**MASTER EN SCIENCES INFORMATIQUES**

# Comportement des systèmes d'exploitation a mémoire virtuelle paginée
# étude des modes de gestion et simulateur de système a mémoire virtuelle paginée

verhaeghe, René

*Award date:*
1978

*Awarding institution:*
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
INSTITUT D'INFORMATIQUE
NAMUR

# COMPORTEMENT DES SYSTEMES D'EXPLOITATION A MEMOIRE VIRTUELLE PAGINEE : ETUDE DES MODES DE GESTION ET SIMULATEUR DE SYSTEME A MEMOIRE VIRTUELLE PAGINEE

Mémoire présenté par
**VERHAEGHE René**
pour l'obtention du diplôme
de Licencié et Maître en
Informatique

Année Académique
1977 – 1978

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
INSTITUT D'INFORMATIQUE
NAMUR

# COMPORTEMENT DES SYSTEMES D'EXPLOITATION A MEMOIRE VIRTUELLE PAGINEE : ETUDE DES MODES DE GESTION ET SIMULATEUR DE SYSTEME A MEMOIRE VIRTUELLE PAGINEE

Mémoire présenté par
**VERHAEGHE René**
pour l'obtention du diplôme
de Licencié et Maître en
Informatique

Année Académique
1977 – 1978

*Nous tenons à remercier Madame Noirhomme-Fraiture, Monsieur Ramaekers et Monsieur Paulus, pour l'aide qu'ils nous ont apporté pendant ce travail.*

# TABLE DES MATIERES

. . .

# INTRODUCTION

Depuis quelques années, de nombreux systèmes d'exploitation se sont tournés vers la technique de la mémoire virtuelle pour améliorer leurs performances.

Dans les premiers ordinateurs, la monoprogrammation était de rigueur : la mémoire centrale ne pouvait contenir qu'un seul programme. Par la suite, elle put contenir plusieurs travaux, chacun d'eux occupant une partie continue de la mémoire centrale (partition), partie de taille fixe ou non. Chaque travail devait disposer d'une structure d'adressage particulière pour pouvoir occuper la partie de la mémoire qui lui était attribuée; cette adresse était calculée à l'exécution en ajoutant à une valeur liée à la position de l'instruction ou de la donnée référencée, une valeur de base fonction de la position de la partition dans la mémoire. Les adresses contenues dans le travail devenaient donc abstraites, amenant la notion d'espace d'adresse (ensemble des adresses référencées par un travail). On peut déjà voir un effet double : amélioration du fonctionnement par augmentation du nombre de travaux, et détérioration par besoin de gestion des conversions d'adresses et de la présence simultanée de travaux.

Dans les systèmes à mémoire virtuelle, chaque travail ne doit plus figurer intégralement en mémoire centrale, mais bien sur une autre mémoire moins chère et plus volumineuse appelée mémoire de réserve ou auxiliaire. Le travail ne dispose plus alors en mémoire centrale que de quelques pages. L'adresse de l'espace d'adresse doit être convertie en adresse de la mémoire centrale, conversion plus longue et plus complexe que dans le cas précédent. Si l'adresse référencée ne figure pas en mémoire centrale, il se produit un défaut de page; on demande alors à un gérant de la mémoire l'allocation d'un élément de mémoire et le transfert de la partie de travail qui contient l'adresse demandée. Le gérant de la mémoire a donc essentiellement pour but de fournir un cadre donné; il doit donc pouvoir le désigner, ce qui implique des choix : attribuer un cadre libre, ou réutiliser un cadre attribué, mais jugé mal utilisé.

Le plus petit volume de mémoire centrale alloué à chaque travail permet un niveau de multiprogrammation plus élevé. Mais la conversion des adresses, la gestion de la mémoire et les attentes consécutives aux défauts de page font perdre une partie de l'efficacité.

Nous ne nous sommes pas intéressés ici au coût en temps processeur nécessité par la gestion d'un système, mais uniquement au bon écoulement des travaux. Ce bon écoulement peut être mesuré par le nombre de travaux prêts à travailler, le nombre et la distribution des défauts de page ou des temps d'attente. Les éléments fondamentaux qui influent sur ces données sont les algorithmes de gestion de la mémoire, le nombre et la taille des travaux concurrent, la taille des mémoire centrales et auxiliaires.

Pour pouvoir analyser l'influence de ces éléments, un système analytique de simulation est inutilisable, vu la complexité des interactions mise en jeu. Nous avons donc décidé de créer un simulateur. Celui-ci doit contenir un modèle du chemin des travaux, des gérants de la mémoire, des entrées-sorties et des travaux. Pour produire des défauts de page, il est nécessaire de générer une suite des références à l'espace d'adresse fait par un travail; donc de disposer d'un modèle de comportement de programme.

Le chapitre I analyse les éléments indispensables à un système de gestion à mémoire virtuelle pour ce qui est lié à la mémoire (gestion des demandes de page, des travaux, etc ...)

Dans le chapitre II, nous avons étudié divers systèmes existants afin de montrer ce que contenaient les éléments en question, ainsi que les interactions entre eux. Les systèmes utilisés sont des systèmes à mémoire virtuelle paginée, Siemens BS2000, IBM VS1 et MVS. Nous nous sommes limités à ces systèmes vu la documentation disponible. On y voit que le mode de gestion adopté tient moins au but du système qu'à l'historique des générations successives qui ont amené à ce système.

Le chapitre III décrit le modèle du simulateur de système, les données nécessaires comme entrée, les méthodes statistique d'analyse et de validation des données. Il décrit brièvement l'application de ce modèle au cas du Siemens BS2000 qui a été choisi vu la meilleure connaissance que nous en avions, et la documentation suffisante.

En annexe, figure le texte complet du simulateur appliqué au système Siemens BS2000.

# CHAPITRE I :

## STRUCTURES DES MÉMOIRES VIRTUELLES

## 1.1. INTRODUCTION

A l'origine de l'informatique, chaque programme exécuté devait être chargé en mémoire centrale à une adresse fixée par le programmeur. Par la suite, la méthode de l'adressage relatif à un point de départ défini à la compilation permit de reloger chaque programme dans un volume quelconque, mais continu de la mémoire, et pouvant contenir tout le programme. C'est à cette occasion qu'apparut pour la première fois une distinction entre l'adresse physique et les adresses apparaissant dans le programme.

D'autres systèmes ont poussé cette séparation à l'extrême de sorte que, à tout moment de l'exécution du programme, le processeur travaille à partir d'adresse de l'espace d'adressage qu'il traduit en adresses physiques permettant ainsi l'éclatement du programme dans la mémoire centrale en volumes de taille fixe (cadres) ou variables (segments). Ces derniers sont liés à la structure logique du programme.

Les systèmes à mémoire virtuelle ont poussé cette évolution plus loin encore, en ne gardant en mémoire centrale que la part du programme qui y est utile pour l'avancement du travail à exécuter. Pour disposer rapidement du reste du programme, celui-ci est stocké sur une troisième mémoire de masse, appelée mémoire auxiliaire ou de réserve.

Un tel système dispose donc pour la mémoire de trois espaces distincts : espace de mémoire centrale, de mémoire auxiliaire et d'adressage. Ces trois pôles constituent la base de la structure d'une mémoire virtuelle, à laquelle s'ajoute alors un système de gestion de ces espaces, de leurs liens et de la réponse aux demandes des divers utilisateurs de la mémoire.

Pour chacun de ces éléments, divers modes généraux de construction et de fonctionnement peuvent être définis. Ce chapitre a comme but de montrer un certain nombre des solutions pensables ou exécutées pour réaliser un système à mémoire virtuelle.

## 1.2. ESPACE D'ADRESSAGE

L'espace d'adressage peut être défini comme l'ensemble des adresses référencées dans un travail. Dans les ordinateurs de la première génération ces adresses correspondaient à l'adresse physique. Si on ne désire avoir qu'une partie d'un travail en mémoire centrale, là ou il y a un cadre libre, il est évident que cette correspondance ne peut plus exister. On a donc trois ensembles d'adresses : les adresses en mémoire centrale, formant l'espace de mémoire centrale, les adresses de la mémoire de masse, autrement dit la position des cadres sur les tambours ou espace de mémoire auxiliaire, et finalement les adresses du programme, espace d'adressage. Les adresses de cet espace ne correspondent donc en rien à une position physique, mais uniquement une position logique, un déplacement par rapport au début du travail.

Afin d'exécuter un processus, il est nécessaire de convertir les adresses d'un espace en une valeur dans une autre. Cela se fait au moyen de tables de conversion qui à une adresse d'un espace fait correspondre une autre adresse, si elle existe.

L'espace d'adresse doit être géré comme un espace quelconque. Sa taille maximum est donné par la plus grande adresse tolérable par l'ordinateur ($2^{23}$ pour Siemens, $2^{24}$ pour IBM, etc ...)

Dans cet espace on peut loger soit un seul travail soit tous les travaux. On a alors un espace d'adresse multiple (IBM/MVS, Siemens 4004) ou unique (IBM/VS1, VS2).

L'espace d'adresse unique peut contenir plusieurs travaux. Il faut donc assurer l'étanchéité entre ceux-ci. Elle est assurée par la propriété qu'a chaque travail d'un élément continu de l'espace adresse. Il y a donc une gestion de cet espace comme il y a une gestion de la mémoire centrale dans les systèmes sans mémoires virtuelles.

Il faut attribuer à chaque travail un espace donné, essayer de placer de nouveaux travaux dans les emplacements vides; on doit donc choisir un mode de gestion, soit partition fixe (IBM/VS1), soit partition variable (IBM/VS2). Ceci implique donc de fortes limitations quant à la taille des travaux, la souplesse du système et la variation de la taille des travaux au cours de l'exécution.

Les systèmes d'exploitation à espaces d'adressage multiples attribuent à chaque travail un espace complet, ce qui élimine les problèmes de concurrence entre travaux dans l'espace d'adressage, ainsi que ceux concernant la sécurité des données.

Le choix entre l'un ou l'autre de ces systèmes se fait donc essentiellement en fonction du type de travail à assurer par l'ordinateur : soit travaux de taille connue ou prévisible, soit travaux de volume difficilement prévisible (systèmes tournés vers l'interactif).

Ce choix n'a pas d'influence sur les procédés de gestion des transferts de données entre les espaces.

---

## 1.3. MEMOIRE CENTRALE

### 1.3.1. Introduction

La mémoire centrale est le lieu où doit se trouver une instruction pour pouvoir être exécutée par le processeur. On devra donc pouvoir facilement y amener ou en retirer un élément de programme inutile pour le remplacer par un autre momentanément plus intéressant. Il faut donc définir quel type d'élément servira de base : élément de taille fixe, non lié à la structure du travail (cadre), élément de taille variable, lié à la structure du programme, ou un système mixte utilisant les deux techniques précédentes.

Le second point important concerne la taille de la mémoire centrale, nécessaire pour faire fonctionner le système de manière satisfaisante.

### 1.3.2. Segmentation

La segmentation consiste à considérer le travail comme étant séparable en unités logiques qu'il est préférable de ne pas décomposer. Cette séparation est évidemment à opérer par le programmeur.

Les transferts entre la mémoire centrale et la mémoire auxiliaire ne se fera que par segments entiers. Chaque transfert sera donc plus long, plus coûteux en place dans la mémoire centrale. Chaque segment de travail est conçu de façon à faire travailler l'ordinateur pendant un temps très long sans nécessiter de nouveaux transferts. La segmentation aura dès lors une influence sur la conception et la structuration du programme.

La segmentation a comme effet de rendre plus complexe la gestion des espaces de mémoire, qui s'apparente à une gestion de mémoire à partition variable, avec tous les problèmes d'évolution des partitions, et de choix de priorité (priorité aux segments de gros volume ou à ceux de faible volume).

On a donc une gestion plus complexe des organisations particulières des travaux qui peuvent amener une meilleure efficacité, la structure physique du travail épousant sa structure logique.

### 1.3.3. Pagination

La pagination prend le contrepied exact de la segmentation. Elle est totalement indépendante de la structure du travail; le volume de base de transfert est de taille fixe (cadre); la gestion de la mémoire centrale devient très simple.

Les espaces de mémoire sont partagés en ensembles de taille fixe. Leur nombre, connu, permet de disposer d'une table permanente qui précise l'état de chacun de ces cadres.

L'inconvénient du système est bien évidemment le manque de relation entre le contenu du cadre et la structure logique du programme, qui peut amener de nombreux défauts de page.

L'avantage fondamental tient dans la simplicité de gestion des divers espaces de mémoire, les volumes allouables étant de taille fixe, et l'état de chacun de ces cadres connu dans une table de gestion.

### 1.3.4. Système mixte (Segmentation pagination)

On pourrait également concevoir un système essayant de réconcilier la pagination et la segmentation; chaque segment est décomposé en unité de bases. On en vient alors à un système où la mémoire physique est organisée en cadres, mais où les transferts se font par segment. On garde la gestion simple de la pagination et la meilleure efficacité des segments.

Il est à noter que dès lors les structures des espaces physiques (mémoire centrale et auxiliaire) diffèreront radicalement de celles de l'espace virtuel d'adressage, les premiers étant essentiellement paginés et le dernier segmenté.

### 1.3.5. Taille de la mémoire centrale

Chaque programme a besoin d'un certain volume de la mémoire centrale pour pouvoir s'exécuter sans trop faire de transferts. Ce volume peut varier selon le type de programme (working set du volume de travail).

Dès lors le nombre de programmes disposant de cadres (dits programmes actifs) sera limité par la taille de la mémoire et par celle de leur working set.

Cette implication servira en général à fixer le nombre de travaux actifs au cours du fonctionnement du système.

Prise en sens inverse (la taille de la mémoire est définie par le nombre de programmes actifs et leur working-set), elle servira à définir à priori une taille minimale à la mémoire centrale en fonction de diverses expériences précédentes.

### 1.3.6. Taille des cadres

Idéalement, si on connaissait l'évolution exacte de chaque travail, il suffirait d'avoir en mémoire les quelques instructions à venir. Cette connaissance absolue étant impossible, on a besoin en mémoire centrale d'un ensemble d'instructions, qui est séparée en cadres dans le cas de la pagination.

Il faut donc établir un équilibre entre la taille de chaque cadre et le nombre de transferts. Si le cadre est petit, il est fort probable que l'instruction nécessaire manquera plus souvent qu'avec un cadre plus grand; il y aura donc plus de défauts de page, mais ils seront de durée plus faible, le temps de transfert étant fonction du volume à transférer. De plus, il y aura plus de cadres disponibles, et donc moins d'urgence à réutiliser des cadres inutilisés.

Cet équilibre est donc fonction de la connaissance de l'histoire des programmes que l'on a, et des interactions entre celles-ci.

---

## 1.4. MEMOIRE AUXILIAIRE (DE RESERVE)

Cette mémoire doit contenir tous les travaux en cours d'exécution, et doit donc être de forte capacité. De plus, la fréquence importante des transferts impose que son accès soit rapide, d'où les solutions habituelles, tambours, système le plus rapide et le plus cher, ou disques, moins rapide et moins cher.

Pour accélérer les transferts, il est intéressant de pouvoir les effectuer en parallèle, avec plusieurs supports physiques. S'il s'agit de disques, on partage en général la mémoire de réserve sur plusieurs disques qui contiennent aussi d'autres données stockées. On accorde alors la priorité aux demandes de pages sur ces disques par rapport aux autres demandes.

La structure est la même que celle de la mémoire centrale, et sera segmentée ou paginée selon la structure de la mémoire centrale.

La taille globale est fonction du nombre de programmes à stocker et de leur taille moyenne. Le nombre de programmes est à calculer à partir d'une estimation du nombre moyen de travaux actifs, et de la proportion entre travaux actifs et non-actifs, qui n'ont aucun cadre en mémoire centrale. A ce volume global, il faut ajouter l'ensemble des programmes composant le système et les utilitaires les plus courants, les autres étant stockés en une mémoire de masse.

---

## 1.5. GERANT DE LA MEMOIRE

### 1.5.1. Introduction

Par gérant de la mémoire, nous entendons l'ensemble des programmes du système chargé de la gestion des mémoires centrales, auxiliaires, et de l'espace d'adressage, des liens entre ces espaces, et chargé de vérifier la bonne utilisation des cadres alloués aux travaux.

La gestion propre des espaces de mémoire est assuré au travers de tables de deux types. Le premier correspond aux tables donnant l'état d'un élément (segment ou page) d'un des espaces : occupation, occupant, usage fait de l'élément. Le second type est constitué des tables de conversion qui permettent de faire le lien entre les espaces. Elles sont en général au nombre de deux reliant par paire les espaces de mémoire centrale, auxiliaire et l'espace d'adressage.

Cette gestion consiste dans le maintien à jour des informations présentes dans ces tables, mise à jour qui s'effectue au fur et à mesure des changements effectués.

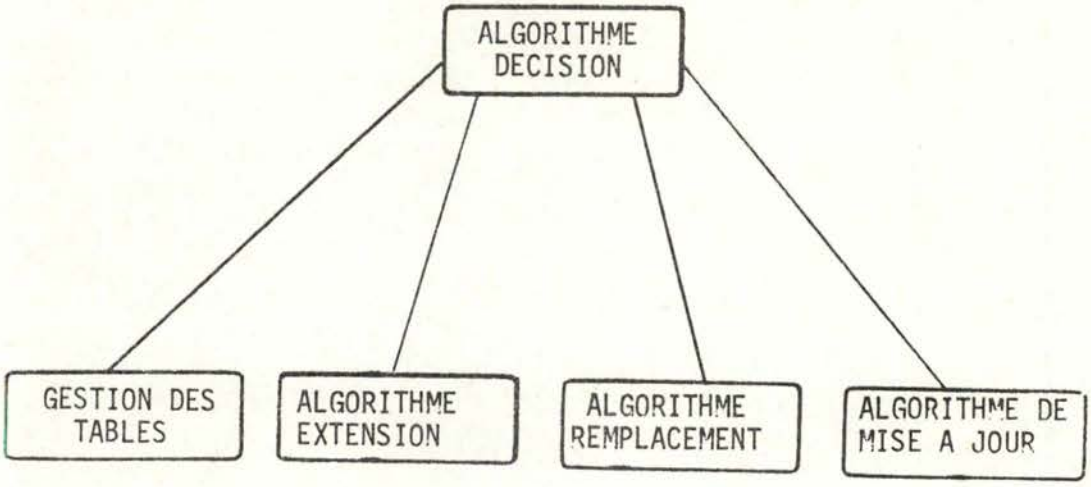
La partie fondamentale de la gestion de la mémoire est constituée par les algorithmes de gestion des cadres alloués aux travaux et de réponse aux défauts de page.

### 1.5.2. Défauts de page

Chaque travail a besoin en mémoire centrale des instructions qu'il doit exécuter. Lorsqu'elles n'y seront pas présentes, il y a défaut de page, autrement dit demande au gérant de la mémoire d'un volume de la mémoire centrale dans lequel loger l'élément du travail où se trouve l'instruction référencée. Le volume disponible étant limité, arrivera un moment de saturation. Un des éléments fondamentaux de la gestion de la mémoire consiste donc à déterminer si le demandeur a droit à un nouveau volume de mémoire ou s'il est nécessaire de le forcer à réutiliser une partie de l'espace qui lui a déjà été alloué pour satisfaire à sa demande. Il faut donc pouvoir décider du mode de réponse à la demande, nouvelle allocation ou réutilisation, et dans ce dernier cas, quelle partie de la zone allouée réutiliser. Lorsque la décision a été prise, cet algorithme de décision doit contrôler sa réalisation, et en cas d'échec, choisir la voie à suivre. Il doit être secondé par des algorithmes de remplacement, d'extension, et d'un algorithme qui fixe quelle est la partie du

volume d'espace alloué au travail le moins utile et donc l'ensemble disponible pour le remplacement.

On a donc un système hiérarchisé représenté à la figure [ I-1]




1.5.3. Algorithme de décision

Il est étroitement lié aux décisions globales de gestion du système. Il doit décider d'étendre le volume de mémoire du travail ou de remplacer au sein de ce volume en fonction d'un critère donné. Ce critère peut être inexistant, et dès lors on alloue jusqu'à atteindre les limites de l'espace disponible, favorisant ainsi les travaux à fort taux de défauts de page, sans être certain qu'un cadre alloué soit encore utile au travail. Il peut être nécessaire de fixer arbitrairement un volume maximum allouable à chaque travail, de manière absolue ou relative à la taille du travail, sans tenir compte des situations particulières à chacun, et donc en pénalisant certains tout en allouant des ressources inutiles à d'autres. Ce volume maximum allouable peut aussi être réévalué périodiquement en fonction de l'utilisation réelle des ressources faites par le travail.

Dans les cas précédents, le volume maximum allouable ne peut qu'augmenter au cours du temps. Il est cependant bien évident qu'un travail peut ne plus avoir besoin d'autant de cadres à un moment donné que dans son passé. Il est donc fort utile de pouvoir de temps à autre éliminer les cadres alloués et inutiles au travail.

Cela peut être réalisé soit à des temps fixes pour tous les travaux, soit lors de la survenance d'événements donnés et considérés comme importants et assez fréquents. Selon les cas, cette remise en cause de la situation peut frapper un seul travail, ou l'ensemble des travaux, suivant la philosophie du système.

