés à l'écran, et indiquons les données qui les caractérisent.

* Objet "fenêtre" : - type de l'objet ("0" par définition)
- chaîne de caratères à afficher dans l'entête de la fenêtre (optionnel, 80 caractères maximum).
- coordonnée xh du coin supérieur gauche de la fenêtre qui est un nombre entier tel que 0 <= xh <= 79
- coordonnée yh du coin supérieur gauche de la fenêtre qui est un nombre entier tel que 0 <= yh <= 199
- coordonnée xl du coin inférieur droit de la fenêtre qui est un nombre entier tel que 0 <= xl <= 79
- coordonnée yl du coin inférieur droit de la fenêtre qui est un nombre entier tel que 0 <= yl <= 199

  Il faut également respecter les conditions suivantes: xh <= xl et yh <= yl

- type de la fenêtre: "0" : fenêtre sans bord et sans entête
  "1" : fenêtre avec bord et sans entête
  "2" : fenêtre avec bord et avec entête
- couleur de fond de l'écran comprise entre 0 et nb_couleurs
- couleur des caractères affichés à l'écran comprise entre 0 et nb_couleurs
- numéro de l'action si on sélectionne cette fenêtre à l'aide du curseur de l'écran. Ce numéro est un entier >= 0

Remarques: - cet objet doit être le dernier élément mémorisé dans le fichier de fenêtres.
********* - le numéro de l'action doit être défini dans la table des actions du programme "AIDAMI", sinon aucune action ne se produit.

* Objet "rectangle": - type de l'objet ("1" par définition)
- coordonnée xh du coin supérieur gauche du bouton qui est un nombre entier tel que 0 <= xh <= X_max_world
- coordonnée yh du coin supérieur gauche du bouton qui est un nombre entier tel que 0 <= yh <= Y_max_world
- coordonnée xl du coin inférieur droit du bouton qui est un nombre entier tel que 0 <= xl <= X_max_world

- coordonnée yl du coin inférieur droit du bouton qui est un nombre entier tel que 0 <= yl <= Y_max_world

  Il faut également respecter les conditions suivantes:
  xh <= xl et yh <= yl

- numéro du groupe du bouton qui est un nombre entier >= 0 et tel que:

  * s'il s'agit d'un bouton permettant d'exécuter un logiciel "standard", ce numéro vaut:
    4 : pour sélectionner un logiciel "standard" non défini dans la table des logiciels "standards" du programme "AIDAMI"
    N (avec 1 <= N) : pour sélectionner un logiciel "standard" défini dans la table des logiciels "standards" du programme "AIDAMI"

  * s'il s'agit d'un bouton caractérisant un chiffre de numéro de téléphone, ce numéro vaut:
    0 à 9 : pour les chiffres de téléphone 0 à 9 respectivement
    10 : pour le "back space" permettant de corriger un numéro de téléphone erronné

- couleur de fond de l'écran comprise entre 0 et nb_couleurs
- couleur des caractères affichés à l'écran comprise entre 0 et nb_couleurs
- numéro de l'action si on sélectionne ce bouton à l'aide du curseur de l'écran. Ce numéro est un entier >= 0

* Objet "chaîne de caractères":
  - type de l'objet ("2" par définition)
  - valeur de la chaîne de caractères (80 caractères maximum)
  - coordonnée xh de la chaîne qui localise l'emplacement du premier caractère de la chaîne et qui est un nombre entier tel que 0 <= xh <= 79
  - coordonnée yh de la chaîne qui localise l'emplacement du premier caractère de la chaîne et qui est un nombre entier tel que 0 <= yh <= 24
  - couleur de fond de l'écran comprise entre 0 et nb_couleurs
  - couleur des caractères affichés à l'écran comprise entre 0 et nb_couleurs

* Objet "emplacement de chaîne de caractères": cet emplacement est réservé dans une fenêtre dans le but d'y afficher une chaîne de caractères pendant l'exécution du programme "AIDAMI" et à la demande de l'utilsateur
  - type de l'objet ("3" par définition)
  - coordonnée xh du premier caractère affichable dans l'emplacement, et qui est un nombre entier tel que 0 <= xh <= 79

- coordonnée yh du premier caractère affichable dans l'emplacement, et qui est un nombre entier tel que 0 <= yh <= 24
- couleur de fond de l' écran comprise entre 0 et nb_couleurs
- couleur des caractères affichés à l'écran comprise entre 0 et nb_couleurs
- numéro de l'emplacement à réserver dans la fenêtre. Ce numéro est un nombre entier >= 1. (Il peut y avoir plusieurs emplacements dans une même fenêtre).

* Objet "touche du clavier":

- type de l'objet ("5" par définition)
- coordonnée xh du coin supérieur gauche du bouton qui est un nombre entier tel que 0 <= xh <= X_max_world
- coordonnée yh du coin supérieur gauche du bouton qui est un nombre entier tel que 0 <= yh <= Y_maxworld
- coordonnée xl du coin inférieur droit du bouton qui est un nombre entier tel que 0 <= xl <= X_max_world
- coordonnée yl du coin inférieur droit du bouton qui est un nombre entier tel que 0 <= yl <= Y_max_world

  Il faut également respecter les conditions suivantes:
  xh <= xl et yh <= yl

- couleur de fond de l'écran comprise entre 0 et nb_couleurs
- couleur des caractères affichés à l'écran comprise entre 0 et nb_couleurs
- le code ASCII associé à la touche, augmenté de 1000. Ce code est représenté par un nombre entier >= 0
- le code de recherche associé à la touche, augmenté de 1000. Ce code est représenté par un nombre entier >= 0

Fichier dans lequel sont mémorisées les erreurs pouvant survenir pendant l'exécution du programme "AIDAMI"

Le fichier dans lequel sont mémorisées les erreurs pouvant survenir pendant l'exécution du programme "AIDAMI" est un fichier de texte dans lequel les messages d'erreur sont mémorisés à l'aide d'un éditeur de textes. Le message d'erreur numéro "i" est mémorisé à la ligne numéro "i" et n'excède pas 80 caractères.

ANNEXE D : Présentation du schéma hiérarchique du composant "système d'exploitation" de la hiérarchie de l'application

Nous présentons dans cette annexe la découpe hiérarchique du composant "système d'exploitation" identifié dans la découpe de l'application au chapitre 2.

La figure D.1 représente le schéma hiérarchique issu de la décomposition du système d'exploitation "idéal" relatif à notre application.

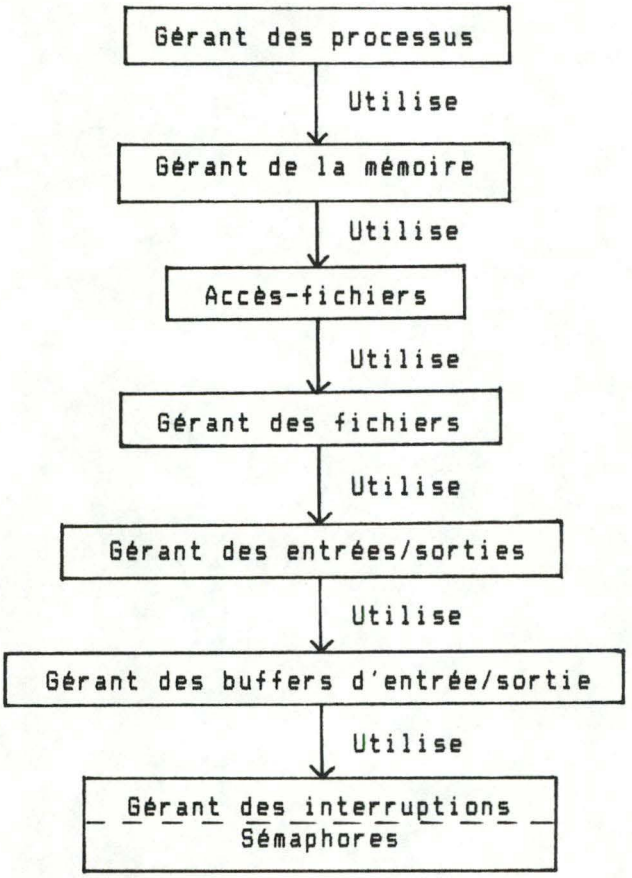

Figure D.1 Découpe hiérarchique du niveau "système d'exploitation"

Ce schéma est composé des modules suivants:

* gérant des processus.
* gérant de la mémoire.
* accès-fichiers.
* gérant des fichiers.
* gérant des entrées/sorties.
* module spécialisé dans la gestion des buffers d'entrée/sortie.
* gérant des interruptions et sémaphores.

Les modules ainsi identifiés sont reliés par une relation "utilise"; il s'agit d'une hiérarchie de type "utilise".

Analyse des modules du graphe hiérarchique

* Le gérant des processus est implémenté au niveau supérieur de l'architecture. En effet, celui-ci assure les créations, suppressions des processus, et alloue un des processus prêts au processeur. La création d'un processus s'accompagne notamment de la mémorisation d'informations dans un descripteur de processus: ceci nécessite une réservation de place mémoire effectuée par le gérant de la mémoire.

* Le gérant de la mémoire permet de créer de l'espace en mémoire centrale lors de l'allocation de nouvelles ressources. Lorsqu'une demande d'allocation en mémoire centrale est effectuée et que cette dernière est pleine, il est souvent utile de libérer des blocs de données de la mémoire soit en les effaçant, soit en les sauvant sur disque sous forme de fichiers. Une utilisation du module "accès-fichiers" s'avère donc nécessaire.

* Les accès fichiers : ce module permet de lire et d'écrire des informations dans des fichiers. Les fonctions du gérant de la mémoire centrale ainsi que celles des niveaux supérieurs au système d'exploitation utilisent ce module.

* Le gérant des fichiers assure entr'autres la création, la suppression, et la mise à jour des fichiers. Ces primitives sont donc utilisées par le module "accès-fichiers" et peuvent être exécutées correctement pour autant que le gérant des entrées/sorties leur fournisse les services dont elles ont besoin.

* Le gérant des entrées/sorties assure les échanges de données entre la mémoire centrale et les périphériques. Ce module utilise les services offerts par un module spécialisé dans la gestion des buffers d'entrée/sortie. Ceci évite de devoir utiliser les services du gérant de la mémoire car dans ce cas, nous observerions un croisement de relations dans la hiérarchie. En effet, considérons le cas suivant: lors d'une demande de lecture sur disque, le gérant des entrées/sorties doit disposer d'un espace libre (buffer) en mémoire centrale pour y mémoriser les données lues sur le disque. La réservation de ces buffers est réalisée par le module spécialisé.

* Le gérant des buffers d'entrée/sortie réserve des buffers d'entrée/sortie permettant d'effectuer des échanges de données avec les périphériques (lecteurs de disques, imprimante,...).

* Le gérant des interruptions permet d'analyser le type d'événement qui génère une interruption. Il assure également un sauvetage de tout l'environnement du programme interrompu.

Les primitives identifiées dans chaque module du système d'exploitation sont présentées dans l'annexe "E".

ANNEXE E : Explicitation des primitives relatives à chaque composant du niveau "système d'exploitation"

L'analyse des fonctionnalités d'un système d'exploitation nous a permis de définir un ensemble de primitives répondant aux besoins de notre application. Nous présentons ci-dessous les primitives relatives au composant "système d'exploitation" de la hiérarchie de l'application.

* Gérant des processus :
  - Créer un processus.
  - Détruire un processus.
  - Allouer un processus au processeur.
  - Désallouer un processus du processeur.
  - Choisir un processus à allouer au processeur.

* Gérant de la mémoire :
  - Allouer un espace mémoire.
  - Libérer un espace mémoire.
  - Echanger des données entre deux blocs de mémoire.

* Accès-fichiers :
  - Ecrire dans un fichier.
  - Lire dans un fichier.
  - Vider le contenu d'un fichier.

* Gérant des fichiers :
  - Créer un fichier.
  - Effacer un fichier.
  - Renommer un fichier.
  - Vérifier l'existence d'un fichier.
  - Ouvrir un fichier.
  - Fermer un fichier.

* Gérant des entrées/sorties :
  - Toutes les primitives d'entrée/sortie implémentées dans les librairies du langage de programmation de haut niveau.
  - Toutes les primitives d'entrée/sortie offertes par le système d'exploitation.
  - Ecriture d'un caractère dans le buffer du clavier.

  Ces primitives assurent des accès à :
    - l'écran.
    - le clavier.
    - les lecteurs de disques.
    - les ports de sortie séries et parallèles.

* Gérant des buffers d'entrée/sortie:
  - Allouer un buffer.
  - Désallouer un buffer.

* Gérant des interruptions :
  - Sauver les registres du processeur.
  - Sauver les données propres au processus interrompu.
  - Autoriser les interruptions externes.
  - Interdire les interruptions externes.
  - Positionner le masque des interruptions et les registres associés.
  - Analyser le type d'événement qui interrompt.

* Primitives d'exclusion mutuelle:
  - Attendre (un événement).
  - Signaler (un événement).

La structure de données utilisée est:

- Table des contenus de la mémoire centrale.

Le lecteur peut se référer à l'annexe "F" pour obtenir une description détaillée des primitives et structures implémentées dans notre programme.

ANNEXE F: Spécifications externes des primitives identifiées dans le composant "système d'exploitation" et implémentées dans notre programme

Nous présentons dans cette annexe les spécifications externes des primitives identifiées dans le composant "système d'exploitation" de la découpe hiérarchique de l'application "AIDAMI" et implémentées dans notre programme.

Composant "Accès-fichiers"

Lire dans un fichier

nom de la primitive: "lire_fich"

Cette primitive permet de lire un enregistrement donné dans un fichier.

arguments: - nom du fichier
- numéro de l'enregistrement à lire dans le fichier
- numéro du pointeur de la structure du fichier

préconditions: - le fichier est fermé
et - le numéro de l'enregistrement à lire est un nombre entier >= 0
et - le nom du type d'enregistrement composant le fichier identifie une structure d'enregistrement déclarée dans le programme et figurant effectivement dans ce fichier
et - le numéro du pointeur est un nombre entier

résultats: - lecture de l'enregistrement dans le fichier
ou - "erreur"

postconditions: - [l'enregistrement est lu dans le fichier si son numéro référence un numéro d'enregistrement existant dans le fichier
et
"erreur" inchangé]
ou - "erreur" = 1 : "fichier non existant"
= 2 : "nom de type d'enregistrement invalide"
= 3 : "numéro d'enregistrement inexistant dans ce fichier"
= 4 : "numéro de pointeur n'appartenant pas à la table des fichiers"

Vider le contenu d'un fichier
---

nom de la primitive: "vider_contenu_fich"

Ce module permet d'effacer les informations contenues dans un fichier.

arguments: - le nom du fichier à vider
- le fichier

préconditions: - -----

résultats: - les données du fichier sont effacées
ou - "erreur"

postconditions: - le fichier est vidé et "erreur" est inchangé
ou - "erreur" = 1 : "fichier inexistant"

Composant "Gérant des fichiers"
+++

Vérifier l'existence d'un fichier
---

nom de la primitive: "exist_fich"

Cette primitive permet de vérifier si un fichier de nom donné est présent sur l'unité de disques courante.

arguments: - nom du fichier

préconditions: - -----

résultats: - booléen : "exist"

postcondition: - "exist" = true si un fichier localisé sur l'unité de disque courante est identifié par le nom du fichier.
= false sinon.

Ouvrir un fichier
---

nom de la primitive: "ouvrir_fich"

Ce module permet d'ouvrir un fichier de nom donné.

arguments: - nom du fichier
- numéro du pointeur de la structure du fichier
- type d'ouverture du fichier

```
préconditions: - si le fichier existe, il est fermé
           et - le type d'ouverture vaut "1" si on effectue une  ouverture
                pour faire une lecture
                ou
                le  type d'ouverture vaut "0" si on effectue une ouverture
                pour faire une écriture
           et - le numéro de pointeur est un nombre entier

résultats: - le fichier est ouvert
       ou - "erreur"

postconditions: - [le  fichier est ouvert si un fichier du même  nom existe
                  sur l'unité de disques courante
                  et
                  "erreur" inchangé]
            ou - "erreur" = 1 : "fichier non existant"
                          = 4 : "numéro de pointeur n'appartenant pas à la
                                 table des fichiers"
```

```
Fermer un fichier
-----------------
nom de la primitive: "fermer_fich"

     Cette primitive permet de fermer un fichier.

arguments: - nom du fichier
           - le fichier
           - numéro du pointeur de la structure du fichier

préconditions: - si le fichier existe, il est ouvert
           et - le numéro de pointeur est un nombre entier

résultats: - fermeture du fichier
       ou - "erreur"

postconditions: - [le fichier est fermé si un fichier du  même  nom  existe
                  sur l'unité de disques courante
                  et
                  "erreur" inchangé]
            ou - "erreur" = 1 : "fichier non existant"
                          = 4 : "numéro de pointeur n'appartenant pas à la
                                 table des fichiers"
```

```
Composant "Gérant des entrées/sorties"
++++++++++++++++++++++++++++++++++++++

Ecriture d'un caractère dans le buffer du clavier
-------------------------------------------------
nom de la primitive: "ecrire_carac_clavier"

     Ce module permet de mémoriser les codes caractérisant  une  touche  du
clavier dans le buffer du clavier.
```

arguments: - le code de recherche correspondant au caractère
           - le code ASCII correspondant au caractère

préconditions: - le code de recherche est un nombre entier
               - le code ASCII est un nombre entier

résultats: - le caractère est mémorisé dans le buffer du clavier
       ou - "erreur"

postconditions: - [les pointeurs gérant le buffer sont mis à jour
                  et
                  "erreur" inchangé]
             ou - "erreur" = 14 : "caractère inexistant"
                           = 15 : "buffer déjà rempli"

Composant "Gérant des interruptions"
++++++++++++++++++++++++++++++++++

Autoriser les interruptions externes
----
nom de la primitive: "autoriser_interrupt"

Cette primitive permet d'autoriser la prise en considération des interruptions externes de type INTR.

arguments: - indicateur des interruptions

préconditions: - -----

résultats: - positionnement à "1" de l'indicateur

postconditions: - -----

Interdire les interruptions externes
----
nom de la primitive: "interdire_interrupt"

Cette primitive permet d'interdire la prise en considération des interruptions externes de type INTR.

arguments: - indicateur des interruptions

préconditions: - -----

résultats: - positionnement à "0" de l'indicateur

postconditions: - -----

Attendre(s)
---

nom de la primitive: "attendre"

Cette primitive permet de positionner un verrou avant d'entrer dans une section critique.

arguments: - le verrou

préconditions: - le verrou est un nombre entier caractérisé par les valeurs "0" ou "1"

résultats: - positionnement du verrou

postconditions: - le verrou est positionné à "0" si sa valeur, lors de l'entrée dans la primitive, vaut "1"
- la valeur du verrou est inchangée sinon

Signaler(s)
---

nom de la primitive: "signaler"

Cette primitive permet de dépositionner un verrou lors de la sortie d'une section critique.

arguments: - le verrou

préconditions: - le verrou est un nombre entier caractérisé par les valeurs "0" ou "1"

résultats: - dépositionnement du verrou

postconditions: - le verrou prend la valeur "1"

ANNEXE G: Présentation du contenu des variables d'état du clavier

Nous présentons dans cette annexe le contenu des variables d'état du clavier, appelées KB_FLAG et KB_FLAG_1. Rappelons que les bits de ces deux variables représentent l'état des touches de changement de mode et de verrouillage du clavier. La figure G.1 donne la signification des bits de ces deux octets. Chaque bit est identifié par un nom. La figure indique pour chaque bit l'état de la touche correspondante du clavier lorsque ce bit est positionné à "1".

KB_FLAG

Numéro du bit	Nom associé	Sémantique si le bit est à "1"
7	INS_STATE	"Insert" actif
6	CAPS_STATE	"Caps Lock" actif
5	NUM_STATE	"Num Lock" actif
4	SCROLL_STATE	"Scroll Lock" actif
3	ALT_SHIFT	"Alternate" enfoncé
2	CTL_SHIFT	"Ctrl" enfoncé
1	LEFT_SHIFT	"Chgt de mode" enfoncé
0	RIGHT_SHIFT	"Chgt de mode" enfoncé
KB_FLAG_1		
7	INS_SHIFT	"Insert" enfoncé
6	CAPS_SHIFT	"Caps Lock" enfoncé
5	NUM_SHIFT	"Num Lock" enfoncé
4	SCROLL_SHIFT	"Scroll Lock" enfoncé
3	HOLD_STATE	"Ctrl Num Lock" actif
2	Inutilisé	
1	Inutilisé	
0	Inutilisé	

Figure G.1 Interprétation des octets KB_FLAG et KB_FLAG_1.

ANNEXE H: Les codes associés aux touches du clavier

A chaque touche du clavier est associé un code de recherche compris entre 1 et 83. Ce code est transmis vers l'ordinateur lorsqu'une touche est enfoncée ou relâchée. Pour indiquer l'appui d'une touche vis-à-vis de sa relâche, le code de recherche, augmenté de la valeur 128h, est envoyé vers la machine dès que l'utilisateur relâche la touche. La figure H.1 présente les codes de recherche associés aux touches du clavier lorsqu'elles sont enfoncées. Les nombres indiqués à côté des touches représentent leur code de recherche respectif.

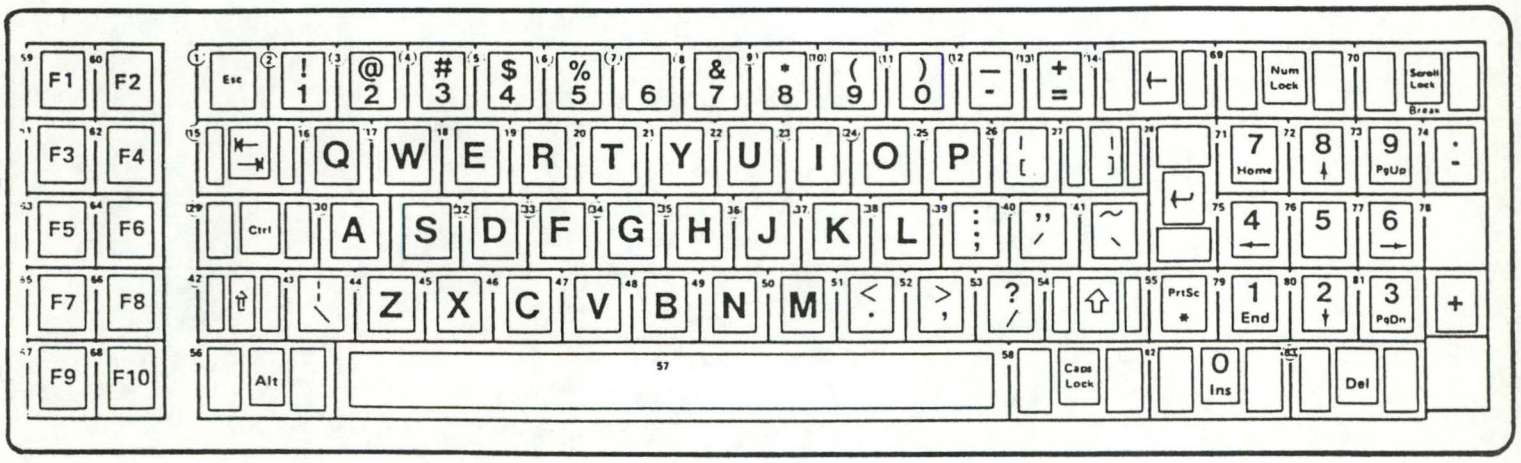

Figure H.1 Codes de recherche associés aux touches du clavier.

Mais d'autres possibilités sont offertes par le clavier. En effet, les codes ci-dessus ne peuvent identifier des séquences d'appuis telles que "Ctrl K", "Alt G", les touches de fonction, les touches de type "home", "PgUp". Il est dans ce cas nécessaire de faire appel à un autre type de codage qui utilise les codes ASCII étendus. Ce codage est effectué sur deux octets: le premier est toujours nul tandis que le second contient généralement la valeur du code de recherche de la séquence entrée au clavier. La figure H.2 donne les valeurs du code ASCII étendu pour les séquences d'appuis possibles au clavier. Les valeurs indiquées sur la figure sont celles qu'il faut attribuer au second octet du code.

3	NUL Character
F	←
10-19	ALT Q, W, E, R, T, Y, U, I, O, P
1E-26	ALT A, S, D, F, G, H, J, K, L
2C-32	ALT Z, X, C, V, B, N, M
3B-44	F1-F10 Function Keys (Base Case)
47	Home
48	↑
49	Page Up and Home Cursor
4B	←
4D	→
4F	End
50	↓
51	Page Down and Home Cursor
52	INS
53	DEL
54-5D	F11-F20 (Upper Case F1-F10)
5E-67	F21-F30 (CTRL F1-F10)
68-71	F31-F40 (ALT F1-F10)
72	CTRL PRTSC (Start/Stop Echo to Printer)
73	CTRL ← (Reverse Word)
74	CTRL → (Advance Word)
75	CTRL END (Erase to End of Line)
76	CTRL PG DN (Erase to End of Screen)
77	CTRL HOME (Clear Screen and Home)
78-83	ALT 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -, = (Top row)
84	CTRL PG UP (Top 25 Lines of Text and Home Cursor)

Figure H.2 Valeurs du code ASCII étendu pour les séquences d'appui possibles au clavier.

Enfin, la figure H.3 présente les 256 caractères ASCII. Cette table établit la correspondance entre la représentation du caractère et la valeur numérique qui lui est associée.

DECIMAL VALUE	HEXA DECIMAL VALUE	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	BLANK (NULL)	►	BLANK (SPACE)	0	@	P	‘	p	Ç	É	á	▒	└	╨	∝	≡
1	1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	ß	±
2	2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▒	┬	╘	Γ	≥
3	3	♥	‼	#	3	C	S	c	s	â	ô	ú	│	├	╒	π	≤
4	4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╓	Σ	⌠
5	5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╫	σ	⌡
6	6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╪	μ	÷
7	7	•	↨	'	7	G	W	g	w	ç	ù	º	╖	╟	┘	τ	≈
8	8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	┌	Φ	°
9	9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	╫	Θ	•
10	A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	╪	Ω	·
11	B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	▄	δ	√
12	C	♀	∟	,	<	L	\	l	¦	î	£	¼	╝	╠	▌	∞	n
13	D	♪	↔	–	=	M	]	m	}	ì	¥	¡	╜	═	▐	φ	2
14	E	♫	▲	.	>	N	^	n	~	Ä	₧	«	╛	╬	▀	∈	■
15	F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	α	∩	BLANK (FF)

Figure H.3 Table des caractères ASCII.

ANNEXE I: Mode d'emploi du programme

Nous exposons dans cette annexe un mode d'emploi très succinct de l'application "AIDAMI" réalisée dans cette étude.

La première opération à effectuer pour exécuter le programme "AIDAMI" consiste à mettre l'ordinateur sous tension. A partir de cet instant, le programme "AIDAMI" est chargé automatiquement en mémoire centrale. (Celui-ci peut être chargé "manuellement" en introduisant la commande "Aidami" dans l'ordinateur).

L'ordinateur affiche ensuite un menu à l'écran ainsi qu'un curseur. Les touches fléchées du clavier réel de la machine peuvent être utilisées afin de mouvoir ce curseur. Pour sélectionner un menu, l'utilisateur doit le pointer à l'aide du curseur et enfoncer la touche "F9" du clavier. Cette touche simule l'appui sur le bouton de la souris ou de l'interface utilisateur. La relâche du bouton de la souris est simulée en enfonçant la touche "F10" du clavier.