L'algorithme de décision a donc deux buts :
- choisir comment répondre à un défaut de page
- évaluer l'état des cadres alloués au travail.

Dans la pratique, cette évaluation se fait par l'algorithme de mise à jour. Cet algorithme est utilisé par l'algorithme de décision suivant la situation, mais il peut aussi être employé à divers moments de manière automatique, sans tenir compte des besoins réels du ou des travaux en cours.

### 1.5.4. Algorithme d'extension

Lorsque la décision d'étendre le volume alloué à un travail a été prise, reste à déterminer quel élément de la mémoire centrale allouer. On peut décider d'allouer le premier élément disponible pour la pagination et de plus suffisamment grand dans un cas de segmentation.

On peut aussi choisir en priorité un élément immédiatement disponible, ne demandant pas de transfert préalable de son contenu sur la mémoire auxiliaire.

### 1.5.5. Algorithme de remplacement

Lorsqu'il y a impossibilité d'étendre le volume du travail demandeur, cet algorithme utilise un volume désigné par l'algorithme de mise à jour, segment ou cadre, comme zone réceptrice de l'information demandée. Il assure aussi la réalisation des opérations préparatoires propres à rendre le volume sélectionné utilisable; c'est-à-dire recopie de son contenu sur la mémoire auxiliaire si nécessaire.

1.5.6. Algorithme de mise à jour

Il permet de désigner la partie de la mémoire centrale qui peut, sans trop de problèmes prévisibles, être remplacée.

Le choix peut se faire suivant différents modes. On peut retenir l'élément, segment ou cadre, le premier entré en mémoire centrale (FIFO), avec le risque d'avoir besoin souvent du contenu remplacé et donc de faire des défauts de pages inutiles. On peut aussi essayer de tenir compte de l'utilisation globale des cadres. Dans ce cas, on remplacerait le cadre qui a été le moins souvent utilisé dans la vie totale du programme. Mais en pratique, les études faites sur le comportement de programme permettent de montrer que les suites de références restent concentrées pendant une période donnée dans une zone du travail, pour ensuite changer radicalement de zone.

On essaie dans ce cas de ne tenir compte que des références faites dans un passé récent. On considère alors que, dès que l'élément a été utilisé, son histoire passée n'a plus d'intérêt et est oblitérée. S'il ne l'a pas été, on tient compte d'une manière ou d'une autre de la durée d'inutilisation.

L'algorithme peut donner un élément à remplacer ou une liste rangée dans un ordre de priorité au remplacement.

De plus, il peut être appliqué indistinctement à tous les travaux pour répondre à un défaut de page, ou à celui qui le commet. Ceci est lié aux choix fondamentaux de priorité : pénalisation des travaux pour demandeur de ressource au profit des travaux gourmands, ou au contraire compétition entre les travaux pour l'obtention d'une ressource commune, la mémoire centrale, et refus des interactions directes entre les programmes.

L'algorithme de mise à jour est particulier en ce sens qu'il n'est pas lié à priori de manière unique au gérant de la mémoire. On peut en effet choisir de mettre à jour les cadres en cas de besoin, donc de demande de page d'un travail impossible à résoudre autrement. On peut aussi choisir de faire cela à des moments donnés de l'histoire du travail, indépendamment des demandes, par exemple lors de la désactivation dans un système de time-sharing, ou lors de la perte du processeur pour un délai assez long par le travail, ou encore à des temps fixes et dans ce cas pour tous les travaux.

On peut non seulement indiquer quel volume est à remplacer en priorité, mais aussi pouvoir éliminer des éléments considérés comme absolument sans intérêt, sans remplacer le contenu dans le cadre du travail. Cette élimination peut être faite dans le cadre de l'algorithme de mise à jour à chaque utilisation de celui-ci, ou seulement dans un cas de saturation de la mémoire centrale.

---

## 1.6. CONCLUSION

Le gérant de la mémoire a donc à sa disposition divers algorithmes de gestion des espaces de mémoire, et des zones allouées aux divers travaux, ainsi que des tables où sont stockés les renseignements nécessaires à cette gestion.

L'allocation, la restitution et la réutilisation des zones allouées à chaque travail varie fortement selon chaque système et selon ses objectifs : quel type de travail favoriser, avoir un bon temps de réponse ou une utilisation maximale des ressources, vouloir ou refuser une interdépendance entre les travaux, etc ...

Dans le chapitre deux, nous allons montrer quelle application a été faite des principes dans les cas d'ordinateurs à mémoire virtuelle paginée.

---

# CHAPITRE II :

## PRESENTATION DE SYSTÈMES À MÉMOIRE VIRTUELLE

## 2.1. INTRODUCTION

Dans le but de montrer l'implémentation pratique des différentes structures montrées au chapitre 1, nous avons étudié divers systèmes à mémoire virtuelle paginée sur de gros ordinateurs, Siemens 4004-151, IBM 370/VS1/VS2/MVS.

Disposant de plus grandes facilités d'analyse pour l'ordinateur Siemens, celui-ci sera décrit de façon plus détaillée et servira de référence aux autres présentations.

Les systèmes Siemens et IBM/MVS sont fort proches, leurs objectifs étant similaires : un système de time-sharing avec forte proportion de travaux interactifs.

Les systèmes IBM/VS1 et VS2 sont les développements des systèmes MFT et MVT avec une mémoire virtuelle, et gardent donc fortement les caractéristiques de ces systèmes initiaux.

La description sera partagée en trois parties : moyens de gestion des espaces de mémoire (tables du système), gestion des espaces et gestion des interactions entre ceux-ci.

---

## 2.2. SIEMENS 4004/BS2000

### 2.2.1. Introduction

Cet ordinateur possède un système à mémoire virtuelle avec espaces d'adresse multiples. Il est basé sur le partage du processeur entre divers utilisateurs par des tranches de temps (time-slice) égaux. L'élément de base est le travail, qui dispose d'un espace d'adresse; chaque travail est concurrent avec d'autres pour l'obtention des ressources, dont la mémoire.

Lors de la fin d'un time-slice, le travail est désactivé, et perd pour un certain temps le droit de concourir pour des ressources. Cette désactivation se produit aussi lors des entrées-sorties avec le terminal, ou sur ordre de l'opérateur. Un travail désactivé perd toutes les ressources dont il dispose, donc également la place en mémoire centrale.

### 2.2.2. Moyens de gestion des espaces de la mémoire

Les moyens sont constitués de tables de deux types. Les unes donnent l'état d'un des espaces de mémoire centrale auxiliaire ou d'adresse. Il s'agit de la PMM (Physicial Memory Map), de la PDMM (Paging Drum Memory Map) et de la VMM (Virtual Memory Map). D'autres assurent les liaisons entre espaces. Dans le Siemens, il s'agit de la PGT (Page table) et de la PPAT (Physical Page Table). Ces deux tables assurent encore d'autres fonctions (voir 2.2.6.). Il n'y a pas de table autonome donnant l'état de l'espace d'adressage. Cette fonction est assurée par le PGT, en sus de son but de lien entre les espaces de la mémoire.

Les tables PGT et PPAT n'assurent pas chacune la liaison entre deux espaces, mais entre les trois espaces directement. La PPAT fait le lien entre la mémoire centrale comme origine de l'information et les deux autres espaces comme cible. La PGT a pour origine quant à elle l'espace d'adresse et pour cible les autres espaces. Une description du contenu de ces tables est faite au paragraphe 2.2.6.

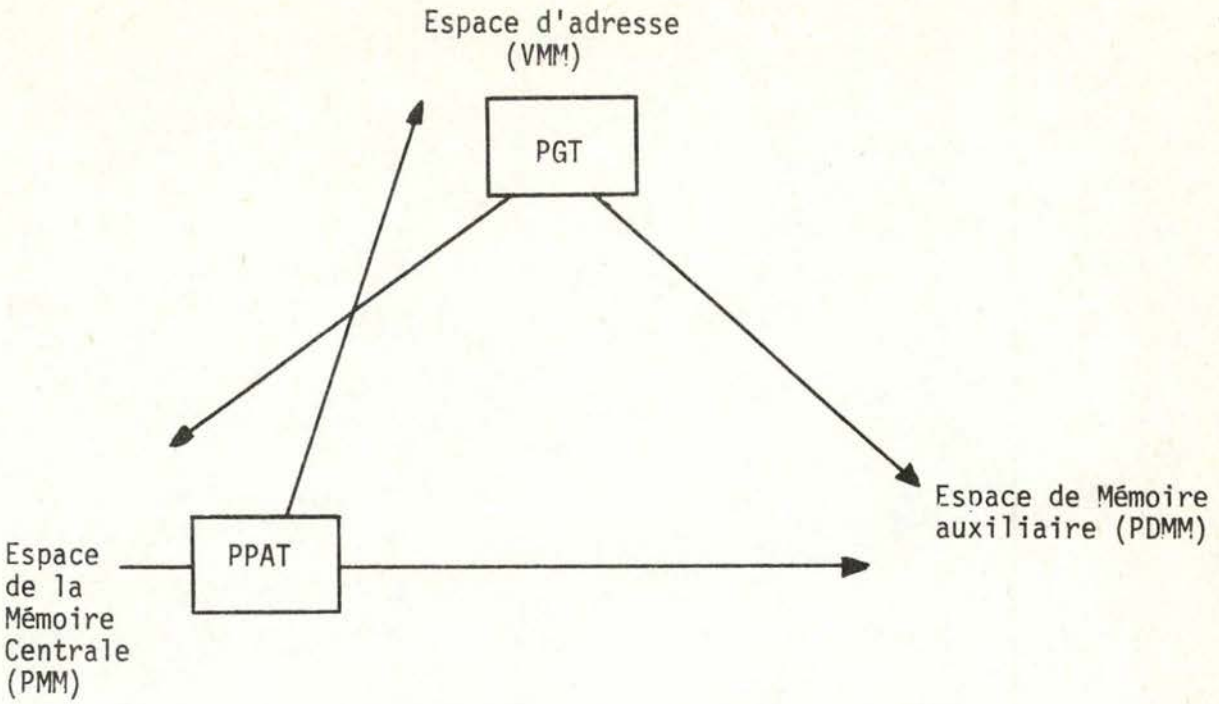



Figure 2.1 : Espaces de la mémoire et tables assurant les relations entre eux.

2.2.3. Mémoire centrale

La mémoire centrale est découpée en cadres de 4036 bytes (ou 4K bytes). Sa taille est de 500 à 800 K en général, mais peut être étendue.

Une partie de la mémoire centrale est utilisée comme zone résidente pour la partie fixe du système. Cette partie ne peut servir dans la pagination, ce qui réduit le nombre de cadres qui seront disponibles pour répondre aux défauts de page.

Une autre partie est remplie par les diverses tables de gestion et les bufferts du système.

Le reste est disponible pour être utilisé par les travaux utilisateurs.

La PMM (Physical Memory Map) donne pour chaque cadre de la mémoire centrale l'état global occupé-libre.

2.2.4. Mémoire auxiliaire

Elle est disposée sur deux tambours, subdivisés en cadres de 4K appelés slots.

Une partie de ces tambours est attribuée aux modules du système non permanent et aux modules utilitaires.

La table PDMM (Paging Drum Memory Map) permet de savoir si un de ses cadres est occupé ou non.

2.2.5. Espace d'adressage

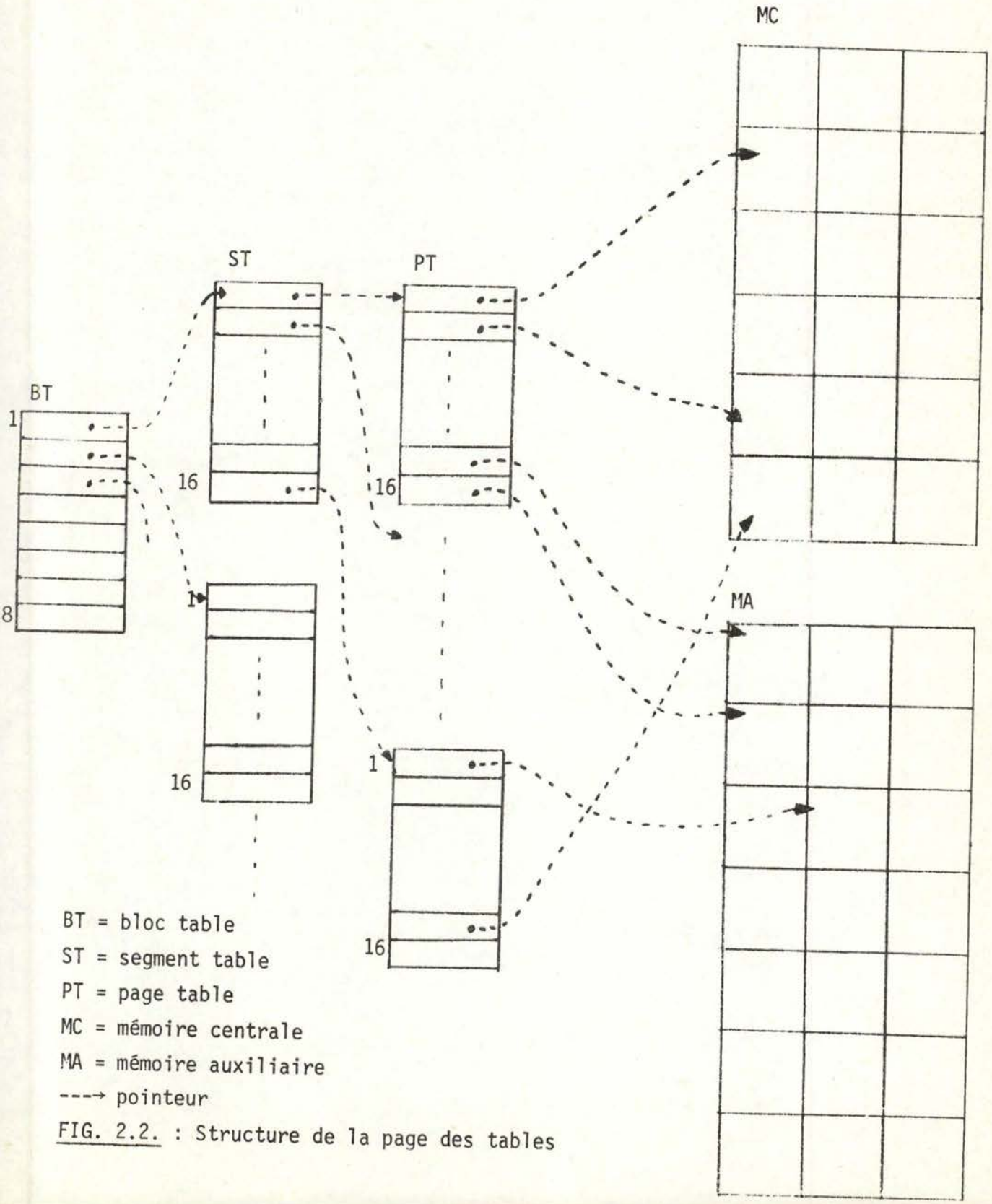
Chaque travail dispose d'un espace d'adresse propre, de 2048 pages de 4K.

La part la plus importante, 1792 pages, est une zone commune à tous les travaux et donne les références nécessaires pour accéder aux utilitaires du système et à celui-ci.

Le reste, 256 pages, contiendra le travail et fixe donc la taille maximale des travaux dans ce système.

Tout programme demandé par le travail doit figurer dans la partie privée de l'espace d'adresse du travail, même s'il s'agit d'un utilitaire du système.

Les adresses contenues dans les programmes font référence à une adresse de l'espace d'adresse du travail qui a demandé l'exécution du programme. Cependant, Siemens permet l'existence d'adresse indépendante de l'espace d'adressage, référençant directement la mémoire centrale. Ces adresses font référence au volume occupé par le système permanent ou les programmes fixés en mémoire centrale.




BT = bloc table
ST = segment table
PT = page table
MC = mémoire centrale
MA = mémoire auxiliaire
----→ pointeur

FIG. 2.2. : Structure de la page des tables

### 2.2.6. Tables de gestion

#### 2.2.6.1. PGT (Page Table)

Cette table fait le lien entre l'espace d'adresse d'un travail et les mémoires centrales et auxiliaires. Pour chaque page, elle donne l'adresse du cadre correspondant en mémoire centrale ou auxiliaire. Comme l'espace d'adresse, la table des pages est séparée en deux groupes d'entrées de table, l'une liée à la partie privée de l'espace d'adresse et l'autre à la partie commune.

La table est structurée en trois niveaux hiérarchiques, blocs, segment et page (voir figure 2.2.)

Chaque entrée de bloc ou de segment se décompose en :
- un pointeur vers le niveau hiérarchique inférieur
- des indicateurs d'état (existence de ce pointeur ou non, le pointeur est une adresse réelle ou non, divers bits de protection).

L'entrée de bloc est constituée de :
- un pointeur vers un cadre de la mémoire auxiliaire ou centrale
- des indicateurs : existence du cadre, présence de celui-ci en mémoire centrale, cadre demandé ou non, cadre dont le contenu a été modifié ou non.

Le niveau de bloc sert à faire la distinction entre la partie privée et la partie commune de l'espace d'adressage. La première entrée de la table des blocs adressera la partie privée, ici un groupe de seize entrées de segments. Sept autres entrées de blocs adressent la partie commune. Lorsqu'un nouveau travail dispose du processeur central, l'adresse contenue dans la première entrée de bloc sera chargée pour contenir l'adresse du groupe de segments privés du nouveau travail. Les autres entrées de blocs sont fixées au début de la session pour la durée de celle-ci.

Le niveau intermédiaire du segment a comme but d'éviter une perte de place. Sans lui, à toute entrée de bloc, correspondrait 156 entrées de pages, même si seulement un petit nombre est nécessaire. Avec lui, on aura un groupe de 16 entrées de segments et autant de groupes de pages qu'il sera nécessaire pour contenir les pages de travail. Le gain de place est non négligeable.

Chaque entrée de page est conçue pour donner le maximum d'informations sur l'état et éviter ainsi des accès-mémoire supplémentaires.

Si un groupe d'entrée est totalement vide, c'est-à-dire qu'aucune des entrées n'adresse quoi que ce soit, l'entrée du niveau hiérarchique supérieur qui correspondait à ce groupe est remise à jour pour indiquer ce fait.

Chaque entrée de page indique donc si la page existe dans l'espace d'adresse. Si oui, elle donnera le numéro du cadre de la mémoire centrale ou auxiliaire qui lui correspond.

2.2.6.2. PPAT(Physical page allocation table)

Cette table permet à la fois d'avoir l'état d'un cadre de la mémoire centrale, et de relier ces cadres avec les cadres correspondant de la mémoire auxiliaire, et la page de l'espace d'adressage.

Chaque cadre appartient soit à l'une des chaînes du système, soit à une chaîne reprenant les cadres attribués à un travail.

Les chaînes du système sont au nombre de quatre : la première contient les cadres directement utilisables (Read Only Queue - ROQ), deux autres les cadres à recopier sur le tambour avant de les utiliser (Read Write Queue, une par tambour), et finalement la chaîne contenant les cadres alloués au système (System Queue, SQ).

Il existe une entrée par cadre de la mémoire centrale, qui contient :
- la référence à l'espace d'adresse, bloc, segment et page qui occupe le cadre
- la position du cadre sur la mémoire auxiliaire
- l'adresse réelle de l'entrée correspondante de la page des tables
- un lien qui précise à quelle queue du système où à quelle queue de travail le cadre appartient.
- l'âge du cadre (le temps depuis lequel le cadre n'a pas été référencé)
- divers indicateurs : cadre bloqué en mémoire centrale ou non, etc ...

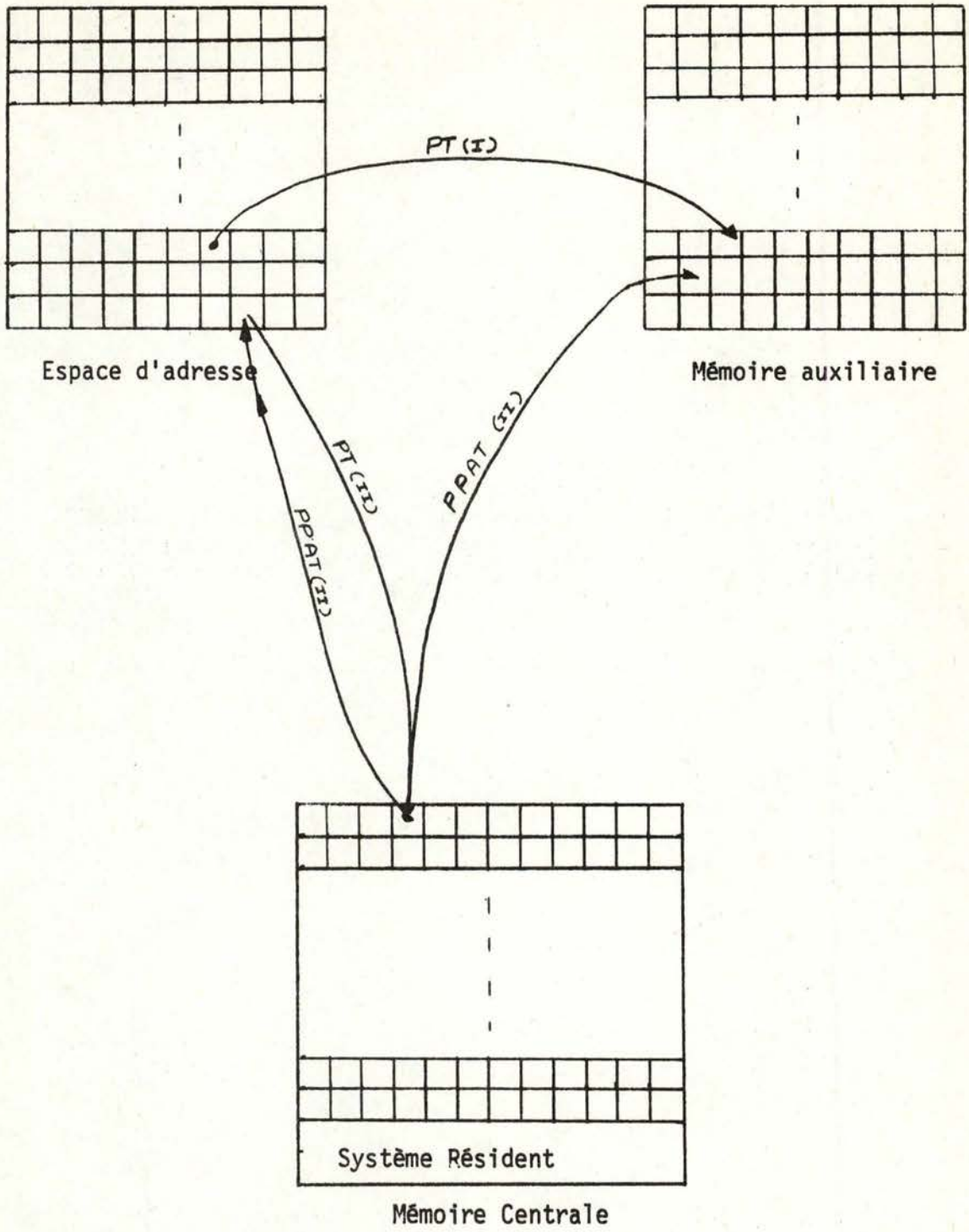



FIG. 2.3. : Espaces de mémoire et relations entre ces espaces par les tables, selon que le cadre soit (II) ou ne soit pas en mémoire centrale (I)

2.2.7. Conversion des adresses

Chaque donnée d'un programme est référencée par une adresse de l'espace d'adressage. Pour pouvoir utiliser la donnée, il faut au préalable disposer de sa position en mémoire centrale, et donc convertir l'adresse de l'espace d'adressage en adresse de l'espace de mémoire centrale.

Cette conversion se fait au travers de la table des pages (PGT). L'adresse virtuelle (de l'espace d'adressage) est structurée de la même manière que cette table. La conversion est faite par hardware.

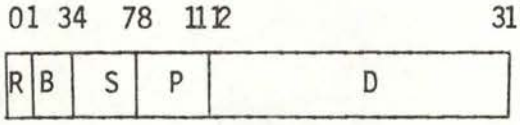



FIG.2.4. : structure de l'adresse dans le programme

Le champ R, s'il vaut 1, signifie que l'adresse est réelle, et indique une zone en mémoire centrale.

Les champs B indiquent quelle entrée prendre dans la table des blocs, S l'entrée dans le groupe d'entrée de segments correspondant à ce bloc et enfin P la page dans le groupe de page pointée par le segment choisi.

Enfin, dans cette page, le déplacement est défini dans le champ D.

A partir des champs B, S, P, on peut donc trouver l'entrée correspondante dans la table des pages et voir si la page cherchée est ou n'est pas en mémoire centrale. Si oui, on ajoute à l'adresse réelle du cadre le déplacement pour obtenir l'adresse demandée. Sinon, il y a défaut de page.

Une telle méthode nécessite trois accès mémoire et est fort lente. Or expérimentalement, on peut voir que en général, un programme donné référence un ensemble d'instruction tenant en peu de pages, et ce pour une durée assez longue.

Cela a donné l'idée d'utiliser une mémoire plus rapide et plus petite, où, pour les dernières pages demandées, on stockerait assez de renseignements pour éviter des accès à la mémoire.

Pour le Siemens, cette mémoire rapide et associative s'appelle la C.A.M. (content addressable memory). Elle se compose de huit entrées. Chacune d'entre elle contient

- le numéro de la page virtuelle (champ BSP)
- l'adresse de l'entrée correspondante de la table des pages
- l'adresse de l'entrée correspondante de la table des segments
- l'adresse du cadre correspondant de la mémoire centrale.

Lors d'une conversion d'adresse, le hardware parcourt chaque entrée de la table. S'il y trouve une référence à la page demandée, il a l'adresse du cadre de mémoire centrale cherché.

Il peut aussi n'y trouver qu'une référence à une autre page située dans le même segment. La CAM donne alors l'adresse de l'entrée correspondante. Un seul accès à la mémoire est alors suffisant pour trouver l'entrée de la table des pages cherchées.

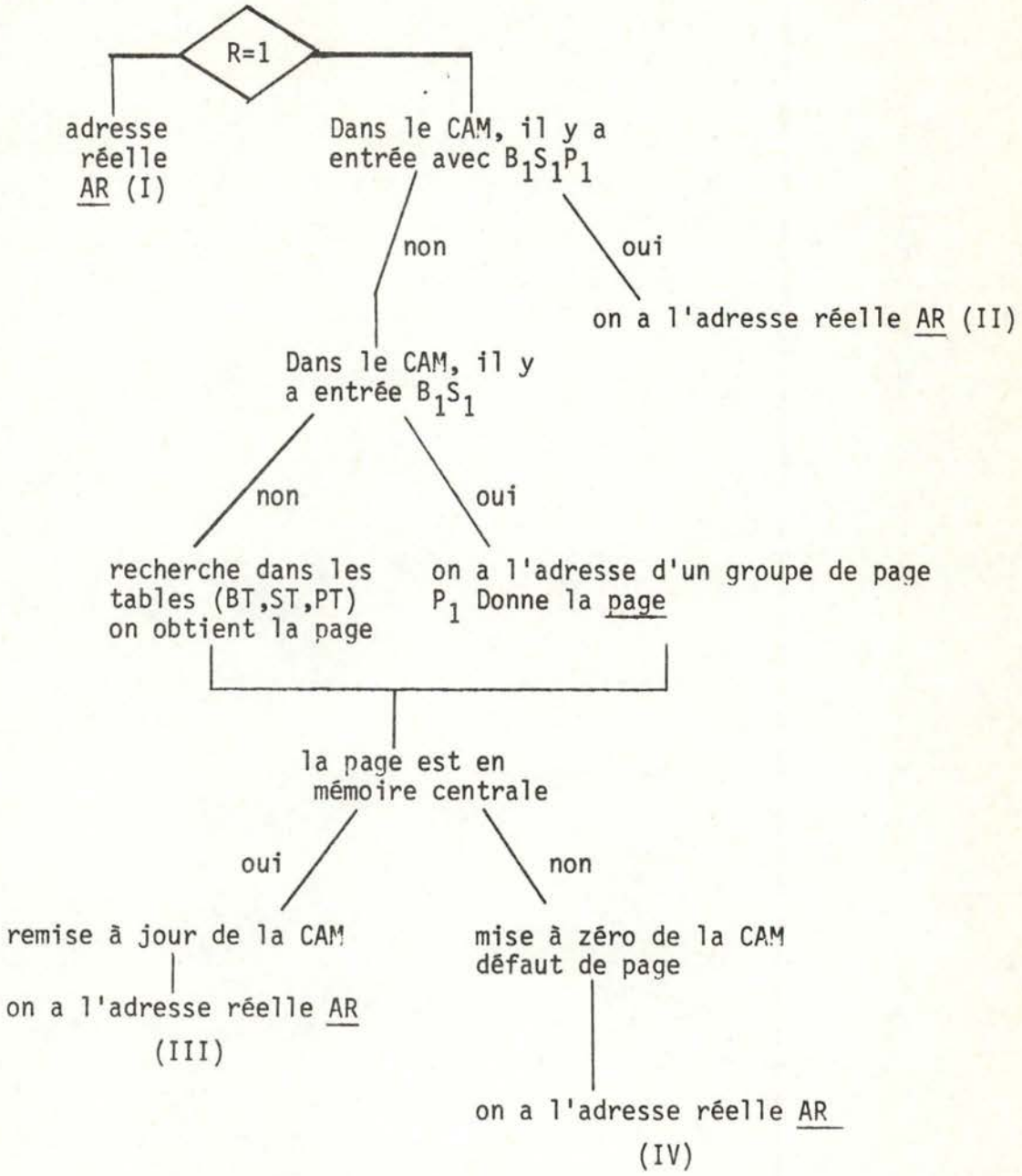
Enfin, il peut ne rien trouver, et alors commencer la recherche par la table des blocs.

Chaque référence à une page qui n'est pas reprise dans la CAM oblige à une remise à jour de celle-ci. Si la page dispose d'un cadre en mémoire centrale, les renseignements nécessaires sont pris pour former une nouvelle entrée dans la CAM. Celle-ci va remplacer la première des références dans la CAM (procédure F.I.F.O.).

Lorsqu'un travail perd le processeur, le contenu de la CAM est mis à zéro.

A ce niveau-ci, défaut de page signifie seulement que la page demandée n'est pas dans la chaîne des cadres alloués au travail. Il peut soit être dans une des queues du système, soit ne plus se trouver en mémoire centrale.

adresse virtuelle $R_1B_1S_1P_1D_1$




Pour les sorties I, II, III, IV, on calcule AR + $D_1$ = adresse demandée

FIG.2.5. : Conversion de l'adresse virtuelle en adresse réelle

## 2.2.8. Algorithmes de gestion

### 2.2.8.1. Introduction

Chaque travail dispose en mémoire centrale d'un volume d'espace appelé mémoire occupée. Ce volume est en général dans un système à mémoire virtuelle inférieur à la taille du travail. Expérimentalement, il doit dépasser un certain minimum variable au cours du temps pour permettre un fonctionnement efficace du système, et éviter un nombre trop important de défauts de page.

On a pu constater (1) que chaque programme avait tendance à faire pendant un certain temps des références à la mémoire localisées à un nombre réduit de pages. Si le volume minimum disponible pour le programme dépasse ce nombre, après une série de défauts de pages initiaux qui amèneront les pages en mémoire centrale, il n'y aura plus de défauts de page pendant le temps où le programme ne référencera que ces pages.

L'application pratique de cette hypothèse est limitée par la non-connaissance précise de l'avenir du programme. Dans le cas du Siemens, on suppose que le comportement futur du programme est calqué sur son passé. La notion de volume minimum est reprise dans celle de volume alloué au travail, qui est le nombre maximum de cadres occupables par le programme.

Le volume alloué est périodiquement revu pour le faire coïncider avec le passé proche du travail. Cette mise à jour se fait à période fixe, lors de l'échéance d'un time-slice, ou à des moments liés au programme, entrée-sortie vers un terminal, défaut de page impossible à résoudre. A ce moment, le travail est désactivé et perd ses cadres de la mémoire centrale. On égale alors le volume à allouer au volume occupé.

Cette mise à jour se fait à des périodes assez éloignées l'une de l'autre. Pour rendre le système plus souple, des mises à jour locales sont faites lors des défauts de page par l'algorithme de mise à jour, qui élimine les cadres considérés comme trop vieux.

---

(1) A.W. Madison, A.P. Batson - University of Virginia, Communication of the ACS, May 1976

2.2.8.2. Algorithme de remplacement

Cet algorithme est utilisé soit lorsque l'espace occupé égale l'espace alloué, soit lorsque il n'y a plus de cadres disponibles pour la demande.

Le travail doit céder un de ses cadres qui lui aura été désigné par l'algorithme de mise à jour. Si celui-ci a trouvé un cadre, un transfert ou le tambour est organisé. Si aucun cadre n'a pu être trouvé, on essaie d'appliquer soit l'algorithme d'extension (si l'espace alloué égale l'espace occupé), soit celui d'urgence (s'il n'y a plus de cadres disponibles).

2.2.8.3. Algorithme d'extension

Le travail essaie d'obtenir un cadre supplémentaire pris parmi les cadres se trouvant dans l'une des chaînes (ROQ, RWQ1, RWQ2). Si le contenu du cadre doit être recopié sur le tambour, le transfert est fait et suivi par un transfert de la page demandée depuis le tambour.

Si aucun cadre n'est disponible, le travail essaie l'algorithme de remplacement si l'espace alloué n'est pas totalement occupé, et l'algorithme d'urgence dans le cas contraire.

La recherche d'un cadre dans les chaînes suit un ordre bien précis : on commence par la chaîne read-only, qui ne nécessite qu'un transfert depuis le disque. En cas d'échec, on va tenter d'allouer un cadre dont le contenu est à recopier sur un tambour. Dans un premier temps, on cherchera un cadre qui sera à transférer sur le même tambour d'où on ramènera la page demandée, en ne bloquant qu'un seul périphérique. Sinon, on devra faire un transfert sur un des tambours pour recopier le contenu du cadre, et un second transfert depuis l'autre tambour pour amener la page demandée en mémoire, le transfert devant attendre que l'autre tambour soit libre.

Lorsqu'on alloue un nouveau cadre, et que les volumes alloués et occupés sont égaux, les deux volumes sont incrémentés d'une unité simultanément.

2.2.8.4. Algorithme de mise à jour

Il permet de donner pour chaque cadre d'un travail une idée de l'utilisation qui a été faite par le travail de ce cadre.

Cette idée est définie par un temps de non-utilisation, qui est le temps depuis lequel le travail n'a fait aucune référence à ce cadre.

Sa première fonction est d'éliminer les cadres considérés comme trop vieux, c'est-à-dire inutilisés depuis plus de 120 millisecondes.

Ensuite, il essaie de trouver le plus vieux cadre ayant un temps d'utilisation compris entre 30 et 120 millisecondes, et le désigne s'il existe, à l'algorithme de remplacement.

2.2.8.5. Algorithme d'urgence

Cet algorithme est utilisé lorsqu'on n'a pu répondre à un défaut de page par l'algorithme de remplacement ou d'extension.

Si un autre travail est en attente du processeur, le travail demandeur est désactivé.

Sinon, on utilisera le cadre le plus vieux qui ait été attribué au travail même s'il ne vérifie pas les critères de l'algorithme de mise à jour.

2.2.8.6. Prise en charge des demandes

La prise en charge se fait en deux temps : établissement de celle-ci, si nécessaire mise en file d'attente et exécution.

Lors du défaut de page, une demande est constituée portant les renseignements nécessaires à son exécution (numéro de page à transférer, travail demandeur, position du cadre en mémoire auxiliaire).

Une vérification a alors lieu. Si la page demandée figure dans l'une des chaînes de système, il y a faux défaut de page. On remet la page dans la chaîne des cadres alloués, on remet à jour les tables (PPAT, PGT)

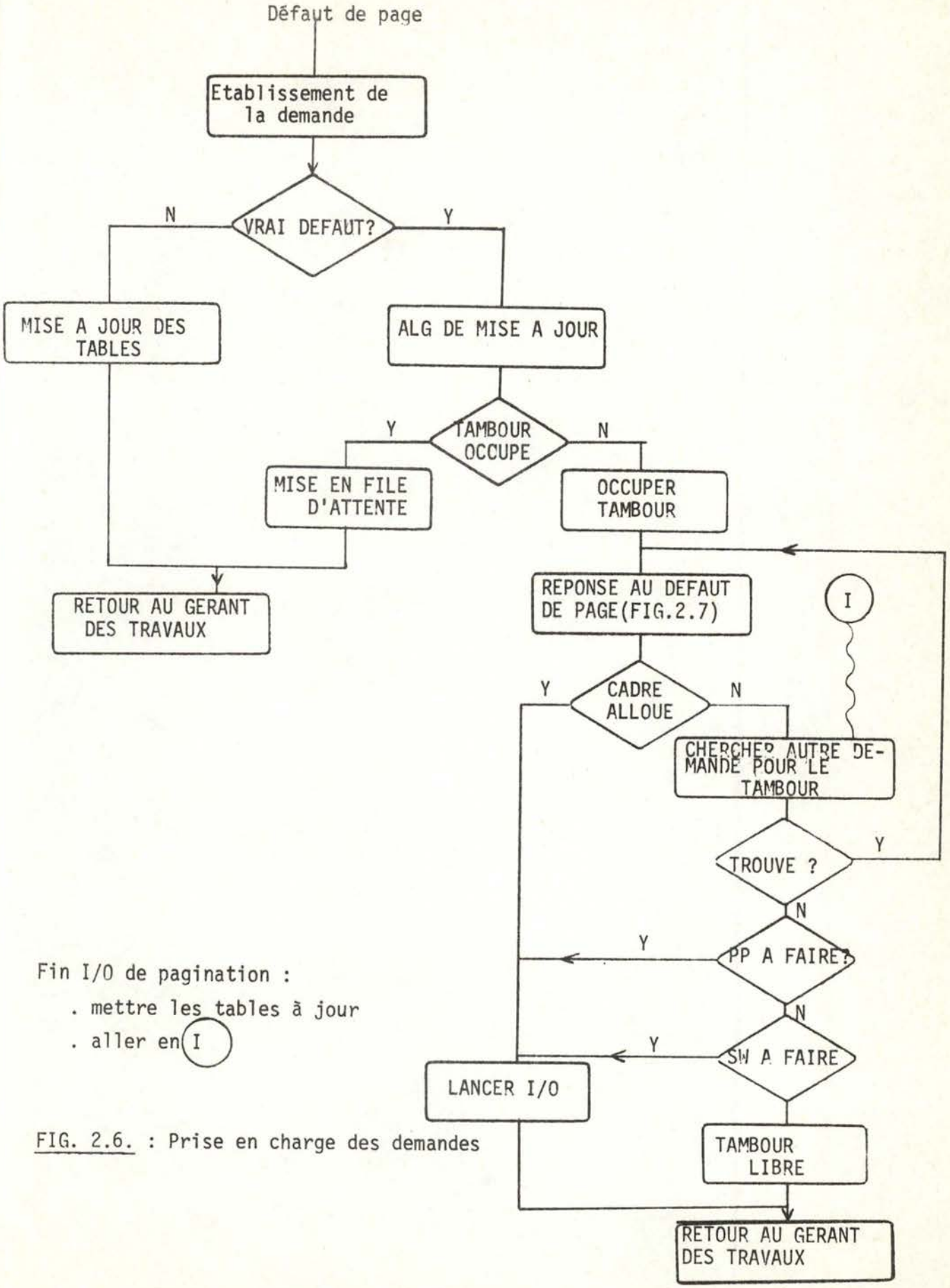
Dans le cas contraire, on appelle la routine de mise à jour et, si le tambour où se trouve la page demandée est occupé, la demande est mise en file d'attente. Sinon, elle est exécutée. Il y a une file d'attente par tambour.

Chaque file est organisée avec un système de priorité tenant compte du type de demande et d'une priorité liéeau travail.

Les types de demande peuvent être séparés en deux groupes. Le premier, le plus prioritaire, contient les demandes des travaux FW(fast write), PW (Priority Write), défauts de page. Le second est constitué des demandes liées à la gestion propre des cadres : PP (postpage) et SW (show write).

. FW : demandes désirant fixer un cadre en mémoire centrale
. PW : le cadre alloué pour répondre à un défaut de page doit être transféré sur un tambour, alors que la page demandée se trouve sur l'autre tambour. La demande de transfert de la page demandée est mise en attente, et sera réveillée lorsque le cadre aura été vidé.
. PP : Lorsque le gérant des demandes n'a plus de demandes des travaux, il transfère les cadres à recopier sur le tambour pour les rendre disponibles à la demande.
. SW : Lorsque les demandes PP sont achevées, il met à zéro les cadres abandonnés par les travaux achevés sur la mémoire auxiliaire.

Un défaut de page peut être résolu de quatre manières :
. le cadre reçu ne doit pas être recopié
. le cadre reçu doit être recopié, sur le même disque où est stockée la page demandée.
. le cadre et la page demandée se trouvent sur deux disques. La demande est mise en attente. Une demande PW est créée pour transférer le contenu du cadre.
. On n'a pas pu trouver de cadre; le travail est désactivé, on ajuste la taille du volume alloué.