Lorsqu'un menu est sélectionné, d'autres menus ou fenêtres apparaissent à l'écran. Nous ne décrivons pas ici en détail les opérations que l'utilisateur doit effectuer pour assurer le bon déroulement du programme, car le contenu de ces menus et fenêtres est suffisamment explicite. Notons simplement qu'ils contiennent des boutons et que l'utilisateur doit les sélectionner s'il désire effectuer une opération particulière.

Enfin, si l'ordinateur exécute un logiciel "standard", l'utilisateur peut interrompre ce logiciel et exécuter le programme "AIDAMI" en appuyant trois fois consécutivement sur la touche "F9" du clavier. La sélection du menu "sortie" permet de rendre le contrôle au logiciel "standard".

ANNEXE J: Texte du programme

Le texte du programme réalisé dans cette étude est exclusivement disponible auprès de Monsieur Jean Ramaekers, Directeur de l'Institut d'Informatique, pour des raisons de confidence.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[1] Andre Ph., Bellofatto B., Duterne J., Lassoie J-M.
Le système AIDAMI Mons-Namur.
Contrat Cadre Spécial Temporaire, CST N° 30319, 1986/1987.

[2] Apple Computer INC.
Inside Machintosh.
Volume 3, N° 5409.
Addison Wesley, Reading, Massachussetts, Août 1986.

[3] Ben-Ari M.
Principles of Concurrent Programming.
Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[4] Brady Company J-R., Scanlon L-J.
IBM-PC and XT Assembly Langage.
A Guide For Programmers.
Prentice Hall, 1983.

[5] Deitel HM., Lorin H.
Operating systems.
Addison Wesley, Reading, Massachussetts. November 1981.

[6] IBM
Disk Operating System.
Technical Reference.
IBM Personnal Computer Software, Février 1985.

[7] MicroSoft Corporation
MicroSoft Mouse Menu.
MicroSoft Mouse for IBM Personnal Computer.
Installation and operation manual.
1983.

[8] Olivetti Ing.C.
8086 Assembler.
Reference Manual.
1984.

[9] Politis R., Vanryb B.
Le système d'exploitation MS-DOS, versions 1 et 2.
Editions Eyrolles, Juillet 1984.

[10] Ramaekers J.
Systèmes d'exploitation, cours de première licence informatique.
FNDP, Namur, Octobre 1985.

[11] Ramaekers J.
Architecture des systèmes d'exploitation, cours de deuxième licence informatique.
FNDP, Namur, Mars 1984.

[12] Singh A., Triebel W.
The 8086 Micro-processor Architecture, Software And Interfacing Techniques.
Prentice Hall, INC Englewood, New Jersey, 1985.

[13] Thièle JB.
Au coeur de l'IBM-PC.
Logique et fonctionnement interne.
Editions Editest, Juin 1986.

[14] Van Lamsweerde A.
Méthodologie de développement de logiciels, cours de deuxième licence informatique.
FNDP, Namur, 1986/1987.

Facultés Universitaires Notre Dame de la Paix Namur

Institut d'informatique

CONCEPTION D'INTERFACES STANDARDS POUR UNE APPLICATION DESTINEE A DES HANDICAPES

ANNEXES

THIRIONET Philippe

Promoteur: Jean Ramaekers

Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique

Année académique 1986 - 1987

TABLE DES MATIERES

ANNEXES C ET F Modifications et compléments apportés aux annexes C et F pour améliorer les performances du système et mettre à la disposition du programmeur d'autres primitives

ANNEXE J Texte du programme

COMPLEMENTS DES ANNEXES C ET F

Modifications et compléments apportés aux annexes "C" et "F" pour améliorer les performances du système et mettre à la disposition du programmeur d'autres primitives.

ANNEXE "C"

Composant "entrees"

Lire le nombre de fois que l'on appuie sur le bouton de la souris

nom de la primitive: "lect_nbre_appuis_bout"

arguments: - -----

préconditions: - -----

résultats: - nombre de fois que l'utilisateur a appuyé sur le bouton de la souris entre deux appels à cette primitive-ci

postconditions: - le nombre d'appuis est un nombre entier >= 0

Lire le nombre de fois que l'utilisateur relâche le bouton de la souris

nom de la primitive: "lect_nbre_relaches_bout"

arguments: - -----

préconditions: - -----

résultats: - nombre de fois que l'utilisateur a relâché le bouton de la souris entre deux appels à cette primitive-ci

postconditions: - le nombre de relâches est un nombre entier >= 0

Composant "sorties"

Transformer le curseur de l'écran pour le rendre compatible avec le mode texte

nom de la primitive: "texte_curseur"

arguments: - -----

préconditions: - -----

résultats: - transformation du curseur pour le mode texte

postconditions: - -----

Composant "gestion_ecran"
++++++++++++++++++++++++

Sauver le contenu de l'écran dans un fichier sur disque
---

nom de la primitive: "sauver_ecran_disque"

arguments: - table d'ordonnancement

préconditions: - table d'ordonnancement valide

résultats: - mémorisation du contenu de l'écran dans un fichier sur le disque

postconditions: - le nom du fichier est "window" + le numéro de la derniére fenêtre affichée à l'écran

Composant "Manip-fichiers"
++++++++++++++++++++++++++

Accès séquentiel en lecture dans un fichier
---

nom de la primitive: "acces_seq"

Cette primitive permet de lire un ou plusieurs éléments d'un fichier. Ces éléments sont mémorisés en mémoire centrale dans une liste chainée et sont accessibles séquentiellement à l'aide d'un pointeur.

arguments: - nom du fichier
- numéro de l'enregistrement à partir duquel on commence la lecture
- numéro de l'enregistrement final jusqu'où on effectue la lecture

préconditions: - les numéros d'enregistrement sont des nombres entiers
et - [N° enregistrement initial <= N° enregistrement final ET N° enregistrement initial >= 0]
OU
[N° enregistrement initial >= 0 ET N° enregistrement final < 0]
et - le fichier est fermé

résultats: - [mémorisation des enregistrements en mémoire centrale
et
un pointeur vers la liste des enregistrements mémorisés en mémoire centrale
et
le numéro de ce pointeur]
ou - "erreur"

postconditions: - (mémorisation en mémoire centrale des enregistrements de:
[N° enregistrement initial à N° enregistrement final inclus si N° enregistrement final >= 0]
OU
[N° enregistrement initial à fin du fichier si N° enregistrement final < 0]
et
le numéro du type de pointeur est un nom déclaré dans le programme
et
"erreur" inchangé)
ou - "erreur" = 1 : "fichier non existant"
= 2 : "nom de fichier n'appartenant pas à la table des fichiers"
= 3 : "numéro d'enregistrement inexistant dans le fichier"
= 4 : "nom de fichier ne correspondant pas à un nom d'enregistrement défini dans le programme / nom de fichier invalide"
= 7 : "les données ne sont pas mémorisées en mémoire centrale"
= 23: "numéro d'enregistrement inconnu dans un fichier de texte"

ANNEXE "F"
==========

Composant "Accès-fichiers"
++++++++++++++++++++++++++

Lire dans un fichier
----
nom de la primitive: "lire_fich"

Cette primitive permet de lire un enregistrement donné dans un fichier.

arguments: - le fichier
- numéro de l'enregistrement à lire dans le fichier
- numéro du pointeur de la structure du fichier

préconditions: - le fichier est ouvert
et - le numéro de l'enregistrement à lire est un nombre entier >= 0
et - le numéro du pointeur est un nombre entier

résultats: - lecture de l'enregistrement dans le fichier
ou - "erreur"

postconditions: - [l'enregistrement est lu dans le fichier si son numéro référence un numéro d'enregistrement existant dans le fichier
et
"erreur" inchangé]
ou - "erreur" = 3 : "numéro d'enregistrement inexistant dans ce fichier"
= 4 : "numéro de pointeur n'appartenant pas à la table des fichiers"

Composant "Gérant des fichiers"
+++++++++++++++++++++++++++++++

Créer un fichier
----

arguments: - nom du fichier à créer

préconditions: - -----

résultats: - création du fichier
ou - "erreur"

postconditions: - le fichier est créé et "erreur" est inchangé
ou - "erreur" = 27 : "fichier existant déjà sur l'unité de disques courante"

Composant "Gérant des entrées/sorties"

Faire émettre un signal sonore par l'ordinateur

nom de la primitive: "son"

arguments: - la fréquence du signal (en Hz)
- le temps pendant lequel ce signal est émis (en msec)

préconditions: - la fréquence est un nombre entier >= 1
et - le temps est un nombre entier >= 1

résultats: - émission d'un signal sonore

postconditions: - -----

Désactiver les touches de fonction non permanentes du clavier

nom de la primitive: "reset_touch_special"

Cette primitive permet de simuler la relâche d'une touche de fonction du clavier réel telle que les touches "control", "alternate".

arguments: - drapeaux "Ctrl, shift_gauche, shift_droit, alternate, caps_lock, num_lock, scroll_lock et insert" indiquant l'état des touches

préconditions: - -----

résultats: - désactivation des touches de fonction non permanentes

postconditions: - "désactivation" signifie mettre les drapeaux à la valeur FALSE

Activer une touche de fonction du clavier

nom de la primitive: "flags"

Cette primitive permet de simuler l'appui sur une touche de fonction du clavier réel telle que les touches "control", "alternate",

arguments: - "code" = code de recherche de la touche de fonction

préconditions: - "code" = nombre entier tel que 0 <= "code" <= max_code

résultats: - activation de la touche identifiée par "code" si le code de recherche identifie bien une touche de fonction du clavier réel

postconditions: - -----

Composant "Gérant des interruptions"
++++++++++++++++++++++++++++++++++

Lire le contenu d'un vecteur d'interruption
---

nom de la primitive: "lire_vecteur_interrupt"

arguments: - "numero_vecteur" : le numéro du vecteur à lire

préconditions: - "numero_vecteur" identifie un vecteur d'interruption de l'ordinateur et est un nombre entier

résultats: - lecture du contenu du vecteur d'interruption
et - "segment_code" = adresse du segment de code contenant la routine de gestion des interruptions pointée par ce vecteur
et - "offset_code" = adresse relative de la routine de gestion des interruptions pointée par ce vecteur

postconditions: - "segment_code" est un nombre entier dont la valeur est comprise entre 0000h et FFFFh
et - "offset_code" est un nombre entier dont la valeur est comprise entre 0000h et FFFFh

Ecrire l'adresse d'une routine dans un vecteur d'interruption
---

nom de la primitive: "ecrire_vecteur_interrupt"

arguments: - "numero_vecteur" : numéro du vecteur dans lequel on désire mémoriser une nouvelle adresse
et - "segment_code" = adresse du segment de code contenant la routine de gestion des interruptions concernée
et - "offset_code" = adresse relative de la routine de gestion des interruptions concernée

préconditions: - "numero_vecteur" est un nombre entier et identifie un vecteur d'interruption de l'ordinateur
et - "segment_code" est un nombre entier tel que 0000h <= "segment_code" <= FFFFh
et - "offset_code" est un nombre entier tel que 0000h <= "offset_code" <= FFFFh

résultats: - écriture de l'adresse de la routine de gestion des interruptions dans le vecteur concerné

postconditions: - -----

ANNEXE "J": TEXTE DU PROGRAMME

```
const bs = ^h;                      (* back space *)
      CR = ^m;                      (* carriage return *)


(*****************************************************************************)
(* fenêtres *)

      nb_max_fenetres = 7;          (* nombre maximum de fenêtres affichables en
                                       même temps à l'écran *)
      n_max_fen_mc = 10;            (* nombre maximum de fenêtres mémorisables en
                                       même temps en mémoire centrale, sous forme
                                       de liste chainée *)

      x_max_world = 1000;           (* coordonnée maximale en "x" des rectangles
                                       affichables dans une fenêtre *)
      y_max_world = 1000;           (* coordonnée maximale en "y" des rectangles
                                       affichables dans une fenêtre *)


(*****************************************************************************)
(* couleurs *)

      nb_couleurs = 15;             (* nombre maximum de couleurs affichables par
                                       l'écran couleur *)
      couleur_string_affiche = 15; (* couleur dans laquelle les caractères sont
                                       affichés dans un emplacement réservé d'une
                                       fenêtre *)


(*****************************************************************************)
(* ports de sortie *)

      nb_max_ports_sortie = 5;      (* nombre maximum de ports de sortie
                                       utilisables en même temps *)


(*****************************************************************************)
(* constantes relatives au clavier. Ces constantes mémorisent les adresses
   des variables du clavier réel de la machine, contenues dans la zone de
   communication du système d'exploitation *)

      buffer_start = $001E;         (* adresse de début du buffer du clavier *)
      buffer_end = $003E;           (* adresse de fin du buffer du clavier *)

      buffer_tail_mem = $001C;      (* adresse du pointeur "buffer_tail" dans la
                                       zone de communication du clavier *)
      buffer_head_mem = $001A;      (* adresse du pointeur "buffer_head" dans la
                                       zone de communication du clavier *)

      seg_buffer_clavier = $0040;   (* adresse du segment contenant la zone de
                                       communication du clavier *)

      kb_flag_mem = $0017;          (* adresse de l'octet de configuration
                                       "KB_FLAG" dans la zone de communication
                                       du clavier *)
      kb_flag_1_mem = $0018;        (* adresse de l'octet de configuration
                                       "KB_FLAG_1" dans la zone de communication
                                       du clavier *)

      max_code = 84;                (* valeur maximale des codes de recherche
                                       émis par le clavier réel *)
```