Fin I/O de pagination :

. mettre les tables à jour

. aller en (I)

FIG. 2.6. : Prise en charge des demandes

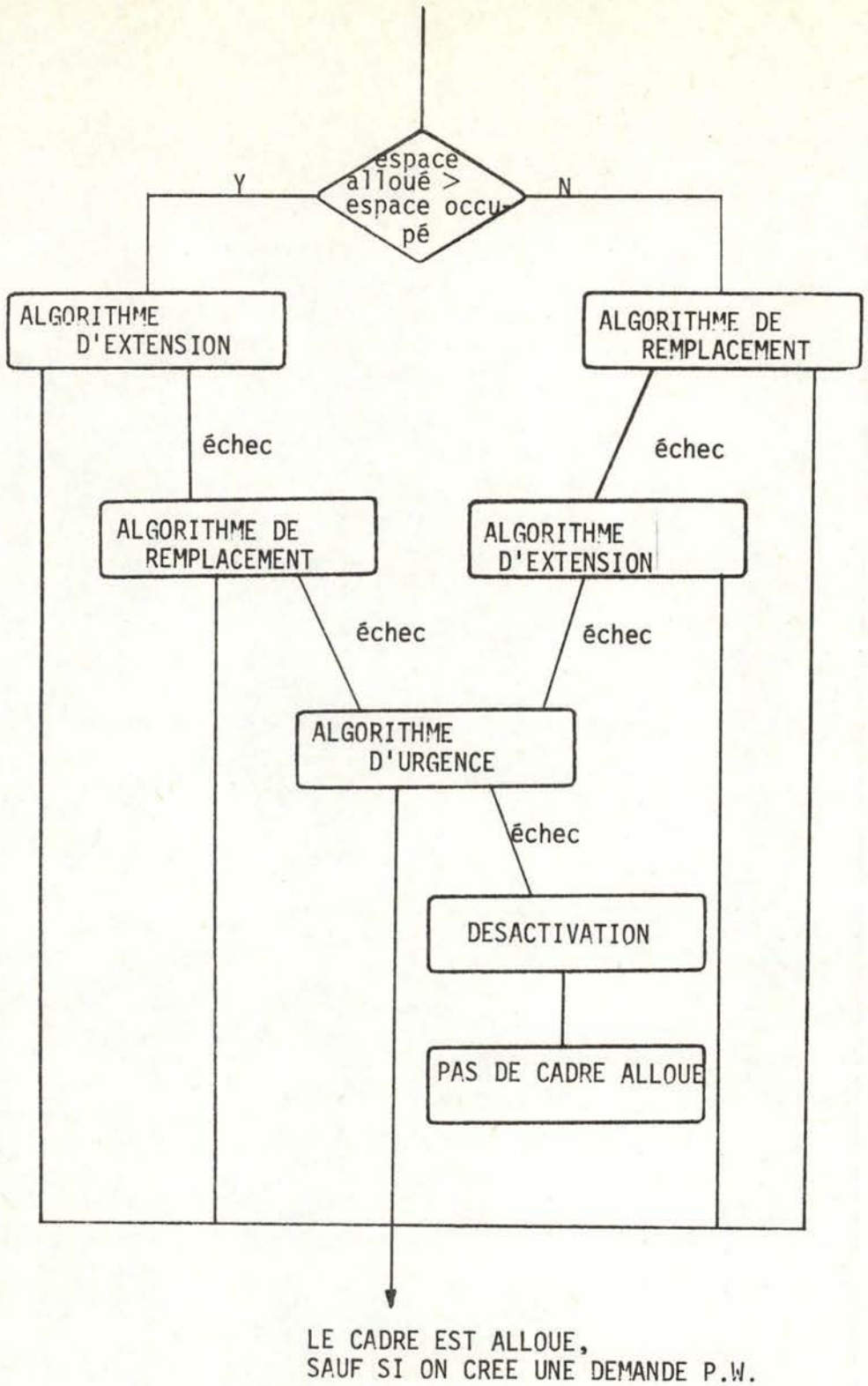



FIG. 2.7. . : Réponse au défaut de page

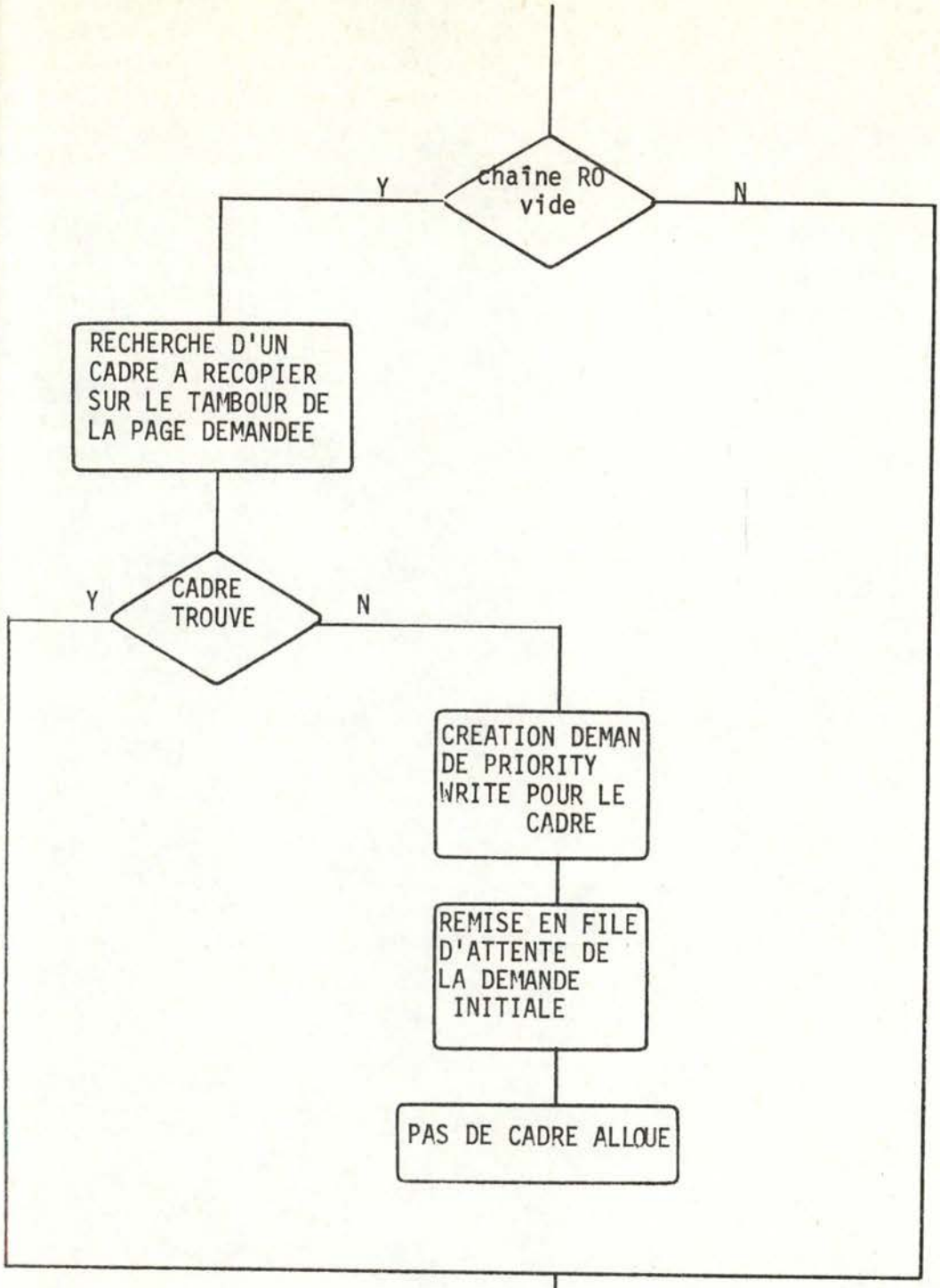



FIG. 2.8. : Algorithme d'extension

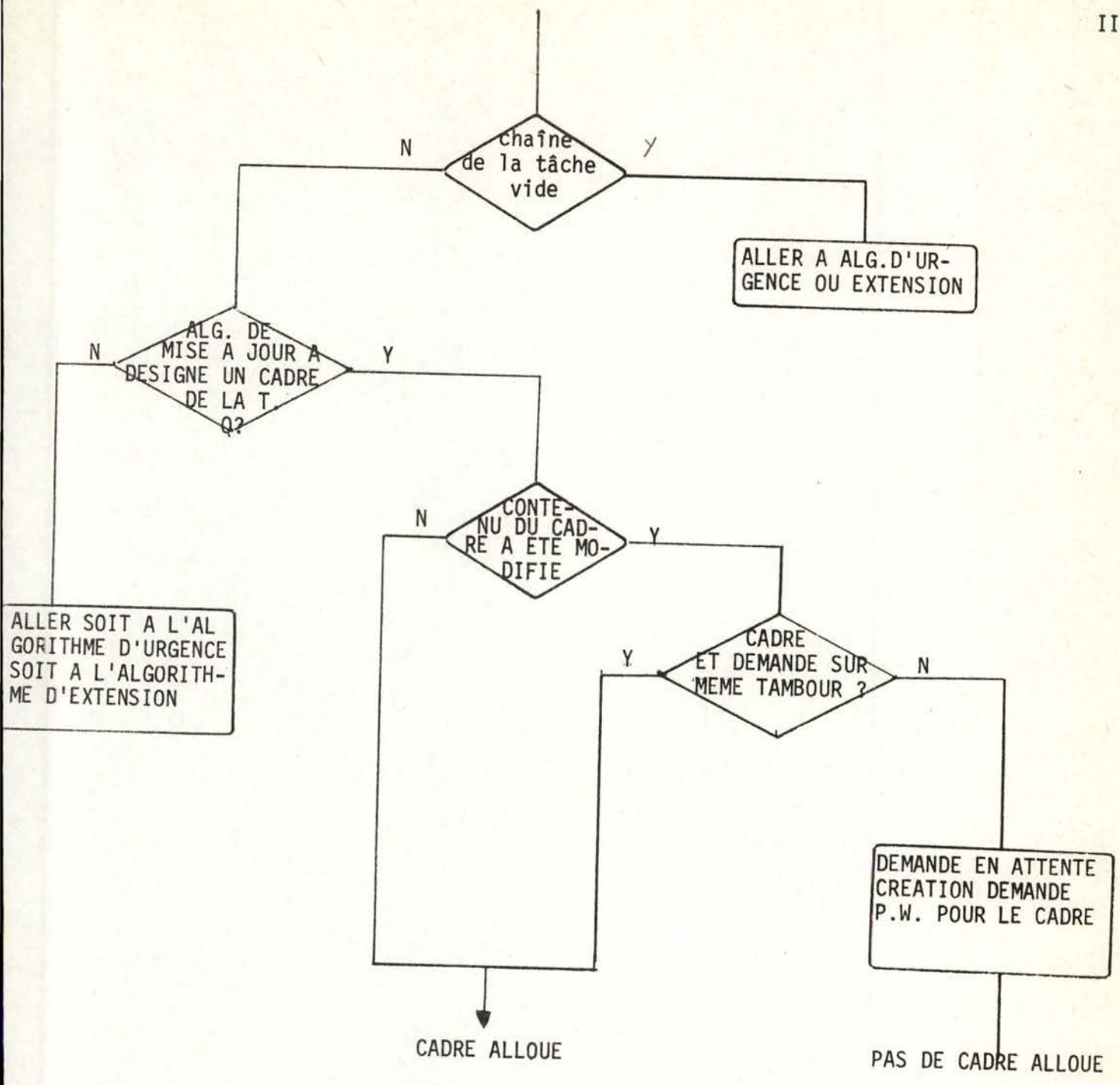



FIG. 2.9. : Algorithme de remplacement

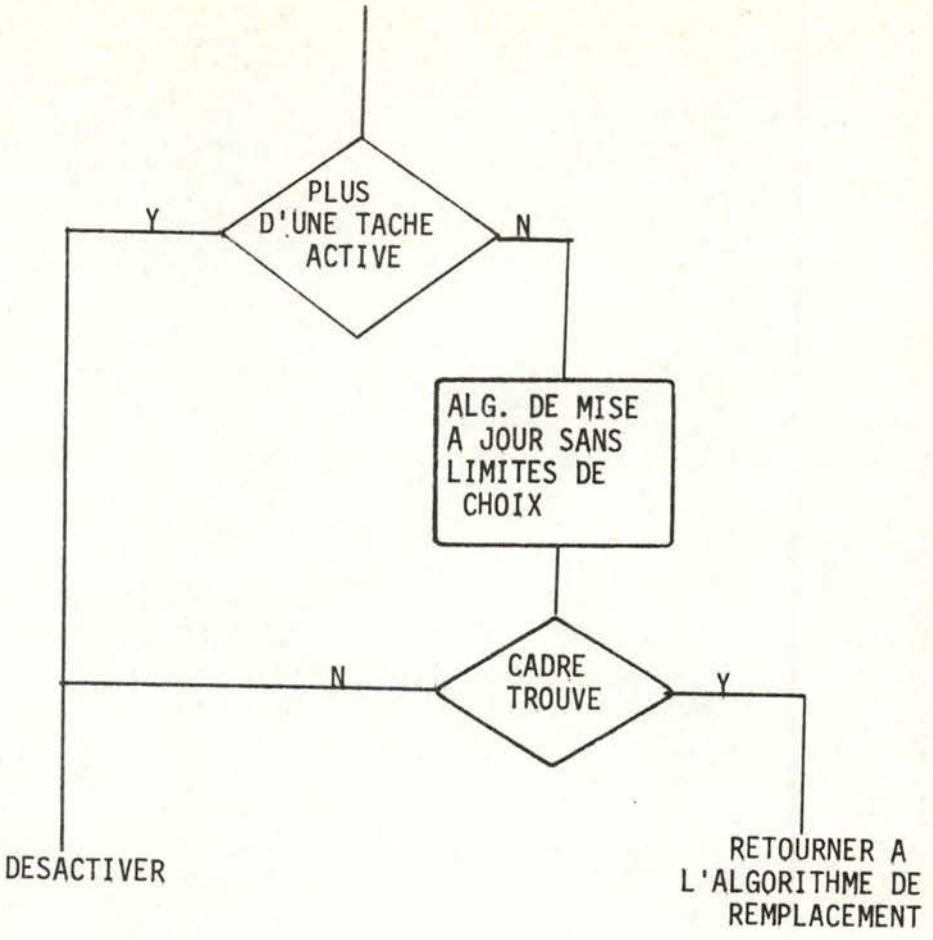



FIG. 2.10. : ALGORITHME D'URGENCE

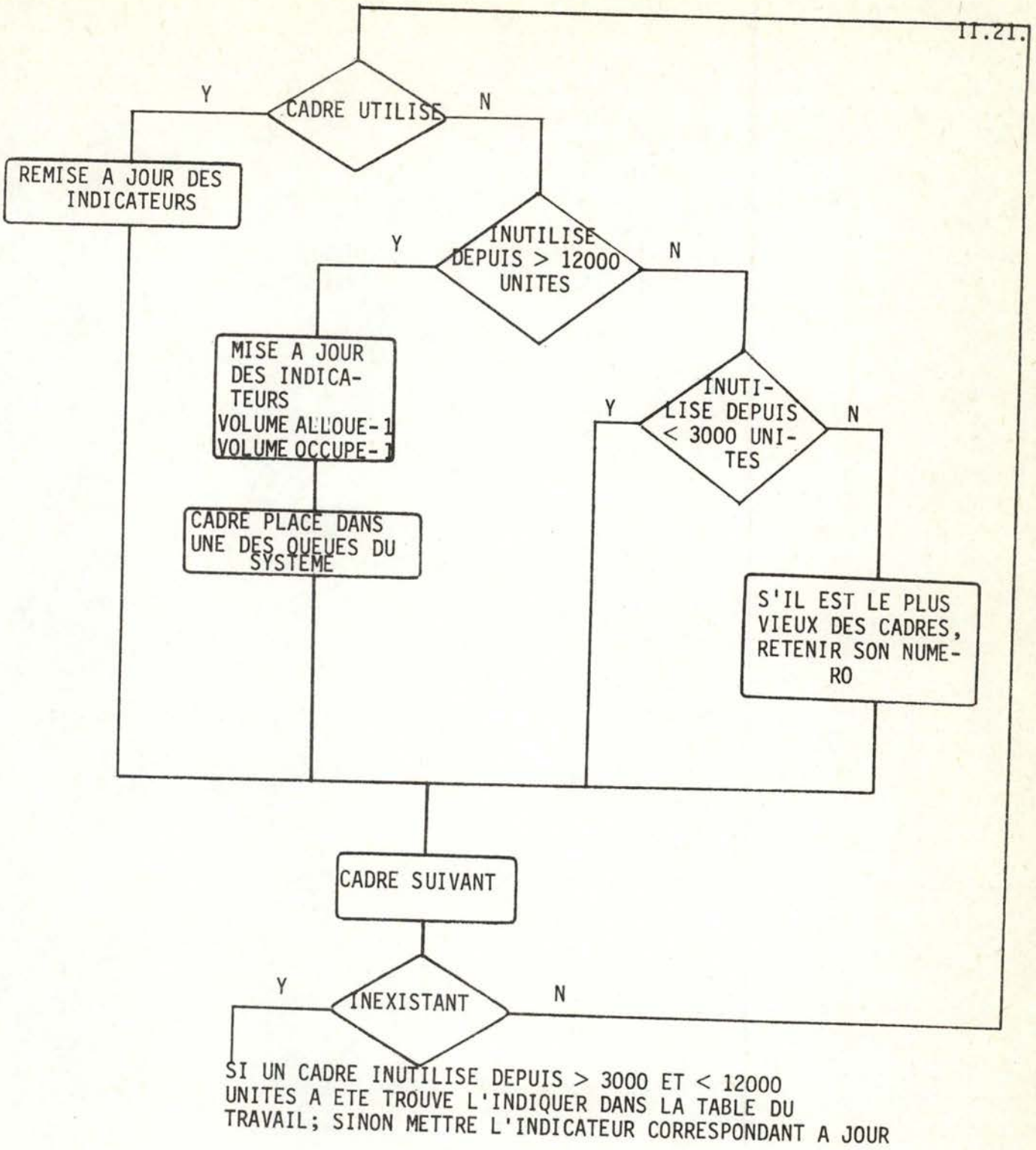



FIG. 2.11. : Algorithme de mise à jour

## 2.3. SYSTEMES IBM VS1 ET MVS

### 2.3.1. Introduction

Après un premier exemple d'implémentation et de choix faits pour le système SIEMENS-BS2000, nous allons examiner ceux réalisés par la firme IBM pour ses systèmes VS1 et MVS.

Ces systèmes contiennent divers éléments communs : structure des tables, structure du ou des espaces d'adresses et mode de conversion d'adresse.

Les algorithmes de gestion sont de même fort semblables. Les deux systèmes utilisent un algorithme de mise à jour et d'élimination des cadres inutilisés. Nous décrirons ces algorithmes séparément.

### 2.3.2. Eléments communs aux deux systèmes

#### 2.3.2.1. Adresse

L'adresse, réelle ou virtuelle se décompose en trois champs : champ de segment (S) (8 bits), champ de la page (P) (4 à 5 bits) et celui du déplacement (D) (12 à 11 bits). Il n'y a pas ici de différence entre adresse virtuelle ou réelle. Une adresse est réelle si sa valeur est inférieure à la taille de la mémoire centrale.

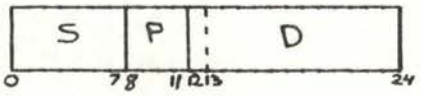



FIG. 2.12. : Structure de l'adresse

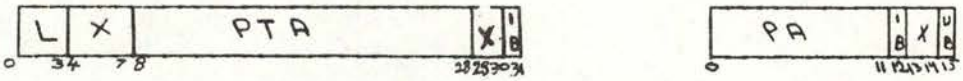
#### 2.3.2.2. Tables

##### 2.3.2.2.1. *Liaison entre les espaces de mémoire centrale et d'adresse*

Dans les systèmes IBM, la page et le cadre peuvent être de deux ou quatre kilobytes. Ceci implique deux structures possibles pour cette liaison. La table qui l'assure (PGT, page table) est divisée en deux niveaux hiérarchiques, segments et pages. Toutes les entrées de segment doivent être créées à l'initialisation de l'espace d'adresse.

Chaque entrée de segment, au nombre de 256, peut adresser seize ou trente-deux pages selon la taille de celles-ci. L'entrée contiendra un indicateur du nombre maximum de pages adressées (2), par le segment 16 ou 32 selon la taille de la page, l'adresse du groupe d'entrées de page qui correspond à ce segment (PTA) et un bit indiquant la validité ou non de l'adresse en question, c'est-à-dire si l'adresse correspond à un cadre présent dans la chaîne des cadres alloués au travail ou non.

Chaque entrée de table a une longueur d'un demi-mot. L'adresse qui y est donnée ne constitue pas l'adresse du début du cadre de la mémoire centrale où se trouve la page demandée, mais bien les 11 ou 12 premiers bits de cette adresse, ce qui permet une économie de volume assez appréciable. (ces bits correspondent en fait au numéro du cadre dans la mémoire).




L : lenght
PTA : Page Table Address
IB : Invalid Bit

PA : Page Address
IB : Invalid Bit
UB : User Bit

FIG. 2.13. : Structure d'une entrée de segment et de table

La première entrée de la table des segments est pointée par un registre du système (CR1, Control Registre 1). Il y a 256 entrées de segment, et par segment 16 pages de 4 kilobytes ou 32 de 2 kilobytes taille déterminée lors de la génération du système.

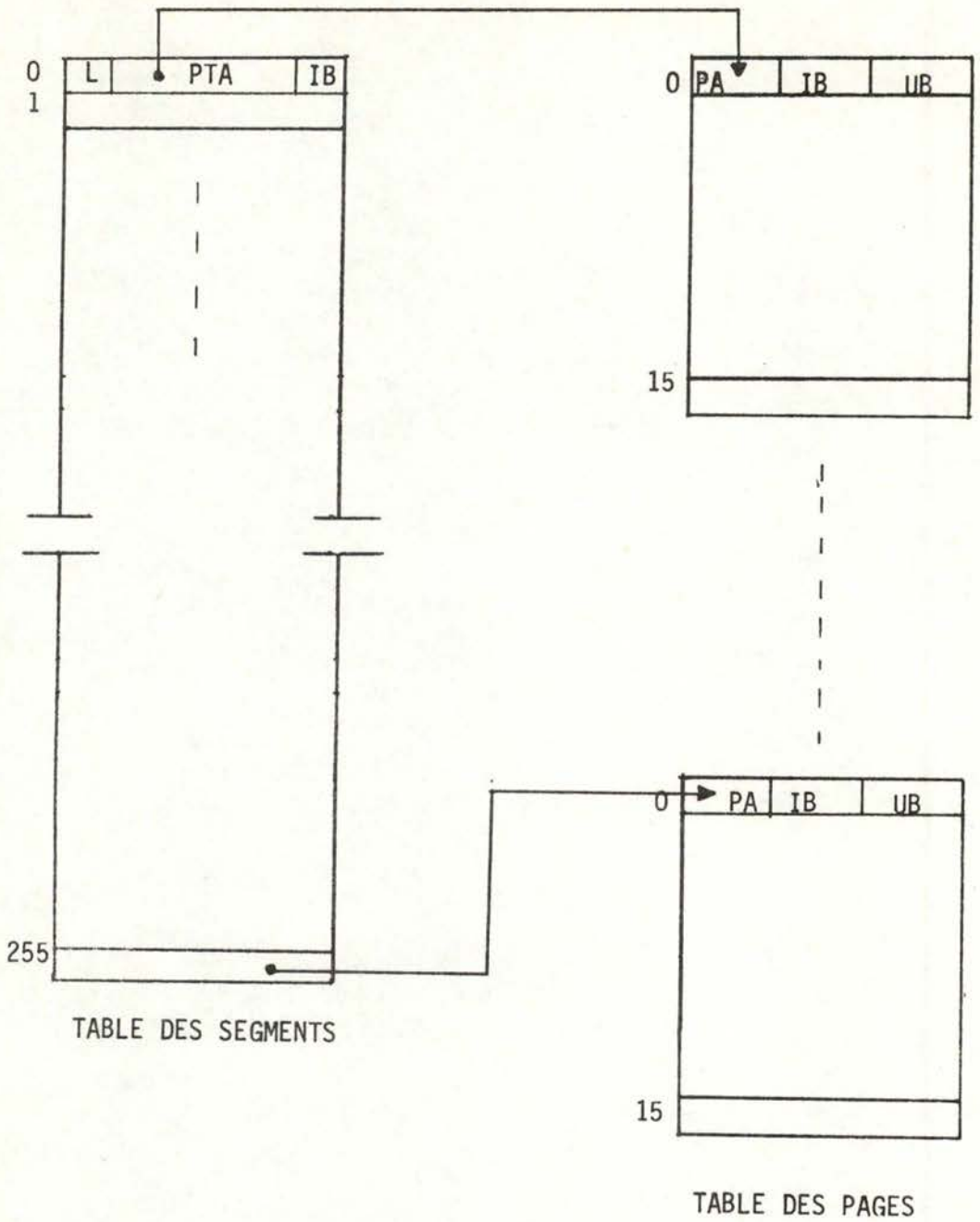



FIG. 2.14. : Structure de la PGT (Page Table)

2.3.2.2.2. *Liaison entre l'espace d'adresse et espace de mémoire auxiliaire*

Pour chaque page de l'espace d'adresse, la table XPT (External Page Table) donne la localisation du cadre sur la mémoire de masse.

2.3.2.2.3. *Table d'état des espaces*

Il y a d'abord des tables donnant les cadres libres (1 bit par cadre) pour l'espace de mémoire centrale, l'espace de mémoire auxiliaire et le ou les espaces d'adresse.

2.3.2.2.4. *Table d'état des cadres*

Dans le cas de la mémoire centrale, il existe de plus une table donnant l'état de chaque cadre. Cette table (PTF, page table frame) permet de préciser quelle page occupe le cadre, à quel travail le cadre est alloué, si la page est libre ou non, (FB, face bit) a été référencée ou non (RF, reference bit), a été modifiée ou non (CB, change bit), et un lien avec les autres pages de la chaîne à laquelle le cadre appartient (chaîne des cadres libres, APQ, (available page queue), chaîne du travail (un par travail)).

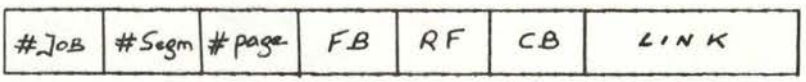



FIG. 2.15. : Schéma d'une entrée de la table PTF (Page Table Frame)

2.3.2.3. Mode de conversion d'adresse (DAT, Direct Access Translation)

La transformation d'une référence à l'espace d'adressage en adresse physique se fait en utilisant une mémoire associative rapide et petite (T.C.B. : Transition Look aside Buffer) et la table de page (PGT).

2.3.2.3.1. *Transition Look Aside Buffer (T.L.B.)*

La mémoire rapide comprend 8 ou 64 entrées selon le système utilisé. Chacune de ces entrées se compose d'un numéro de page (segment et page), d'un bit de référence (RB, reference bit) et d'une adresse en mémoire centrale.

2.3.2.3.2. *Fonctionnement*

Lors d'une référence à une adresse, la mémoire rapide est parcourue. Si la page demandée a son numéro (champ S et P) dans l'une des entrées, le bit de référence est mis à un, et l'on obtient les bits d'ordre supérieur de l'adresse réelle. Ces bits sont concaténés avec ceux du déplacement figurant dans le registre d'instruction pour fournir l'adresse réelle en mémoire centrale.

En cas d'échec, le hardware parcourt la table des segments, et la table des pages pour localiser la page correspondante. Si l'invalid bit est positionné, la page n'est plus en chaîne des cadres alloués au travail. Il y a défaut de page. Sinon, les bits de référence et, si nécessaire, de changement sont mis à jour, et l'adresse réelle est calculée.

2.3.2.3.3. *Gestion du TLB*

A chaque référence à une page non encore reprise dans le TLB, la page référencée entre dans le TLB., S'il n'y a plus de places libres, le mode de remplacement consiste à remplacer une entrée non utilisée (reference bit à zéro).

Dans le cas du système MVS, lors d'un changement d'utilisateur, la mémoire associative est mise à zéro.

2.3.2.4. Structure de l'espace d'adresse

L'espace d'adresse est partagée en trois zones : zone du système, où se trouvent les utilitaires du système et la partie non-résidente de celui-ci, zone privée ou de l'utilisateur, où ceux-ci logent leurs travaux, et zone V=R. Toute adresse de cette dernière zone correspond directement à une adresse de la mémoire centrale.

Les zones systèmes et utilisateurs ont une image sur la mémoire de réserve; certaines de leurs pages ont une image en mémoire centrale au cours du travail. La zone V=R n'a pas d'image sur mémoire auxiliaire, mais toutes ses pages occupées ont une image en mémoire centrale. L'adresse réelle a la même valeur que l'adresse virtuelle. Cette zone comprendra la partie résidente du système, et les programmes qui ont été fixés en mémoire centrale.

## 2.3.3. Système VS1

### 2.3.3.1. Introduction

Le système VS1 dispose d'un espace d'adresse unique; celui-ci fonctionne par partition fixe. Cette caractéristique implique des travaux planifiés et de volume connu.

La différence fondamentale avec le système Siemens est de n'effectuer les mises à jour et enlèvements de cadre, non pas à chaque défaut de page, mais en cas de manque de cadres.

Structure de l'espace d'adressage

L'espace d'adressage est unique. Connu tel, plusieurs travaux coexistent dans un espace logique continu. Cela nécessite un système de gestion de cet espace qui place les travaux, leur attribue les ressources nécessaires dans cet espace.

Dans le VS1, le mode de gestion choisi a été celui des partitions fixes. Les partitions ont pour taille un multiple de 64 kilobytes et donc du segment. La répartition des partitions se fait à l'initialisation du système.

Chaque partition est elle-même subdivisée en sous-ensembles contenant les programmes, les zones de tables privées et de tampon d'entrée-sortie, et une zone de communication avec le système.

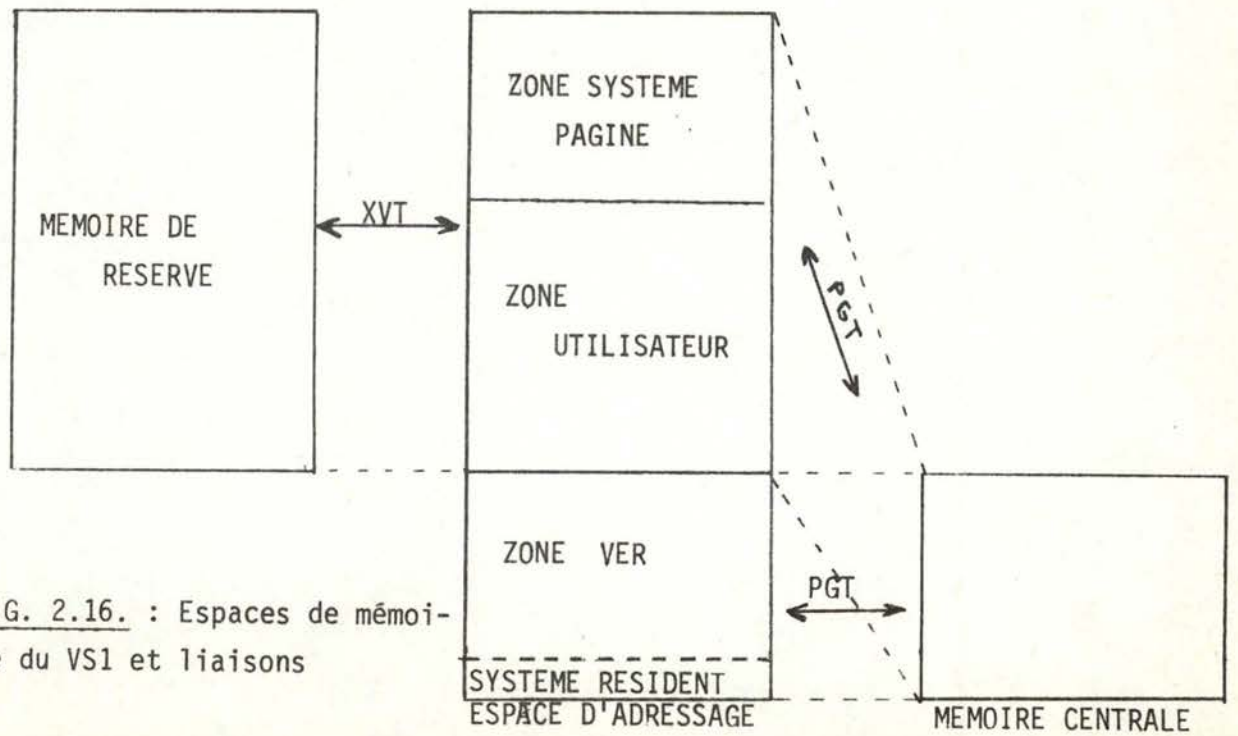



FIG. 2.16. : Espaces de mémoire du VS1 et liaisons

2.3.3.2. Gérant de la mémoire

2.3.3.2.1. *Introduction*

Dans le système VS1, les fonctions de mise à jour et de nettoyage des cadres alloués aux travaux ne sont pas faites travail par travail, suivant les événements liés à ce travail. Elles sont au contraire remplies pour tous les travaux à des moments dépendant de l'état global du système (manque de cadres, tranche de temps, entrée dans l'état d'attente du processeur).

Les fonctions de gestion peuvent se séparer en trois groupes. La première (P.M.R., page management routine) est une tâche qui a pour but de répondre aux défauts de page et autres demandes de place en mémoire centrale. La seconde fonction est celle de mise à jour des cadres et de nettoyage. La troisième est une tâche de désactivation et de réactivation.

2.3.3.2.2. *Gestion des demandes (P.M.R.)*

Lors d'un défaut de page, il y a interruption du processeur et passage dans le mode privilégié à une routine qui analyse s'il s'agit d'un vrai ou d'un faux défaut de page, c'est-à-dire si la page demandée est toujours en mémoire centrale ou non. Si la page est toujours en mémoire centrale, il n'y a pas défaut de page; on met les indicateurs à jour; le cadre rejoint la chaîne des cadres alloués.

En cas de vrai défaut de page, le système éveille la tâche de gestion des demandes (P.M.R.)

Elle va commencer par traiter tous les défauts de page, et leur attribuer un cadre pris dans la chaîne des cadres non alloués (A.P.Q., available page queue). Il n'y a pas ici de limitation dans le nombre de cadres qui peuvent être réclamés par un travail. En cas d'incapacité à satisfaire les demandes, le gérant des demandes va éveiller la tâche de mise à jour (P.Me.R., page measurement routine)

Si cette dernière ne parvient pas à fournir assez de cadres, le gérant des demandes éveille alors la tâche de désactivation.

A chacun de ces éveils, le gérant des demandes cède le processeur au gérant des tâches (T.D., task dispatiher) qui va le donner aux tâches appelées. Celles-ci, leur rôle achevé, rendront le processeur ou gérant des tâches, qui amènera au processeur le gérant des demandes.

Après avoir rempli toutes les demandes, le gérant des demandes va préparer les programmes canaux nécessaires pour amener dans les cadres alloués les pages demandées. Lors d'une fin d'entrée-sortie de pagination, le système fera débuter le premier des programmes canaux en attente.

Enfin, le gérant des demandes va répondre aux demandes de zone V=R, c'est-à-dire de programmes à fixer en mémoire centrale. Si nécessaire, il éliminera les cadres occupés et non fixés dans la zone allouée.

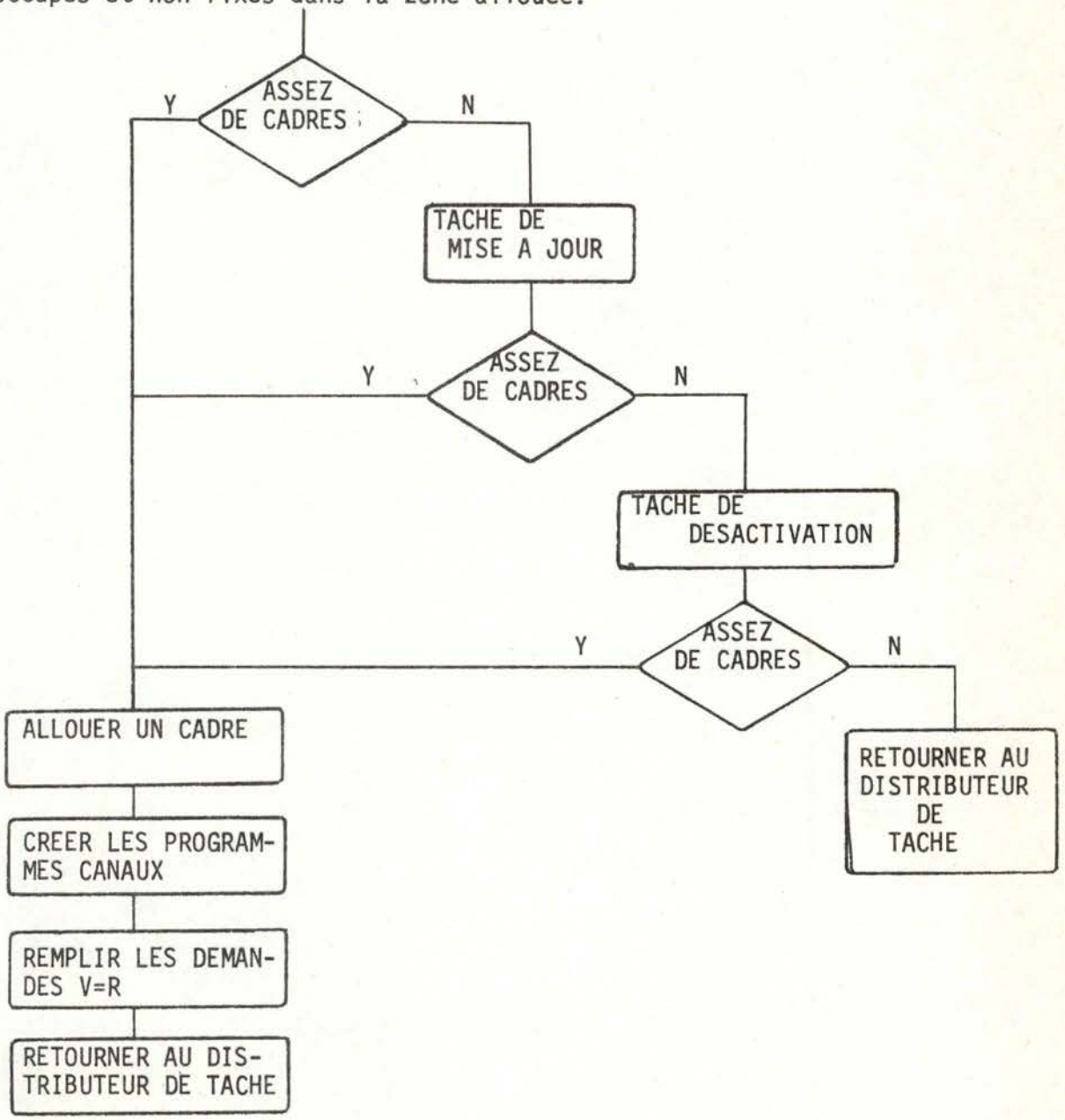



FIG. 2.17. : Tâche de gestion des demandes

2.3.3.2.3. *Tâche de mise à jour*

La routine de mise à jour a pour but de vérifier périodiquement l'état des cadres au point de vue de leur utilisation. Elle sera aussi appelée à éliminer les cadres trop vieux. Elle est exécutée à des périodes indépendantes des demandes des travaux, lors de la fin d'une tranche de temps et lors de l'incapacité de répondre aux demandes de cadres.

La différence entre ce système et le système Siemens étudié précédemment tient essentiellement pour ce point dans le mode de mesure de l'inutilisaton des cadres. Au lieu de mener la durée d'inutilisation et de la comparer avec une limite fixée, le système VS1 utilise une limite de temps variable.

Les cadres sont distribués sur des chaînes au nombre de deux plus le nombre de travaux actifs. Chaque cadre est lors de son acquisition placé sur la chaîne de niveau le plus élevé. A chaque passage de la routine de mise à jour, les cadres passent de leur chaîne à la chaîne de niveau inférieur. Ceux qui sont alors dans la chaîne de niveau zéro sont analysés. Les cadres qui n'ont pas été utilisés sont enlevés au travail qui les possédait. Les autres sont replacés dans la chaîne de niveau maximum, et leur bit de référence est mis à zéro.

La fréquence de chargements de chaîne est fonction non pas du temps de processeur utilisé, mais des demandes de pages non remplies et des fins de tranches de temps, donc du volume disponible en mémoire centrale.

L'élimination des cadres se fait donc non pas seulement en fonction de la durée d'inutilisation mais aussi de l'état du système. Elle se fait en cas de besoin et non de manière automatique. Cette philosophie se retrouvera sur le système MVS.

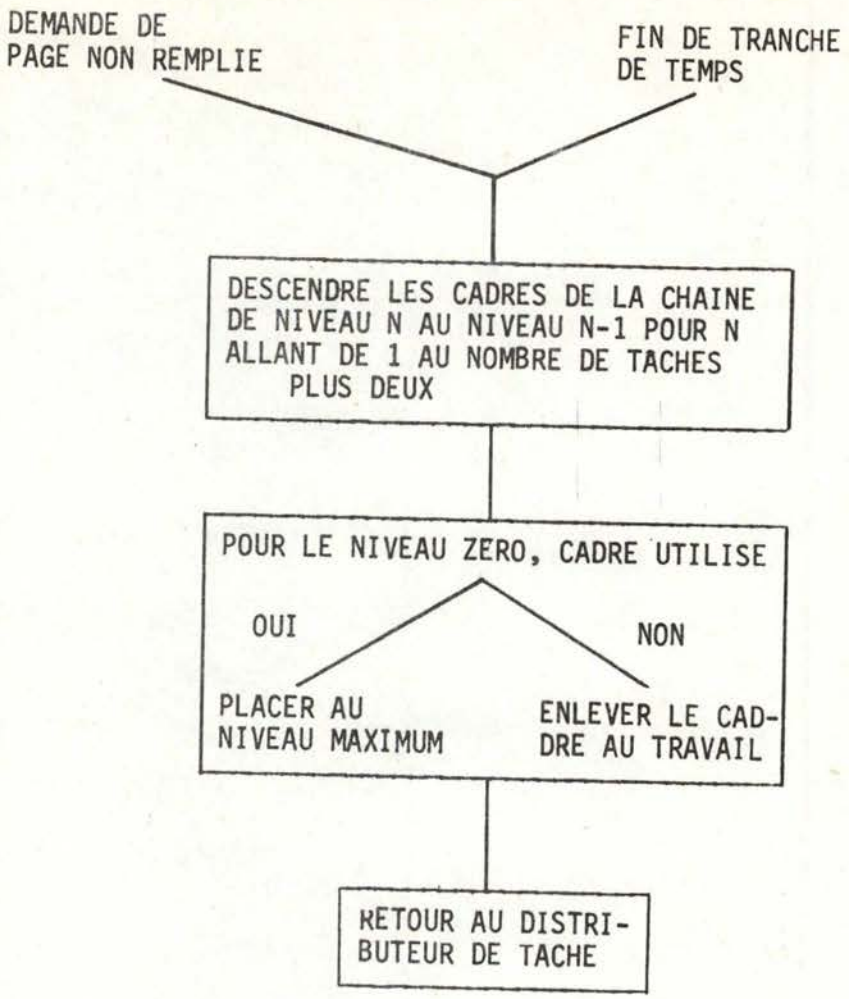



FIG. 2.18. : Tâche de mise à jour

2.3.3.2.4. *Tâches d'activation et de désactivation*

Introduction

En cas de pénurie des cadres, si l'appel à la tâche de mise à jour ne suffit pas, le gérant des tâches (T.D., task dispatcher) va éveiller la tâche de désactivation. Celle-ci a pour but d'enlever à un travail les cadres dont il dispose et de l'endormir pour une période donnée. Lorsqu'il n'y a plus de travaux actifs prêts à utiliser le processeur central, le gérant des tâches réveille la tâche d'activation.