CONSTANT.PAS

```
(*****************************************************************************)
(* piles utilisées dans le programme "aidami" *)

      pile_OS = $FFFF;              (* pile associée aux routines "entrée" et "sortie" *)
      pile_souris1 = $FBFF;         (* pile associée à la routine "entree_souris" *)
      pile_clavier = $F7FF;         (* pile associée à la routine de détournement
                                       du clavier *)
      pile_souris2 = $F3FF;         (* pile associée à la routine "sortie_souris" *)
      pile_fct25 = $F000;           (* pile associée à la routine "inter_21_25h"*)
      pile_aidami = $EA00;          (* pile associée au programme "AIDAMI" *)


(*****************************************************************************)
(* emplacement d'éléments en mémoire centrale *)

      taille_heap = $500;           (* place réservée pour mémoriser les variables
                                       dynamiques du programme "aidami" *)
      data_seg = $0186;             (* adresse relative de la cellule mémoire
                                       contenant la valeur du segment de données
                                       du programme "AIDAMI". La valeur du segment
                                       de données associé est 0000h. *)


(*****************************************************************************)
(* numéros de vecteurs d'interruption utilisés *)

      num_vecteur_souris = $33;     (* numéro du vecteur d'interruption appelé
                                       par le logiciel de la souris *)

      num_vecteur_clavier_detourne = $60; (* numéro du vecteur d'interruption
                                             appelé par la routine de détournement
                                             du clavier réel *)
      inter_1C_detournee = $70;     (* numéro du vecteur d'interruption appelé par
                                       la routine de gestion des interruptions
                                       numéro 1Ch *)


(*****************************************************************************)
(* bouton de la souris *)

      tps_inter_clic = 5;           (* temps maximum d'attente entre deux appuis
                                       successifs sur le bouton de la souris. Ce
                                       temps est exprimé en multiple de 50 msec.
                                       Si le temps entre deux appuis est supérieur
                                       à "tps_inter_clic * 50 msec", l'ordinateur
                                       ne considère que le nombre d'appuis succes-
                                       sifs sur le bouton avant que ce temps ne
                                       soit supérieur *)


(*****************************************************************************)
```

```
type strWrk = string[80];          (* string de travail *)
     str3 = string[3];


(* registres du processeur *)

     registre = record case integer of
               1 : (ax, bx, cx, dx, bp, si, di, es, flags : integer);
               2 : (al, ah, bl, bh, cl, ch, dl, dh        : byte);
               end;


(*****************************************************************************)
(* table des fenêtres présentes en mémoire centrale *)

(* record décrivant les objets affichables dans une fenêtre *)

     obj_fenetre = ^objet_fen;
     objet_fen = record
                 num_objet : integer;            (* type de l'objet *)
                 valeur_string : strWrk;         (* chaine de caractères associée
                                                    à l'objet *)
                 coord : array[1..4] of integer; (* coordonnées de l'objet *)
                 couleur_caractere : integer;    (* couleur des caractères associés
                                                    à l'objet *)
                 couleur_fond : integer;         (* couleur de fond de l'écran *)
                 etat_bouton : integer;          (* état du bouton *)
                 num_gr_bouton : integer;        (* N° du groupe de l'objet *)
                 num_action : integer;           (* N° de l'action associée à
                                                    l'objet *)
                 next : obj_fenetre;
                 end;


     (* records d'objets *)
     tab_fen_mc_ptr = array[1..n_max_fen_mc] of obj_fenetre;

     (* nom des fichiers dans lesquels sont décrites les fenêtres *)
     tab_fen_mc_nom = array[1..n_max_fen_mc] of strWrk;


(*****************************************************************************)
(* record décrivant les objets d'une fenêtre, de type "rectangle" et "chaine de
   caractères" *)

     obj_ecran = ^objets_ecran;
     objets_ecran = record
                    type_objet : integer;      (* type de l'objet *)
                    coord_objet : array[1..4] of integer;  (* coordonnées de
                                                              l'objet *)
                    num_gr_objet : integer;    (* N° du groupe de l'objet *)
                    num_action : integer;      (* N° de l'action associée à
                                                  l'objet *)
                    next : obj_ecran;
                    end;


(*****************************************************************************)
(* record contenant les informations nécessaires pour envoyer des signaux vers
   le "mains-libres" *)

     code_teleph = ^teleph_rec;
     teleph_rec = record
```

```
                tps_inter_chiffre : integer;  (* temps entre l'envoi de deux
                                                 chiffres d'un même numéro de
                                                 téléphone *)
                tps_inter_impuls : integer;   (* temps entre l'envoi de deux
                                                 impulsions relatives à un même
                                                 chiffre de téléphone *)
                tps_duree_impuls : integer;   (* temps de durée d'une impulsion
                                                 relative à un chiffre de
                                                 téléphone *)
                port_sortie : integer;        (* port de sortie vers lequel les
                                                 signaux de téléphone doivent
                                                 être envoyés *)
                num_bit_imp : integer;        (* numéro du bit du port de sortie
                                                 sur lequel les signaux doivent
                                                 être émis *)
                next : code_teleph;
                end;


    nbr_teleph = array[1..nb_max_fenetres] of strWrk;    (* tables des numéros de
                                                            téléphone *)

(*****************************************************************************)

    tab_chaine_cla = array[1..nb_max_fenetres] of strWrk;(* table des chaines de
                                                            caractères introduites
                                                            au clavier simulé *)

(*****************************************************************************)
(* table de conversion *)

    etat_obj_ecran = record
                     actif : boolean;           (* état de la fenêtre *)
                     obj_ecran_ptr : obj_ecran; (* pointeur vers les objets
                                                   affichés dans la fenêtre à
                                                   l'écran *)
                     end;


    table_conversion = array[1..nb_max_fenetres] of etat_obj_ecran;


(*****************************************************************************)
(* table des fenêtres. Cette table contient les coordonnées du coin supérieur
   gauche et du coin inférieur droit de chaque fenêtre affichée à l'écran *)

    tableau_fenetre = array[1..nb_max_fenetres] of array[1..4] of integer;


(*****************************************************************************)
(* table d'ordonnancement *)

    tableau_ordre_fen = array[1..nb_max_fenetres] of integer;


(*****************************************************************************)
(* table contenant l'adresse de chaque port de sortie utilisé ainsi que la
   valeur du code que l'ordinateur lui a envoyé en dernier lieu *)

    description_port = array[1..2] of integer;
```

```
    pteur_obj_ecran : obj_ecran;      (* pointeur vers un objet d'une fenêtre,
                                         situé en mémoire centrale et dessiné à
                                         l'écran *)

(*****************************************************************************)
(* téléphone *)

    ptr_teleph : code_teleph;         (* pointeur vers le record contenant les
                                         informations nécessaires pour envoyer
                                         des signaux vers le "mains-libres" *)
    pteur_telephone : code_teleph;    (* pointeur vers le record contenant les
                                         informations nécessaires pour envoyer
                                         des signaux vers le "mains-libres" *)
    pteur_teleph : code_teleph;       (* pointeur vers le record contenant les
                                         informations nécessaires pour envoyer
                                         des signaux vers le "mains-libres" *)

    teleph : teleph_rec;              (* record contenant les informations
                                         nécessaires pour envoyer des signaux
                                         vers le "mains-libres" *)

    num_telephone : nbr_teleph;       (* table des numéros de téléphone *)
    der_num_tel : strWrk;             (* dernier N° de téléphone introduit par
                                         l'utilisateur et mémorisé *)

(*****************************************************************************)
(* fichiers de texte *)

    ptr_text : fich_text_buf;         (* pointeur vers un ligne de fichier de
                                         texte mémorisée en mémoire centrale *)
    pteur_text : fich_text_buf;       (* pointeur vers un ligne de fichier de
                                         texte mémorisée en mémoire centrale *)
    prec_ptr_text : fich_text_buf;    (* pointeur vers un ligne de fichier de
                                         texte mémorisée en mémoire centrale *)

    buffer_text : buf_text;           (* record contenant une ligne de fichier de
                                         texte *)

    tampon : tab_tampon;              (* table contenant les lignes de texte à
                                         mémoriser dans un fichier de texte *)

    num_tampon : integer;       (* indice des éléments de la table contenant les
                                   lignes à mémoriser dans un fichier de texte *)

(*****************************************************************************)

    table_fenetre : tableau_fenetre;      (* table des fenêtres *)

(*****************************************************************************)

    table_ordre_fen : tableau_ordre_fen; (* table d'ordonnancement *)

(*****************************************************************************)

    table_fen_mc_nom : tab_fen_mc_nom;    (* table des fenêtres présentes en *)
    table_fen_mc_ptr : tab_fen_mc_ptr;    (* mémoire centrale *)

    ptr_table_fen_mc : integer;           (* pointeur vers les éléments de la
                                             table des fenêtres présentes en
                                             mémoire centrale *)

(*****************************************************************************)
```

```
   table_port_sortie : tableau_port_sortie; (* table des ports de sortie *)

(*****************************************************************************)

   table_convers : table_conversion;    (* table de conversion *)

(*****************************************************************************)

   erreur : integer;                    (* numéro d'une erreur survenue dans le
                                           programme "aidami" *)

(*****************************************************************************)

   sortie_aida : boolean;   (* flag de sortie du programme "aidami" *)


   affiche_menu : boolean;  (* flag indiquant qu'un menu est affiché à l'écran *)

(*****************************************************************************)
(* clavier *)

   nbre_car_buffer : integer;       (* nombre de caractères mémorisés dans
                                       le buffer du clavier simulé *)

   chaine_cla : tab_chaine_cla;     (* table des chaines de caractères introduites
                                       au clavier simulé *)

   taille_buffer : integer;         (* taille du buffer du clavier simulé *)

   logiciel_clavier : boolean; (* flag indiquant que le clavier simulé est
                                  utilisé pour mémoriser et exécuter un
                                  logiciel "standard" *)

   carac_present_buffer : boolean; (* flag indiquant qu'un caractère est mémorisé
                                      dans le buffer du clavier réel *)

   touche_active_souris : boolean;  (* flag indiquant que les touches fléchées
                                       du clavier réel contrôlent le curseur
                                       de la souris *)
   reset_touche_active : boolean;   (* flag activant ou désactivant les
                                       touches fléchées du clavier réel pour
                                       actionner le curseur de la souris *)

   clavier_fenetre_affichee : boolean; (* flag indiquant si le clavier simulé
                                          permettant de travailler en interactif
                                          avec un logiciel "standard" est affiché
                                          à l'écran *)

   buffer_tail : integer;       (* pointeur vers le premier caractère à lire dans
                                   le buffer du clavier réel *)

   code_asci_clavier,           (* codes de recherche et asci introduits au *)
   code_rech_clavier : integer; (* clavier simulé *)

   special : boolean;   (* flag indiquant qu'une touche de fonction a été
                           sélectionnée au clavier simulé *)

(*****************************************************************************)
(* souris *)

   seg_souris : integer;            (* adresse du segment contenant le logiciel
                                       de la souris *)
```

```
    clic_bouton,                (* nombre de fois que le bouton de la souris est
                                   enfoncé consécutivement *)

    tps_bouton : integer;       (* temps écoulé entre deux appuis successifs
                                   sur le bouton de la souris *)

    compte_appuis_bout,           (* nombre de fois que le bouton de la souris
                                     est enfoncé entre deux appels de la procédure
                                     "lect_nbre_appuis_bout" *)

    compte_relaches_bout : integer; (* nombre de fois que le bouton de la souris
                                     est relaché entre deux appels de la procédure
                                     "lect_nbre_relaches_bout" *)

   actif_bouton_souris : boolean;  (* flag indiquant si le bouton de la souris
                                    vient d'être enfoncé *)

(*****************************************************************************)

    logiciel : boolean;           (* flag indiquant si on est en train d'exécuter
                                     la tâche "logiciels" du programme
                                     "aidami" *)

(*****************************************************************************)
(* paramètres de gestion de la mémoire centrale *)

    taille_seg_donnees : integer;   (* taille du segment de données nécessaire
                                       pour mémoriser les variables du programme
                                       "aidami" *)
    nbre_para_seg_donnees : integer;(* nombre de paragraphes occupés par les variables
                                       du programme "aidami" *)

(*****************************************************************************)
(* exclusion mutuelle *)

    exclusion_souris,
    exclusion_DS,
    exclusion_aidami,
    exclusion_clavier,
    exclusion_ES : integer;         (* verrous pour assurer l'exclusion
                                       mutuelle des diverses sections
                                       critiques du programme *)

(*****************************************************************************)
(* gestion du curseur *)

    deplace_vertical_curseur,     (* déplacement vertical du curseur de
                                     l'écran en un "pas", exprimé en nombre
                                     de points de l'écran *)
    deplace_horizontal_curseur : integer; (* déplacement vertical du curseur de
                                         l'écran en un "pas", exprimé en nombre
                                         de points de l'écran *)

    curseur_non_present : integer;  (* flag permettant de vérifier si le curseur
                                     de la souris est affiché à l'écran *)

    x_curseur, y_curseur : integer;     (* coordonnées du curseur de l'écran
                                       avant exécution du programme "aidami" *)

(*****************************************************************************)
(* interruptions *)

    retour_chargement : boolean;        (* flag indiquant que le programme utilisateur
```

```
                                        vient de faire appel à une interruption
                                        du système d'exploitation, permettant
                                        de lancer ou de terminer un programme *)

    change_1C, changement_1C : boolean; (* flags indiquant qu'un logiciel "standard"
                                        a modifié l'adresse du vecteur d'inter-
                                        ruption numéro 1Ch *)


    offset : array[$00..$3F] of integer;  (* adresses relatives des routines de
                                          gestion d'interruptions du système
                                          d'exploitation *)
    segment : array[$00..$3F] of integer; (* adresses des segments dans lesquels
                                          sont localisées les routines du
                                          système d'exploitation *)

    int_offset : array[$00..$3F] of integer; (* adresses relatives des routines
                                             "inter i" du programme "aidami" *)

    num_vecteur : integer;                (* numéro du vecteur d'interruption
                                          appelé par le programme utilisateur *)

    detourne_clavier : boolean;  (* flag indiquant si les routines de gestion
                                 d'interruptions du système d'exploitation
                                 permettant de lire un caractère au clavier,
                                 doivent être détournées avant de passer le
                                 contrôle au système d'exploitation, en vue
                                 d'exécuter le programme "aidami" *)

    intro_data_clavier : boolean; (* flag signalant qu'un processus vient d'être
                                  détourné avant d'accéder à une routine du
                                  système d'exploitation permettant de lire
                                  un caractère au clavier *)

    ofs_entree : integer;          (* adresse d'entrée de la procédure "entree"
                                   ou de la procédure "sortie directe" *)

    css_sortie, ipp_sortie : integer; (* adresse d'entrée de la procédure "sortie" *)

(*****************************************************************************)

    fichier_inexistant : strWrk;  (* nom d'un fichier dont on désire lire le
                                  contenu, mais qui n'est pas présent sur le
                                  disque *)

    actif_son : boolean;          (* flag indiquant si le son est coupé dans le
                                  programme "aidami" *)

    ofs_prog_aidami : integer;    (* adresse relative du début du programme
                                  "aidami" *)
    lecture_fichier : boolean;    (* flag indiquant que le programme "aidami" est
                                  susceptible de faire des accès dans des

                                   fichiers *)

    mode_ecran : integer;          (* mode de l'écran avant d'exécuter le
                                   programme "aidami" *)

(*****************************************************************************)
```

```
procedure init_verif_prog_aidami(var erreur : integer);
(*******************************************************)
(* Vérifie si le programme "aidami" est déjà chargé en mémoire centrale *)

begin
ofs_prog_aidami := ofs(prog_aidami);
if offset[$1C] = ofs_prog_aidami then erreur := 26;
end;


(*****************************************************************************)


procedure init_son(actif : boolean);
(***********************************)
(* initialisation de la variable de son *)

begin
if actif = true then actif_son := true
               else actif_son := false;
end;


(*****************************************************************************)


procedure init_8255(var erreur : integer);
(*****************************************)
(* Initialisation du circuit électronique N° 8255 du système "aidami" *)

var port_s : integer;            (* adresse des ports de sortie initialisés *)
    pteur_teleph : code_teleph;  (* pointeur vers les éléments du fichier de
                                    téléphone *)
    n_type_pteur : integer;      (* N° du pointeur de la structure du fichier *)

begin
acces_seq('telep1',0,-1, n_type_pteur, erreur);
pteur_teleph := ptr_teleph;
if erreur = 0
    then begin
         port_s := ptr_teleph^.port_sortie;
         interdire_interrupt;
         table_port_sortie[1,1] := port_s;
         table_port_sortie[1,2] := $FF;
         table_port_sortie[2,1] := port_s + 1;
         table_port_sortie[2,2] := $FF;
         table_port_sortie[3,1] := port_s + 2;
         table_port_sortie[3,2] := $FF;
         table_port_sortie[4,1] := port_s + 3;
         table_port_sortie[4,2] := $81;
         envoyer_code_appareils(0, port_s + 3, 50, 1, erreur);
         envoyer_code_appareils(0, port_s    , 50, 1, erreur);
         envoyer_code_appareils(0, port_s + 1, 50, 1, erreur);
         autoriser_interrupt;
         dispose(ptr_teleph);
         end;
end;


(*****************************************************************************)


procedure init_souris(var erreur : integer);
```

```
(*****************************************)
(* Initialisation des variables de la souris et du logiciel de la souris *)

var present : integer;  (* flag indiquant que le logiciel de la souris est
                           présent sur le disque *)

begin
actif_bouton_souris := false;
modif_action_souris := false;
action_souris := 0;
touche_active_souris := false;
reset_touche_active := true;
clic_bouton := 0;
tps_bouton := 0;
compte_appuis_bout := 0;
compte_relaches_bout := 0;

(* initialisation du logiciel de la souris *)

seg_souris := segment[num_vecteur_souris];
if (seg_souris <> 0) AND (offset[num_vecteur_souris] <> 0)
    then begin
         regs.ax := 0;
         regs.bx := 2;
         intr(num_vecteur_souris,regs);
         end
    else erreur := 20;
end;


(*****************************************************************************)


procedure init_table_fenetre;
(****************************)
(* Initialisation de la table des fenêtres *)

var i, j : integer;

begin
for i := 1 to nb_max_fenetres do
    begin
    for j := 1 to 4 do
        begin
        table_fenetre[i,j] := -1;
        end;
    end;
end;


(*****************************************************************************)


procedure init_table_ordre_fen;
(*****************************)
(* Initialisation de la table d'ordonnancement *)

var i : integer;

begin
for i := 1 to nb_max_fenetres do
    begin
    table_ordre_fen[i] := 0;
```

```
    end;
end;


(*****************************************************************************)


procedure init_table_fen_mc;
(**************************)
(* Initialisation de la table des fenêtres présentes en mémoire centrale *)

var i : integer;

begin
for i := 1 to n_max_fen_mc do
    begin
    table_fen_mc_nom[i] := '';
    table_fen_mc_ptr[i] := nil;
    end;
end;


(*****************************************************************************)


procedure init_ptr_table_fen_mc;
(******************************)
(* Initialisation du pointeur vers les éléments de la table des fenêtres
   présentes en mémoire centrale *)

begin
ptr_table_fen_mc := 1;
end;


(*****************************************************************************)


procedure init_table_convers;
(***************************)
(* Initialisation de la table de conversion *)

var i : integer;

begin
for i := 1 to nb_max_fenetres do
    begin
    table_convers[i].actif := false;
    table_convers[i].obj_ecran_ptr := nil;
    end;
end;


(*****************************************************************************)


procedure init_procedure_inter_i;
(*******************************)
(* Mémorisation des adresses d'entrée dans les procédures "inter i" *)

begin
int_offset[$00] := ofs(inter_00h);
int_offset[$01] := ofs(inter_01h);
```

```
int_offset[$02] := ofs(inter_02h);
int_offset[$03] := ofs(inter_03h);
int_offset[$04] := ofs(inter_04h);
int_offset[$05] := ofs(inter_05h);
int_offset[$06] := ofs(inter_06h);
int_offset[$07] := ofs(inter_07h);

int_offset[$0A] := ofs(inter_0Ah);
int_offset[$0B] := ofs(inter_0Bh);
int_offset[$0C] := ofs(inter_0Ch);
int_offset[$0D] := ofs(inter_0Dh);
int_offset[$0E] := ofs(inter_0Eh);
int_offset[$0F] := ofs(inter_0Fh);
int_offset[$10] := ofs(inter_10h);
int_offset[$11] := ofs(inter_11h);
int_offset[$12] := ofs(inter_12h);
int_offset[$13] := ofs(inter_13h);
int_offset[$14] := ofs(inter_14h);
int_offset[$15] := ofs(inter_15h);
int_offset[$16] := ofs(inter_16h);
int_offset[$17] := ofs(inter_17h);
int_offset[$18] := ofs(inter_18h);
int_offset[$19] := ofs(inter_19h);
int_offset[$1A] := ofs(inter_1Ah);
int_offset[$1B] := ofs(inter_1Bh);

int_offset[$20] := ofs(inter_20h);
int_offset[$21] := ofs(inter_21h);
int_offset[$25] := ofs(inter_25h);
int_offset[$26] := ofs(inter_26h);
int_offset[$27] := ofs(inter_27h);
int_offset[$28] := ofs(inter_28h);
int_offset[$29] := ofs(inter_29h);
int_offset[$2A] := ofs(inter_2Ah);
int_offset[$2B] := ofs(inter_2Bh);
int_offset[$2C] := ofs(inter_2Ch);
int_offset[$2D] := ofs(inter_2Dh);
int_offset[$2E] := ofs(inter_2Eh);
int_offset[$2F] := ofs(inter_2Fh);
int_offset[$30] := ofs(inter_30h);
int_offset[$31] := ofs(inter_31h);
int_offset[$32] := ofs(inter_32h);
int_offset[$33] := ofs(souris_entree);
int_offset[$34] := ofs(inter_34h);
int_offset[$35] := ofs(inter_35h);
int_offset[$36] := ofs(inter_36h);
int_offset[$37] := ofs(inter_37h);
int_offset[$38] := ofs(inter_38h);
int_offset[$39] := ofs(inter_39h);
int_offset[$3A] := ofs(inter_3Ah);
int_offset[$3B] := ofs(inter_3Bh);
int_offset[$3C] := ofs(inter_3Ch);
int_offset[$3D] := ofs(inter_3Dh);
int_offset[$3E] := ofs(inter_3Eh);
int_offset[$3F] := ofs(inter_3Fh);
end;


(*****************************************************************************)


procedure init_piles;
(********************)
```

```
(* Initialisation des segments de pile, permettant de localiser les piles
   utilisées en mémoire centrale *)

var debut_segment_pile : integer;  (* début du segment de pile associé aux
                                      différentes piles utilisées dans le
                                      programme "aidami" *)

begin
debut_segment_pile := Dseg + nbre_para_seg_donnees + taille_heap;

nss_entree := debut_segment_pile;
nss_prog_aidami := debut_segment_pile;
nss_clavier := debut_segment_pile;
nss_sortie := debut_segment_pile;
nss_souris1 := debut_segment_pile;
nss_souris2 := debut_segment_pile;
nss_fct25 := debut_segment_pile;
end;


(*****************************************************************************)


procedure init_prim_exclusion;
(****************************)
(* Initialisation des verrous *)

begin
exclusion_souris := 1;
exclusion_OS := 1;
exclusion_aidami := 1;
exclusion_clavier := 1;
exclusion_ES := 1;
end;


(*****************************************************************************)




procedure init_fenetres;
(**********************)
(* initialisation de variables de fenêtres *)

begin
affiche_menu := false;
num_fenetre_menu := 0;
end;


(*****************************************************************************)


procedure init_vecteurs_interrupt_changes;
(****************************************)
(* initialisation de variables associées aux routines "inter_21_25h" et "inter_21h"
   pour gérer les vecteurs d'interruption dont l'adresse est modifiée par un
   logiciel "standard" *)

begin
changer_vecteur := false;
```

```
interdire_interrupt;
change_1C := false;
changement_1C := false;
autoriser_interrupt;
end;


(*****************************************************************************)


procedure init_procedure_os;
(**************************)
(* Initialisation des variables contenant l'adresse d'entrée dans les procédures
   utilisées pour réaliser la multi-programmation *)

begin
retour_chargement := false;

ipp_sortie := ofs(sortie) + 7;
css_sortie := Cseg;
ipp_souris := ofs(souris_sortie) + 7;
css_souris := Cseg;
ofs_entree := ofs(entree) + 4;
interrupt_21_25h := ofs(inter_21_25h) + 4;

ofs_saut_1 := ofs(saut) + 8;
ofs_saut_2 := ofs(saut) + 10;
ofs_saut_3 := ofs(saut_cla) + 8;
ofs_saut_4 := ofs(saut_cla) + 10;
ofs_saut_5 := ofs(saut_souris) + 8;
ofs_saut_6 := ofs(saut_souris) + 10;
end;


(*****************************************************************************)


procedure init_clavier;
(**********************)
(* Initialisation des variables du clavier simulé et du clavier réel *)

begin
ctrl := false;
shift_gauche := false;
shift_droit := false;
alternate := false;
caps_lock := false;
caps_lock_etat := false;
caps_lock_etat_simul := false;
num_lock := false;
num_lock_etat := false;
scroll_lock := false;
scroll_lock_etat := false;
insert := false;
insert_etat := false;
fct := false;

nbre_car_buffer := 0;

interdire_interrupt;
memw[seg_buffer_clavier:kb_flag_mem] := $00;
autoriser_interrupt;
```

```
effacer_carac_buffer := false;
detourne_clavier := true;
intro_data_clavier := false;
clavier_fenetre_affichee := false;
end;


(*****************************************************************************)


procedure init_telephone;
(************************)
(* Initialisation des variables du téléphone *)

var i : integer;

begin
der_num_tel := '';
for i := 1 to nb_max_fenetres do
    begin
    num_telephone[i] := '';
    end;
end;


(*****************************************************************************)


procedure init_logiciel;
(***********************)
(* Initialisations des variables de la tâche "logiciel" *)

begin
logiciel := false;
logiciel_clavier := false;
memw[$0000:$0188] := 0;
memw[$0000:$0184] := 1;
end;


(*****************************************************************************)


procedure init_graphique;
(************************)
(* Initialisation du logiciel "Turbo-Graphix" *)

begin
deplace_vertical_curseur := 4;
deplace_horizontal_curseur := 8;
initgraphic;
LeaveGraphic;
end;


(*****************************************************************************)


procedure init_fichiers_batch;
(*****************************)

var erreur : integer;
```

```
begin
creer_fich('batch1.bat',erreur);
vider_contenu_fich('batch1.bat',erreur);
tampon[1] := ' aida_go';
tampon[2] := ' ';
tampon[3] := ' c:';
tampon[4] := ' batch2';
ecrire_fich_seq_text('batch1.bat',0,3,erreur);
end;


(*****************************************************************************)


procedure init(var erreur : integer);
(*********************************)
(* Initialisation *)

begin
init_verif_prog_aidami(erreur);
init_son(false);
if erreur = 0
    then begin
         init_8255(erreur);
         if erreur = 0
             then begin
                  init_souris(erreur);
                  if erreur = 0 then begin
                                     init_table_fenetre;
                                     init_table_ordre_fen;
                                     init_table_fen_mc;
                                     init_ptr_table_fen_mc;
                                     init_table_convers;
                                     init_procedure_inter_i;
                                     init_piles;
                                     init_prim_exclusion;
                                     init_fenetres;
                                     init_vecteurs_interrupt_changes;
                                     init_procedure_os;
                                     init_clavier;
                                     init_telephone;
                                     init_logiciel;
                                     init_graphique;
                                     init_son(true);
                                     init_fichiers_batch;
                                     end;
                  end;
         end;
end;


(*****************************************************************************)


procedure init_lire_vect_interrupt;
(*********************************)
(* Lecture des vecteurs d'interruption du système d'exploitation *)

begin
interdire_interrupt;
for numero_vecteur := $00 to $3F do
    begin
```

```
    lire_vecteur_interrupt;