Désactivation et activation sont ici liés à l'état de la mémoire centrale, et non à la notion de tranche de temps.

Désactivation

Lors d'un manque de cadre qui ne peut être résolu par la tâche de mise à jour, le gérant des tâches appelle la tâche de désactivation. Celle-ci va désactiver la partition la moins prioritaire et lui enlever ses cadres, placé dans la chaîne des cadres libres.

La tâche est affectée d'une durée de désactivation; on mémorisera aussi le nombre de cadres dont elle disposait.

Activation

L'activation se fait sur appel du gérant des tâches, s'il n'y a plus de tâches à utiliser le processeur.

On ne réactivera une tâche que si elle a fini son temps, et si le nombre de cadres libres dépasse ou atteint le nombre de cadres possédés par le travail avant sa désactivation.

Si aucune des tâches désactivées ne vérifie les deux conditions, on réveillera la plus prioritaire.

## 2.3.4. Système MVS

### 2.3.4.1. Introduction

Le système MVS est un système de time-sharing, à mémoire virtuelle. Chaque travail a un espace d'adresse propre, permettant une plus grande souplesse quant à l'introduction des travaux et une plus grande taille maximale des travaux.

La différence fondamentale vis-à-vis du système Siemens BS-2000 est d'effectuer le nettoyage des chaînes des cadres alloués aux travaux sur demande du gérant de la mémoire, pour tous les travaux au même moment. Pas plus que le VS1, il n'utilise la notion de volume allouable maximum de cadre.

Structure de l'espace d'adresse

Les espaces d'adresse sont séparés en trois zones : la zone commune, la zone privée et la zone V=R, suivant par là la structure du système VS1. A la différence de celui-ci, la zone privée ou zone utilisateur se trouve reproduite en autant d'exemplaires qu'il y a de travaux, chacun d'entre eux y logeant ses programmes.

Lorsqu'un travail perd le processeur et qu'un autre le reprend en charge, les entrées de segment correspondant à la zone privée doivent être changées pour contenir les informations liées au nouveau travail. Chaque travail a donc une série de groupes d'entrées de table qui lui sont propre et qui seront adressées au travers des segments.

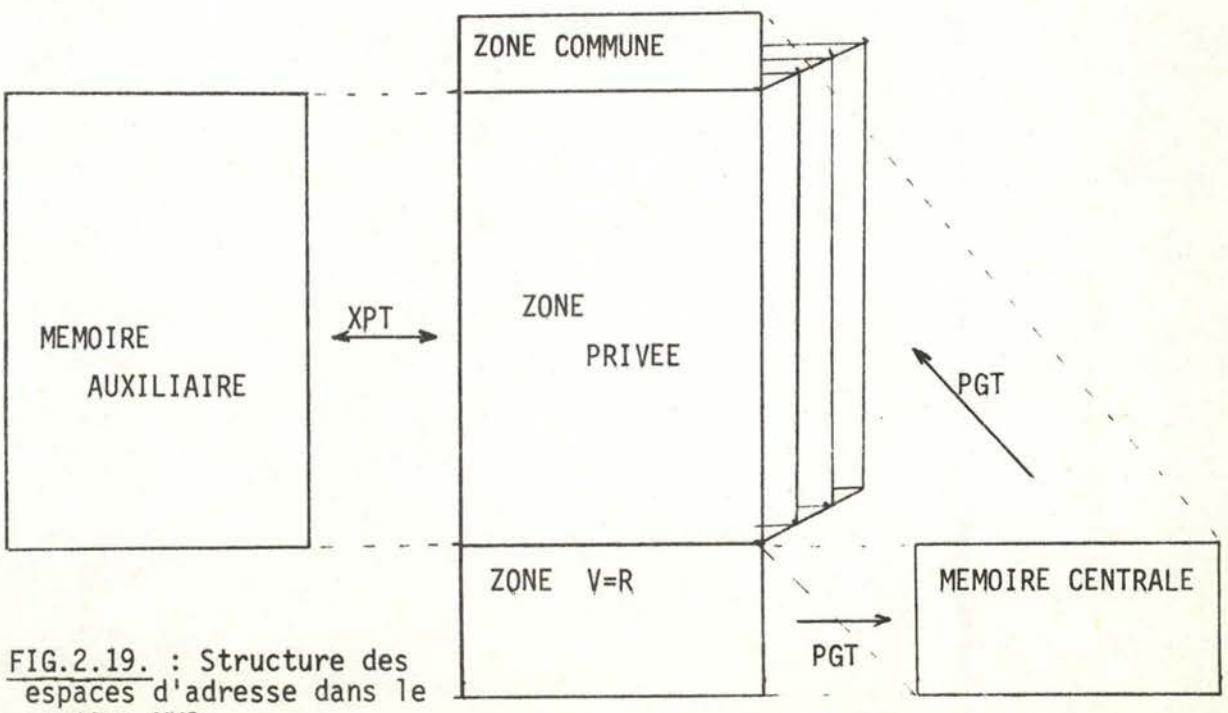



FIG.2.19. : Structure des espaces d'adresse dans le système MVS

2.3.4.2. Gérant de la mémoire

2.3.4.2.1. *Introduction*

Le gérant de la mémoire est fort semblable à celui du système VS-1. Une différence importante consiste dans le mode de mesure de l'inutilisation d'un cadre. Il ne s'agit plus ici d'appartenir à un niveau ou un autre dans les chaînes de cadres, avec vérification d'utilisation au niveau zéro, mais d'un compteur lié au temps d'utilisation du processeur par chaque travail.

Une autre provient du fait que le système MVS utilise des tranches de temps pour assurer une bonne rotation des tâches au processeur. A chaque tranche de temps, le travail est désactivé et ses cadres sont libérés. Les notions de désactivation et de réactivation n'ont donc que plus à voir avec celles contenues dans le système VS1.

2.3.4.2.2. *Prise en charge*

Le gérant des demandes va constituer pour chaque demande un bloc contenant les données nécessaires. Les types de demandes sont similaires à ceux du VS1 ainsi que leur ordre de prise en charge.

Le schéma de gestion présenté pour le VS1 reste valable ici, sauf pour le traitement en cas d'échec. Si une demande ne peut être remplie, elle reste en attente. Il y a éveil d'une tâche de mise à jour et d'élimination des cadres dès que le nombre de cadres disponibles tombe sous une limite donnée.

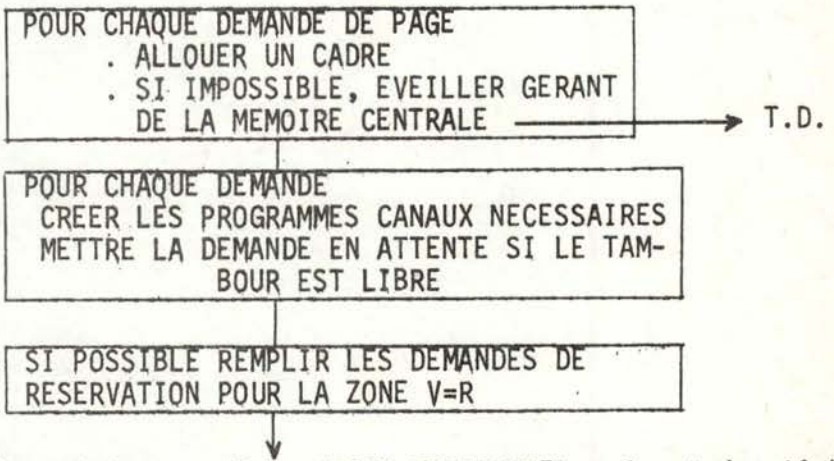



FIG. 2.20. : Gérant des tâches T.D. (TASK DISPATCHER, gérant des tâches)

2.3.4.2.3. *Gérant de la mémoire centrale*

Ce gérant a pour but de vérifier les disponibilités en cadres de la mémoire centrale. Il dispose pour cela de deux indicateurs. L'un est la limite inférieure du nombre de cadres libres, l'autre la limite supérieure du nombre de cadres fixés en mémoire centrale.

Si le nombre de cadres libres tombe sous la limite indiquée, le gérant appelle une routine de mise à jour, qui va calculer pour chaque cadre de chaque travail le temps d'inutilisation, à partir du temps pendant lequel le travail a occupé le processeur depuis le dernier passage de la routine, et du bit de référence de la page. Si celle-ci a été référencée, le temps d'inutilisation (UIC, unoccuped interval count) est nul. Sinon, il est augmenté du temps d'occupation du processeur. La routine trie ensuite les cadres par ordre croissant du temps d'inutilisation.

Le gérant appelle ensuite une seconde routine qui enlèvera les cadres avec le temps d'inutilisation le plus élevé, en nombre suffisant pour que le nombre de pages libres dépasse à nouveau la limite fixée.

Il préparera les entrée-sorties, si un des cadres enlevé à son travail doit être recopié sur la mémoire auxiliaire.

Si le nombre de page fixées dépasse une limite donnée, le gérant va désactiver les travaux fixés qui ont la priorité la plus faible pour revenir sous la limite. Ces travaux devront être réactivés par la suite et occuper à nouveau la place qu'ils occupent.

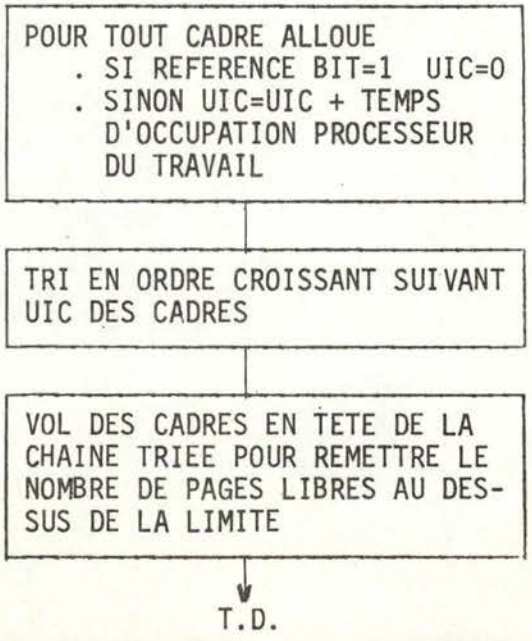



FIG. 2.21. : Mise à jour et vol des cadres

## 2.4. CONCLUSION

Nous avons décrit dans ce chapitre trois systèmes à mémoire virtuelle : le Siemens BS-2000, les IBM VS1 et MVS.

Ces systèmes présentent entre eux de grandes différences situées à trois niveaux : structure de l'adresse et des tables de gestion, structure du ou des espaces d'adresse et structure des algorithmes de gestion.

Des systèmes, proche au point de vue de leur but, comme le BS-2000 et MVS ont des différences fondamentales dans le mode de gestion des mises à jour et des nettoyages de cadres, dans la structure d'adresse.

Par contre, pour ces points de vue, MVS et VS1 sont fort proches. Mais la structure de l'espace d'adresse change du tout au tout.

On peut donc en conclure que le but adopté pour le système, time-sharing ou non, interactif ou non, n'influe pas de façon déterminante sur les structures des mémoires virtuelles.

Chaque solution implémentée a ses avantages et ses inconvénients. Il est impossible de trancher à priori. La seule méthode utilisable pour vérifier l'efficacité dans un cadre donné d'un système reste donc le simulateur.

° ° °

# CHAPITRE III :

## SIMULATEUR DE SYSTÈME D'EXPLOITATION À MÉMOIRE VIRTUELLE PAGINÉE

## 3.1. INTRODUCTION

Le but d'un simulateur est de fournir une idée du comportement du système étudié dans un cadre donné. Dans notre cas, il s'agissait de voir le comportement d'un système d'exploitation à mémoire virtuelle paginée placée dans une situation choisie.

Ceci implique la création d'un modèle pour le système, et donc l'étude de sa structure. Cette étude a été faite dans le premier chapitre. Elles permettent de dégager les éléments structurels indispensables en fonctionnement d'un tel système.

Ces éléments se partagent en deux groupes. Le premier qui tient compte des structures des tables de gestion, du ou des espaces d'adresse, de méthode de conversion d'adresse, nécessite une simulation avec comme unité de temps de base la dizaine de manoseconde. Elle servirait à vérifier l'efficacité en temps processeur consommé des algorithmes de conversion d'adresse. Vu le peu de données disponibles à ce sujet, nous avons préféré nous intéresser au second groupe.

Celui-ci tient compte des algorithmes de gestion des demandes et plus généralement le cheminement des travaux dans le système, avec une structure plus détaillée pour ce qui est lié à la mémoire.

Un simulateur demande des données initiales définissant la charge à laquelle la structure doit réagir. Ces données comprennent des données liées à l'ordinateur (taille de la mémoire centrale, des mémoires auxiliaires, vitesse de transfert des données vers les périphériques retenus) et à la charge (nombre de travaux, taille, durée, comportement des travaux).

Le simulateur devant suivre les travaux, il doit tenir compte non seulement de ce qui touche à la mémoire (gérant des demandes), mais aussi au parcours des travaux (entrée-sortie, activation et désactivation).

Les différents éléments de base de la structure du système d'exploitation que nous prenons en ligne de compte (gérant de la mémoire, des entrées-sorties et de la mémoire) interagissent. La forme pratique de cette interaction, ainsi que le contenu de ces gérants, dépendra du système simulé. Il est donc nécessaire d'étudier avec soin la possibilité d'adapter le simulateur à un autre système; la facilité d'adap-

tation est un critère important. Une manière de remplir cet objectif est de faire remplir par des routines, ou une autre forme de suite d'instruction, une fonction bien déterminée, faisant elle-même appel à d'autres routines si besoin en est. Le choix qui a été fait est de créer une routine à chaque décision mettant en jeu la politique de gestion du système d'exploitation.

Il est également important de rendre le simulateur le plus indépendant possible des données initiales, liées à l'ordinateur (et non au système) ou à la charge. C'est pourquoi nous avons choisi de réunir ces données dans un fichier dont le simulateur se servira pour son initialisation.

Nous avons conçu un simulateur pour le système Siemens BS2000 comme illustration des idées développées ci-dessus. Nous avons été obligé d'introduire certaines simplifications dans le modèle par rapport à la réalité, simplifications liées essentiellement au comportement des travaux pour les références à la mémoire, à la réactivation de ceux-ci, ainsi qu'à l'introduction de travaux nouveaux.

## 3.2. PRESENTATION DES DONNEES INITIALES

### 3.2.1. Charge

Par charge nous entendons nombre, taille et durée des travaux. Nous avons choisi de fixer le volume total en pages des travaux. La répartition du nombre de travaux et de leur taille propre se fait en fonction de la distribution des tailles de travaux observés aux facultés de Namur.

Il faut préciser ici que la taille est en fait le nombre maximul de pages de l'espace d'adresse occupées par le travail pendant son exécution. La durée du travail est laissée libre.

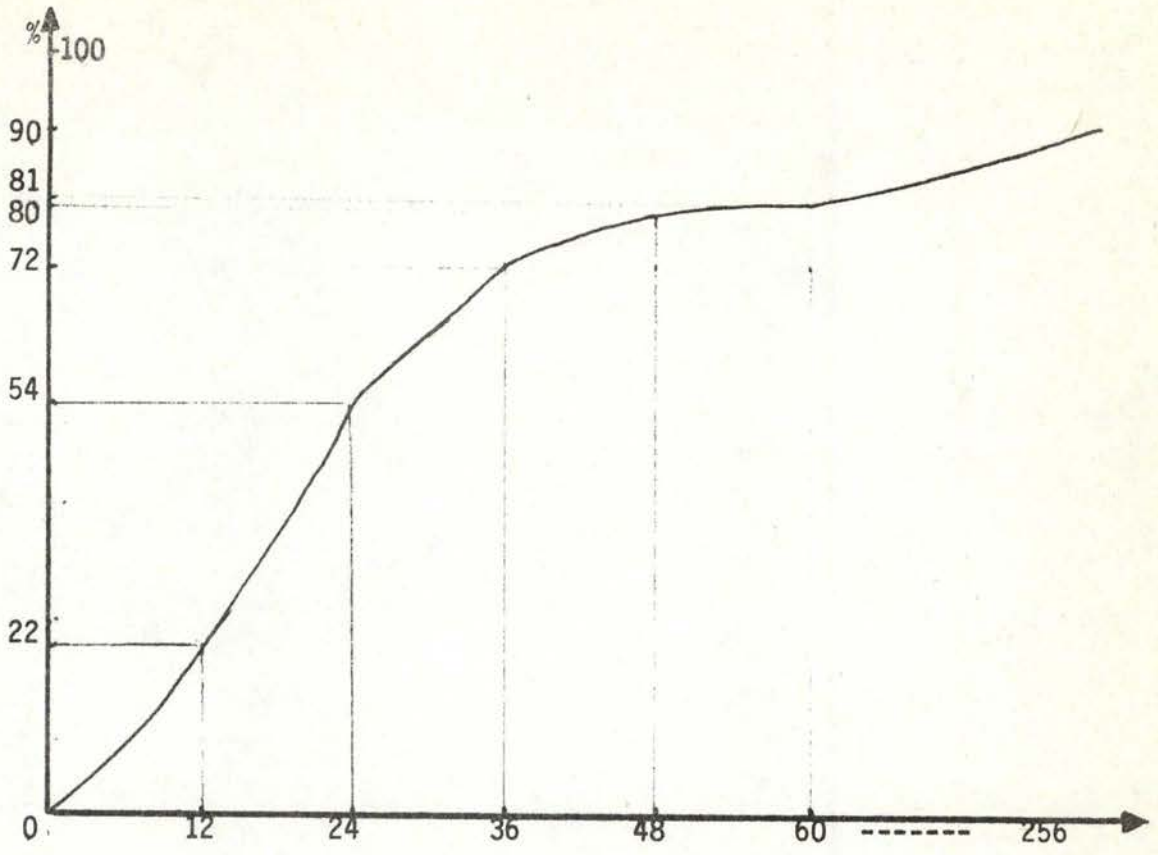



FIG. 3.1. : Distribution des tailles des travaux

3.2.2. Comportement des travaux

3.2.2.1. Introduction

Pour pouvoir simuler le système de gestion de la mémoire, il est nécessaire de disposer des défauts de page (qui les fait, quelle page est demandée) et de maintenir l'état des cadres (cadre utilisé ou non, modifié ou non). Cela implique la connaissance de la suite des références à l'espace d'adresse faite par chaque travail. Introduire cette suite telle quelle comme donnée est irréalisable, pour des raisons de taille et vu l'impossibilité de sa constitution. Nous avons donc préféré utiliser un modèle de comportement des travaux pour les références à l'espace d'adresse. Ce modèle va permettre de générer une suite de références à l'espace d'adresse, de gérer les cadres et de fournir au gérant de la mémoire les données nécessaires à son travail.

3.2.2.2. Modèle de comportement de programme

Le comportement des programmes varie assez fortement selon le type de travail à effectuer et la programmation du travail. Cependant on considère actuellement que le comportement par localité est une caractéristique de tout travail. Chaque programme ne référence qu'un sous-ensemble de son espace d'adresse pendant une durée assez longue; à la fin de celle-ci, il référence un autre sous-exemple; entre les localités, se déroule une phase intermédiaire, de comportement assez mal défini.

Nous avons choisi de ne pas tenir compte de cette dernière phase. Le modèle simplifié adopté consiste donc en localités, dont la durée et la taille sont générés aléatoirement à partir de distributions données. On doit également fixer les numéros de pages intervenant dans la localité.

Pour les tailles et durées des localités, nous nous sommes inspirés des travaux de Madison et Batson.(1)

Ces travaux analysaient le comportement en localité de divers programmes, et donnaient les distributions observées sur ces programmes des durées et des tailles de localité. Ils ne disposaient pas de la corrélation entre ces deux distributions, corrélation qui existe certainement. N'ayant aucune donnée précise sur celle-ci, nous n'en avons pas tenu compte. La distribution des tailles a été généralisée en ramenant l'unité à une valeur standard, celle-ci étant mesurée en pourcent de la taille du travail.