    offset[numero_vecteur] := offset_code;
    segment[numero_vecteur] := segment_code;
    end;
autoriser_interrupt;
end;


(*****************************************************************************)


procedure init_ecrire_vect_interrupt;
(***********************************)
(* Ecriture des adresses des routines "inter i" et de quelques autres procédures
   dans les vecteurs d'interruption du système d'exploitation et dans certains
   vecteurs disponibles pour l'utilisateur *)

begin
interdire_interrupt;
for numero_vecteur := $00 to $3F do
    begin
    if not(numero_vecteur in [$08,$09,$1C,$1D,$1E,$1F,$21,$22,$23,$24,
                              $28..$2E])
         then begin
              segment_code := Cseg;
              offset_code := int_offset[numero_vecteur];
              ecrire_vecteur_interrupt;
              end;
    end;

numero_vecteur := $60;
segment_code := segment[$09];
offset_code := offset[$09];
ecrire_vecteur_interrupt;

numero_vecteur := $09;
segment_code := Cseg;
offset_code := ofs(clavier);
ecrire_vecteur_interrupt;

numero_vecteur := $1C;
segment_code := Cseg;
offset_code := ofs(prog_aidami);
ecrire_vecteur_interrupt;

numero_vecteur := $21;
segment_code := Cseg;
offset_code := int_offset[$21];
ecrire_vecteur_interrupt;

autoriser_interrupt;
end;


(*****************************************************************************)
```

```
(* GERANT DES INTERRUPTIONS *)


var
    numero_vect,                        (* numéro d'un vecteur d'interruption,
                                           utilisé exclusivement par les procédures
                                           "lire_vecteur_interrupt" et "écrire_
                                           vecteur_interrupt" *)
    numero_vecteur : integer;           (* numéro d'un vecteur d'interruption à
                                           lire ou à écrire. Cette variable est
                                           globale pour les procédures "lire_
                                           vecteur_interrupt" et "écrire_vecteur_
                                           interrupt" *)
    segment_code, offset_code : integer;(* adresse du segment et adresse relative
                                           d'une procédure à lire ou à écrire
                                           dans un vecteur d'interruption *)

procedure lire_vecteur_interrupt;
(*********************************)
(* lecture de l'adresse contenue dans un vecteur d'interruption *)
(* "numero_vect" = numéro du vecteur à lire *)
(* "segment_code" = segment de code dans lequel se situe la routine de gestion
   des interruptions pointée par ce vecteur *)

begin
numero_vect := $3500 + numero_vecteur;
inline($A1/numero_vect);
inline($CD/$21);
inline($06/$58/$A3/segment_code/$53/$58/$A3/offset_code);
end;


(*****************************************************************************)


procedure ecrire_vecteur_interrupt;
(*********************************)
(* écriture d'une adresse dans un vecteur d'interruption *)
(* "numero_vect" = numéro du vecteur à écrire *)
(* "segment_code" = segment de code dans lequel se situe la routine de gestion
   des interruptions pointée par ce vecteur *)

begin
numero_vect := $2500 + numero_vecteur;
inline($1E);
inline($A1/offset_code/$92);
inline($A1/segment_code/$50/$A1/numero_vect/$1F);
inline($CD/$21);
inline($1F);
end;


(*****************************************************************************)


procedure autoriser_interrupt;
(*****************************)
(* autorisation des interruptions externes de type "INTR" *)

begin
inline($FB);
end;
```

```
(*****************************************************************************)
(*****************************************************************************)


procedure interdire_interrupt;
(*****************************)
(* interdiction des interruptions externes de type "INTR" *)

begin
inline($FA);
end;


(*****************************************************************************)
(*****************************************************************************)


procedure attendre(var prim_exclusion_mutuelle : integer);
(*********************************************************)
(* première primitive d'exclusion mutuelle précédant une section critique *)

begin
interdire_interrupt;
if prim_exclusion_mutuelle = 1 then prim_exclusion_mutuelle := 0;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure signaler(var prim_exclusion_mutuelle : integer);
(*********************************************************)
(* deuxième primitive d'exclusion mutuelle suivant une section critique *)

begin
interdire_interrupt;
prim_exclusion_mutuelle := 1;
end;


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)

(*       GERANT DES ENTREES/SORTIES *)


procedure son(frequence,tps : integer);
(*************************************)
(* fait émettre un signal sonore par l'ordinateur *)

begin
if actif_son = true then begin
                         sound(frequence);
                         delay(tps);
                         nosound;
                         end;
end;
```

```
(*****************************************************************************)
(*****************************************************************************)


procedure changer_flag(code_flag : integer; etat_touche : boolean);
(*-----------------------------------------------------------*)
(* modification des octets de configuration du clavier *)
(* "code_flag" = nouveau code permettant de modifier les octets de
   configuration *)
(* "etat_touche" = flag indiquant si une touche de fonction du clavier est
   enfoncée ou relâchée *)

var key_flag : integer;  (* mot de configuration désignant les octets
                            "KB_FLAG" et "KB_FLAG_1" du clavier réel *)

begin
interdire_interrupt;
key_flag := ($100 * mem[seg_buffer_clavier:kb_flag_1_mem])
            + mem[seg_buffer_clavier:kb_flag_mem];
if etat_touche = true
    then memw[seg_buffer_clavier:kb_flag_mem] := (key_flag) OR (code_flag)
    else memw[seg_buffer_clavier:kb_flag_mem] := (key_flag) AND (code_flag);
autoriser_interrupt;
end;



(*****************************************************************************)
(*****************************************************************************)


(* variables décrivant l'état des touches de fonction du clavier réel *)

var
    ctrl, shift_gauche, shift_droit, alternate,
    caps_lock, caps_lock_etat, caps_lock_etat_simul,
    num_lock, num_lock_etat,
    scroll_lock, scroll_lock_etat,
    insert, insert_etat : boolean;

    fct : boolean;                  (* flag indiquant si une touche de fonction
                                       "Fi" du clavier simulé est acivée *)


procedure reset_touch_special;
(*****************************)
(* désactivation des touches d'état du clavier préalablement activées *)

begin
if (ctrl = true) then begin
                      changer_flag($FFFB, false);
                      ctrl := false;
                      end;
if (shift_gauche = true) then begin
                              changer_flag($FFFD, false);
                              shift_gauche := false;
                              end;
if (shift_droit = true) then begin
                             changer_flag($FFFE, false);
                             shift_droit := false;
                             end;
if (alternate = true) then begin
                           changer_flag($FFF7, false);
```

```
                        alternate := false;
                        end;
if (caps_lock = true) then begin
                        changer_flag($BFFF, false);
                        caps_lock := false;
                        end;
if (num_lock = true) then begin
                       changer_flag($DFFF, false);
                       num_lock := false;
                       end;
if (scroll_lock = true) then begin
                          changer_flag($EFFF, false);
                          scroll_lock := false;
                          end;
if (insert = true) then begin
                      changer_flag($7FFF, false);
                      insert := false;
                      end;
fct := false;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure flags(code : integer);
(*****************************)
(* modification de l'état des variables décrivant l'état du clavier, en fonction
   de la valeur du code de recherche *)
(* "code" = code de recherche *)

var code_flag : integer;  (* code permettant de modifier l'état des variables
                             du clavier *)

begin
case code of
    29 : begin
         ctrl := true;
         code_flag := $0004;
         changer_flag(code_flag, true);
         end;
    42 : begin
         shift_gauche := true;
         code_flag := $0002;
         changer_flag(code_flag, true);
         end;
    54 : begin
         shift_droit := true;
         code_flag := $0001;
         changer_flag(code_flag, true);
         end;
    56 : begin
         alternate := true;
         code_flag := $0008;
         changer_flag(code_flag, true);
         end;
    58 : begin
         caps_lock_etat := not caps_lock_etat;
         caps_lock := true;
         if caps_lock_etat = true
            then code_flag := $4040
            else begin
```

```
                code_flag := $FFBF;
                changer_flag(code_flag, false);
                code_flag := $4000;
                end;
          changer_flag(code_flag, true);
          end;
     69 : begin
          num_lock_etat := not num_lock_etat;
          num_lock := true;
          if num_lock_etat = true
             then code_flag := $2020
             else begin
                code_flag := $FFDF;
                changer_flag(code_flag, false);
                code_flag := $2000;
                end;
          changer_flag(code_flag, true);
          end;
     70 : begin
          scroll_lock_etat := not scroll_lock_etat;
          scroll_lock := true;
          if scroll_lock_etat = true
             then code_flag := $1010
             else begin
                code_flag := $FFEF;
                changer_flag(code_flag, false);
                code_flag := $1000;
                end;
          changer_flag(code_flag, true);
          end;
     82 : begin
          insert_etat := not insert_etat;
          insert := true;
          if insert_etat = true
             then code_flag := $8080
             else begin
                code_flag := $FF7F;
                changer_flag(code_flag, false);
                code_flag := $8000;
                end;
          changer_flag(code_flag, true);
          end;
     end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure ecrire_carac_clavier(asci, code : integer; var erreur : integer);
(*************************************************************************)
(* écriture d'un caractère dans le buffer du clavier *)
(* "asci" et "code" = codes ascii et de recherche associés au caractère à
   mémoriser dans le buffer du clavier réel de la machine *)


procedure memoriser_buffer(asci, code : integer; var erreur : integer);
(*-------------------------------------------------------------*)
(* mémorisation des codes ASCII et de recherche dans le buffer du clavier
   et mise à jour des pointeurs associés à ce buffer *)
(* "asci" et "code" = codes ascii et de recherche *)
```

```
var valeur_buffer,              (* code combinant les codes ascii et de recherche,
                                   qu'il faut mémoriser dans le buffer du clavier *)
    buffer_tail,                (* pointeur vers le dernier caractère disponible
                                   dans le buffer du clavier réel *)
    buffer_head,                (* pointeur vers le premier caractère disponible
                                   dans le buffer du clavier réel *)
    ptr_buffer_tail : integer;  (* nouvelle valeur à donner au pointeur
                                   "buffer_tail" après mémorisation des codes
                                   ascii et de recherche dans le buffer du
                                   clavier réel *)

begin

(* lecture des pointeurs "buffer_tails" et "buffer_head" du clavier réel *)

interdire_interrupt;
buffer_tail := ($100 * mem[seg_buffer_clavier:buffer_tail_mem + 1])
               + mem[seg_buffer_clavier:buffer_tail_mem];
buffer_head := ($100 * mem[seg_buffer_clavier:buffer_head_mem + 1])
               + mem[seg_buffer_clavier:buffer_head_mem];
autoriser_interrupt;

(* détermination de la nouvelle valeur à donner au pointeur "buffer_tail" *)

if (buffer_tail + 2) = buffer_end
    then ptr_buffer_tail := buffer_start
    else ptr_buffer_tail := buffer_tail + 2;

(* mémorisation des codes dans le buffer du clavier réel et mise à jour du
   pointeur "buffer_tail" *)

if ptr_buffer_tail = buffer_head
    then son(400,200)
    else begin
         valeur_buffer := asci + ($100 * code);
         interdire_interrupt;
         memw[seg_buffer_clavier:buffer_tail] := valeur_buffer;
         memw[seg_buffer_clavier:buffer_tail_mem] := ptr_buffer_tail;
         autoriser_interrupt;
         end;
end;




begin       (**** ecrire_carac_clavier ****)

if (asci >= 0) and (asci <= 255) and (code >= 0) and (code <= max_code)
    then begin
         flags(code);
         if not(code in [42,54,58,69,70])
              then memoriser_buffer(asci, code, erreur);
         end
    else erreur := 14;
end;


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
```

```
(* GERANT DES FICHIERS *)


procedure exist_fich( filename : strWrk; var exist : boolean);
(*************************************************************)
(* vérification de l'existence d'un fichier *)

var f : file;  (* fichier *)

begin
assign(f, filename);    (*$I-*)
reset(f);
exist := (ioresult = 0);
if exist = false then fichier_inexistant := filename;
close(f);               (*$I+*)
end;


(*****************************************************************************)
(*****************************************************************************)


procedure creer_fich(filename : strWrk; var erreur : integer);
(*************************************************************)
(* crée le fichier "filename" sur disque *)

var exist : boolean;  (* flag d'existence du fichier "filename" *)
    f : file;         (* fichier *)

begin
exist_fich(filename,exist);
if exist = false then begin
                      assign(f,filename);
                      rewrite(f);
                      close(f);
                      end
                 else erreur := 27;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure ouvrir_fich( filename : strWrk; n_type_pteur, lect : integer;
                       var erreur : integer);
(*********************************************************************)
(* ouverture d'un fichier *)
(* "n_type_pteur" = N° du pointeur de la structure du fichier *)
(* 'lect" = type d'ouverture: en lecture ou en écriture *)

var exist : boolean;  (* flag d'existence du fichier "filename" *)

begin
exist_fich( filename, exist);
if exist = true then begin
                     case n_type_pteur of
                        1 : begin
                            assign(f_teleph, filename);
                            if lect = 1 then reset(f_teleph)
                                        else rewrite(f_teleph);
                            end;
                        2 : begin
```

```
                        assign(f_fen, filename);
                        if lect = 1 then reset(f_fen)
                                    else rewrite(f_fen);
                        end;
                    3 : begin
                        assign(f_text, filename);
                        if lect = 1 then reset(f_text)
                                    else rewrite(f_text);
                        end;
                    else erreur := 4;
                    end;
                end
            else erreur := 1;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure fermer_fich( filename : strWrk; n_type_pteur : integer; var erreur : integer);
(*****************************************************************************************)
(* fermeture d'un fichier *)
(* "n_type_pteur" = N° du pointeur de la structure du fichier *)

var exist : boolean;  (* flag d'existence du fichier "filename" *)

begin
exist_fich( filename, exist);
if exist = true then begin
                case n_type_pteur of
                    1 : close(f_teleph);
                    2 : close(f_fen);
                    3 : close(f_text);
                    else erreur := 4;
                    end;
                end
            else erreur := 1;
end;


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)

(* ACCES FICHIERS *)


procedure vider_contenu_fich(filename : strWrk; var erreur : integer);
(*********************************************************************)
(* efface les informations contenues dans un fichier *)

var f : file;         (* fichier *)
    exist : boolean;  (* flag d'existence du fichier "filename" *)

begin
exist_fich(filename, exist);
if exist = true then begin
                assign(f, filename);
                rewrite(f);
                close(f);
                end
```

```
              else erreur := 1;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure lire_fich( num_rec, n_type_pteur : integer; var erreur : integer);
(*****************************************************************************)
(* lecture d'un enregistrement dans un fichier *)
(* "num_rec" = N° de l'enregistrement à lire *)
(* "n_type_pteur" = N° du pointeur de la structure du fichier *)


procedure lecture_ligne_text(num_rec : integer; var text_buffer : strWrk;
                             var erreur : integer);
(*-------------------------------------------------------------------*)
(* lecture d'une ligne dans un fichier de texte, dont le numéro est donné
   par "num_rec" *)
(* "num_rec" = N° de la ligne à lire *)

var i : integer;

begin
i := 0;
reset(f_text);
while not(eof(f_text)) AND (i <= num_rec) do
    begin
    readln(f_text,text_buffer);
    i := i + 1;
    end;
if i <= num_rec then erreur := 3;
end;




begin               (**** lire_fich ****)
case n_type_pteur of
     1 : begin
         if num_rec < filesize(f_teleph)
            then begin
                 seek(f_teleph, num_rec);
                 read(f_teleph, teleph);
                 end
            else erreur := 3;
         end;
     2 : begin
         if num_rec < filesize(f_fen)
            then begin
                 seek(f_fen, num_rec);
                 read(f_fen, fenetre);
                 end
            else erreur := 3;
         end;
     3 : lecture_ligne_text(num_rec,buffer_text.tampon,erreur);
     else erreur := 4;
     end;
end;


(*****************************************************************************)
```

```
    var
    ofs_saut_3, ofs_saut_4 : integer;   (* adresse d'entrée de la routine de gestion
                                           des interruptions appelée par le
                                           programme utilisateur *)


procedure sortie_directe;
(************************)
(* fait appel à une primitive du système d'exploitation sans exécuter de
   primitive d'exclusion mutuelle *)

begin

(* mémorisation du numéro du vecteur appelé par le programme de l'utilisateur *)

memw[Cseg:ofs_saut_3] := offset[num_vecteur];
memw[Cseg:ofs_saut_4] := segment[num_vecteur];

(* restauration des registres du processeur *)

inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58);
inline($5D/$8B/$E5/$5D);

(* saut inconditionnel absolu vers une procédure du système d'exploitation
   à exécuter *)
(* ATTENTION !!! ce bloc d'instructions ne peut être scindé !!! *)

(************************************)
inline($E9/$0E/$00);              (****)
end;                                (**)
procedure saut_cla;                 (**)
begin                               (**)
inline($EA/$00/$00/$00/$00);        (**)
end;                              (****)
(************************************)


(*****************************************************************************)


var
    num_vecteur_inter : integer;    (* mémorisation du numéro de vecteur appelé *)


procedure modif_inter_clavier;
(****************************)
(* exécution du programme "AIDAMI" avant de passer le contrôle à une routine
   permettant de lire des caractères au clavier, située dans le système
   d'exploitation *)

begin
num_vecteur_inter := num_vecteur;
ofs_entree := ofs(sortie_directe) + 4;

son(500,70);
intro_data_clavier := true;

(* appel de l'interruption numéro 1Ch *)

intr($1C,regs);

ofs_entree := ofs(sortie_directe) + 4;
num_vecteur := num_vecteur_inter;
```

```
end;

(*****************************************************************************)

var

    (* registres du processeur mémorisés dans a pile lors de la génération
       de l'interruption par le programme utilisateur *)
    ds_entree, ax_entree, dx_entree, sp_entree, bp_entree,

    ass_entree,              (* ancien Stack Segment *)
    asp_entree,              (* ancien Stack Pointer *)
    abp_entree,              (* ancien Base Pointer *)
    nss_entree : integer;    (* nouveau Stack Segment *)

    ipp_prog,                (* registres IP, CS, FR sauvés dans la pile lors *)
    css_prog,                (* de la génération d'une interruption par un *)
    frr_prog : integer;      (* programme utilisateur *)

    retour_sortie : boolean; (* flag indiquant si un processus exécutant la procédure
                                "entree" doit exécuter la procédure "sortie" après
                                avoir exécuté la routine de gestion des interruptions
                                du système d'exploitation désirée *)

    num_fonct : integer;                (* numéro d'une fonction du système
                                           d'exploitation appelée *)
    type_fonction : integer; (* type particulier d'une fonction de l'interruption
                                numéro 06H du système d'exploitation, permettant
                                de lire un caractère au clavier réel *)

    css, ipp : integer;      (* mémorise l'adresse de retour aprés avoir exécuté
                                la routine de gestion des interruptions du système
                                d'exploitation, désirée *)

    ofs_saut_1, ofs_saut_2 : integer; (* adresse d'entrée de la routine de gestion
                                         des interruptions appelée par le
                                         programme utilisateur *)


procedure entree;
(****************)
(* permet l'exécution d'une primitive d'exclusion mutuelle avant de donner
   le contrôle à une primitive du système d'exploitation *)

begin

(* mémorisation de l'adresse de retour du programme de l'utilisateur, située
   dans la pile *)

inline($07);
inline($58/$A3/ds_entree/$5F/$5E/$58/$A3/dx_entree/$59/$5B);
inline($58/$A3/ax_entree);
inline($58/$A3/sp_entree/$58/$A3/bp_entree);
inline($58/$A3/ipp_prog/$58/$A3/css_prog/$58/$A3/frr_prog);

inline($A1/frr_prog/$50/$A1/css_prog/$50/$A1/ipp_prog/$50);
inline($A1/bp_entree/$50/$A1/sp_entree/$50);
inline($A1/ax_entree/$50);
inline($53/$51/$A1/dx_entree/$50/$56/$57/$A1/ds_entree/$50);
inline($06);
```

```
(* changement de pile *)

inline($16/$58/$A3/ass_entree/$A1/nss_entree/$50/$17);
inline($94/$A3/asp_entree/$95/$A3/abp_entree);
inline($B8/pile_OS/$94/$B8/pile_OS/$95);          (* changement de pile *)

(* test sur le numéro de vecteur appelé et action exécutée en conséquence,
   et test sur le numéro de fonction éventuel *)

retour_sortie := true;

regs.ax := ax_entree;
num_fonct := regs.ah;
case num_vecteur of
   $16 : if (num_fonct = $00) AND (detourne_clavier = true)
            then modif_inter_clavier;
   $21 : begin
         case num_fonct of
            $01,$07,$08,$0A,$0C : if detourne_clavier = true then modif_inter_clavier;
            $06 : begin
                  regs.dx := dx_entree;
                  type_fonction := regs.dl;
                  if (detourne_clavier = true) AND (type_fonction = $FF)
                      then modif_inter_clavier;
                  end;
            $00,$31,$4B,$4C : begin
                              retour_sortie := false;
                              retour_chargement := true;
                              end;
            end;
         end;
   $20,$27 : begin
             retour_sortie := false;
             retour_chargement := true;
             end;
   end;

(* mémorisation de l'adresse de retour au programme utilisateur interrompu si on
   appelle une routine permettant de lancer ou de terminer un programme;
   mémorisation de l'adresse de retour à la procédure "sortie" sinon *)

if retour_sortie = true then begin
                             css := css_sortie;
                             ipp := ipp_sortie;
                             attendre(exclusion_OS);
                             end
                        else begin
                             css := css_prog;
                             ipp := ipp_prog;
                             attendre(exclusion_ES);
                             end;

ofs_entree := ofs(sortie_directe) + 4;

(* restauration de la pile *)

inline($A1/ass_entree/$50/$17);
inline($A1/asp_entree/$94/$A1/abp_entree/$95);

(* mémorisation de l'adresse de la routine "sortie" ou de celle du programme
   de l'utilisateur, dans la pile *)

inline($07);
```

```
inline($58/$A3/ds_entree/$5F/$5E/$5A/$59/$5B);
inline($58/$A3/ax_entree);
inline($58/$A3/sp_entree/$58/$A3/bp_entree);
inline($58/$A3/ipp_prog/$58/$A3/css_prog);

inline($A1/css/$50/$A1/ipp/$50);
inline($A1/bp_entree/$50/$A1/sp_entree/$50);
inline($A1/ax_entree/$50);
inline($53/$51/$52/$56/$57/$A1/ds_entree/$50);
inline($06);

(* mémorisation de l'adresse de la routine du système d'exploitation à exécuter *)

memw[Cseg:ofs_saut_1] := offset[num_vecteur];
memw[Cseg:ofs_saut_2] := segment[num_vecteur];

inline($FB);

(* restauration des registres du processeur *)

inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58);
inline($5D/$8B/$E5/$5D);

(* saut inconditionnel absolu vers une procédure du système d'exploitation
   à exécuter *)
(* ATTENTION !!! ce bloc d'instructions ne peut être scindé !!! *)

(***********************************)
inline($E9/$0E/$00);           (****)
end;                             (**)
procedure saut;                  (**)
begin                            (**)
inline($EA/$00/$00/$00/$00);     (**)
end;                           (****)
(***********************************)


(*****************************************************************************)


(* variables de mémorisation des registres du processeur dans la procédure
   "sortie" du programme *)

var
    (* registres du processeur mémorisés dans a pile lors de la génération
       de l'interruption par le programme utilisateur *)
     ds_sortie, ax_sortie, sp_sortie, bp_sortie,

     ass_sortie,               (* ancien Stack Segment *)
     asp_sortie,               (* ancien Stack Pointer *)
     abp_sortie,               (* ancien Base Pointer *)
     nss_sortie : integer;     (* nouveau Stack Segment *)


procedure sortie;
(***************)
(* permet l'exécution d'une primitive d'exclusion mutuelle à la sortie d'une
   routine du système d'exploitation *)

begin

(* sauvetage des registres du processeur *)
```

```
inline($9C/$50/$50/$FA);
inline($55/$8B/$EC/$55);
inline($50/$53/$51/$52/$56/$57/$1E/$06);

(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

(* changement de pile *)

inline($16/$58/$A3/ass_sortie/$A1/nss_sortie/$50/$17);
inline($94/$A3/asp_sortie/$95/$A3/abp_sortie);
inline($B8/pile_OS/$94/$B8/pile_OS/$95);

ofs_entree := ofs(entree) + 4;
signaler(exclusion_OS);

(* restauration de la pile *)

inline($A1/ass_sortie/$50/$17);
inline($A1/asp_sortie/$94/$A1/abp_sortie/$95);

(* mémorisation de l'adresse de retour au programme utilisateur, dans la pile *)

inline($07);
inline($58/$A3/ds_sortie/$5F/$5E/$5A/$59/$5B);
inline($58/$A3/ax_sortie);
inline($58/$A3/sp_sortie/$58/$A3/bp_sortie);
inline($58/$58);

inline($A1/css_prog/$50/$A1/ipp_prog/$50);
inline($A1/bp_sortie/$50/$A1/sp_sortie/$50);
inline($A1/ax_sortie/$50);
inline($53/$51/$52/$56/$57/$A1/ds_sortie/$50);
inline($06);

inline($FB);

(* restauration des registres du processeur *)

inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58);
inline($5D/$8B/$E5/$5D);

(* retour au programme utilisateur interrompu *)

inline($CF);
end;


(*****************************************************************************)


procedure inter_00h;
(******************)
(* procédure appelée par le vecteur d'interruption numéro 00h du système
   d'exploitation *)

begin

(* sauvetage des registres du processeur *)

inline($50/$53/$51/$52/$56/$57/$1E/$06);
```

```
(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

(* mémorisation du numéro du vesteur appelé *)

num_vecteur := $00;

(* saut inconditionnel indirect vers la procédure "entrée" (ou "sortie directe") *)

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_01h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $01;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_02h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $02;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_03h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);
```

```
inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $03;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


{*****************************************************************************}


procedure inter_04h;
{******************}

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $04;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


{*****************************************************************************}


procedure inter_05h;
{******************}

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $05;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


{*****************************************************************************}


procedure inter_06h;
{******************}

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $06;
```

```
inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_07h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $07;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_0Ah;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $0A;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_0Bh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $0B;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)
```

```
procedure inter_0Ch;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $0C;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*******************************************************************************)


procedure inter_0Dh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $0D;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*******************************************************************************)


procedure inter_0Eh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $0E;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*******************************************************************************)


procedure inter_0Fh;
(******************)

begin
```

```
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $0F;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_10h;
(*******************)

begin

(* sauvetage des registres du processeur *)

inline($50/$53/$51/$52/$56/$57/$1E/$06);

(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

(* mémorisation du numéro du vecteur appelé *)

num_vecteur := $10;

(* dépositionnement d'un verrou si une routine de terminaison de programme
   ou de lancement de programme, figurant dans le système d'exploitation,
   vient d'être exécutée *)

if retour_chargement = true
    then begin
         if exclusion_clavier = 1
             then begin
                  ofs_entree := ofs(entree) + 4;
                  retour_chargement := false;
                  exclusion_ES := 1;
                  end;
         end;

(* saut inconditionnel indirect vers la procédure "entree" (ou "sortie directe") *)

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_11h;
(*******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);
```

```
inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $11;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_12h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $12;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_13h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $13;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_14h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $14;

inline($A1/ofs_entree);
```

```
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_15h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $15;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_16h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $16;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_17h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $17;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)
```

```
procedure inter_18h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $18;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*******************************************************************************)


procedure inter_19h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $19;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*******************************************************************************)


procedure inter_1Ah;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $1A;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*******************************************************************************)


procedure inter_1Bh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);
```

```
inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $1B;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_20h;
(*******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $20;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


var
     ax_21h, ds_21h : integer;   (* registres du processeur mémorisées dans la pile *)

     numero_fonction : integer;  (* N° de la fonction appelée par le programme
                                    utilisateur, faisant partie de l'interruption
                                    numéro 21h *)
     interrupt_21_25h : integer; (* adresse d'entrée de la procédure "inter_21_25h" *)

     changer_vecteur : boolean; (* flag indiquant si un vecteur d'interruption du
                                   système d'exploitation va être modifié par la
                                   fonction 25h de l'interruption 21h du système
                                   d'exploitation *)


procedure inter_21h;
(*******************)

begin

(* sauvetage de certains registres du processeur *)

inline($50/$1E);

(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

(* sauvetage des registres du processeur *)
```

```
inline($58/$A3/ds_21h/$58/$A3/ax_21h);
inline($50/$53/$51/$52/$56/$57/$A1/ds_21h/$50/$06);

(* identification de la fonction 25h de l'interruption 21h *)

regs.ax := ax_21h;
numero_fonction := regs.ah;
if (numero_fonction = $25) AND (changer_vecteur = false)
    then begin

         (* saut inconditionnel indirect vers la procédure "inter_21_25h" *)

         inline($A1/interrupt_21_25h);
         inline($FF/$E0);
         end
    else begin

         (* saut inconditionnel indirect vers la procédure "entree"
            (ou "sortie directe") *)

         num_vecteur := $21;
         inline($A1/ofs_entree);
         inline($FF/$E0);
         end;
end;


(*****************************************************************************)


(* variables de mémorisation des registres du processeur dans la procédure
   "inter_21_25h" du programme *)

var
    ds_21_25h, dx_21_25h, ax_21_25h,  (* registres du processeur suvés dans la pile
                                         lors de l'interruption *)

    num_vecteur_21_25h : integer;     (* N° du vecteur dont le programme
                                         utilisateur désire changer l'adresse *)

    ass_fct25,            (* ancien Stack Segment *)
    nss_fct25,            (* nouveau Stack Segment *)
    asp_fct25,            (* ancien Stack Pointer *)
    abp_fct25 : integer;  (* ancien Base Pointer *)


procedure restaurer_vecteur_interrupt(num_vect : integer);
(*-------------------------------------------------------*)
(* cette procédure permet de restaurer l'adresse des routines "inter i" du
   programme "aidami" dans les vecteurs d'interruption dont l'adresse a été
   modifiée par un logiciel "standard" *)

begin
if num_vect = $1C
     then begin
          numero_vecteur := num_vect;
          lire_vecteur_interrupt;
          if (ds_21_25h = segment_code) AND (dx_21_25h = offset_code)
              then change_1C := false
              else begin
                   numero_vecteur := inter_1C_detournee;
                   segment_code := DS_21_25h;
                   offset_code := DX_21_25h;
```

```
                ecrire_vecteur_interrupt;
                change_1C := true;
                end;
         end
    else begin
         numero_vecteur := num_vect;
         segment_code := ds_21_25h;
         offset_code := dx_21_25h;
         ecrire_vecteur_interrupt;
         end;
end;


(*****************************************************************************)


procedure inter_21_25h;
(**********************)

begin

(* mémorisation des paramètres nécessaires pour exécuter la fonction 25h de
   l'interruption 21h du système d'exploitation *)
(* ces paramètres sont fournis par le programme utilisateur dans les registres
   du processeur *)

inline($07);
inline($58/$A3/ds_21_25h/$5F/$5E/$58/$A3/dx_21_25h/$59/$5B);
inline($58/$A3/ax_21_25h);

inline($A1/ax_21_25h/$50);
inline($53/$51/$A1/dx_21_25h/$50/$56/$57/$A1/ds_21_25h/$50);
inline($06);

(* changement de pile *)

inline($16/$58/$A3/ass_fct25/$A1/nss_fct25/$50/$17);
inline($94/$A3/asp_fct25/$95/$A3/abp_fct25);
inline($B8/pile_fct25/$94/$B8/pile_fct25/$95);

changer_vecteur := true;
regs.ax := ax_21_25h;
num_vecteur_21_25h := regs.al;
restaurer_vecteur_interrupt(num_vecteur_21_25h);
changer_vecteur := false;

(* restauration de la pile *)

inline($A1/ass_fct25/$50/$17);
inline($A1/asp_fct25/$94/$A1/abp_fct25/$95);

(* restauration des registres du processeur *)

inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58);
inline($5D/$8B/$E5/$5D);
inline($CF);
end;


(*****************************************************************************)


procedure inter_25h;
```

```
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $25;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_26h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $26;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_27h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $27;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_28h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
```

```
inline($A1/data_seg/$50/$1F);

num_vecteur := $28;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_29h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $29;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_2Ah;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $2A;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_2Bh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $2B;

inline($A1/ofs_entree);
inline($FF/$E0);
```

```
end;

(*****************************************************************************)

procedure inter_2Ch;
(*******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $2C;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_2Dh;
(*******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $2D;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)

procedure inter_2Eh;
(*******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $2E;

inline($A1/ofs_entree);
inline($FF/$E0);
end;

(*****************************************************************************)
```

```
procedure inter_2Fh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $2F;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_30h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $30;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_31h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $31;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_32h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);



inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $34;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_35h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $35;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_36h;
```

```
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_37h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $37;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_38h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $38;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_39h;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $39;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)
```

```
procedure inter_3Ah;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $3A;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_3Bh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $3B;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_3Ch;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $3C;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_3Dh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);
```

```
inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $3D;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_3Eh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $3E;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)


procedure inter_3Fh;
(******************)

begin
inline($50/$53/$51/$52/$56/$57/$1E/$06);

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

num_vecteur := $3F;

inline($A1/ofs_entree);
inline($FF/$E0);
end;


(*****************************************************************************)
```

```
procedure num_der_elem_fich( var num_rec_final, erreur : integer; n_type_pteur : integer);
(*****************************************************************************************)
(* détermination du numéro du dernier enregistrement dans un fichier *)
(* "n_type_pteur" = N° du pointeur de la structure du fichier *)


procedure nbre_elem_fich_text(var nbre_elem : integer);
(*-----------------------------------------------*)
(* détermination du nombre de lignes dans un fichier de texte *)

var buffer_txt : strWrk;  (* buffer temporaire pour la lecture *)

begin
nbre_elem := -1;
reset(f_text);
while not(eof(f_text)) do
    begin
    readln(f_text,buffer_txt);
    nbre_elem := nbre_elem + 1;
    end;
end;



begin          (**** num_der_elem_fich ****)

case n_type_pteur of
    1 : num_rec_final := filesize(f_teleph) - 1;
    2 : num_rec_final := filesize(f_fen) - 1;
    3 : nbre_elem_fich_text(num_rec_final);
    else erreur := 4;
    end;
if num_rec_final < 0 then erreur := 23;
end;


(*****************************************************************************************)
(*****************************************************************************************)


procedure determ_rec_pteur( filename : strWrk; var n_type_pteur, erreur : integer);
(*****************************************************************************************)
(* TABLE DES FICHIERS du programme "AIDAMI". Détermination du numéro du
   pointeur d'une structure de fichier associé à un fichier défini dans
   la table des fichiers *)

var filename_court : strWrk;  (* abrégé du nom du fichier *)
    trouve : boolean;

begin
trouve := false;
filename_court := copy(filename,1,5);
if filename_court = 'telep' then begin                (* fichier du téléphone *)
                                 n_type_pteur := 1;
                                 trouve := true;
                                 end;
if filename_court = 'fenet' then begin                (* fichiers de fenêtres *)
                                 n_type_pteur := 2;
                                 trouve := true;
                                 end;
                                                      (* fichiers de texte *)
if (filename_court = 'texte') OR (filename_court = 'batch') then begin
                                                                n_type_pteur := 3;
```

```
                                                        trouve := true;
                                                        end;
if trouve = false then erreur := 2;
end;


(*****************************************************************************)
(*****************************************************************************)



procedure acces_seq( filename : strWrk; num_rec_init, num_rec_final : integer;
                     var n_type_pteur, erreur : integer);
(*****************************************************************************)
(* lecture d'un ou de plusieurs records dans un fichier *)
(* "num_rec_init" = N° du record initial à lire *)
(* "num_rec_final" = N° du record final à lire *)
(* "n_type_pteur" = N° de la structure du fichier *)

var num_record : integer;  (* N° du record en cours de lecture *)
    premier : boolean;     (* flag indiquant qu'on va lire le premier record *)


procedure memoriser_data( n_type_pteur : integer);
(*-------------------------------------------*)
(* mémorisation dans une liste chainée en mémoire centrale, des records lus
   dans un fichier *)
(* "n_type_pteur" = N° du pointeur de la structure du fichier *)

var i : integer;
    pteur_fenetre : obj_fenetre;  (* pointeur vers un record de fenêtre *)
    pteur_teleph : code_teleph;   (* pointeur vers un record de téléphone *)
    pteur_text : fich_text_buf;   (* pointeur vers un record de fichier de texte *)

begin
case n_type_pteur of
                1 : begin
                    new(pteur_teleph);
                    pteur_teleph^.tps_inter_chiffre := teleph.tps_inter_chiffre;
                    pteur_teleph^.tps_inter_impuls := teleph.tps_inter_impuls;
                    pteur_teleph^.tps_duree_impuls := teleph.tps_duree_impuls;
                    pteur_teleph^.port_sortie := teleph.port_sortie;
                    pteur_teleph^.num_bit_imp := teleph.num_bit_imp;
                    pteur_teleph^.next := ptr_teleph;
                    ptr_teleph := pteur_teleph;
                    end;
                2 : begin
                    new( pteur_fenetre);
                    pteur_fenetre^.num_objet := fenetre.num_objet;
                    pteur_fenetre^.valeur_string := fenetre.valeur_string;
                    for i := 1 to 4 do
                        begin
                        pteur_fenetre^.coord[i] := fenetre.coord[i];
                        end;
                    pteur_fenetre^.couleur_caractere := fenetre.couleur_caractere;
                    pteur_fenetre^.couleur_fond := fenetre.couleur_fond;
                    pteur_fenetre^.etat_bouton := fenetre.etat_bouton;
                    pteur_fenetre^.num_gr_bouton := fenetre.num_gr_bouton;
                    pteur_fenetre^.num_action := fenetre.num_action;
                    pteur_fenetre^.next := ptr_fenetre;
                    ptr_fenetre := pteur_fenetre;
                    end;
```

```
            3 : begin
                new(pteur_text);
                pteur_text^.tampon := buffer_text.tampon;
                if ptr_text = nil then prec_ptr_text := pteur_text
                                  else ptr_text^.prec := pteur_text;
                pteur_text^.next := ptr_text;
                ptr_text := pteur_text;
                end;
            end;
end;



begin         (**** acces_seq ****)

premier := true;
determ_rec_pteur( filename, n_type_pteur, erreur);
if erreur <> 2
   then begin
        ouvrir_fich(filename,n_type_pteur,1,erreur);
        if not(erreur in [1,4])
           then begin
                num_der_elem_fich( num_record, erreur, n_type_pteur);
                if erreur <> 23
                   then begin
                        if (num_rec_init <= num_record) and ( num_rec_final <= num_record)
                           then begin
                                if num_rec_final < 0 then num_rec_final := num_record;

                                (* lecture des records dans le fichier *)

                                for num_record := num_rec_init to num_rec_final do
                                    begin
                                    lire_fich(num_record, n_type_pteur, erreur);
                                    if not(erreur in [3,4])
                                       then begin
                                            if premier = true then begin
                                                                   premier := false;
                                                                   case n_type_pteur of
                                                                       1 : ptr_teleph := nil;
                                                                       2 : ptr_fenetre := nil;
                                                                       3 : ptr_text := nil;
                                                                       end;
                                                                   end;
                                            memoriser_data( n_type_pteur);
                                            end
                                       else erreur := 7;
                                    end;
                                if n_type_pteur = 3 then ptr_text^.prec := nil;
                                end
                           else erreur := 3;
                        end;
                fermer_fich(filename,n_type_pteur,erreur);
                end;
        end;
end;


(*****************************************************************************)
(*****************************************************************************)
```

```
procedure ecrire_fich_seq_text(filename : strWrk; num_rec_init, num_rec_fin : integer;
                               var erreur : integer);
(*****************************************************************************)
(* écriture d'une ou de plusieurs lignes dans un fichier de texte *)
(* "num_rec_init" = N° de la première ligne à écrire *)
(* "num_rec_fin" = N° de la dernière ligne à écrire *)

var i : integer;
    num_record : integer;      (* N° de la dernière ligne du fichier de texte *)
    erreur_entree : integer;   (* valeur de la variable "erreur" à l'entrée de la
                                  procédure *)
    n_type_pteur : integer;    (* N° du pointeur de la structure du fichier *)

begin
erreur_entree := erreur;
acces_seq(filename,0,-1,n_type_pteur,erreur);
if not(erreur in [1,2,4,7])
    then begin
         ouvrir_fich(filename,3,1,erreur);
         num_der_elem_fich(num_record, erreur, 3);
         fermer_fich(filename,3,erreur);
         erreur := erreur_entree;
         if (num_rec_init <= num_record + 1)
             then begin
                  vider_contenu_fich(filename,erreur);
                  ouvrir_fich(filename,3,0,erreur);

(* mémorisation dans le fichier des "num_rec_init - 1" premières lignes qui se
   trouvaient déjà dans le fichier et que l'on désire garder *)

                  for i := 0 to (num_rec_init - 1) do
                      begin
                      writeln(f_text,prec_ptr_text^.tampon);
                      prec_ptr_text := prec_ptr_text^.prec;
                      end;

(* mémorisation des nouvelles lignes mémorisées dans le tableau "tampon" *)

                  num_tampon := 1;
                  for i := num_rec_init to num_rec_fin do
                      begin
                      writeln(f_text,tampon[num_tampon]);
                      num_tampon := num_tampon + 1;
                      if prec_ptr_text <> nil
                          then prec_ptr_text := prec_ptr_text^.prec;
                      end;

(* mémorisation des éventuelles autres lignes lues précédemment dans le fichier *)

                  while (prec_ptr_text <> nil) do
                      begin
                      writeln(f_text, prec_ptr_text^.tampon);
                      prec_ptr_text := prec_ptr_text^.prec;
                      end;
                  fermer_fich(filename,3,erreur);
                  end
             else erreur := 3;
         end;
end;


(*****************************************************************************)
(*****************************************************************************)
```

```
procedure rech_ancien_code(port_s : integer; var ancien_code, indice_table_ports,
                           erreur : integer);
(*****************************************************************************)
(* Recherche de l'ancien code "ancien_code" qui est le dernier code envoyé vers
   le port de sortie adressé par "port_s" *)
(* "indice_table_port" = indice de la table des ports de sortie identifiant
   l'élément dans lequel se trouve l'ancien code *)

var trouve : boolean;
    i : integer;

begin
trouve := false;
i := 1;
while (trouve = false) AND (i <= nb_max_ports_sortie) do
    begin
    if table_port_sortie[i,1] = port_s
        then begin
             trouve := true;
             indice_table_ports := i;
             ancien_code := table_port_sortie[i,2];
             end
        else i := i + 1;
    end;
if trouve = false then erreur := 24;
end;



procedure envoyer_code_appareils(num_bit, port_s, delai, etat_bit : integer;
                                 var erreur : integer);
(*****************************************************************************)
(* envoi d'un signal vers un port de sortie extérieur *)
(* "num_bit" = N° du bit sur lequel le signal est émis *)
(* "port_s" = adresse du port de sortie *)
(* "delai" = temps d'attente pendant l'envoi du signal *)
(* "etat_bit" = niveau logique binaire du signal à envoyer *)

var code : integer;               (* code intermédiaire *)
    nouveau_code : integer;
    indice_table_ports : integer;
    ancien_code : integer;

begin

(* détermination du "masque" du code à envoyer vers le port de sortie *)

if etat_bit = 0
    then begin
         case num_bit of
             0 : code := $FE;
             1 : code := $FD;
             2 : code := $FB;
             3 : code := $F7;
             4 : code := $EF;
             5 : code := $DF;
             6 : code := $BF;
             7 : code := $7F;
             end;
         end
    else begin
         case num_bit of
             0 : code := $01;
```

```
            1 : code := $02;
            2 : code := $04;
            3 : code := $08;
            4 : code := $10;
            5 : code := $20;
            6 : code := $40;
            7 : code := $80;
            end;
        end;

rech_ancien_code(port_s,ancien_code,indice_table_ports,erreur);
if erreur <> 24
    then begin
        case etat_bit of
            0 : nouveau_code := (ancien_code) AND (code);
            1 : nouveau_code := (ancien_code) OR (code);
            end;

        (* envoi du code vers le port de sortie *)

        port[port_s] := nouveau_code;

        table_port_sortie[indice_table_ports,2] := nouveau_code;
        if etat_bit = 1 then son(200,delai)
                        else delay(delai);
        end;
end;
```

```
procedure dessiner_fond_fenetre( xh, yh, xl, yl : integer);
(*************************************************************)
(* "met à blanc" une zone de l'écran pour y afficher une fenêtre *)

var i, j : integer;

begin
for j := yh to (yl - 1) do
    begin
    for i := xh to (xl - 1) do
       begin
       gotoxy(i,j);
       write(' ');
       end;
    end;
end;



procedure dessiner_bord_fenetre(xh, yh, xl, yl : integer);
(*************************************************************)
(* affichage du bord d'une fenêtre *)

var i : integer;

begin
for i := (xh + 1) to (xl - 1) do
   begin
   gotoxy(i,yh);
   write('-');
   end;
for i := (yh + 1) to (yl - 1) do
   begin
   gotoxy(xh,i);
   write('|');
   end;
for i := (yh + 1) to (yl - 1) do
   begin
   gotoxy(xl,i);
   write('|');
   end;
for i := (xh + 1) to (xl - 1) do
   begin
   gotoxy(i,yl);
   write('-');
   end;
gotoxy(xh,yh);
write('┌');
gotoxy(xl,yh);
write('┐');
gotoxy(xh,yl);
write('└');
gotoxy(xl,yl);
write('┘');
end;


(*****************************************************************************)
(*****************************************************************************)


procedure affiche_fen( num_fen, xh, yh, xl, yl, type_fen, couleur_fond,
```

```
                    couleur_caractere : integer; header : strWrk);
(*************************************************************************)
(* affichage d'une fenêtre d'un certain type à l'écran en tenant compte du mode
   de l'écran *)
(* "num_fen" = N° de la fenêtre *)
(* "type_fen" = type de la fenêtre *)
(* "header" = entête de la fenêtre *)

begin
case mode_ecran of
    1,4,5,6 : begin
              defineWindow( num_fen, xh, yh, xl, yl);

              (* affichage de l'entête de la fenêtre *)

              if type_fen = 2 then begin
                                   DefineHeader(num_fen, header);
                                   SetHeaderOn;
                                   end;

              defineWorld( num_fen, 0, 0, x_max_world, y_max_world);
              SelectWorld(num_fen);
              SelectWindow(num_fen);
              SetBackGround(couleur_fond);

              (* affichage du bord de la fenêtre *)

              if (type_fen in [1,2]) then DrawBorder;
              end;
    3     :   begin
              yh := trunc(yh / 8) + 1;
              yl := trunc(yl / 8) + 1;
              xh := xh + 1;
              xl := xl + 1;
              dessiner_fond_fenetre(xh,yh,xl,yl);
              if type_fen = 1 then dessiner_bord_fenetre(xh,yh,xl,yl);
              end;
    end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure rech_emplace_string(num_string, num_fenetre : integer; var x, y,
                              couleur_caractere, echelle, erreur : integer);
(*************************************************************************)
(* Recherche d'un emplacement réservé pour une chaine de caractères, dans la
   fenêtre numéro "num_fenetre" *)
(* "num_string" = N° de l'emplacement *)
(* "x, y" = coordonnées de l'emplacement *)

var trouve : boolean;
    ptr_obj_ecran : obj_ecran;

begin
trouve := false;
ptr_obj_ecran := table_convers[num_fenetre].obj_ecran_ptr;
while (trouve = false) AND (ptr_obj_ecran <> nil) do
    begin
    if ptr_obj_ecran^.type_objet = 3
        then begin
```

```
            if ptr_obj_ecran^.num_gr_objet = num_string
               then begin
                    trouve := true;
                    x := ptr_obj_ecran^.coord_objet[1];
                    y := ptr_obj_ecran^.coord_objet[2];
                    echelle := ptr_obj_ecran^.coord_objet[3];
                    couleur_caractere := couleur_string_affiche;
                    end;
            end;
    ptr_obj_ecran := ptr_obj_ecran^.next;
    end;
if trouve = false then erreur := 19;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure affiche_bouton( num_gr_bouton, etat_bouton, couleur_fond, couleur_caractere,
                          xh, yh, xl, yl,num_fenetre, type_objet : integer);
(*****************************************************************************************)
(* affichage d'un bouton à l'écran *)
(* "num_fenetre" = N° de la fenêtre *)
(* "type_objet" = type de l'objet *)

begin
if mode_ecran in [1,4,5,6]
    then begin
         if type_objet in [1,5]
            then DrawSquare( xh, y_max_world - yl, xl, y_max_world - yh, false);
         end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure affiche_carac( x, y, couleur_caractere, echelle, num_fenetre :
                         integer; chaine_carac : strWrk);
(*************************************************************************)
(* affichage d'une chaîne de caractères à l'écran *)
(* "chaine_carac" = chaine de caractères à afficher *)
(* "x, y" = coordonnées du premier caractère de la chaine *)

begin
gotoxy(x,y);
write(chaine_carac);
end;


(*****************************************************************************)
(*****************************************************************************)


procedure global_to_local(var x, y, erreur : integer; num_fenetre : integer);
(*************************************************************************)
(* transformation de coordonnées globales relatives à tout l'écran en
   coordonnées locales relatives à une fenêtre donnée *)
(* "x, y" = coordonnées globales/locales du curseur de l'écran *)

var xh, yh, xl, yl : integer;  (* coordonnées de la fenêtre *)
```

```
begin
xh := table_fenetre[num_fenetre,1];
yh := table_fenetre[num_fenetre,2];
xl := table_fenetre[num_fenetre,3];
yl := table_fenetre[num_fenetre,4];
if (xh < 0) or (yh < 0) or (xl < 0) or (yl < 0)
    then erreur := 13
    else begin

         (* transformation des coordonnées *)

         x := trunc(x_max_world - (x_max_world * ((xl - x) / (xl - xh))));
         y := trunc(y_max_world - (x_max_world * ((yl - y) / (yl - yh))));
         end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure maj_table_conv( num_fenetre : integer);
(*********************************************)
(* activation de la fenêtre numéro "num_fenetre" *)

var i : integer;

begin
for i := 1 to nb_max_fenetres do
    begin
    table_convers[i].actif := false;
    end;
table_convers[num_fenetre].actif := true;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure inserer_table_conv( type_dobjet, xh, yh, xl, yl, numero_action,
                              num_gr_bouton, num_fenetre : integer);
(*************************************************************************)
(* ajout d'un objet dans une liste chainée de la table de conversion *)
(* "num_gr_bouton" = N° du groupe du bouton *)

var ptr_obj_ecran : obj_ecran;  (* pointeur vers la structure de données du
                                   nouvel objet à insérer dans la table de
                                   conversion *)

begin
         new(ptr_obj_ecran);
         ptr_obj_ecran^.type_objet := type_dobjet;
         ptr_obj_ecran^.coord_objet[1] := xh;
         ptr_obj_ecran^.coord_objet[2] := yh;
         ptr_obj_ecran^.coord_objet[3] := xl;
         ptr_obj_ecran^.coord_objet[4] := yl;
         ptr_obj_ecran^.num_gr_objet := num_gr_bouton;
         ptr_obj_ecran^.num_action := numero_action;
         ptr_obj_ecran^.next := table_convers[num_fenetre].obj_ecran_ptr;
         table_convers[num_fenetre].obj_ecran_ptr := ptr_obj_ecran;
end;
```

```
(*****************************************************************************)
(*****************************************************************************)


procedure supprim_table_conv( num_fenetre : integer);
(*************************************************)
(* suppression des objets relatifs à une fenêtre, de la table de conversion *)

var pteur_obj_ecran : obj_ecran; (* pointeur vers les objets à supprimer *)
    ptr_obj_ecran : obj_ecran;   (* pointeur vers les objets à supprimer *)

begin
ptr_obj_ecran := table_convers[num_fenetre].obj_ecran_ptr;
while ptr_obj_ecran <> nil do
    begin
    pteur_obj_ecran := ptr_obj_ecran^.next;
    dispose(ptr_obj_ecran);
    ptr_obj_ecran := pteur_obj_ecran;
    end;
table_convers[num_fenetre].obj_ecran_ptr := nil;
table_convers[num_fenetre].actif := false;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure determiner_action( x, y : integer; var num_fenetre, type_objet,
                             num_gr_objet, num_action, erreur : integer);
(*************************************************************************)
(* "x, y" = coordonnées du curseur *)
(* "num_fenetre" = N° de la fenêtre dans laquelle se trouve le curseur *)
(* "type_objet" = type de l'objet pointé par le curseur *)
(* "num_gr_objet" = N° du groupe de l'objet pointé *)
(* "num_action" = N° de l'action à exécuter *)

var pteur_obj_ecran : obj_ecran; (* pointeur vers la liste chainée contenant les
                                    objets de la fenêtre pointée *)
    identique : boolean;         (* = true si le curseur de l'écran est à l'intérieur
                                    d'un objet de la fenêtre *)
    trouve : boolean;


procedure determ_num_fenetre_coord(var num_fenetre, erreur : integer;
                                   x, y : integer);
(*-----------------------------------------------------------------*)
(* détermination du numéro de la fenêtre dans laquelle le curseur de l'écran
   est localisé *)
(* "num_fenetre" = N° de la fenêtre *)

var i,
    xl, yl, xh, yh : integer;  (* coordonnées de la fenêtre *)
    trouve : boolean;

begin
trouve := false;
i := nb_max_fenetres;
while (trouve = false) and (i >= 1) do
    begin
    if table_ordre_fen[i] <> 0
```

```
        then begin
             num_fenetre := table_ordre_fen[i];
             xh := table_fenetre[ num_fenetre,1];
             yh := table_fenetre[ num_fenetre,2];
             xl := table_fenetre[ num_fenetre,3];
             yl := table_fenetre[ num_fenetre,4];
             if (x >= xh) and (x <= xl) and (y >= yh) and (y <= yl)
                then trouve := true;
             end;
    i := i - 1;
    end;
if trouve = false then erreur := 5;
end;


procedure comparer_coord( pteur_obj_ecran : obj_ecran; x, y : integer;
                          var identique : boolean);
(*-------------------------------------------------------------------*)
(* comparaison des coordonnées du curseur de l'écran avec les coordonnées
   d'un objet affiché dans une fenêtre à l'écran, pour voir si le curseur
   est à l'intérieur de cet objet. La structure de données de l'objet est
   pointée par le pointeur "pteur_obj_ecran" *)
(* "identique" = true si le curseur de l'écran est à l'intérieur de l'objet *)

begin
identique := false;
case pteur_obj_ecran^.type_objet of
     1,5 : begin
           if (x >= pteur_obj_ecran^.coord_objet[1]) and
              (x <= pteur_obj_ecran^.coord_objet[3]) and
              (y >= pteur_obj_ecran^.coord_objet[2]) and
              (y <= pteur_obj_ecran^.coord_objet[4])
              then identique := true;
           end;
    end;
end;




begin                           (**** determiner_action ****)

determ_num_fenetre_coord( num_fenetre, erreur, x, y);
if erreur <> 5
    then begin
         if table_convers[num_fenetre].actif = true
             then begin
                  global_to_local(x, y, erreur, num_fenetre);
                  pteur_obj_ecran := table_convers[num_fenetre].obj_ecran_ptr;
                  trouve := false;
                  while (pteur_obj_ecran <> nil) AND (trouve = false) do
                      begin
                      comparer_coord( pteur_obj_ecran, x, y, identique);
                      if identique = true
                          then begin
                               type_objet := pteur_obj_ecran^.type_objet;
                               num_gr_objet := pteur_obj_ecran^.num_gr_objet;
                               num_action := pteur_obj_ecran^.num_action;
                               trouve := true;
                               end
                          else pteur_obj_ecran := pteur_obj_ecran^.next;
                      end;
                  if trouve = false then erreur := 12;
```

```
                end
            else erreur := 8;
        end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure rech_entree_tableau(num_fenetre : integer; var entree,
                              erreur : integer);
(*-----------------------------------------------------------*)
(* recherche d'une valeur d'indice de la table de conversion. Cet indice
   identifie un élément de la table dans lequel le numéro de fenêtre
   "num_fenetre" est mémorisé *)

var trouve : boolean;

begin