---

(1) Characteristic of Program Localities
Madison, Batson, Communication of the A.C.M. - May 1976

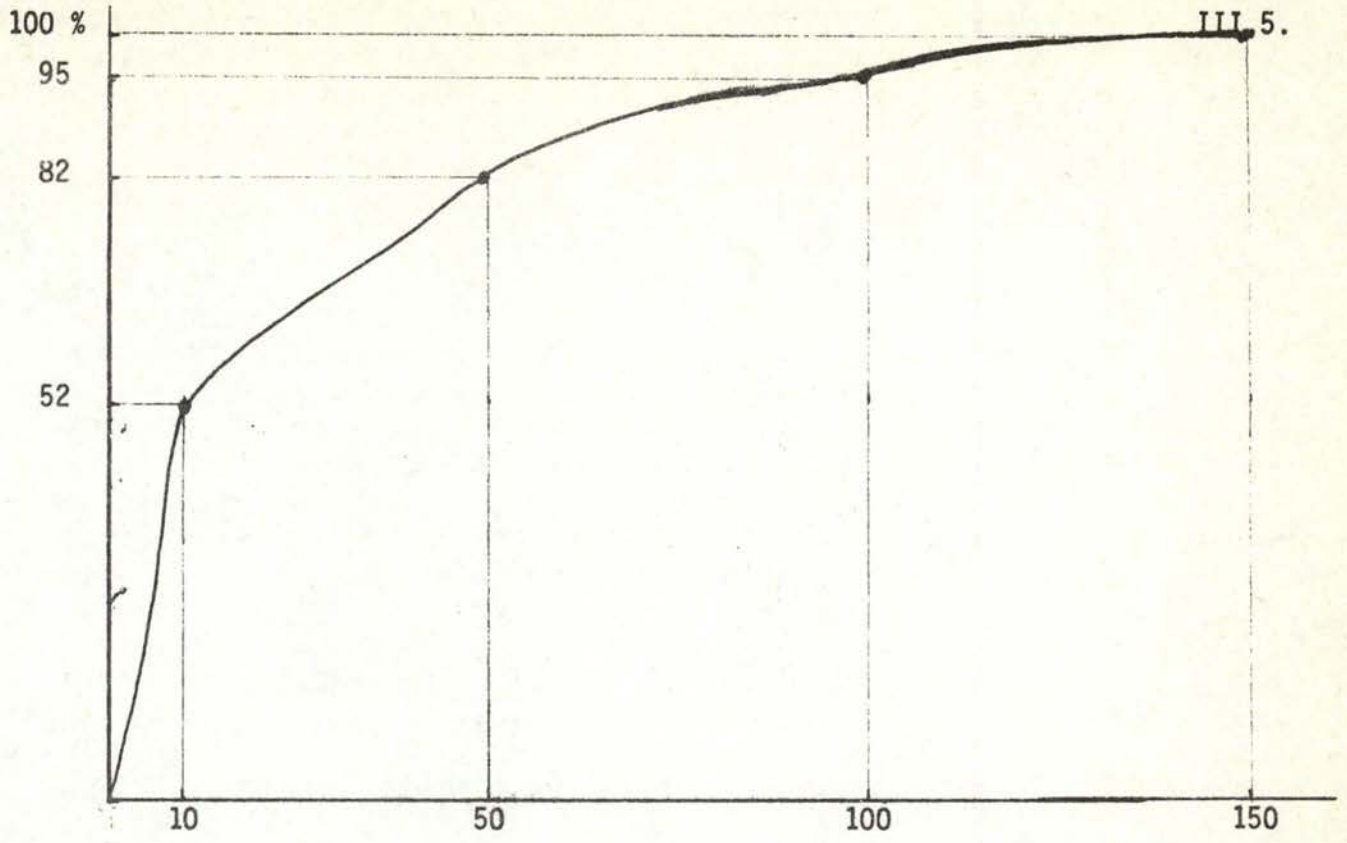



FIG. 3.2. : Distribution des durées de localité

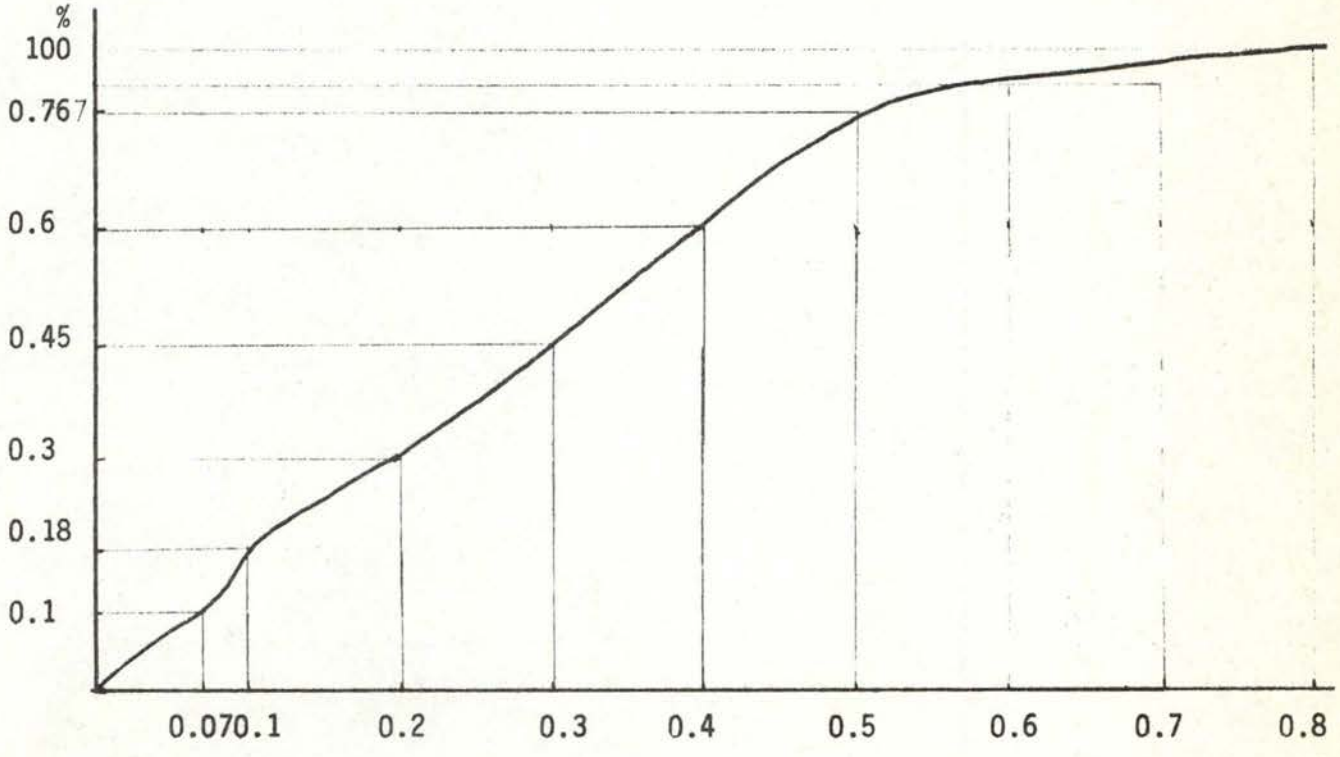



FIG. 3.3. : Distribution des tables de localité (en % du volume du travail)

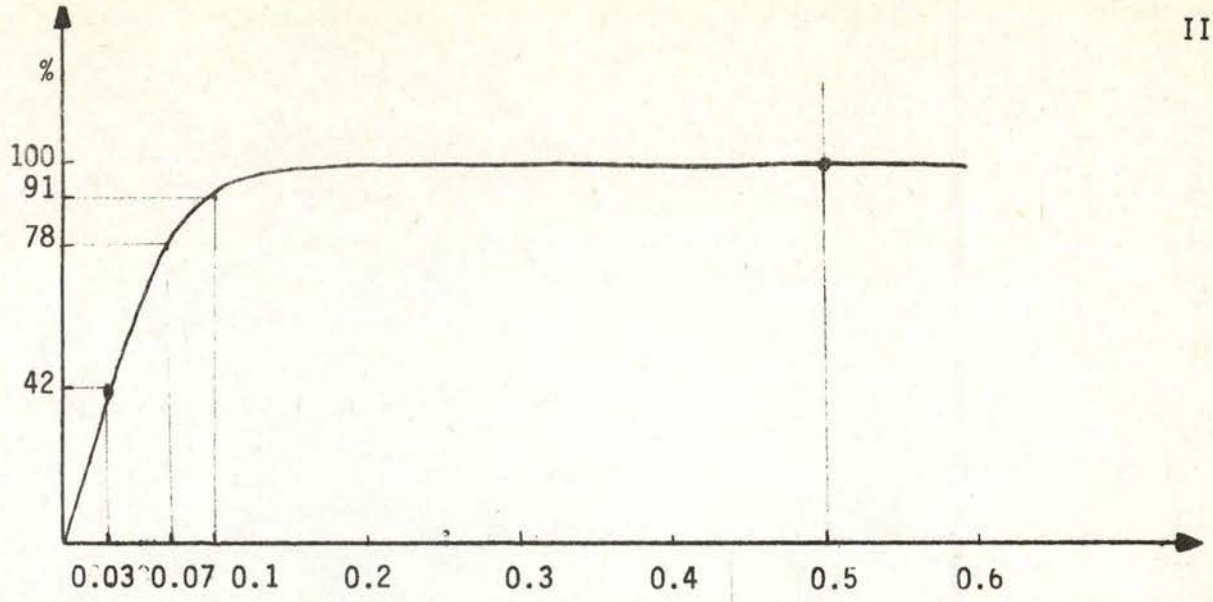
Les pages contenues dans la localité doivent être fixées lors de l'entrée dans celle-ci. Pour des raisons de facilité de gestion, nous avons décidé de constituer chaque localité d'une suite continue de pages, dont la première est choisie aléatoirement.

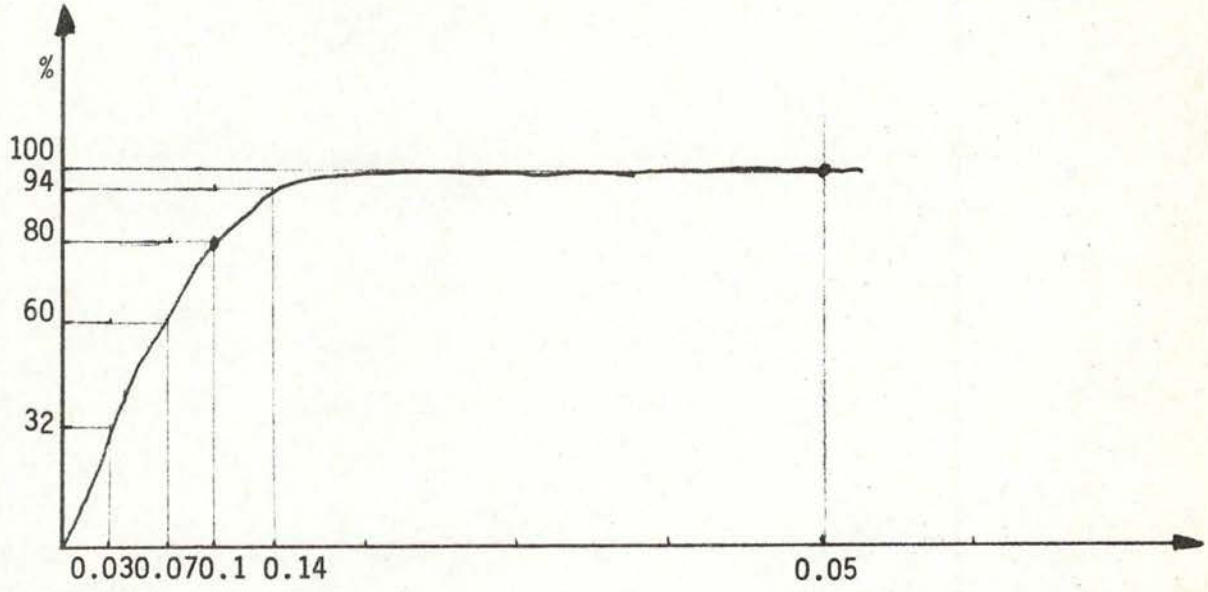
Reste à fixer le mode de référence aux pages dans une localité. Nous nous sommes inspirés des traux de J. Lenfant et P. Burgevin(1) qui, pour divers programmes, ont analysé la distribution des références aux pages. Ces distributions, étudiées au niveau du programme peuvent être utilisées comme telles au niveau de chaque localité. Un programme peut être vu comme une localité qui aurait pour taille et pour données, celles du programme; le comportement d'une localité quelconque ne peut être fort éloignée du comportement du programme global.

Comme les trois distributions étudiées par Lenfant (1) sont assez différentes, nous avons préféré pour généraliser le comportement dans la localité, tenir compte des trois. Lors de l'entrée dans une nouvelle localité, on choisit aléatoirement une des trois distributions qui servira pour la durée de la localité. Ces distributions étant exprimées en pages pour un programme donné, nous les avons ramenés à une unité commune à tous les travaux en les exprimant en pourcentage de la taille du travail.

---

(1) Comportement des programmes dans leur espace d'adresse. Application à la gestion des mémoires hiérarchisées. J. Lenfant.
Thèse présentée devant l'Université de Rennes, 1974



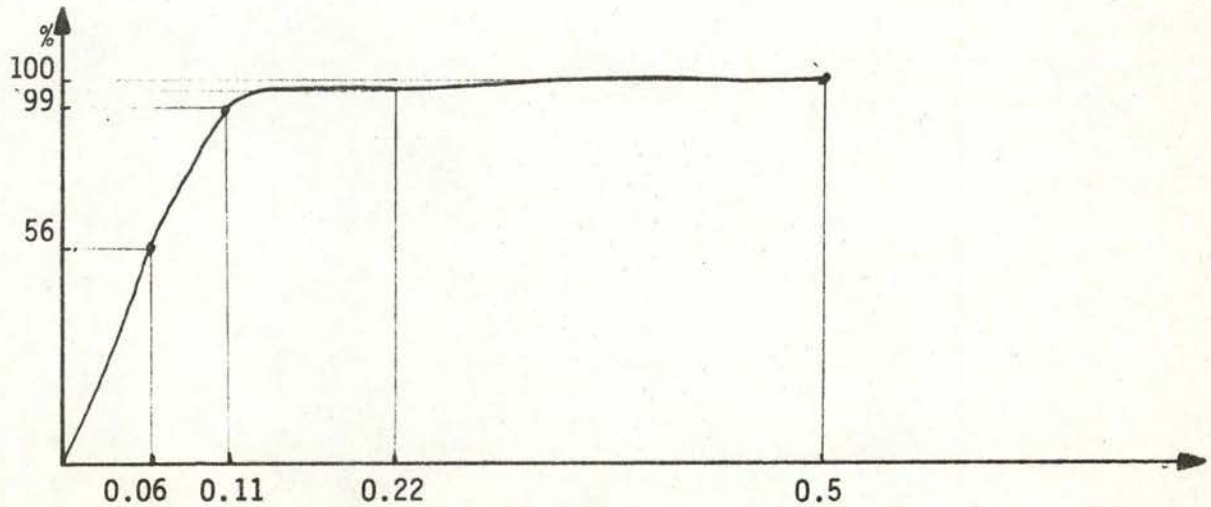




FIG. 3.4. : Distribution des références aux pages, en fraction de la table de la localité

Diverses critiques peuvent être émises. La première tient à l'utilisation de données provenant d'études sur quelques programmes, comme base de modèle global. Mais, si les données numériques sont liées aux programmes étudiés, les structures observées ne s'éloignent guère des structures de comportements de programmes généralement admises. Une solution alternative aurait été d'utiliser un modèle mathématique; mais se serait alors posé le problème des données nécessaires pour fixer la valeur des paramètres du modèle, ce qui nous ramène au même problème. Nous avons donc choisi d'adopter la solution la plus simple au point de vue de la recherche des données, et de leur utilisation.

La seconde tient aux simplications apportées au modèle. On ne tient en effet pas compte de la corrélation entre taille et durée de localité, ni de l'existence d'une période instable entre deux localités. Cette simplification est due en partie à l'absence de données, en partie à la nécessité d'avoir un modèle utilisable dans la simulation.

Les conclusions seront donc valables uniquement pour des programmes bien "localisés" (voir programmation structurée) et uniquement pour les périodes stables.

On peut donc conclure en disant que le problème principal tient au jeu de données disponibles et utilisables quant au problème du comportement des programmes.

### 3.2.3. Unité de temps

Nous avons choisi comme unité de temps la durée entre deux références à la mémoire. Le calcul de cet intervalle de temps a été fait à partir d'analyses de programme faites par Lenfant (1) à Rennes et par le groupe MIMOSA aux Facultés de Namur. Pour différents programmes, on dispose du pourcentage de références par type d'instruction. Connaissant le temps moyen d'exécution d'une instruction de ce type et le nombre moyen de pages référencées par instruction, on peut calculer le temps entre deux références. Les trois exemples fournis par Burgenin et Lenfant donnaient des durées de l'ordre de 0.9 à 1.2 microsecondes. Le programme traité par M. Noirhomme-Fraiture fournissait un intervalle de 1.05 microseconde entre deux références.

---

(1) Comportement des programmes dans leur espace d'adresse. Application à la gestion des mémoires hiérarchisées. J. Lenfant
Thèse présentée devant l'Université de Rennes, 1974

Cette durée varie évidemment en fonction du programme et du type d'instruction traitée, mais aussi en fonction du hardware de l'ordinateur et de la vitesse d'exécution des instructions. Nous avons donc pris la mesure faite sur le Siemens BS2000 comme base, puisqu'elle n'est pas contredite par les autres mesures.

Un correctif a cependant dû être apporté. Il est dû au fait qu'une partie des références à la mémoire sont faites dans des routines fixées en mémoire centrale, routines de gestion du système en général. La partie du temps de fonctionnement du processeur qui est passée dans ces routines est de l'ordre de quinze à vingt pourcents.

Comme nous ne tenons pas compte des instructions exécutées pour les programmes ou routines résidentes, nous avons choisi d'intégrer ces périodes en les distribuant dans les durées de chaque instruction. Nous avons augmenté la durée entre deux références à une page de 20 pour cent. Ceci ramène l'unité de base à 1.25 microseconde.

### 3.2.4. Taille des mémoires

La taille de l'espace d'adresse du travail est limitée à 256 pages. Cette limite est à respecter lors de l'introduction des tailles des travaux.

La taille de la mémoire auxiliaire est de 1024 cadres partagés sur deux tambours de 512 cadres.

Pour la mémoire centrale, nous ne tenons compte que de la partie paginable de la mémoire, c'est-à-dire la taille physique nmoins les volumes ficés par le système résident et les travaux (blocs de contrôle, zone d'entrée-sortie, tables diverses). Nous avons considéré cette taille comme fixe au cours de la simulation.

La valeur maximale a été fixée à 100 cadres; on doit introduire la valeur réelle à chaque simulation comme donnée d'entrée.

### 3.2.5. Entrées-sorties

Pour pouvoir suivre avec précision l'historique de chaque travail, il est nécessaire de tenir compte, dans le simulateur, des entrée-sortie. Le modèle utilisé est un modèle simplifié; il ne tient en effet compte ni des entrées-sorties lentes (terminal, bande magnétique, lecteur de carte ou imprimante), ni du parallélisme

possible pour les entrées-sorties rapides, ni enfin d'une priorité dans la prise en charge des demandes.

Il y a deux serveurs, symbolisant chacun un canal; les demandes en attente sont services dans leur ordre d'arrivée, le temps de service est le temps moyen de service d'une demande sur disque.

La distribution des intervalles entre entrées-sorties provient de valeurs observées sur le Siemens. Nous avons réparti par tranches de dis minutes les entrées-sorties et le temps processeur utilisé par les travaux; par à partir de ces données, on peut facilement calculer les intervalles entre entrées-sorties et créer une distribution de ceux-ci.