entree := 1;
trouve := false;
while (trouve = false) and (entree <= nb_max_fenetres) do
    begin
    if table_ordre_fen[entree] = num_fenetre then trouve := true
                                             else entree := entree + 1;
    end;
if trouve = false then erreur := 6;
end;



procedure supprim_fen_ordre( num_fenetre : integer; var erreur : integer);
(*************************************************************************)
(* suppression du numéro de fenêtre "num_fenetre" de la table d'ordonnancement *)

var j,
    entree : integer; (* indice de la table identifiant l'élément dans lequel
                         le numéro de fenêtre "num_fenetre" est mémorisé *)

begin
rech_entree_tableau( num_fenetre, entree, erreur);
if erreur <> 6
    then begin
         for j := entree to (nb_max_fenetres - 1) do
           begin
           table_ordre_fen[j] := table_ordre_fen[j+1];
           end;
         table_ordre_fen[nb_max_fenetres] := 0;
         end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure ajout_fen_ordre( num_fenetre : integer; var erreur : integer);
(*************************************************************************)
(* ajout du numéro "num_fenetre" dans la table d'ordonnancement *)

var entree : integer;       (* indice du premier élément nul en partant de la
```

```
                                gauche, dans la table de conversion *)
    erreur_verif : integer;

begin
rech_entree_tableau( num_fenetre, entree, erreur_verif);
if erreur_verif = 6
    then begin
         rech_entree_tableau( 0, entree, erreur);
         if erreur <> 6 then table_ordre_fen[entree] := num_fenetre;
         end
    else erreur := 9;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure rech_der_table_ordre(var num_fenetre, erreur : integer);
(*****************************************************************)
(* Recherche du numéro de la dernière fenêtre affichée à l'écran *)

var trouve : boolean;
    i : integer;

begin
num_fenetre := 0;
trouve := false;
i := nb_max_fenetres;
while (trouve = false) and (i >= 1) do
    begin
    if table_ordre_fen[i] <> 0
        then begin
             num_fenetre := table_ordre_fen[i];
             trouve := true;
             end
        else i := i - 1;
    end;
if trouve = false then erreur := 16;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure efface_fenetre_mc(ptr_obj_fen : obj_fenetre);
(*****************************************************)
(* efface de la mémoire centrale une liste chainée contenant la description des
   objets d'une fenêtre et de la fenêtre elle-même *)
(* 'ptr_obj_fen" = pointeur vers cette liste chainée *)

var pter_fenetre : obj_fenetre;

begin
while ptr_obj_fen <> nil do
    begin
    pter_fenetre := ptr_obj_fen^.next;
    dispose(ptr_obj_fen);
    ptr_obj_fen := pter_fenetre;
    end;
end;
```

```
(*****************************************************************************)
(*****************************************************************************)


procedure inserer_table_fen_mc( pointeur_fenetre : obj_fenetre; filename : strWrk);
(*****************************************************************************)
(* Insertion d'une nouvelle fenêtre dans la table des fenêtres présentes en
   mémoire centrale *)
(* "pointeur_fenetre" = pointeur vers la liste chainée contenant la description
   des objets de la fenêtre, et de la fenêtre elle-même *)
(* "filename" = nom du fichier dans lequel est décrite la fenêtre *)

var pter_fenetre : obj_fenetre; (* pointeur vers la liste chainée d'une fenêtre,
                                   à supprimer de la table des fenêtres présentes
                                   en mémoire centrale *)

begin
pter_fenetre := table_fen_mc_ptr[ptr_table_fen_mc];
table_fen_mc_nom[ptr_table_fen_mc] := filename;
table_fen_mc_ptr[ptr_table_fen_mc] := pointeur_fenetre;

(* mise à jour du pointeur vers les éléments de la table des fenêtres présentes
   en mémoire centrale *)

if ptr_table_fen_mc < n_max_fen_mc
   then ptr_table_fen_mc := ptr_table_fen_mc + 1
   else ptr_table_fen_mc := 1;
efface_fenetre_mc(pter_fenetre);
end;


(*****************************************************************************)
(*****************************************************************************)


procedure determ_num_fen( var num_fenetre, erreur : integer);
(*********************************************************)
(* Détermination d'un nouveau numéro de fenêtre *)

var i, j : integer;
    trouve,                 (* flag indiquant que le numéro envisagé existe
                               déjà dans le système *)
    trouve_numero : boolean; (* flag indiquant que l'on a trouvé un nouveau
                               numéro *)

begin
i := 1;
j := 1;
trouve := true;
trouve_numero := false;
while (j <= nb_max_fenetres) and (trouve_numero = false) do
    begin
    while (trouve = true) and (i <= nb_max_fenetres) do
        begin
        if table_ordre_fen[i] = j then trouve := false
                                  else i := i + 1;
        end;
    if trouve = true
        then begin
             trouve_numero := true;
             num_fenetre := j;
             end
```

```
        else begin
             j := j + 1;
             i := 1;
             trouve := true;
             end;
    end;
if trouve_numero = false then erreur := 10;

end;


(*****************************************************************************)
(*****************************************************************************)


procedure maj_table_fenetre( num_fenetre, xh, yh, xl, yl : integer);
(*********************************************************************)
(* Mise à jour de la table des fenêtres *)

begin
table_fenetre[num_fenetre,1] := xh;
table_fenetre[num_fenetre,2] := yh;
table_fenetre[num_fenetre,3] := xl;
table_fenetre[num_fenetre,4] := yl;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure sauver_ecran_disque;
(*****************************)
(* Sauve le contenu de l'écran sur le disque *)

var num_fenetre : integer;  (* N° de la dernière fenêtre affichée à l'écran *)
    num_fen_string : str3;  (* N° de la dernière fenêtre affichée à l'écran *)
    nom_fich : StrWrk;      (* nom du fichier dans lequel l'écran est sauvé *)

begin
rech_der_table_ordre(num_fenetre, erreur);
if erreur <> 16
    then begin
         str(num_fenetre, num_fen_string);
         nom_fich := 'window' + num_fen_string;
         saveScreen(nom_fich);
         end;
end;


(*****************************************************************************)
(*****************************************************************************)
```

```
procedure lecture_string_fenetre(var chaine_carac : strWrk; caract : char;
                couleur_caractere, echelle, num_fenetre, x, y : integer);
(*************************************************************************)
(* Lecture d'une chaine de caractères dans une fenêtre *)
(* "chaine_carac" = chaine de caractères à lire *)
(* "caract" = caractère à ajouter en fin de chaine et à afficher dans la fenêtre *)
(* "echelle" = hauteur des caractères à afficher *)
(* "num_fenetre" = N° de la fenêtre *)
(* "x, y" = coordonnées de l'emplacement de la fenêtre *)

begin
if caract <> bs
    then chaine_carac := chaine_carac + caract
    else begin
         chaine_carac := copy(chaine_carac, 1, length(chaine_carac) - 1);
         affiche_carac(x + length(chaine_carac),y,couleur_caractere,echelle,
                       num_fenetre,' ');
         end;
affiche_carac(x, y, couleur_caractere, echelle, num_fenetre, chaine_carac);
end;


(*****************************************************************************)


procedure lect_bout_souris(var actif_bouton : boolean);
(*****************************************************)
(* lecture de l'état du bouton de la souris *)

begin
regs.ax := 3;
intr(num_vecteur_souris,regs);
if regs.bx in [1,2,3] then actif_bouton := true
                      else actif_bouton := false;
end;


(*****************************************************************************)


procedure lect_nbre_appuis_bout(var nbre_appuis_bout : integer);
(*************************************************************)
(* détermine le nombre d'appuis successifs sur le bouton de la souris entre deux
   appels à cette focntion *)

begin
compte_appuis_bout := 0;
regs.ax := 5;
regs.bx := 3;
intr(num_vecteur_souris,regs);
nbre_appuis_bout := regs.bx;
end;


(*****************************************************************************)


procedure lect_nbre_relaches_bout(var nbre_relaches_bout : integer);
(*****************************************************************)
(* détermine le nombre de relâches successives du bouton de la souris entre deux
   appels à cette focntion *)

begin
```

```
compte_relaches_bout := 0;
regs.ax := 6;
regs.bx := 3;
intr(num_vecteur_souris,regs);
nbre_relaches_bout := regs.bx;
end;


(*****************************************************************************)


procedure lect_pos_souris(var x,y : integer);
(*****************************************)
(* lecture de la position du curseur à l'écran *)

begin
regs.ax := 3;
intr(num_vecteur_souris,regs);
x := regs.cx;
y := regs.dx;
if x < 0 then x := 0;
if y < 0 then y := 0;
if x > 639 then x := 639;
if y > 199 then y := 199;
end;


(*****************************************************************************)
```

```
procedure affiche_fenetre( filename : strWrk; var erreur : integer);
(*****************************************************************)
(* Affichage d'une fenêtre, identifiée par son nom de fichier, et de son
   contenu à l'écran *)

var pteur_fenetre : obj_fenetre;  (* pointeur vers la liste chainée contenant la
                                     description de la fenêtre et de son contenu *)
    num_fenetre : integer;        (* N° de la fenêtre à afficher *)



procedure chercher_pteur_fen( var pteur_fenetre : obj_fenetre; filename : strWrk;
                              var erreur : integer);
(*-----------------------------------------------------------------------------*)
(* Recherche, dans la table des fenêtres présentes en mémoire centrale, du
   pointeur "pteur_fenetre" de la fenêtre identifiée par son nom de fichier
   "filename". *)
(* Ce pointeur pointe vers la liste chainée contenant la desription des éléments
   de cette fenêtre, et de la fenêtre elle-même. Si ce pointeur n'existe pas,
   le fichier "filename" est chargé en mémoire centrale et une liste chainée est
   créée *)

var trouve : boolean;
    i : integer;
    n_type_pteur : integer;    (* N° du pointeur de la structure du fichier
                                  "filename" *)

begin
trouve := false;
i := n_max_fen_mc;
while (trouve = false) and (i >= 1) do
    begin
    if table_fen_mc_nom[i] = filename
        then begin
             trouve := true;
             pteur_fenetre := table_fen_mc_ptr[i];
             end
        else i := i - 1;
    end;
if trouve = false
    then begin
         acces_seq( filename, 0, -1, n_type_pteur, erreur);
         pteur_fenetre := ptr_fenetre;
         if not(erreur in [1,2,3,4,7,23]) then inserer_table_fen_mc( pteur_fenetre, filename);
         end;
end;



procedure afficher_objet_fenetre(pteur_fenetre : obj_fenetre; num_fenetre : integer);
(*-----------------------------------------------------------------------------*)
(* affichage d'un objet d'une fenêtre à l'écran *)
(* "pteur_fenetre" = pointeur vers cet objet *)
(* "num_fenetre" = N° de la fenêtre à afficher *)

var type_dobjet,                  (* type de l'objet à afficher *)
    xl, yl, xh, yh,               (* coordonnées de l'objet *)
    num_action,                   (* N° de l'action associée à l'objet à afficher *)
    couleur_caractere,
    echelle,                      (* hauteur des caractères à afficher *)
    num_gr_bouton,                (* N° du groupe du bouton à afficher *)
    type_fen : integer;           (* type de la fenêtre à afficher *)
    chaine_carac : strWrk;        (* chaine de caractères à afficher *)
```

```
    couleur_fond : integer;
    etat_bouton : integer;        (* état du bouton à afficher *)
    header : strWrk;              (* entête de la fenêtre à afficher *)

begin
type_dobjet := pteur_fenetre^.num_objet;
case type_dobjet of
   1, 5  : begin
           xh := pteur_fenetre^.coord[1];
           yh := pteur_fenetre^.coord[2];
           xl := pteur_fenetre^.coord[3];
           yl := pteur_fenetre^.coord[4];
           num_action := pteur_fenetre^.num_action;
           num_gr_bouton := pteur_fenetre^.num_gr_bouton;
           etat_bouton := pteur_fenetre^.etat_bouton;
           couleur_caractere := pteur_fenetre^.couleur_caractere;
           couleur_fond := pteur_fenetre^.couleur_fond;
           inserer_table_conv( type_dobjet, xh, yh, xl, yl, num_action,
                               num_gr_bouton, num_fenetre);
           affiche_bouton( num_gr_bouton, etat_bouton, couleur_fond, couleur_caractere,
                           xh, yh, xl, yl, num_fenetre, type_dobjet);
           end;
   2 : begin
       xh := pteur_fenetre^.coord[1];
       yh := pteur_fenetre^.coord[2];
       echelle := pteur_fenetre^.coord[3];
       couleur_caractere := pteur_fenetre^.couleur_caractere;
       chaine_carac := pteur_fenetre^.valeur_string;
       affiche_carac( xh, yh, couleur_caractere, echelle, num_fenetre, chaine_carac);
       end;
   3 : begin
       xh := pteur_fenetre^.coord[1];
       yh := pteur_fenetre^.coord[2];
       num_gr_bouton := pteur_fenetre^.num_gr_bouton;
       inserer_table_conv(type_dobjet, xh,yh,0,0,0,num_gr_bouton,num_fenetre);
       end;
   0 : begin
       xh := pteur_fenetre^.coord[1];
       yh := pteur_fenetre^.coord[2];
       xl := pteur_fenetre^.coord[3];
       yl := pteur_fenetre^.coord[4];
       maj_table_fenetre( num_fenetre, xh, yh, xl, yl);
       type_fen := pteur_fenetre^.etat_bouton;
       couleur_fond := pteur_fenetre^.couleur_fond;
       couleur_caractere := pteur_fenetre^.couleur_caractere;
       header := pteur_fenetre^.valeur_string;
       affiche_fen( num_fenetre, xh, yh, xl, yl, type_fen, couleur_fond,
                    couleur_caractere, header);
       end;
   end; (* endcase *)
end;




begin                  (**** affiche_fenetre ****)

determ_num_fen(num_fenetre, erreur);
if erreur <> 10
    then begin
         if filename = 'feneti' then num_fenetre_menu := num_fenetre;
         chercher_pteur_fen( pteur_fenetre, filename, erreur);
```

```
        if not (erreur in [1, 2, 3, 4, 7, 23])
           then begin
                if num_fenetre in [1..nb_max_fenetres]
                   then begin
                        maj_table_conv( num_fenetre);
                        ajout_fen_ordre( num_fenetre, erreur);
                        if not (erreur in [6, 9])
                           then begin
                                while pteur_fenetre <> nil do
                                   begin
                                   afficher_objet_fenetre(pteur_fenetre, num_fenetre);
                                   pteur_fenetre := pteur_fenetre^.next;
                                   end;
                                end;
                        end
                   else erreur := 11;
                end;
        end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure efface_fenetre(type_fenetre : integer; var erreur : integer);
(*************************************************************************)
(* Effacement de la dernière fenêtre affichée à l'écran *)
(* "type_fenetre" = type de fenêtre à effacer. On distingue les types:
   "1" : pour une fenêtre ne nécessitant pas de sauvetage de l'écran sur
         disque, et qui doit donc être effacée de l'écran dès que l'utilisateur
         sélectionne un objet quelconque de l'écran avec la souris. (c'est le
         cas des fenêtres de menus).
   "2" : pour une fenêtre nécessitant un sauvetage de l'écran sur disque *)

var num_fenetre : integer;  (* N° de la fenêtre à effacer *)


procedure effacer_fenetre_disque(num_fenetre : integer; var erreur : integer);
(*-----------------------------------------------------------------------*)
(* Efface une fenêtre affichée à l'écran, en chargeant le contenu de l'écran
   qui a été mémorisé sur disque avant l'affichage de cette fenêtre *)

var num_fen_string : str3;  (* N° de la fenêtre *)
    nom_fich : StrWrk;      (* nom du fichier désiré *)
    exist : boolean;        (* flag indiquant si le fichier désiré existe *)

begin
str(num_fenetre, num_fen_string);
nom_fich := 'window' + num_fen_string;
exist_fich(nom_fich, exist);
if exist = true then loadScreen(nom_fich)
                else erreur := 1;
end;




begin            (**** efface_fenetre ****)

rech_der_table_ordre(num_fenetre,erreur);
if erreur <> 16
```

```
    then begin
         supprim_fen_ordre(num_fenetre,erreur);
         supprim_table_conv(num_fenetre);
         maj_table_fenetre(num_fenetre,-1,-1,-1,-1);
         rech_der_table_ordre(num_fenetre,erreur);
         if erreur <> 16 then maj_table_conv(num_fenetre);
         case type_fenetre of
             1 : begin
                 SwapScreen;
                 SelectScreen(1);
                 end;
             2 : effacer_fenetre_disque(num_fenetre,erreur);
             end;
         end;
end;


(*****************************************************************************)
(*****************************************************************************)


procedure deplacer_curseur( x,y : integer);
(****************************************)
(* modification des coordonnées du curseur de l'écran *)

begin
regs.ax := 4;
regs.cx := x;
regs.dx := y;
intr(num_vecteur_souris,regs);
end;


(*****************************************************************************)
(*****************************************************************************)


procedure montrer_curseur;
(************************)
(* rend le curseur de l'écran visible *)

begin
regs.ax := 1;
intr(num_vecteur_souris,regs);
end;


(*****************************************************************************)
(*****************************************************************************)


procedure cacher_curseur;
(***********************)
(* rend le curseur de l'écran invisible *)

begin
regs.ax := 2;
intr(num_vecteur_souris,regs);
end;


(*****************************************************************************)
(*****************************************************************************)
```

```
procedure texte_curseur;
(**********************)
(* active le curseur "hardware" du logiciel de la souris, pour le mode texte *)

begin
regs.ax := 10;
regs.bx := 1;
regs.cx := 0;
regs.dx := 7;
intr(num_vecteur_souris,regs);
end;


(*****************************************************************************)
(*****************************************************************************)
```

```
procedure preparer_code_appareils(code, n_type_pteur : integer; var erreur : integer);
(*********************************************************************************)
(* Préparation des codes à envoyer aux appareils électriques extérieurs *)
(* "code" = code à envoyer après mise en forme *)
(* "n_type_pteur" = N° du pointeur de la structure du fichier *)

var tps_chif,    (* temps entre l'envoi de deux chiffres d'un même N° de téléphone *)
    tps_imp,     (* temps entre deux impulsions d'un même chiffre de téléphone *)
    tps_duree,   (* temps de durée d'une impulsion pour le téléphone *)
    port_s,      (* adresse du port de sortie *)
    num_bit,     (* N° du bit sur lequel le signal est envoyé *)
    i : integer;

begin
case n_type_pteur of

    (* envoi de signaux vers le "mains-libres" *)

    1 : begin
        if code in [0,1,2,3,4,5,6,7,8,9]
           then begin
                tps_chif := pteur_telephone^.tps_inter_chiffre;
                tps_imp := pteur_telephone^.tps_inter_impuls;
                tps_duree := pteur_telephone^.tps_duree_impuls;
                end;
        port_s := pteur_telephone^.port_sortie;
        num_bit := pteur_telephone^.num_bit_imp;
        case code of
            0,1,2,3,4,5,6,7,8,9 : begin
                                  if code = 0 then code := 10;
                                  for i := 1 to code do
                                      begin
                                      envoyer_code_appareils(num_bit, port_s, tps_duree, 1, erreur);
                                      envoyer_code_appareils(num_bit, port_s, tps_imp, 0, erreur);
                                      end;
                                  envoyer_code_appareils(num_bit, port_s, tps_chif - tps_imp, 0, erreur);
                                  end;
            10 : envoyer_code_appareils(num_bit, port_s, 10, 0, erreur);
            11 : envoyer_code_appareils(num_bit, port_s, 10, 1, erreur);
            else erreur := 22;
            end;
        end;
    else erreur := 4;
    end;(* end case *)
end;
```

```
procedure bouger_curseur_ecran(action_souris : integer);
(*****************************************************)
(* déplacement du curseur visible à l'écran, à un certain endroit de l'écran *)

var x, y : integer;  (* coordonnées du curseur à l'écran *)

begin
cacher_curseur;
lect_pos_souris(x,y);
case action_souris of
    72 : y := y - deplace_vertical_curseur;
    75 : x := x - deplace_horizontal_curseur;
    77 : x := x + deplace_horizontal_curseur;
    80 : y := y + deplace_vertical_curseur;
    end;
deplacer_curseur(x,y);
montrer_curseur;
end;


(*****************************************************************************)


procedure activer_bouton_souris;
(*******************************)
(* active le bouton de la souris et met à jour les variables du logiciel de la
   souris *)

begin
if compte_appuis_bout = MAXINT then compte_appuis_bout := 30;
compte_appuis_bout := compte_appuis_bout + 1;
memw[seg_souris:$0118] := 3;
memw[seg_souris:$0132] := compte_appuis_bout;
regs.ax := 3;
intr(num_vecteur_souris,regs);
memw[seg_souris:$0134] := regs.cx;
memw[seg_souris:$0136] := regs.dx;
memw[seg_souris:$011A] := 1;
memw[seg_souris:$011C] := regs.cx;
memw[seg_souris:$011E] := regs.dx;
end;


(*****************************************************************************)


procedure desactiver_bouton_souris;
(*********************************)
(* désactive le bouton de la souris et met à jour les variables du logiciel
   de la souris *)

begin
if compte_relaches_bout = MAXINT then compte_relaches_bout := 30;
compte_relaches_bout := compte_relaches_bout + 1;
memw[seg_souris:$0118] := 0;
memw[seg_souris:$0138] := compte_relaches_bout;
regs.ax := 3;
intr(num_vecteur_souris,regs);
memw[seg_souris:$013A] := regs.cx;
memw[seg_souris:$013C] := regs.dx;
memw[seg_souris:$0120] := 1;
memw[seg_souris:$0122] := regs.cx;
memw[seg_souris:$0124] := regs.dx;
```

```
end;


(*****************************************************************************)


procedure activer_souris(action_souris : integer);
(***********************************************)
(* effectue une action spécifique avec le logiciel de la souris, en fonction
   de la valeur contenue dans la variable "action_souris" *)

begin
autoriser_interrupt;
case action_souris of
    72,75,77,80 : bouger_curseur_ecran(action_souris);
             67 : activer_bouton_souris;
             68 : desactiver_bouton_souris;
             end;
interdire_interrupt;
end;


(*****************************************************************************)


procedure affiche_erreur(erreur : integer);
(*****************************************)
(* affichage d'une erreur dans un encadrement *)

var chaine : strWrk;         (* chaine de caractères à afficher *)
    string_erreur : str3;    (* numéro de l'erreur *)

begin
cacher_curseur;
interdire_interrupt;
clic_bouton := 0;
autoriser_interrupt;
saveScreen('erreur.ecr');

dessiner_fond_fenetre(21,8,59,15);
dessiner_bord_fenetre(21,8,59,15);
affiche_carac(23,9,15,1,1,'présence d''une erreur !!!! ');
str(erreur,string_erreur);
chaine := 'erreur : ' + string_erreur;
if erreur = 1 then chaine := chaine + '  "' + fichier_inexistant + '" ';
affiche_carac(23,11,15,1,1,chaine);
affiche_carac(23,13,15,1,1,'Appuyez pour continuer ');

interdire_interrupt;
lecture_fichier := false;
repeat                    (* attente d'une pression sur le bouton de la souris *)
    autoriser_interrupt;
    interdire_interrupt;
until clic_bouton <> 0;
lecture_fichier := true;
autoriser_interrupt;

loadScreen('erreur.ecr');
montrer_curseur;
end;


(*****************************************************************************)
```

```
(*****************************************************************************)


procedure affiche_nouv_fen(filename : strWrk; type_fenetre : integer;
                           var erreur : integer);
(*************************************************************************)
(* Affichage d'une nouvelle fenêtre à l'écran avec sauvetage du contenu de
   l'écran avant l'affichage *)
(* "filename" = nom du fichier contenant la description de la fenêtre *)
(* "type_fenetre" = type de fenêtre à afficher. On distingue les types:
   "1" : pour une fenêtre ne nécessitant pas de sauvetage de l'écran sur
         disque, et qui doit donc être effacée de l'écran dès que l'utilisateur
         sélectionne un objet quelconque de l'écran avec la souris. (c'est le
         cas des fenêtres de menus).
   "2" : pour une fenêtre nécessitant un sauvetage de l'écran sur disque *)


var num_fenetre : integer;

begin
determ_num_fen(num_fenetre, erreur);
if erreur <> 10
    then begin
         case type_fenetre of
           1 : begin
               SelectScreen(1);
               CopyScreen;
               end;
           2 : sauver_ecran_disque;
           end;
         affiche_fenetre(filename,erreur);
         end;
if erreur <> 0 then affiche_erreur(erreur);
end;


(*****************************************************************************)


procedure effacer_fenetres_ecran;
(*******************************)
(* Efface toutes les fenêtres affichées à l'écran *)

var num_fenetre : integer;       (* N° des fenêtres à effacer *)
    i : integer;
    erreur : integer;
    ptr_fenetre,                 (* pointeur vers les objets des fenêtres à *)
    pteur_fenetre : obj_fenetre; (* effacer *)

begin
erreur := 0;

(* réinitialisation des tables du programme "aidami" *)

rech_der_table_ordre(num_fenetre,erreur);
while erreur = 0 do
    begin
    supprim_table_conv(num_fenetre);
    supprim_fen_ordre(num_fenetre,erreur);
    maj_table_fenetre(num_fenetre,-1,-1,-1,-1);
    rech_der_table_ordre(num_fenetre,erreur);
    end;
```

```
(* effacement des listes chainées contenant la description des fenêtres en
   mémoire centrale *)

end;


(*****************************************************************************)


procedure resetter_buffer_clavier(memoriser_carac : boolean);
(*********************************************************)

var erreur : integer;

begin
interdire_interrupt;
memw[seg_buffer_clavier:buffer_tail_mem] := buffer_start;
memw[seg_buffer_clavier:buffer_head_mem] := buffer_start;
autoriser_interrupt;
if memoriser_carac = true then ecrire_carac_clavier(0,1,erreur);
end;


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)

(* TACHE "TELEPHONE" *)


procedure Tache_telephone(num_fenetre : integer);
(************************************************)
(* Affichage du "sous-menu" de la tâche "téléphone" *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
affiche_nouv_fen('fenet2',1,erreur);
if erreur = 0 then affiche_menu := true;
end;


(*****************************************************************************)


procedure selectionner_numero(num_fenetre : integer; var erreur : integer);
(*************************************************************************)
(* Affichage de la fenêtre permettant de sélectionner un N° de téléphone *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
efface_fenetre(1,erreur);
affiche_nouv_fen('fenet5',2,erreur);
affiche_menu := false;
end;


(*****************************************************************************)


procedure raccro_decro(code,num_fenetre : integer);
(*-----------------------------------------*)
(* Décroche ou raccroche le téléphone *)
```

```
(* "code" = code à envoyer vers le "mains-libres", après transformations *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

var n_type_pteur,      (* N° du pointeur de la structure du fichier *)
    erreur : integer;

begin
erreur := 0;
acces_seq('telep1',0,-1,n_type_pteur,erreur);
pteur_telephone := ptr_teleph;
if erreur = 0 then begin
                   preparer_code_appareils(code,n_type_pteur,erreur);
                   dispose(pteur_telephone);
                   end;
efface_fenetre(1,erreur);
if erreur <> 0 then affiche_erreur(erreur);
delay(50);
end;



procedure decrocher(num_fenetre : integer);
(*****************************************)
(* Décrochage du téléphone *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
raccro_decro(10,num_fenetre);
affiche_menu := false;
end;


(*****************************************************************************)


procedure raccrocher(num_fenetre : integer);
(*****************************************)
(* Raccrochage du téléphone *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
raccro_decro(11,num_fenetre);
affiche_menu := false;
end;


(*****************************************************************************)


procedure composer_nouv_numero_teleph;
(************************************)
(* Composition d'un nouveau numéro de téléphone et initialisation des variables
   associées *)

var numero_fenetre : integer;

begin
efface_fenetre(2,erreur);
determ_num_fen(numero_fenetre,erreur);
if erreur = 0 then begin
                   num_telephone[numero_fenetre] := '';
                   affiche_nouv_fen('fenet11',2,erreur);
```

```
                end;
if erreur <> 0 then affiche_erreur(erreur);
end;


(*****************************************************************************)


procedure selection_der_num_tel;
(*******************************)
(* Sélection du dernier numéro de téléphone mémorisé *)

begin
efface_fenetre(2,erreur);
if erreur = 0 then affiche_nouv_fen('fenet12',2,erreur)
            else affiche_erreur(erreur);
end;


(*****************************************************************************)


procedure selection_num_annuaire;
(*******************************)
(* Sélection d'un numéro de téléphone dans l'annuaire *)

begin
delay(200);
end;


(*****************************************************************************)


procedure intro_num_telephone(code, num_fenetre : integer);
(**********************************************************)
(* Introduction et mémorisation d'un chiffre de numéro de téléphone. Le chiffre
   est affiché dans un emplacement réservé de la fenêtre N° "num_fenetre" *)
(* "code" = code à envoyer vers le "mains-libres" aprés mise en forme *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

var x, y,                                   (* coordonnées de l'emplacement réservé
                                               dans la fenêtre "num_fenetre", pour
                                               afficher une chaine de caractères *)
    couleur_caractere, echelle : integer;
    chiffre_telephone : string[1];

begin
rech_emplace_string(1,num_fenetre,x,y,couleur_caractere,echelle,erreur);
if erreur <> 19
    then begin
         if code = 10 then chiffre_telephone := bs
                      else str(code,chiffre_telephone);
         lecture_string_fenetre(num_telephone[num_fenetre],chiffre_telephone,
                      couleur_caractere,echelle,num_fenetre,x,y);
         end
    else affiche_erreur(erreur);
delay(60);
end;


(*****************************************************************************)
```

```
procedure numero_tel_cor(num_fenetre : integer);
(*********************************************)
(* Mémorisation du numéro de téléphone introduit *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
der_num_tel := num_telephone[num_fenetre];
delay(150);
end;


(*****************************************************************************)


procedure sortie_telephone(num_fenetre : integer);
(************************************************)
(* Sortie après avoir introduit un nouveau numéro de téléphone *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
efface_fenetre(2,erreur);
if erreur <> 0 then affiche_erreur(erreur);
end;


(*****************************************************************************)


procedure envoyer_num_tel(tel_num : strWrk);
(******************************************)
(* Envoi du numéro de téléphone "tel_num" vers le "mains-libres" *)

var chiffre_teleph : string[1];  (* un chiffre du N° de téléphone à envoyer *)
    chiffre : integer;           (* un chiffre du N° de téléphone à envoyer *)
    n_type_pteur : integer;      (* N° du pointeur vers la structure du fichier
                                    de téléphone *)
    code : integer;              (* flag indiquant si chaque chiffre du numéro
                                    de téléphone est bien un nombre entier *)
    faux_numero : boolean;       (* flag indiquant si le numéro de téléphone composé
                                    est valide *)
    i : integer;


begin

(* lecture du fichier de téléphone *)

acces_seq('telep1',0,-1,n_type_pteur,erreur);
pteur_telephone := ptr_teleph;

if erreur = 0
   then begin
        i := 1;
        faux_numero := false;
        while (i <= length(tel_num)) AND (faux_numero = false) do
           begin
           chiffre_teleph := copy(tel_num,i,1);
           val(chiffre_teleph, chiffre,code);
           if code <> 0 then begin
                             faux_numero := true;
                             erreur := 22;
                             end
```

```
                    else preparer_code_appareils(chiffre,n_type_pteur,erreur);
          if erreur = 22 then faux_numero := true;
          i := i + 1;
          end;
        dispose(pteur_telephone);
        end;
if erreur <> 0 then affiche_erreur(erreur);
end;


(*****************************************************************************)


procedure affiche_der_num_tel(num_fenetre : integer);
(*************************************************)
(* Affichage du dernier numéro de téléphone mémorisé *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

var echelle,
    x, y,                          (* coordonnées de l'emplacement situé dans la
                                      fenêtre N° "num_fenetre" pour y afficher le
                                      numéro de téléphone *)
    couleur_caractere : integer;

begin
rech_emplace_string(1,num_fenetre,x,y,couleur_caractere,echelle,erreur);
if erreur <> 19 then affiche_carac(x,y,couleur_caractere,echelle,num_fenetre,der_num_tel)
               else affiche_erreur(erreur);
end;


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)

(* TACHE "OUTILS" *)


procedure Tache_outils(num_fenetre : integer);
(*********************************************)
(* Affichage du "sous-menu" de la tâche "outils" *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
affiche_nouv_fen('fenet3',1,erreur);
if erreur = 0 then affiche_menu := true;
end;


(*****************************************************************************)


procedure afficher_horloge(num_fenetre : integer; var erreur : integer);
(***********************************************************************)
(* Affichage de l'horloge *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
efface_fenetre(1,erreur);
affiche_nouv_fen('fenet6',2,erreur);
affiche_menu := false;
end;
```

```
(*****************************************************************************)

procedure vitesse_clavier(num_fenetre : integer; var erreur : integer);
(*************************************************************************)
(* Sélection de la vitesse du clavier *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
efface_fenetre(1,erreur);
affiche_nouv_fen('fenet7',2,erreur);
affiche_menu := false;
end;


(*****************************************************************************)


procedure activer_detournement_clavier;
(************************************)
(* active ou désactive le détournement des appels aux routines de gestion des
   interruptions qui permettent de lire un caractère au clavier *)

begin
efface_fenetre(1,erreur);
interdire_interrupt;
detourne_clavier := not(detourne_clavier);
autoriser_interrupt;
delay(60);
affiche_menu := false;
end;


(*****************************************************************************)


procedure activer_touches_flechees;
(*********************************)
(* active ou désactive les touches fléchées du clavier réel lorsque le logiciel
   "standard" qui s'exécute utilise un curseur mobile à l'écran, manipulable
   avec la souris *)

begin
efface_fenetre(1,erreur);
interdire_interrupt;
reset_touche_active := not(reset_touche_active);
autoriser_interrupt;
delay(60);
affiche_menu := false;
end;


(*****************************************************************************)


procedure desactiver_son;
(***********************)
(* désactive le son du programme "aidami" *)

begin
efface_fenetre(1,erreur);
```

```
actif_son := not(actif_son);
delay(60);
affiche_menu := false;
end;


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)

(* TACHE "LOGICIELS" *)


procedure Tache_logiciel(num_fenetre : integer);
(*********************************************)
(* Affichage du "sous-menu" de la tâche "logiciel" *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
affiche_nouv_fen('fenet4',1,erreur);
if erreur = 0 then affiche_menu := true;
end;


(*****************************************************************************)


procedure afficher_repertoire(num_fenetre : integer; var erreur : integer);
(*************************************************************************)
(* Affichage du répertoire *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
efface_fenetre(1,erreur);
affiche_nouv_fen('fenet8',2,erreur);
affiche_menu := false;
end;


(*****************************************************************************)


procedure afficher_clavier(num_gr_objet : integer; var erreur : integer);
(*************************************************************************)
(* Affichage du clavier et initialisation des variables associées *)
(* "num_gr_objet" = N° du groupe de l'objet pointé *)

var numero_fenetre : integer;

begin
if logiciel_clavier = true then efface_fenetre(2,erreur)
                           else efface_fenetre(1,erreur);
determ_num_fen(numero_fenetre,erreur);
if erreur <> 10
    then begin
         case num_gr_objet of
             1 : affiche_nouv_fen('fenet9',2,erreur);
             2 : affiche_nouv_fen('fenet13',2,erreur);
             end;
         chaine_cla[numero_fenetre] := '';
         taille_buffer := 80;
         nbre_car_buffer := 0;
```

```
        reset_touch_special;
        affiche_menu := false;
        end;
end;


(*****************************************************************************)


procedure lire_clavier(code_rech, code_asci, num_fenetre : integer);
(*****************************************************************)
(* Lecture d'un caractère introduit au clavier simulé de l'écran *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)
(* "code_rech" et "code_asci" = codes de recherche et ascii augmentés de 1000,
   caractérisant la touche du clavier *)

var code, asci,    (* codes de recherche et ascii *)
    x, y,          (* coordonnées de l'emplacement réservé dans la fenêtre N°
                      "num_fenetre" pour y afficher le caractère sélectionné *)
    echelle,
    couleur_caractere : integer;
    caract : char;         (* caractère sélectionné *)

begin
if nbre_car_buffer < taille_buffer  (* teste si le buffer du clavier simulé est
                                       rempli *)
then begin
     code := code_rech - 1000;
     asci := code_asci - 1000;
     if logiciel = false then flags(code);
     if (asci in [97..122]) AND (caps_lock_etat = true)
         then asci := asci - 32;    (* détermination du code ascii pour les
                                       majuscules *)
     code_asci_clavier := asci;
     code_rech_clavier := code;
     if (asci in [8,21,32..255]) AND (code > 1)
         then begin

              (* affichage du caractère sélectionné dans l'emplacement réservé
                 de la fenêtre à l'écran *)

              rech_emplace_string(1,num_fenetre,x,y,couleur_caractere,echelle,erreur);
              if erreur = 0
                  then begin
                       caract := chr(asci);
                       lecture_string_fenetre(chaine_cla[num_fenetre],caract,
                                 couleur_caractere,echelle,num_fenetre,x,y);
                       end;
              end;
     nbre_car_buffer := nbre_car_buffer + 1;
     end
else son(300,400);

if erreur <> 0 then affiche_erreur(erreur);
delay(50);
end;


(*****************************************************************************)


procedure lancer_logiciel(num_fenetre, num_gr_objet : integer);
(*************************************************************)
```

```
(* Lance l'exécution d'un logiciel "standard" *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)
(* "num_gr_objet" = N° du groupe de l'objet pointé *)

var filename : strWrk;  (* nom du logiciel "standard" à exécuter *)


procedure determ_logiciel(num_gr_objet : integer; var filename : strWrk);
(*-------------------------------------------------------------------*)
(* TABLE DES LOGICIELS "STANDARDS" du programme "aidami" *)
(* Détermination d'un nom de fichier "filename" en fonction du numéro de
   groupe d'objet "num_gr_objet *)

begin
case num_gr_objet of
   1 : filename := 'turbo';
   2 : filename := 'a:pctools';
   3 : filename := 'pcpaint';
   5 : filename := 'ws';
   6 : filename := 'dbase';
   7 : filename := 'gwbasic';
   8 : filename := 'chess';
   end;
end;



procedure inserer_nom_fich_batch(filename : strWrk; var erreur : integer);
(*-------------------------------------------------------------------*)
(* Insertion du nom du fichier "filename" dans le fichier "batch" appelé
   "batch1" *)

begin
tampon[1] := filename;
ecrire_fich_seq_text('batch1.bat',1,1,erreur);
end;




begin                  (**** lancer_logiciel ****)
effacer_fenetres_ecran;
if num_gr_objet <> 4 then determ_logiciel(num_gr_objet,filename)
                     else filename := chaine_cla[num_fenetre];
inserer_nom_fich_batch(filename,erreur);
memw[$0000:$0184] := 0;   (* permet de sortir du programme "aida_go.com" *)
sortie_aida := true;
end;


(*****************************************************************************)


procedure sortie_clavier(num_fenetre : integer);
(*********************************************)
(* Sortie d'une fenêtre représentant un clavier simulé à l'écran *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
if logiciel_clavier = true then begin
                                lancer_logiciel(num_fenetre,4);
                                logiciel_clavier := false;
```

```
                        end
                    else efface_fenetre(2,erreur);
if erreur <> 0 then affiche_erreur(erreur);
end;


(*****************************************************************************)


procedure afficher_IBM_PC(num_fenetre : integer; var erreur : integer);
(*************************************************************************)
(* Affichage des logiciels exécutables sur IBM-PC *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
efface_fenetre(1,erreur);
affiche_nouv_fen('fenet10',2,erreur);
affiche_menu := false;
end;


(*****************************************************************************)


procedure choix_logiciel(num_gr_objet, num_fenetre : integer);
(*************************************************************)
(* Affichage de la fenêtre permettant de choisir un logiciel "standard" en
   vue de son exécution *)
(* num_gr_objet" = N° du groupe de l'objet pointé par le curseur *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)

begin
if num_gr_objet <> 0
    then begin
         logiciel := true;
         logiciel_clavier := true;
         if num_gr_objet = 4 then afficher_clavier(2,erreur)
                            else lancer_logiciel(num_fenetre, num_gr_objet);
         end;
end;


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)

(* TACHE "SORTIE" *)


procedure retablir_vecteurs_interrupt;
(*--------------------------------*)
(* remémorise dans tous les vecteurs d'interruption du système d'exploitation leur
   ancienne adresse avant de lancer le programme "aidami" *)

var i : integer;

begin
interdire_interrupt;

for i := $00 to $3F do
    begin
    if i <> $21 then begin
```

```
                numero_vecteur := i;
                segment_code := segment[i];
                offset_code := offset[i];
                ecrire_vecteur_interrupt;
                end;
    end;

numero_vecteur := $21;
segment_code := segment[$21];
offset_code := offset[$21];
ecrire_vecteur_interrupt;

numero_vecteur := $1C;
segment_code := segment[$1C];
offset_code := offset[$1C];
ecrire_vecteur_interrupt;

memw[$0000:$0184] := 0;
autoriser_interrupt;

vider_contenu_fich('batch1.bat',erreur);
end;



procedure tache_sortie;
(**********************)
(* Sortie du programme "aidami" *)

begin
sortie_aida := true;
affiche_menu := false;
if logiciel = false then retablir_vecteurs_interrupt;
end;



(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)


procedure executer_action(num_action, num_fenetre, type_objet, num_gr_objet :
                          integer);
(*****************************************************************************)
(* TABLE DES ACTIONS du programme "aidami" *)
(* "num_action" = N° de l'action à effectuer *)
(* "num_fenetre" = fenêtre pointée par le curseur de l'écran *)
(* "typeobjet" = type de l'objet pointé par le curseur *)
(* "num_gr_objet" = N° du groupe de l'objet pointé par le curseur *)

begin
erreur := 0;
cacher_curseur;
case num_action of
    0 : ;
    1 : Tache_telephone(num_fenetre);
    2 : Tache_outils(num_fenetre);
    3 : Tache_logiciel(num_fenetre);
    4 : selectionner_numero(num_fenetre, erreur);
    5 : decrocher(num_fenetre);
```

```
    6 : raccrocher(num_fenetre);
    7 : afficher_horloge(num_fenetre, erreur);
    8 : vitesse_clavier(num_fenetre, erreur);
    9 : afficher_repertoire(num_fenetre, erreur);
    10 : afficher_clavier(num_gr_objet,erreur);
    11 : afficher_IBM_PC(num_fenetre, erreur);
    12 : tache_sortie;
    13 : efface_fenetre(2,erreur);
    14 : composer_nouv_numero_teleph;
    15 : selection_der_num_tel;
    16 : selection_num_annuaire;
    17 : intro_num_telephone(num_gr_objet, num_fenetre);
    18 : numero_tel_cor(num_fenetre);
    19 : sortie_telephone(num_fenetre);
    20 : envoyer_num_tel(num_telephone[num_fenetre]);
    21 : affiche_der_num_tel(num_fenetre);
    22 : envoyer_num_tel(der_num_tel);
    23 : choix_logiciel(num_gr_objet, num_fenetre);
    24 : sortie_clavier(num_fenetre);
    25 : activer_detournement_clavier;
    26 : activer_touches_flechees;
    27 : desactiver_son;
    else if type_objet = 5 then lire_clavier(num_action, num_gr_objet,num_fenetre);
    end;
montrer_curseur;
end;


(*****************************************************************************)


procedure bouton_actif;
(**********************)
(* Opérations à effectuer lorsque le bouton de la souris est enfoncé *)

var num_fen,                (* N° de la dernière fenêtre affichée à l'écran *)
    erreur,
    num_action,             (* N° de l'action associée à l'objet pointé *)
    num_gr_objet,           (* N° du groupe de l'objet pointé par le curseur *)
    type_objet,             (* type de l'objet pointé par le curseur *)
    x, y,                   (* coordonnées du curseur de l'écran *)
    num_fenetre : integer;  (* N° de la fenêtre pointée par le curseur *)


begin
erreur := 0;
lect_pos_souris(x, y);
x := trunc(x / 8);
determiner_action(x, y, num_fenetre, type_objet, num_gr_objet, num_action,
                  erreur);
case erreur of
    0 : executer_action(num_action, num_fenetre, type_objet, num_gr_objet);
    5 : begin
        if affiche_menu = true
           then begin

                (* effacement d'un "sous-menu" si celui-ci est affiché à l'écran
                   et si le pointeur de l'écran sélectionne un endroit de
                   l'écran où aucune fenêtre n'est affichée *)

                cacher_curseur;
                erreur := 0;
                rech_der_table_ordre(num_fen,erreur);
```

```
var
    (* registres du processeur sauvés dans la pile lors de la génération de
       l'interruption par le programme utilisateur *)
    ds_souris, ax_souris, sp_souris, bp_souris, ip_souris, cs_souris : integer;

    ass_souris1,            (* ancien Stack Segment *)
    asp_souris1,            (* ancien Stack Pointer *)
    abp_souris1,            (* ancien Base Pointer *)
    nss_souris1 : integer;  (* nouveau Stack Segment *)

    modif_action_souris : boolean; (* flag indiquant si des fonctions du logiciel
                                      de la souris doivent être exécutées. Cette
                                      fonction est mémorisée dans la routine de
                                      détournement du clavier réel, pour empêcher
                                      toute réentrance dans le logiciel de la
                                      souris *)
    ipp_souris, css_souris : integer; (* adresse vers laquelle le processus qui
                                         exécute le logiciel de la souris, est
                                         détourné, après exécution d'une primitive
                                         de ce logiciel *)
    ofs_saut_5, ofs_saut_6 : integer; (* adresse de la routine du logiciel de la
                                         souris que le programme utilisateur
                                         désire exécuter *)


procedure souris_entree;
(**********************)
(* permet d'exécuter une primitive d'exclusion mutuelle avant de donner le
   contrôle au logiciel de la souris *)

begin

(* mémorisation des registres du processeur *)

inline($50/$53/$51/$52/$56/$57/$1E/$06);

(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

(* mémorisation de l'adresse de retour au programme utilisateur interrompu *)

inline($07);
inline($58/$A3/ds_souris/$5F/$5E/$5A/$59/$5B);
inline($58/$A3/ax_souris);
inline($58/$A3/sp_souris/$58/$A3/bp_souris);
inline($58/$A3/ip_souris/$58/$A3/cs_souris);

inline($A1/cs_souris/$50/$A1/ip_souris/$50);
inline($A1/bp_souris/$50/$A1/sp_souris/$50);
inline($A1/ax_souris/$50);
inline($53/$51/$52/$56/$57/$A1/ds_souris/$50);
inline($06);

(* changement de pile *)

inline($16/$58/$A3/ass_souris1/$A1/nss_souris1/$50/$17);
inline($94/$A3/asp_souris1/$95/$A3/abp_souris1);
inline($B8/pile_souris1/$94/$B8/pile_souris1/$95);

attendre(exclusion_souris);
```

```
if modif_action_souris = true
    then begin
         ipp_souris := ip_souris;
         css_souris := cs_souris;
         end;

(* restauration de la pile *)

inline($A1/ass_souris1/$50/$17);
inline($A1/asp_souris1/$94/$A1/abp_souris1/$95);

(* mémorisation de l'adresse de retour à la procédure "souris_sortie" *)

inline($07);
inline($58/$A3/ds_souris/$5F/$5E/$5A/$59/$5B);
inline($58/$A3/ax_souris);
inline($58/$A3/sp_souris/$58/$A3/bp_souris);
inline($58/$A3/ip_souris/$58/$A3/cs_souris);

inline($A1/css_souris/$50/$A1/ipp_souris/$50);
inline($A1/bp_souris/$50/$A1/sp_souris/$50);
inline($A1/ax_souris/$50);
inline($53/$51/$52/$56/$57/$A1/ds_souris/$50);
inline($06);

(* mémorisation de l'adresse d'entrée de la procédure du logiciel de la souris
   appelée *)

memw[Cseg:ofs_saut_5] := offset[num_vecteur_souris];
memw[Cseg:ofs_saut_6] := segment[num_vecteur_souris];

(* restauration des registres du processeur *)

inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58);
inline($5D/$8B/$E5/$5D);

(* saut inconditionnel absolu vers une procédure du logiciel de la souris
   à exécuter *)
(* ATTENTION !!! ce bloc d'instructions ne peut être scindé !!! *)

(************************************)
inline($E9/$0E/$00);           (****)
end;                             (**)
procedure saut_souris;           (**)
begin                            (**)
inline($EA/$00/$00/$00/$00);     (**)
end;                           (****)
(************************************)


(*****************************************************************************)


var
    ass_souris2,           (* ancien Stack Segment *)
    asp_souris2,           (* ancien Stack Pointer *)
    abp_souris2,           (* ancien Base Pointer *)
    nss_souris2 : integer; (* nouveau Stack Segment *)

    action_souris : integer;(* identifie les fonctions du logiciel de la souris
                               à appeler pour satisfaire une demande faite par la
                               routine du clavier détourné *)
    csss_souris, ippp_souris : integer; (* adresse de retour au programme utilisateur
```

```
                                        interrompu *)


procedure souris_sortie;
(**********************)
(* permet d'exécuter une primitive d'exclusion mutuelle aprés l'exécution
   du logiciel de la souris *)

begin

(* mémorisation des registres du processeur *)

inline($9C/$50/$50/$FA);
inline($55/$8B/$EC/$55);
inline($50/$53/$51/$52/$56/$57/$1E/$06);

(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);
inline($A1/data_seg/$50/$1F);

(* changement de pile *)

inline($16/$58/$A3/ass_souris2/$A1/nss_souris2/$50/$17);
inline($94/$A3/asp_souris2/$95/$A3/abp_souris2);
inline($B8/pile_souris2/$94/$B8/pile_souris2/$95);

csss_souris := cs_souris;
ippp_souris := ip_souris;

(* exécution de certaines fonctions du logiciel de la souris, qui ont été
   mémorisées dans la routine de détournement du clavier réel de la machine.
   Ceci permet d'éviter toute réentrance dans le logiciel de la souris *)

if action_souris <> 0 then begin
                           modif_action_souris := true;
                           activer_souris(action_souris);
                           action_souris := 0;
                           modif_action_souris := false;
                           ipp_souris := ofs(souris_sortie) + 7;
                           css_souris := Cseg;
                           end;

signaler(exclusion_souris);

(* restauration de la pile *)

inline($A1/ass_souris2/$50/$17);
inline($A1/asp_souris2/$94/$A1/abp_souris2/$95);

(* restauration de l'adresse de retour au programme utilisateur interrompu *)

inline($07);
inline($58/$A3/ds_souris/$5F/$5E/$5A/$59/$5B);
inline($58/$A3/ax_souris);
inline($58/$A3/sp_souris/$58/$A3/bp_souris);
inline($58/$58);

inline($A1/csss_souris/$50/$A1/ippp_souris/$50);
inline($A1/bp_souris/$50/$A1/sp_souris/$50);
inline($A1/ax_souris/$50);
inline($53/$51/$52/$56/$57/$A1/ds_souris/$50);
inline($06);
```

```
inline($FB);

(* restauration des registres du processeur *)

inline($07/$1F/$5F/$5E/$5A/$59/$5B/$58);
inline($5D/$8B/$E5/$5D);
inline($CF);
end;


(*****************************************************************************)


procedure lecture_etat_bouton;
(*-------------------------*)
(* détermination du nombre de fois que le bouton de la souris est enfoncé *)

begin
if clic_bouton = 2 then begin
                        clic_bouton := 3;
                        tps_bouton := 0;
                        end;
if clic_bouton = 1 then begin
                        clic_bouton := 2;
                        tps_bouton := 0;
                        end;
if clic_bouton = 0 then begin
                        clic_bouton := 1;
                        tps_bouton := 0;
                        end;
end;


(*****************************************************************************)


var
    ass_clavier,            (* ancien Stack Segment *)
    asp_clavier,            (* ancien Stack Pointer *)
    abp_clavier,            (* ancien Base Pointer *)
    nss_clavier : integer;  (* nouveau Stack Segment *)

    effacer_carac_buffer : boolean; (* flag indiquant qu'il faut effacer le caractère
                                       introduit au clavier réel, du buffer du
                                       clavier réel *)
    caractere : byte;       (* caractère introduit au clavier réel *)


procedure clavier;
(*****************)
(* procédure de détournement du clavier *)

begin

(* sauvetage des registres du processeur *)

inline ($50/$53/$51/$52/$56/$57/$1E/$06);

(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);                    (* DS = $0000 *)
inline($A1/data_seg/$50/$1F);                   (* DS = [data_seg] *)
```

```
exclusion_clavier := 0;

if lecture_fichier = false
   then begin

        (* changement de pile *)

        inline($16/$58/$A3/ass_clavier/$A1/nss_clavier/$50/$17);
        inline($94/$A3/asp_clavier/$95/$A3/abp_clavier);
        inline($B8/pile_clavier/$94/$B8/pile_clavier/$95);

        (* lecture du caractère présent sur le port du clavier et actions
           exécutées lors de l'appui sur la touche "F9" du clavier réel *)

        caractere := port[$60];
        if caractere = 67 then begin
                               effacer_carac_buffer := true;
                               lecture_etat_bouton;
                               actif_bouton_souris := true;
                               end;


        if touche_active_souris = true
           then begin
                case caractere of

                 (* actions des touches fléchées du clavier réel *)

                 72,75,77,80 : if exclusion_souris = 1 then activer_souris(caractere)
                                                       else action_souris := caractere;

                 (* action de la touche "F9" du clavier réel *)

                         67 : if exclusion_souris = 1 then activer_souris(caractere)
                                                       else action_souris := caractere;

                 (* action de la touche "F10" du clavier réel *)

                         68 : if exclusion_souris = 1 then activer_souris(caractere)
                                                       else action_souris := caractere;
                         end;  (* endcase *)

                if caractere in [67,68,72,75,77,80] then effacer_carac_buffer := true;
                end;

        (* restauration de la pile *)

        inline($A1/ass_clavier/$50/$17);
        inline($A1/asp_clavier/$94/$A1/abp_clavier/$95);

        (* exécution de la routine de gestion du clavier du système
           d'exploitation *)

        inline($CD/num_vecteur_clavier_detourne);

        inline($FA);

        (* suppression du caractère mémorisé dans le buffer du clavier si
           l'utilisateur a appuyé sur une touche fléchée ou sur une des
           touches "F9" ou "F10" du clavier réel *)

        if effacer_carac_buffer = true
```

```
          then begin
               effacer_carac_buffer := false;
               buffer_tail := ($100 * mem[seg_buffer_clavier:buffer_tail_mem + 1])
                              + mem[seg_buffer_clavier:buffer_tail_mem];
               if buffer_tail = buffer_start then buffer_tail := buffer_end - 2
                                             else buffer_tail := buffer_tail - 2;
               memw[seg_buffer_clavier:buffer_tail_mem] := buffer_tail;
               end;

        end

    (* exécution de la routine de gestion du clavier du système d'exploitation
       lorsque les touches fléchées et "F9" et "F10" du clavier réel n'ont plus
       leur sémantique modifiée par le programme "aidami" *)

    else inline($CD/num_vecteur_clavier_detourne);

inline($FA);
exclusion_clavier := 1;
inline($FB);

(* restauration des registres du processeur *)

inline ($07/$1F/$5F/$5E/$5A/$59/$5B/$58/$8B/$E5/$5D/$CF);
end;


(*****************************************************************************)
```

```
program aidami;

(*{$iphil\constant.pas}*)
(*{$iphil\variable.pas}*)

(* insertion des routines du logiciel "Turbo-Graphix" *)

(*{$iphil\typedef.sys}
{$iphil\graphix.sys}
{$iphil\kernel.sys}
{$iphil\windows.sys}*)

(* insertion des composants de la hiérarchie du programme "aidami" *)

(*{$iphil\syst_exp.pas}
{$iphil\routi_os.pas}
{$iphil\man_fich.pas}
{$iphil\aux_impr.pas}
{$iphil\gest_ecr.pas}
{$iphil\entrees.pas}
{$iphil\sorties.pas}
{$iphil\form_sig.pas}
{$iphil\coordina.pas}
{$iphil\cla_sour.pas}*)

{$iconstant.pas}
{$ivariable.pas}

(* insertion des routines du logiciel "Turbo-Graphix" *)

{$ib:typedef.sys}
{$ib:graphix.sys}
{$ib:kernel.sys}
{$ib:windows.sys}

(* insertion des composants de la hiérarchie du programme "aidami" *)

{$isyst_exp.pas}
{$irouti_os.pas}
{$iman_fich.pas}
{$iaux_impr.pas}
{$igest_ecr.pas}
{$ientrees.pas}
{$isorties.pas}
{$iform_sig.pas}
{$icoordina.pas}
{$icla_sour.pas}


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)


procedure tester_mode_ecran_entree;
(*********************************)
(* détermine le mode de l'écran à l'entrée du programme "AIDAMI" et met l'écran
   en mode graphique si le mode utilisé par le programme utilisateur n'est pas
   le mode graphique ou le mode texte 80 colonnes couleur *)


procedure init_font;
```

```
(*---------------*)
(* (re)chargement du générateur de caractères pour le mode graphique de l'écran *)

var FontFile : file of IBMFont;

begin
if not FontLoaded then
   begin
   assign(FontFile,'8x8.FON');
   {$i-} reset(FontFile); {$i+}
   if IOresult = 0 then begin
                        read(FontFile,font);
                        close(FontFile);
                        end
                   else fillchar(font,sizeOf(font),0);
   FontLoaded := true;
   end;
if GrafModeGlb  = false then ConOutPtrSave := ConOutPtr;
ConOutPtr := ofs(DisplayChar);
GrafModeGlb := true;
end;



begin          (**** tester_mode_ecran_entree ****)

regs.ah := $0F;
intr($10,regs);
mode_ecran := regs.al;
case mode_ecran of
   3 :       begin
             texte_curseur;
             deplace_vertical_curseur := 8;
             deplace_horizontal_curseur := 8;
             end;
   1,4,5,6 : begin
             init_font;
             deplace_vertical_curseur := 4;
             deplace_horizontal_curseur := 8;
             end;
   end;
if mode_ecran = 1 then begin
                       hires;
                       hiresColor(white);
                       end;
end;


(*****************************************************************************)


procedure tester_mode_ecran_sortie;
(*********************************)
(* restaure le mode de l'écran à la sortie du programme "AIDAMI" *)

begin
if mode_ecran = 1 then begin
                       textMode(C40);
                       textColor(white);
                       textBackGround(black);
                       end;
```

```
if mode_ecran in [1,4,5,6]
    then begin
         if GrafModeGlb = true then ConOutPtr := ConOutPtrSave;
         GrafModeGlb := false;
         end;
end;


(*****************************************************************************)


procedure sauver_environnement;
(*****************************)
(* sauve l'environnement du programme interrompu avant d'entrer dans le
   programme "AIDAMI" *)

begin
interdire_interrupt;
if change_1C = true then begin
                         change_1C := false;
                         changement_1C := true;
                         end;

(* activation des touches fléchées et "F9" et "F10" du clavier réel pour pouvoir
   exécuter le programme "aidami" correctement *)

touche_active_souris := true;
autoriser_interrupt;

x_curseur := wherex;
y_curseur := wherey;

(* sauve sur disque le contenu de l'écran du programme utilisateur interrompu *)

savescreen('logici');
end;


(*****************************************************************************)


procedure restaurer_environnement;
(*********************************)
(* restaure l'environnement du programme interrompu à la sortie du programme
   "AIDAMI" *)

begin

(* restaure le contenu de l'écran *)

loadscreen('logici');

if mode_ecran in [1,3] then gotoxy(x_curseur,y_curseur);

interdire_interrupt;
if reset_touche_active = true then touche_active_souris := false
                              else touche_active_souris := true;
if changement_1C = true then begin
                             change_1C := true;
                             changement_1C := false;
                             end;
autoriser_interrupt;
end;
```

```
(*****************************************************************************)


procedure lect_fich(lecture_fich : boolean);
(*****************************************)
(* interdit temporairement toute action sur le curseur de l'écran à l'aide
   des touches fléchées du clavier *)

begin
interdire_interrupt;
if lecture_fich = true then lecture_fichier := true
                       else lecture_fichier := false;
autoriser_interrupt;
end;


(*****************************************************************************)


procedure traiter_curseur(var curseur_cache : boolean);
(*-----------------------------------------------*)
(* gère l'apparition et la disparition du curseur de la souris à l'écran *)

begin
curseur_cache := false;
if mem[seg_souris:$0156] <> $00
    then begin
         while mem[seg_souris:$0156] <> $FF do
            begin
            montrer_curseur;
            end;
         end
    else begin
         cacher_curseur;
         curseur_cache := true;
         end;
end;



procedure executer_aidami;
(************************)
(* exécution du programme "aidami" *)

var i : integer;
    curseur_cache : boolean;         (* flag indiquant que le curseur était visible
                                        à l'écran avant de faire un appel au
                                        programme "aidami" *)

begin

(* sauvetage de l'environnement du programme utilisateur interrompu *)

interdire_interrupt;
actif_bouton_souris := false;
autoriser_interrupt;
lect_fich(true);
if mem[$0000:$0184] = 1 then begin
                             logiciel := false;
                             regs.ax := 0;
                             regs.bx := 2;
```

```
                        intr(num_vecteur_souris,regs);
                        end;

traiter_curseur(curseur_cache);
sauver_environnement;
tester_mode_ecran_entree;

(* affichage du menu principal à l'écran *)

erreur := 0;
SelectScreen(1);
affiche_fenetre('feneti',erreur);
if erreur = 0
   then begin
        montrer_curseur;
        lect_fich(false);

        (* boucle testant si le bouton de la souris est enfoncé. Exécution
           du programme "aidami" s'il est enfoncé *)

        sortie_aida := false;
        while sortie_aida = false do
           begin
           interdire_interrupt;
           if actif_bouton_souris = true then begin
                                              autoriser_interrupt;
                                              lect_fich(true);
                                              bouton_actif;
                                              interdire_interrupt;
                                              actif_bouton_souris := false;
                                              autoriser_interrupt;
                                              lect_fich(false);
                                              end;
           autoriser_interrupt;
           end;

        lect_fich(true);
        cacher_curseur;
        effacer_fenetres_ecran;
        end
   else begin
        affiche_erreur(erreur);
        delay(5000);
        end;

(* restauration de l'environnement du programme interrompu *)

tester_mode_ecran_sortie;
restaurer_environnement;
if curseur_cache = true then montrer_curseur;
lect_fich(false);
interdire_interrupt;
memw[$0000:$0188] := 0;
autoriser_interrupt;
end;


(*****************************************************************************)


procedure afficher_fenetre_clavier;
(*********************************)
(* permet d'afficher la fenetre du clavier *)
```

```
var erreur : integer;
    i : integer;
    curseur_cache : boolean;     (* flag indiquant que le curseur était visible
                                    à l'écran avant de faire un appel au
                                    programme "aidami" *)

begin

(* sauvetage de l'environnement du programme utilisateur interrompu *)

lect_fich(true);
traiter_curseur(curseur_cache);
tester_mode_ecran_entree;

if clavier_fenetre_affichee = false
    then begin
         SelectScreen(1);
         CopyScreen;
         erreur := 0;
         affiche_fenetre('fenet9',erreur);
         if erreur = 0
            then begin
                 if curseur_cache = true then curseur_non_present := 0
                                         else begin
                                              curseur_non_present := 1;
                                              montrer_curseur;
                                              end;
                 interdire_interrupt;
                 touche_active_souris := true;
                 autoriser_interrupt;
                 clavier_fenetre_affichee := not(clavier_fenetre_affichee);
                 end
            else begin
                 affiche_erreur(erreur);
                 delay(5000);
                 end;
         end
    else begin
         effacer_fenetres_ecran;
         if curseur_non_present <> 0 then cacher_curseur;
         clavier_fenetre_affichee := not(clavier_fenetre_affichee);
         end;

son(300,100);

(* restauration de l'environnement du programme interrompu *)

special := false;
resetter_buffer_clavier(false);
tester_mode_ecran_sortie;
if curseur_cache = true then montrer_curseur;
lect_fich(false);
interdire_interrupt;
if clavier_fenetre_affichee = false
    then if reset_touche_active = true then touche_active_souris := false
                                       else touche_active_souris := true;
autoriser_interrupt;
end;


(*****************************************************************************)
```

```
procedure ecrire_1_carac_buffer_clavier;
(*************************************)
(* écrit un caractère dans le buffer du clavier de l'ordinateur *)

var code, asci : integer;  (* codes ascii et de recherches mémorisés dans le
                              buffer du clavier *)
    erreur : integer;
    numero_fenetre : integer;
    touch_spec : integer;  (* identifie une touche de fonction du clavier *)


procedure touche_special(var touch_spec : integer; asci, code : integer);
(*-------------------------------------------------------------------*)
(* identification de certaines touches de fonction du clavier en fonction de
   leurs codes ascii et de recherche *)

begin
if (asci = 0) AND (code = 29) then touch_spec := 1;  (* ctrl *)
if (asci = 1) AND (code = 58) then touch_spec := 2;  (* majuscule *)
if (asci = 0) AND (code = 56) then touch_spec := 3;  (* alternate *)
if (asci = 0) AND (code = 59) then touch_spec := 4;  (* fonctions "Fi" *)
end;



procedure traiter_touches_speciales(var asci, code : integer);
(*-----------------------------------------------------------*)
(* détermination des codes de recherche et ascii pour certaines touches de
   fonction du clavier *)

begin
if (ctrl = true) then begin
                      if (asci in [65..90]) OR (asci in [97..122])
                          then begin
                               if asci in [65..90] then asci := asci - 64;
                               if asci in [97..122] then asci := asci - 96;
                               end;
                      end;
if (alternate = true) then begin
                           if (asci in [65..90]) OR (asci in [97..122])
                               then begin
                                    asci := 0;
                                    if code = 16 then code := 30;
                                    if code = 39 then code := 50;
                                    if code = 30 then code := 16;
                                    if code = 44 then code := 17;
                                    if code = 17 then code := 44;
                                    end;
                           end;
if (fct = true) then begin
                     if (asci in [49..57])
                         then begin
                              code := asci + 10;
                              asci := 0;
                              end;
                     end;
end;



procedure traiter_touch_spec(touch_spec, code : integer);
```

```
(*-------------------------------------------------*)
(* active une touche de fonction du clavier *)

begin
case touch_spec of
    1 : begin
        if ctrl = false then begin
                             if special = true then reset_touch_special;
                             flags(code);
                             special := true;
                             end
                        else reset_touch_special;
        end;
    2 : begin
        flags(code);
        reset_touch_special;
        end;
    3 : begin
        if alternate = false then begin
                                  if special = true then reset_touch_special;
                                  flags(code);
                                  special := true;
                                  end
                             else reset_touch_special;
        end;
    4 : begin
        if fct = false then begin
                            if special = true then reset_touch_special;
                            fct := true;
                            special := true;
                            end
                       else reset_touch_special;
        end;
    end;
end;




begin                  (**** ecrire_1_carac_buffer_clavier ****)

lect_fich(true);
cacher_curseur;

taille_buffer := 15;
nbre_car_buffer := 0;
rech_der_table_ordre(numero_fenetre,erreur);
chaine_cla[numero_fenetre] := '';

bouton_actif;          (* recherche du caractère sélectionné *)

asci := code_asci_clavier;
code := code_rech_clavier;
touch_spec := 0;
touche_special(touch_spec,asci,code);
if touch_spec = 0
    then begin
         if (not( (asci = 0) AND (code = 0) )) AND (nbre_car_buffer <> 0)
             then begin
                  erreur := 0;
                  traiter_touches_speciales(asci,code);
                  resetter_buffer_clavier(false);
```

```
                ecrire_carac_clavier(asci,code,erreur);
                special := false;
                reset_touch_special;
                if not (code in [42,54,58,69,70]) then carac_present_buffer := true;
                end;
        end
    else begin
        son(400,75);
        traiter_touch_spec(touch_spec,code);
        end;

montrer_curseur;
lect_fich(false);
end;


(*****************************************************************************)


procedure executer_action_aidami;
(*****************************)
(* action effectuée en fonction du nombre d'appuis successifs sur le bouton de
   la souris *)

var fenetre_clavier : boolean; (* flag indiquant si le clavier simulé était affiché
                                  à l'écran avant d'appeler le programme "aidami" *)

begin
interdire_interrupt;
if (clic_bouton = 1) AND (clavier_fenetre_affichee = true)
   then begin
        autoriser_interrupt;
        ecrire_1_carac_buffer_clavier;
        interdire_interrupt;
        end;
if clic_bouton = 2 then begin
                        autoriser_interrupt;
                        afficher_fenetre_clavier;
                        interdire_interrupt;
                        end;
if clic_bouton = 3
    then begin
        autoriser_interrupt;
        fenetre_clavier := false;
        if clavier_fenetre_affichee = true then begin
                                                afficher_fenetre_clavier;
                                                fenetre_clavier := true;
                                                end;
        executer_aidami;
        if fenetre_clavier = true then begin
                                       afficher_fenetre_clavier;
                                       fenetre_clavier := false;
                                       end;
        if ((detourne_clavier = true) AND (intro_data_clavier = false))
            then resetter_buffer_clavier(true);
        if detourne_clavier = false
            then begin
                 resetter_buffer_clavier(true);
                 carac_present_buffer := true;
                 end;
        interdire_interrupt;
        end;
end;
```

```
(*****************************************************************************)


procedure choix_aidami_clavier_detourne;
(**************************************)
(* action effectuée en fonction du nombre de fois que le bouton de la souris est
   enfoncé, lorsque les interruptions permettant de lire un caractère au clavier
   sont détournées dans la procédure "entree" *)

begin
interdire_interrupt;
carac_present_buffer := false;

repeat
    clic_bouton := 0;
        repeat
            autoriser_interrupt;
            interdire_interrupt;
        until ((clic_bouton <> 0) AND (tps_bouton > tps_inter_clic));

    if ((clic_bouton = 1) AND (clavier_fenetre_affichee = false))
       then begin
            autoriser_interrupt;
            afficher_fenetre_clavier;
            interdire_interrupt;
            end;
    autoriser_interrupt;
    executer_action_aidami;
    interdire_interrupt;
until carac_present_buffer = true;

intro_data_clavier := false;
autoriser_interrupt;
end;


(*****************************************************************************)


procedure programme_aidami;
(*************************)
(* exécution d'une action spécifique en fonction du nombre de fois que
   l'utilisateur a enfoncé successivement le bouton de la souris *)

begin
interdire_interrupt;
if intro_data_clavier = true
    then begin
         autoriser_interrupt;
         choix_aidami_clavier_detourne;
         interdire_interrupt;
         end
    else begin
         interdire_interrupt;
         if mem[$0000:$0188] = 1 then clic_bouton := 3;
         autoriser_interrupt;
         executer_action_aidami;
         end;

interdire_interrupt;
clic_bouton := 0;
```

```
autoriser_interrupt;
end;


(*****************************************************************************)


var
    (* registres du processeur sauvés dans la pile lors de la génération de
       l'interruption par le programme utilisateur *)
    ass_prog_aidami,           (* ancien Stack Segment *)
    nss_prog_aidami,           (* nouveau Stack Segment *)
    asp_prog_aidami,           (* ancien Stack Pointer *)
    abp_prog_aidami : integer; (* ancien Base Pointer *)


procedure prog_aidami;
(*********************)
(* interception de l'interruption numéro 1Ch pour pouvoir exécuter le programme
   "aidami" *)

begin

(* sauvetage des registres du processeur *)

inline($50/$53/$51/$52/$56/$57/$1E/$06);

(* restauration du registre DS *)

inline($B8/$00/$00/$50/$1F);                    (* DS = $0000 *)
inline($A1/data_seg/$50/$1F);                   (* DS = [data_seg] *)

(* détournement vers une routine éventuelle d'un logiciel "standard", devant
   être appelée par l'interruption 1Ch *)

if change_1C = true then inline($CD/inter_1C_detournee);

(* incrémentation du compteur de temps entre deux appuis successifs sur le
   bouton de la souris *)

if (clic_bouton <> 0) AND (tps_bouton < tps_inter_clic)
    then tps_bouton := tps_bouton + 1
    else tps_bouton := tps_inter_clic + 1;


(* tests sur les verrous avant de pouvoir exécuter le programme "aidami" *)

if ((tps_bouton > tps_inter_clic) AND (clic_bouton <> 0))
   OR (mem[$0000:$0188] <> 0) OR (intro_data_clavier = true)
    then begin
         if exclusion_clavier = 1
            then begin
                 if exclusion_OS = 1
                    then begin
                      if exclusion_ES = 1
                      then begin
                         if exclusion_souris = 1
                         then begin
                              if exclusion_aidami = 1
                                 then begin

                                      (* changement de pile *)
```

```
                                    inline($16/$58/$A3/ass_prog_aidami/$A1/nss_prog_aidami/$50/$17);
                                    inline($94/$A3/asp_prog_aidami/$95/$A3/abp_prog_aidami);
                                    inline($B8/pile_aidami/$94/$B8/pile_aidami/$95);

                                    attendre(exclusion_aidami);
                                    ofs_entree := ofs(sortie_directe) + 4;

                                    (* envoi du signal "EOI" vers le 8259 *)

                                    port[$20] := $60;
                                    autoriser_interrupt;
(*!!!
```

```
init_lire_vect_interrupt;

(* initialisation *)

erreur := 0;
init(erreur);

if erreur = 0 then begin
                   memw[$0000:data_seg] := Dseg;
                   memw[$0000:$0184] := 1;

                   (* écriture des adresses des routines "inter i" dans les
                      vecteurs d'interruption du système d'exploitation *)

                   init_ecrire_vect_interrupt;

                   writeln('programme "AIDAMI" chargé en mémoire centrale');

                   lect_fich(false);

                   (* mise en résidance du programme en mémoire centrale *)

                   regs.ax := $3100;
                   regs.dx := (Dseg - Cseg) + nbre_para_seg_donnees + $1000
                              + taille_heap;
                   intr($21,regs);
                   end
              else begin
                   clrscr;
                   gotoxy(3,21);
                   writeln('Présence d''une erreur!!! ');
                   gotoxy(3,22);
                   writeln('Erreur : ',erreur);
                   lect_fich(false);
                   end;
end.
```

```
program aida_go;
(* programme de lancement du programme "aidami" *)

var sortie : integer;
    regs : record
           ax,bx,cx,dx : integer;
           end;

begin
regs.ax := 0;
regs.bx := 2;
intr($33,regs);

hires;
hiresColor(white);

memw[$0000:$0188] := 1;
memw[$0000:$0184] := 1;

sortie := 1;
while sortie = 1 do
   begin
   sortie := mem[$0000:$0184];
   end;
end.
```

```
program creer_fich;
(* programme permettant de créer et de mettre à jour les fichiers utilisés par
   le programme "aidami" *)

const return = ^m;
      bs = ^h;
      bell = ^g;
      objet_min = 0;
      objet_max = 5;
      WrkStr = 80;


type strWrk = string[WrkStr];
     character = set of char;
     obj_fenetre = ^objet_fen;
     objet_fen = record
                 num_objet : integer;
                 valeur_string : strWrk;
                 coord : array[1..4] of integer;
                 couleur_caractere : integer;
                 couleur_fond : integer;
                 etat_bouton : integer;
                 num_gr_bouton : integer;
                 num_action : integer;
                 next : obj_fenetre;
                 end;
     code_teleph = ^teleph_rec;
     teleph_rec = record
                  tps_inter_chiffre : integer;
                  tps_inter_impuls : integer;
                  tps_duree_impuls : integer;
                  port_sortie : integer;
                  num_bit_imp : integer;
                  next : code_teleph;
                  end;

var fenetre : objet_fen;
    filename : strWrk;
    exist : boolean;
    sortie : char;
    f : file of objet_fen;
    f_teleph : file of teleph_rec;
    teleph : teleph_rec;
    caractere : character;
    carac : char;



      PROCEDURE clearline(u,v,lg : integer);
     {-------------------------------------}
     {assure l'effacement des caracteres erronnes introduits au clavier et le repositionnement du curseur}
     {à l'endroit initial de la lecture}

      var i : integer;

      begin
      write(bell);
      gotoxy(u,v);
      for i := 1 to lg do
        begin
        write(' ');
        end;
```

```
gotoxy(u,v);
end;



PROCEDURE correction(var strings : strWrk ; strin : char ; lgueur,x1,y : integer);
{-------------------------------------------------------------------------------}
{lecture caractere par caractere d'un string; renvoie ce string}

var x : integer;  {compteur du nombre de lettres dont le string est formé}
   back : boolean;

begin
back := false;
if strin <> bs then write(strin)
               else back := true;
strings := strin;
x := 1;
read(kbd,strin);
if (strin = bs) and (back = true) then write(' ');
while (strin <> return) or (x = 0) do
  begin
  if (x < lgueur) or (strin = bs) then
     begin
     if (strin <> bs) then begin                        {memorisation du string caractere par caractere}
                           gotoxy(x1+x,y);
                           write(strin);
                           strings := strings + strin;
                           x := x + 1;
                           end
                      else if x > 0 then begin          {permet de supprimer un caractere du string a la fois}
                                         write(bs,' ',bs);
                                         x := x - 1;
                                         strings := copy(strings,1,x);
                                         end;
     end;
  read(kbd,strin)
  end;
end;



PROCEDURE introString(var strings : strWrk ; lgueur : integer);
{-------------------------------------------------------------}
{lecture d'un string introduit au clavier; renvoie la valeur de ce string }

var strin : char;   {premier caractere du string introduit au clavier}
    x,y : integer;  {coordonnees du curseur sur l'ecran}

begin
x := wherex;
y := wherey;
read(kbd,strin);
while strin = return do
  begin
  clearline(x,y,lgueur);
  read(kbd,strin);
  end;
correction(strings,strin,lgueur,x,y);
writeln;
end;
```

```
PROCEDURE IntroEntier(var Entier : integer ; min,max,lgueur : integer);
{-------------------------------------------------------------------}
{lecture d'un entier introduit au clavier; renvoie la valeur de cet entier}
{cet entier doit être compris dans l'intervalle "min-max" et être formé de "lgueur" chiffres maximum }

var
 code,                {variable detectant si la valeur introduite dans le string est bien un reel}
 x,y : integer;       {coordonnees du curseur }
 entree : strWrk; {string introduit au clavier devant etre converti en reel}

begin
x := wherex;
y := wherey;
introstring(entree,lgueur);
val(entree,Entier,code);
while ((code <> 0) or (entree = '-') or (Entier < min) or (Entier > max)) do
  begin
  clearline(x,y,lgueur);
  introstring(entree,lgueur);
  val(entree,entier,code);
  end;
end;



PROCEDURE IntroCharacter(var caractere : char ; ensemble : character ; lgueur : integer);
{-------------------------------------------------------------------------------------}
{lecture d'un caractere introduit au clavier ; retourne ce caractere }
{lgueur = nombre de caractères effacés à l'écran sur la ligne courante}
{lorsque l'on a introduit un caractère n'appartenant pas à "ensemble }

var
  x,y : integer;    {coordonnees de la position actuelle du curseur sur l'ecran}
begin
x := wherex;
y := wherey;
readln(caractere);
while (not(caractere in ensemble)) do
  begin
  clearline(x,y,lgueur);
  readln(caractere);
  end;
end;



PROCEDURE rep(var sortie :char);
{------------------------------}

{lecture d'un caractere introduit au clavier, celui-ci etant limite aux valeurs Y ou N }
{renvoie la valeur de ce caractere }

begin
caractere := ['Y','y','N','n'];
IntroCharacter(sortie,caractere,1);
end;
```

```
 PROCEDURE correcintroString(var strings : strWrk ; lgueur : integer);
{-----------------------------------------------------------------}
{lecture d'un string introduit au clavier; renvoie la valeur de ce string }

var strin : char;   {premier caractere du string introduit au clavier}
    x,y : integer;  {coordonnees du curseur sur l'ecran}

 begin
 read(kbd,strin);
 if strin <> return then correction(strings,strin,lgueur,x,y);
 writeln;
 end;



 PROCEDURE correcIntroEntier(var Entier : integer ; min,max,lgueur : integer);
{-------------------------------------------------------------------------}
{lecture d'un entier introduit au clavier; renvoie la valeur de cet entier}
{cet entier doit être compris dans l'intervalle "min-max" et être formé de "lgueur" chiffres maximum }


 var
  code,               {variable detectant si la valeur introduite dans le string est bien un reel}
  x,y : integer;      {coordonnees du curseur }
  entree : strWrk; {string introduit au clavier devant etre converti en reel}

 begin
 x := wherex;
 y := wherey;
 str(entier, entree);
 correcintrostring(entree,lgueur);
 val(entree,Entier,code);
 while ((code <> 0) or (entree = '-') or (Entier < min) or (Entier > max)) do
   begin
   clearline(x,y,lgueur);
   correcintrostring(entree,lgueur);
   val(entree,entier,code);
   end;
 end;


procedure exist_fich( filename : strWrk; var exist : boolean);
(*****************************************************************)
(* vérification de l'existence d'un fichier *)

begin
assign(f, filename);    (*$I-*)
reset(f);
exist := (ioresult = 0);
close(f);               (*$I+*)
end;



procedure intro_donnees;
(***********************)
(* introduction de données à mémoriser dans un fichier de fenêtres *)

begin
    with fenetre do
```

```
begin
write('Entrez le type de l''objet : ');
introentier(num_objet,objet_min,objet_max,2);
if num_objet in [0,2] then begin
                           write('Entrez la valeur de la chaine de caractères : ');
                           introstring(valeur_string,WrkStr);
                           end;
case num_objet of
    0 : begin
          write('Entrez la coordonnée "x" du coin supérieur gauche : ');
          introentier(coord[1],0,79,2);
          end;
    2,3 : begin
          write('Entrez la coordonnée en "x" du début de la chaine : ');
          introentier(coord[1],0,79,2);
          end;
    1,5 : begin
          write('Entrez la coordonnée "x" du coin supérieur gauche : ');
          introentier(coord[1],0,1000,4);
          end;
    end;
case num_objet of
    0 : begin
        write('Entrez la coordonnée "y" du coin supérieur gauche : ');
        introentier(coord[2],0,199,3);
        end;
    2,3 : begin
          write('Entrez la coordonnée en "y" du début de la chaine : ');
          introentier(coord[2],0,24,2);
          end;
    1,5 : begin
          write('Entrez la coordonnée "y" du coin supérieur gauche : ');
          introentier(coord[2],0,1000,4);
          end;
     end;
case num_objet of
   0 : begin
       write('Entrez la coordonnée "x" du coin inférieur droit : ');
       introentier(coord[3],0,79,2);
       end;
   1,5 : begin
         write('Entrez la coordonnée "x" du coin inférieur droit : ');
         introentier(coord[3],0,1000,4);
         end;
    end;
case num_objet of
   0 : begin
       write('Entrez la coordonnée "y" du coin inférieur droit : ');
       introentier(coord[4],0,199,3);
       end;
   1,5 : begin
         write('Entrez la coordonnée "y" du coin inférieur droit : ');
         introentier(coord[4],0,1000,4);
         end;
    end;
write('Entrez la couleur du caractère : ');
introentier(couleur_caractere,0,15,2);
write('Entrez la couleur de fond : ');
introentier(couleur_fond,0,15,2);
if num_objet = 0 then begin
                      write('Entrez le type de la fenêtre : ');
                      introentier(etat_bouton,0,2,1);
                      end;
```

```
        case num_objet of
            1,5 : begin
                  write('Entrez l''état du bouton : ');
                  introentier(etat_bouton,0,1,1);
                  write('Entrez le groupe du bouton : ');
                  introentier(num_gr_bouton,0,9999,4);
                  end;
            3 : begin
                write('Entrez le numéro de la chaîne de caractères : ');
                introentier(num_gr_bouton,1,99,2);
                end;
            end;
        if num_objet in [0,1,5] then begin
                                     write('Entrez le numéro de l''action : ');
                                     introentier(num_action,0,9999,4);
                                     end;
        next := nil;
        end;
end;



procedure intro_donnees_cor;
(**************************)
(* correction de données introduites dans un fichier de fenêtres *)

begin
    with fenetre do
        begin
        write('Entrez le type de l''objet (',num_objet,') : ');
        correcintroentier(num_objet,objet_min,objet_max,2);
        if num_objet in [0,2] then begin
                                   write('Entrez la valeur de la chaine de caractères (',valeur_string,') : ');
                                   correcintrostring(valeur_string,WrkStr);
                                   end;
        case num_objet of
            0 : begin
                  write('Entrez la coordonnée "x" du coin supérieur gauche (',coord[1],') : ');
                  correcintroentier(coord[1],0,79,2);
                  end;
            2,3 : begin
                  write('Entrez la coordonnée en "x" du début de la chaine (',coord[1],') : ');
                  correcintroentier(coord[1],0,79,2);
                  end;
            1,5 : begin
                  write('Entrez la coordonnée "x" du coin supérieur gauche (',coord[1],') : ');
                  correcintroentier(coord[1],0,1000,4);
                  end;
            end;
        case num_objet of
            0 : begin
                write('Entrez la coordonnée "y" du coin supérieur gauche (',coord[2],') : ');
                correcintroentier(coord[2],0,199,3);
                end;
            2,3 : begin
                  write('Entrez la coordonnée en "y" du début de la chaine (',coord[2],') : ');
                  correcintroentier(coord[2],0,24,2);
                  end;
            1,5 : begin
                  write('Entrez la coordonnée "y" du coin supérieur gauche (',coord[2],') : ');
                  correcintroentier(coord[2],0,1000,4);
                  end;
             end;
```

```
        case num_objet of
           0 : begin
               write('Entrez la coordonnée "x" du coin inférieur droit (',coord[3],') : ');
               correcintroentier(coord[3],0,79,2);
               end;
           1,5 : begin
                 write('Entrez la coordonnée "x" du coin inférieur droit (',coord[3],') : ');
                 correcintroentier(coord[3],0,1000,4);
                 end;
            end;
        case num_objet of
           0 : begin
               write('Entrez la coordonnée "y" du coin inférieur droit (',coord[4],') : ');
               correcintroentier(coord[4],0,199,3);
               end;
           1,5 : begin
                 write('Entrez la coordonnée "y" du coin inférieur droit (',coord[4],') : ');
                 correcintroentier(coord[4],0,1000,4);
                 end;
            end;
        write('Entrez la couleur du caractère (',couleur_caractere,') : ');
        correcintroentier(couleur_caractere,0,15,2);
        write('Entrez la couleur de fond (',couleur_fond,') : ');
        correcintroentier(couleur_fond,0,15,2);
        if num_objet = 0 then begin
                              write('Entrez le type de la fenêtre (',etat_bouton,') : ');
                              correcintroentier(etat_bouton,0,2,1);
                              end;
        case num_objet of
            1,5 : begin
                  write('Entrez l''état du bouton (',etat_bouton,') : ');
                  correcintroentier(etat_bouton,0,1,1);
                  write('Entrez le groupe du bouton (',num_gr_bouton,') : ');
                  correcintroentier(num_gr_bouton,0,9999,4);
                 end;
            3 : begin
                write('Entrez le numéro de la chaîne de caractères (',num_gr_bouton,') : ');
                correcintroentier(num_gr_bouton,1,99,2);
                end;
            end;
        if num_objet in [0,1,5] then begin
                                     write('Entrez le numéro de l''action (',num_action,') : ');
                                     correcintroentier(num_action,0,9999,4);
                                     end;
        next := nil;
        end;
end;



procedure intro_donnees_creer;
(*****************************)
(* création d'un nouveau record de fenêtre pour le mémoriser dans un fichier
   de fenêtres *)

var sortie_intro : char;
    num : integer;

begin
num := 1;
sortie_intro := 'y';
while sortie_intro in ['Y','y'] do
    begin
```

```
    clrscr;
    writeln('Objet numéro ',num);
    writeln;

    intro_donnees;

    write(f,fenetre);
    num := num + 1;
    writeln;
    write('Désirez-vous entrer un autre objet? (Y/N) : ');
    rep(sortie_intro);
    end;
end;



procedure creer_fichier(filename : strWrk);
(*****************************************)
(* création d'un fichier *)

var sortie_creer_fichier : char;
    creation : char;

begin
writeln;
sortie_creer_fichier := 'y';
writeln('Ce fichier n''existe pas sur le disque;');
write('désirez-vous créer le fichier "',filename,'" ? (Y/N) : ');
rep(creation);
if creation in ['Y', 'y']
    then begin
         assign(f,filename);
         rewrite(f);
         intro_donnees_creer;
         close(f);
         end;
end;



procedure intro_donnees_corriger;
(*******************************)
(* lecture d'un record d'un fichier de fenêtres en vue de le corriger *)


var sortie_intro : char;
    num : integer;

begin
sortie_intro := 'y';
while sortie_intro in ['Y','y'] do
    begin
    clrscr;
    write('Entrez le numéro de l''objet à modifier : ');
    introentier(num,1,filesize(f) + 1 ,3);
    writeln;
    seek(f,num - 1);
    if num <> (filesize(f) + 1) then read(f,fenetre);
    if num <> (filesize(f) + 1) then intro_donnees_cor
                                else intro_donnees;
    seek(f,num - 1);
    write(f,fenetre);
    writeln;
```

```
    write('Désirez-vous entrer un autre objet? (Y/N) : ');
    rep(sortie_intro);
    end;
end;



procedure corriger_fichier(filename : strWrk);
(*******************************************)

var sortie_corriger : char;

begin
write('Ce fichier existe déjà; désirez-vous effectuer des corrections? (Y/N) : ');
rep(sortie_corriger);
if sortie_corriger in ['Y','y']
    then begin
         assign(f,filename);
         reset(f);
         intro_donnees_corriger;
         close(f);
         end;
end;


procedure fich_fenetre;
(**********************)

begin
    clrscr;
    write('Entrez le nom du fichier de fenêtre : ');
    readln(filename);
    exist_fich(filename,exist);
    if exist = true then corriger_fichier(filename)
                    else creer_fichier(filename);
    writeln;
end;



procedure in_data_teleph;
(************************)
(* introduction des données relatives au téléphone *)

begin
clrscr;
with teleph do
    begin
    write('Entrez le temps d''attente entre l''envoi de deux chiffres en msec : ');
    introentier(tps_inter_chiffre,0,9999,4);
    write('Entrez le temps entre deux impulsions en msec : ');
    introentier(tps_inter_impuls,0,999,3);
    write('Entrez le temps de durée d''une impulsion en msec : ');
    introentier(tps_duree_impuls,0,999,3);
    write('Entrez le numéro du port de sortie en décimal : ');
    introentier(port_sortie,0,maxint,5);
    write('Entrer le numéro du bit du port vers lequel le signal doit être envoyé : ');
    introentier(num_bit_imp,0,7,1);
    next := nil;
    end;
write(f_teleph,teleph);
end;
```

```
procedure in_data_cor_teleph;
(***************************)
(* correction de données mémorisées dans le fichier de téléphone *)

begin
clrscr;
with teleph do
    begin
    writeln('Entrez le temps d''attente entre l''envoi de deux ');
    write('chiffres en msec (',tps_inter_chiffre,') : ');
    correcintroentier(tps_inter_chiffre,0,9999,4);
    write('Entrez le temps entre deux impulsions en msec (',tps_inter_impuls,') : ');
    correcintroentier(tps_inter_impuls,0,999,3);
    write('Entrez le temps de durée d''une impulsion en msec (',tps_duree_impuls,') : ');
    correcintroentier(tps_duree_impuls,0,999,3);
    write('Entrez le numéro du port de sortie en décimal (',port_sortie,') : ');
    correcintroentier(port_sortie,0,maxint,5);
    write('Entrer le numéro du bit du port vers lequel le signal doit être envoyé (',num_bit_imp,') : ');
    correcintroentier(num_bit_imp,0,7,1);
    next := nil;
    end;
write(f_teleph,teleph);
end;



procedure corriger_fich_teleph;
(*****************************)

var sortie_corriger : char;

begin
write('Ce fichier existe déjà; désirez-vous effectuer des corrections? (Y/N) : ');
rep(sortie_corriger);
if sortie_corriger in ['Y','y']
    then begin
         assign(f_teleph,filename);
         reset(f_teleph);
         in_data_cor_teleph;
         close(f_teleph);
         end;
end;



procedure creer_fich_teleph;
(**************************)
(* création d'un fichier de téléphone *)

var sortie_creer_fichier : char;
    creation : char;

begin
writeln;
sortie_creer_fichier := 'y';
writeln('Ce fichier n''existe pas sur le disque;');
write('désirez-vous créer le fichier "',filename,'" ? (Y/N) : ');
rep(creation);
if creation in ['Y', 'y']
    then begin
         assign(f_teleph,filename);
```

```
        rewrite(f_teleph);
        in_data_teleph;
        close(f_teleph);
        end;
end;



procedure fich_telephone;
(************************)

begin
clrscr;
write('Entrez le nom du fichier de téléphone : ');
readln(filename);
exist_fich(filename,exist);
if exist = true then corriger_fich_teleph
                else creer_fich_teleph;
writeln;
end;




begin                (**** main ****)

sortie := 'n';
while sortie = 'n'
    begin
    clrscr;
    write('Entrez le type de fichier désiré : (F)enetre/(T)éléphone/(S)ortie : ');
    readln(carac);
    case carac of
       'F','f' : fich_fenetre;
       'T','t' : fich_telephone;
       'S','s' : sortie := 'y';
       end;
    end;
end.
```