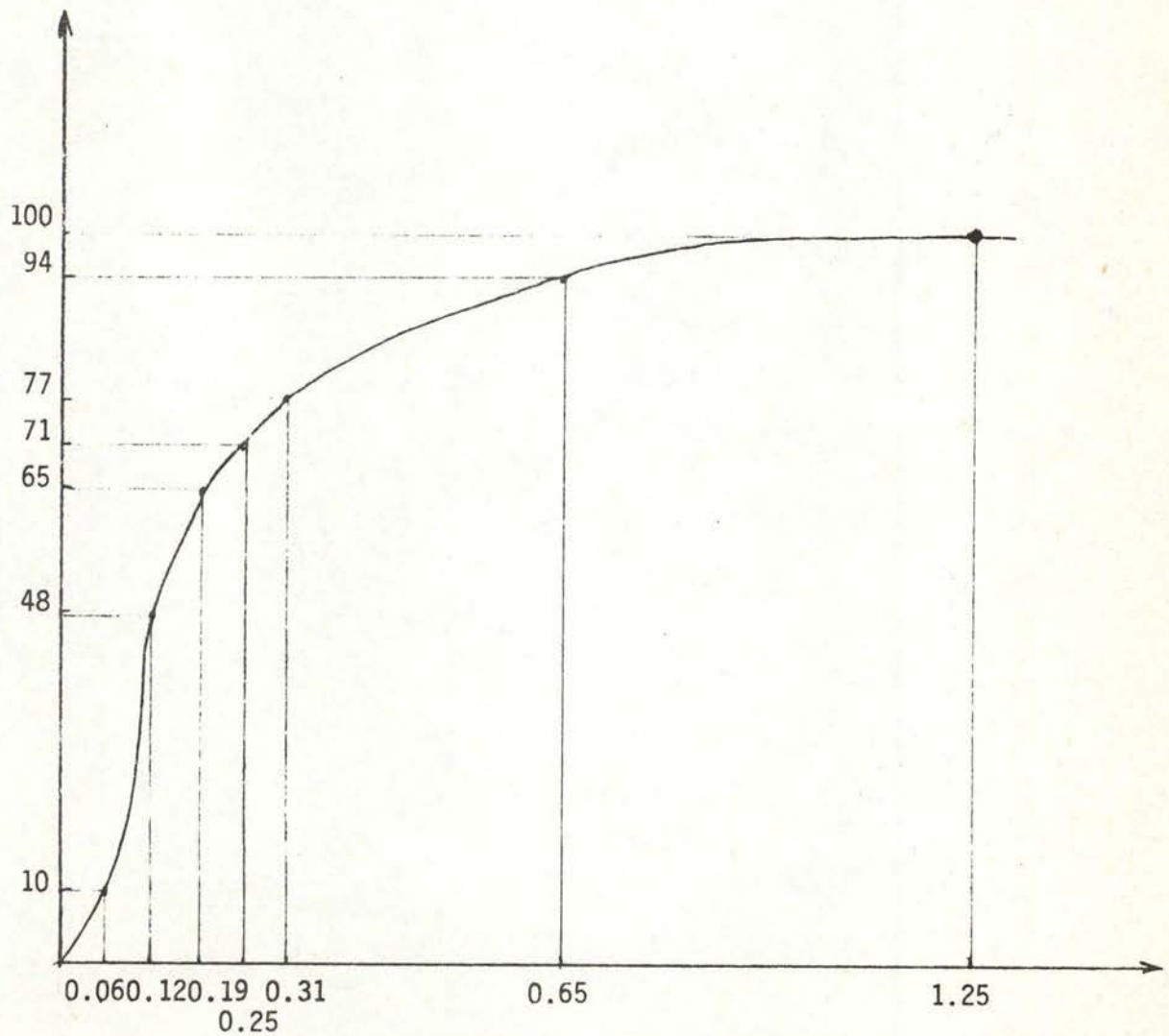



FIG.3.5. : Distribution des entrées-sorties, exprimées en unité de 100 secondes

La distribution obtenue est cependant peu précise. Outre que le calcul du temps entre entrée-sorties se fait sur base d'une valeur moyenne calculée par tranche de dix minutes, elle ne fait pas là distinction entre entrée-sortie rapide sur disque ou lente (bande magnétique, imprimante ou terminal), et ne tient compte ni du parallélisme possible entre diverses demandes pour un canal, ni d'un système de priorité.

Cependant, la distribution obtenue a une forme proche de celles observées par M. Noirhomme-Fraiture(1). On peut donc l'utiliser en temps que première approximation.

### 3.2.6. Conclusions

Nous avons voulu créer un simulateur où un maximum de données liées à l'installation proprement dite et non à la gestion de la mémoire, soient introduites au début de la simulation.
Ces données sont :

- la taille de la mémoire centrale
- le nombre de travaux, leur taille et leur durée
- les distributions de taille et de durée des localités
- les distributions de références aux pages
- la distribution des intervalles entre entrées-sorties.

Les principaux problèmes rencontrés ont été ceux liés à la recherche des données nécessaires, tant au point de vue comportement des programmes, entrée-sorties, ou données liées à l'ordinateur.

---

(1) Observations on programs used for feeding an operating system simulator, M. Noirhomme-Fraiture, Computer Performance Evaluation, Online Conferences Limited, Uxbridge, Enfland, 1976

## 3.3. ANALYSE STATISTIQUE DES RESULTATS

### 3.3.1. Résultats donnés par la simulation

Ces résultats sont de deux types. Le premier concerne le nombre de passages dans chaque zone du simulateur, chaque type de traitement à effectuer. Ces données permettent de fixer les itinéraires les plus souvent utilisés, et donc d'optimiser les algorithmes de traitement pour accélérer le traitement.

Le second type concerne essentiellement les interactions entre travaux et serveurs de la mémoire virtuelle. Il s'agit de mesures concernant les durées d'activité du processeur, les intervalles entre défauts de page, les temps d'attente avant d'être pris en compte des serveurs de défauts de page, et enfin temps de non-disponibilité d'un travail suite à un défaut de page, c'est-à-dire temps d'attente et temps de service.

Ces résultats permettent d'estimer l'efficacité du système de gestion de la mémoire : gestion des défauts de pages das cadres, des désactivations et des réactivations.

### 3.3.2. Validité des résultats

Les résultats obtenus par la simulation ne peuvent être considérés comme valables que s'ils restent stables au cours du temps. Il va de soi qu'on ne peut considérer le début de la simulation comme période stable : en effet, à cette période, la mémoire centrale est libre, et les travaux n'ont aucun cadre à leur disposition. Il s'agit d'une période transitoire et les résultats obtenus pendant cette période ne caractérisent pas le système. Il faut donc déterminer à quel moment la simulation devient stable.

Nous avons choisi de découper la simulation en un banc de simulations, la première utilisant des données externes pour s'initialiser, les autres utilisant comme entrée l'état final de la simulation précédente.

Pour chaque simulation, on obtient les distributions liées aux durées observées.

La méthode choisie pour tester la stabilité consiste pour chaque distribution observée, à tester son homogénéité pour un nombre donné de simulations consécutives.

On répartit les survenances observées dans des classes pour chaque simulation et chaque distribution. On utilise pour chaque distribution et k simulations un test d'homogénéité en calculant :

$$D^2_{k,d} = \sum_{i=1}^{k} \sum_{j=1}^{n} \frac{(R^e_{ij} - R^t_j)^2}{R^t_j}$$

où n est le nombre de classes

k est le nombre de simulations

$R^e_{ij}$ est le nombre de survenances de l'événement observé pour la simulation i et la classe j

$R^t_j$ est la valeur théorique du nombre de survenance dans la classe j sous l'hypothèse d'homogénéité

d est l'indice de l'événement observé.

Cette mesure est faite pour chaque événement, pour un ensemble de k simulations. Si pour chaque événement, la valeur de $D^2_{k,d}$ confirme l'hypothèse, on peut dire que la simulation est arrivée dans une zone stable.

On peut également observer l'évolution temporelle de la stabilité en réalisant m simulat-ons, et en appliquant la technique développée ci-dessus pour les k premières, puis les simulations de 2 à k+1, etc ...

On peut alors avoir une idée du gain dans la stabilité fait à chaque nouvelle simulation, et voir s'il est encore utile de continuer ou, si la stabilité ne s'accroit plus, stopper.

Pour rendre ces mesures efficaces, il importe que chaque simulation soit de durée constante.

### 3.3.3. Statistiques liées aux résultats

Pour chaque distribution retenue, nous donnons la moyenne, l'écart-type, et un intervalle de confiance à 95%, avec la supposition d'indépendance des mesures.

Nous donnons de plus une distribution des observations par classe de tailles égales, sauf celle qui va de la dernière valeur retenue à l'infini.

## 3.4. STRUCTURE DU SIMULATEUR

### 3.4.1. Introduction

Le simulateur de système orienté vers la gestion de la mémoire, a pour but de permettre d'analyser la conséquence sur l'efficacité du système d'un changement effectué au niveau de l'algorithme de gestion de la mémoire (et non au niveau des structures de tables ou d'espace d'adresse).

Pour faciliter cette adaptation, nous avons adopté une structure modulaire, chaque module correspondant à une fonction de la gestion. Le contenu de ces modules ainsi que le moment de leur appel seront fonction du système de gestion à simuler. Cette technique permet d'isoler facilement les effets d'un changement, et simplifie la mosidication au niveau du programme.

Le simulateur doit de plus permettre une simulation de durée suffisante, donc être rapide.

Il doit pouvoir être adapté facilement au niveau de l'analyse statistique, ce qui implique des routines statistiques autonomes, et un accès simple aux données à retenir pour cette analyse.

Le simulateur qui a été implémenté modelise le système Siemens BS2000, qui a été décrit au chapitre deux.

### 3.4.2. Chemin des travaux

Le chemin suivi par les travaux simule celui du système Siemens BS2000. Un travail peut être actif, prêt et attendre la libération du processeur central; il peut être en attente de fin d'entrée-sortie vers un disque ou vers la mémoire auxiliaire. Il peut enfin être désactivé, et attendre sa réactivation. Dans ce cas, contrairement au système réel, tout travail désactivé est automatiquement réactivé.

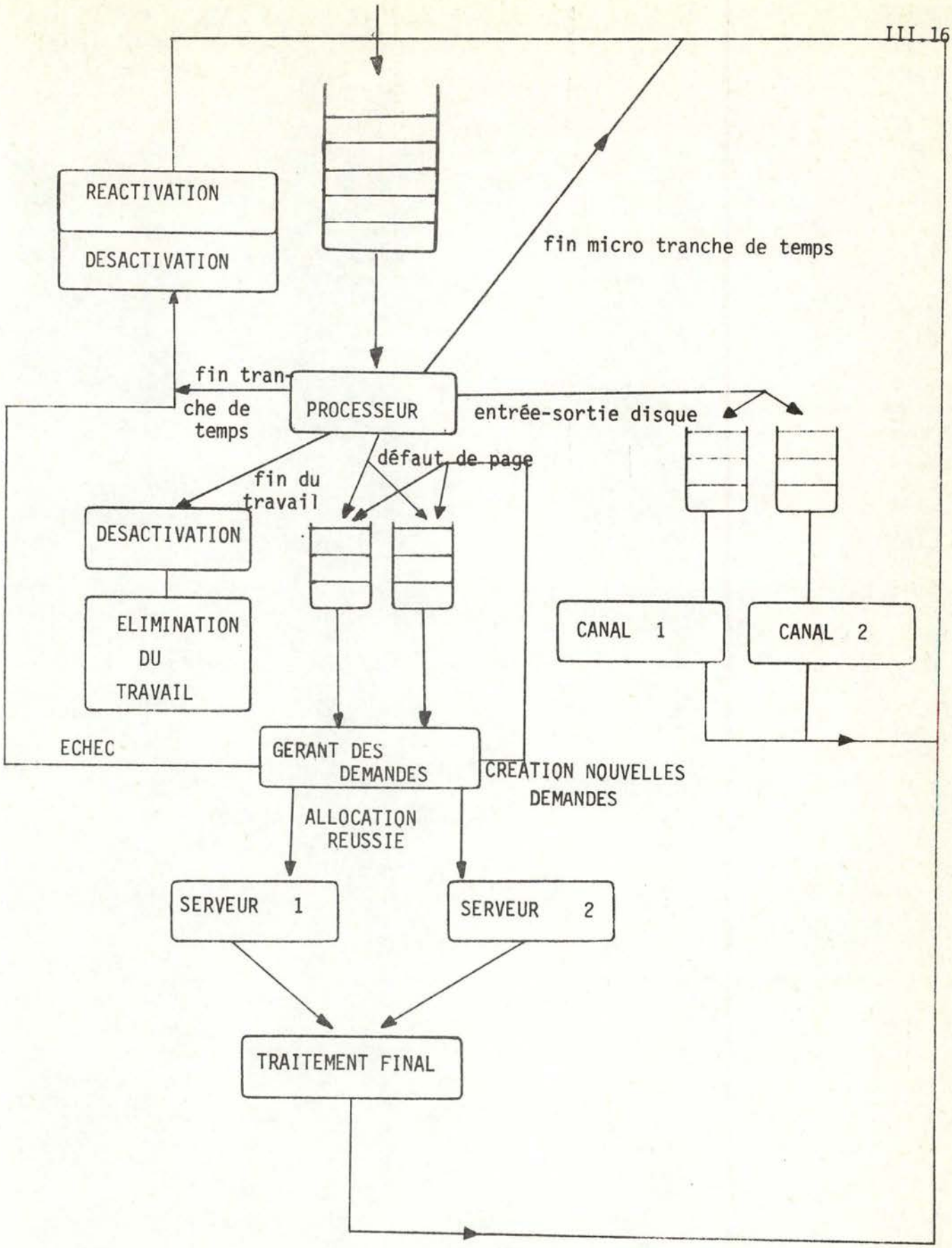



FIG.3.6: SCHEMA DES CHEMINS DES TRAVAUX DANS LE SIMULATEUR

### 3.4.3. Langage choisi par le simulateur

Il nous était possible de choisir soit un langage général de programmation soit un langage spécialisé; le seul disponible étant le langage SIAS, dérivé de GPSS seul à être implémenté sur le Siemens BS2000.

Le SIAS a pour avantage de gérer automatiquement la succession des évènements et les statistiques. Il présente cependant des inconvénients assez importants. Il est incapable de gérer des tables de données, tables indispensables pour stocker les données d'état des travaux, des pages et des cadres. Cette gestion se réalise dans des routines écrites en Fortran, posant alors le problème du passage des paramètres d'un langage à l'autre qui alourdit le programme SIAS. De plus, le nombre de routines Fortran utilisable est réduit à vingt. Cela limite donc fortement les capacités d'extension du simulateur actuel. Enfin, la difficulté de communication entre le Fortran et le SIAS créera également des problèmes lors d'un changement dans les algorithmes de gestion.

C'est pourquoi nous avons décidé de créer un gérant de la simulation écrit en Fortran. Ce gérant est très simple vu le petit nombre de serveurs, et surtout de type de serveurs différents par les traitements qui sont liés : deux serveurs d'entrée-sortie, deux serveurs de transfert de page et le processeur actuel. Il y a donc cinq serveurs, mais seulement trois types de serveurs, ce qui simplifie la gestion.

Nous avons donc aussi créé des routines statistiques de stockage des données au cours de la simulation, et de traitement à la fin de celle-ci.

### 3.4.4. Initialisation et terminaison de la simulation

Pour des raisons de facilité, la simulation débute dans un etat abstrait : tous les travaux sont prêts à acquérir le processeur, aucun cadre ne leur a été alloué. Cet état a pour conséquence de rendre le début de la simulation sans intérêt au point de vue statistique.

La simulation s'achève après une durée déterminée, à introduire dans la simulation.

### 3.4.5. Ecarts entre le modèle et le système réel

Nous n'avons pas introduit dans le simulateur la notion de chargeur de travaux, qui charge le contenu du programme que le travail veut faire exécuter, dans la mémoire de réserve. Nous n'avons pas non plus tenu compte des travaux interactifs et de leur comportement propre (désactivation lors d'une entrée-sortie vers le terminal).

Ceci peut se justifier par la durée d'une entrée-sortie au terminal ou celle de l'exécution d'un programme par rapport à la durée d'exécution du simulateur, qui est de l'ordre de quelques secondes.

D'autres points, comme le gérant des entrées-sorties ne sont qu'ébauchés et demanderaient un développement ultérieur.

### 3.4.6. Fonctions du simulateur

Le simulateur comprend cinq fonctions : le gérant de la simulation, un modèle du processeur central, du gérant des demandes celui des travaux et des entrées-sorties.

Le gérant de la simulation a pour but de choisir le prochain événement à traiter et d'appeler les routines appropriées. Il doit aussi gerer l'ecoulement du temps stopper la simulation à la fin de la durée prévue, et initialiser la simulation.

Le modèle du processeur central simulera la suite des références, à la mémoire faite par un travail, tenant compte des hypothèses de localité qui ont été faites. Il gèrera l'état des cadres, en fonction des références. En cas de défaut de page, de fin de micro-time-slice ou de time-slice, en cas de demande d'entrée-sortie, ou de fin de travail, il répond à la demande faite et retourne au gérant du simulateur.

Le gérant des demandes choisit la demande à prendre en compte, et effectue le traitement approprié au type de demande. S'il n'y a plus de demandes provenant des travaux, il gère les demandes internes (recopie des cadres de la mémoire centrale, et mise à zéro de cadres de la mémoire auxiliaire). Si la demande nécessite un transfert, le gérant occupe le serveur correspondant et retourne au gérant de la simulation. Dans le cas contraire, elle traite la demande suivante. Le gérant n'est appelé que si un des serveurs est libre.

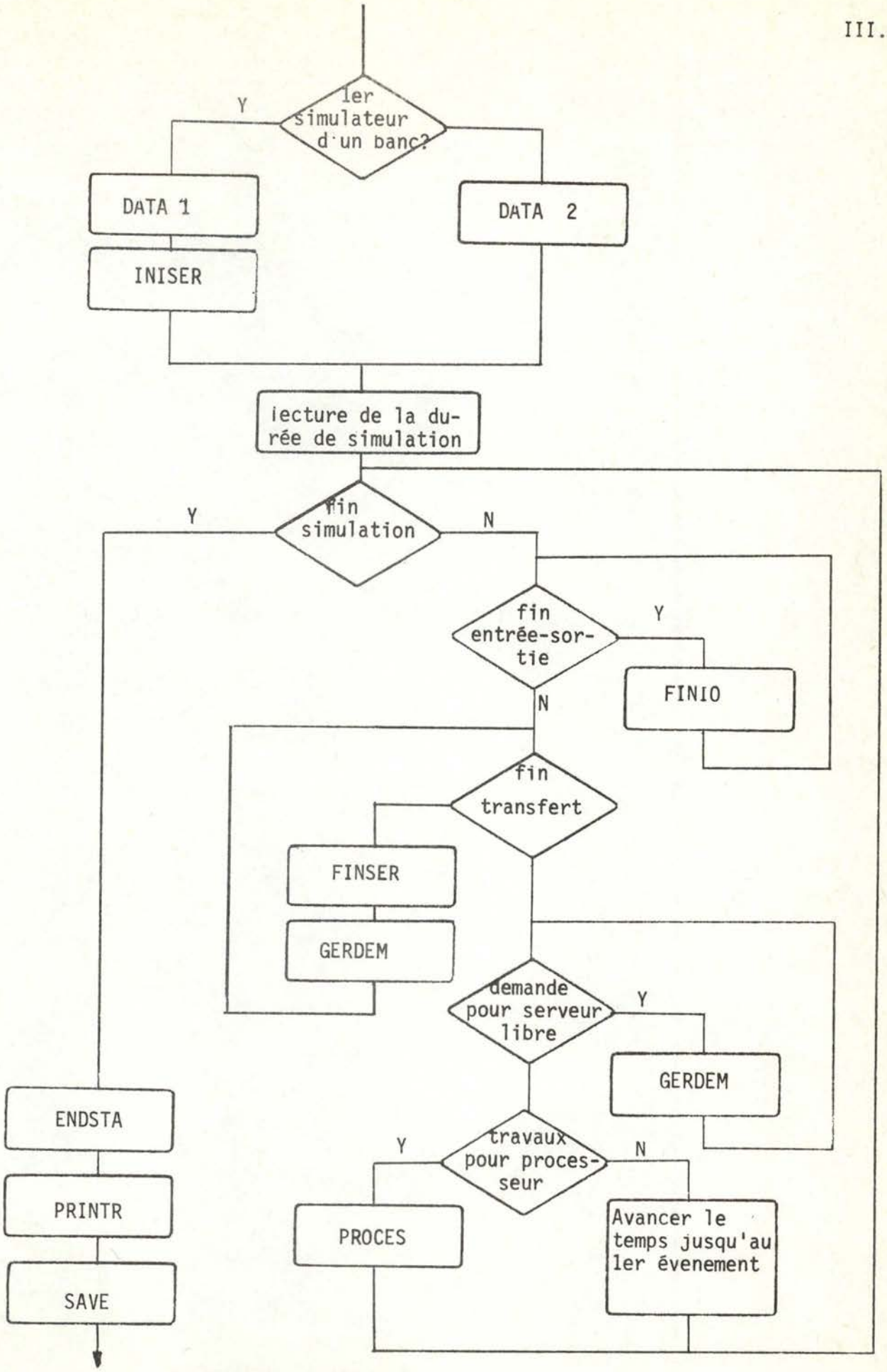
Le gérant des travaux s'occupe de la réactivation des travaux et de leur introduction dans la file d'attente du processeur.

Le gérant des entrées-sorties met les demandes en attente si le serveur est occupé et l'attribue sinon. Nous avons considéré comme serveur, les canaux. Dès qu'un canal prend une demande en main, il reste occupé jusqu'à la fin de la demande. Nous n'avons gardé dans notre modèle que les entrées-sorties vers les disques, dans un but de simplification.

### 3.4.7. Enchaînement des fonctions

Chacune des fonctions du simulateur décrites au paragraphe 3.4.6. est composée d'une ou plusieurs routines : une pour le modèle du processeur (PROCES), le gérant des demandes (GERDEM), deux pour les gérants des travaux (DESACT, desactivation d'un travail, et REACT, réactivation) et celui des entrées-sorties (une pour les débuts d'entrée-sortie (DEBIO) et une pour les terminaisons, FINIO). Le gérant de la simulation constitue le programme principal.

Chacune de ces routines en appelle d'autres, en fonction des décisions à prendre. Nous allons donner ici brièvement le schéma d'enchaînement des fonctions, et le contenu général des routines.




GERANT DE SIMULATION

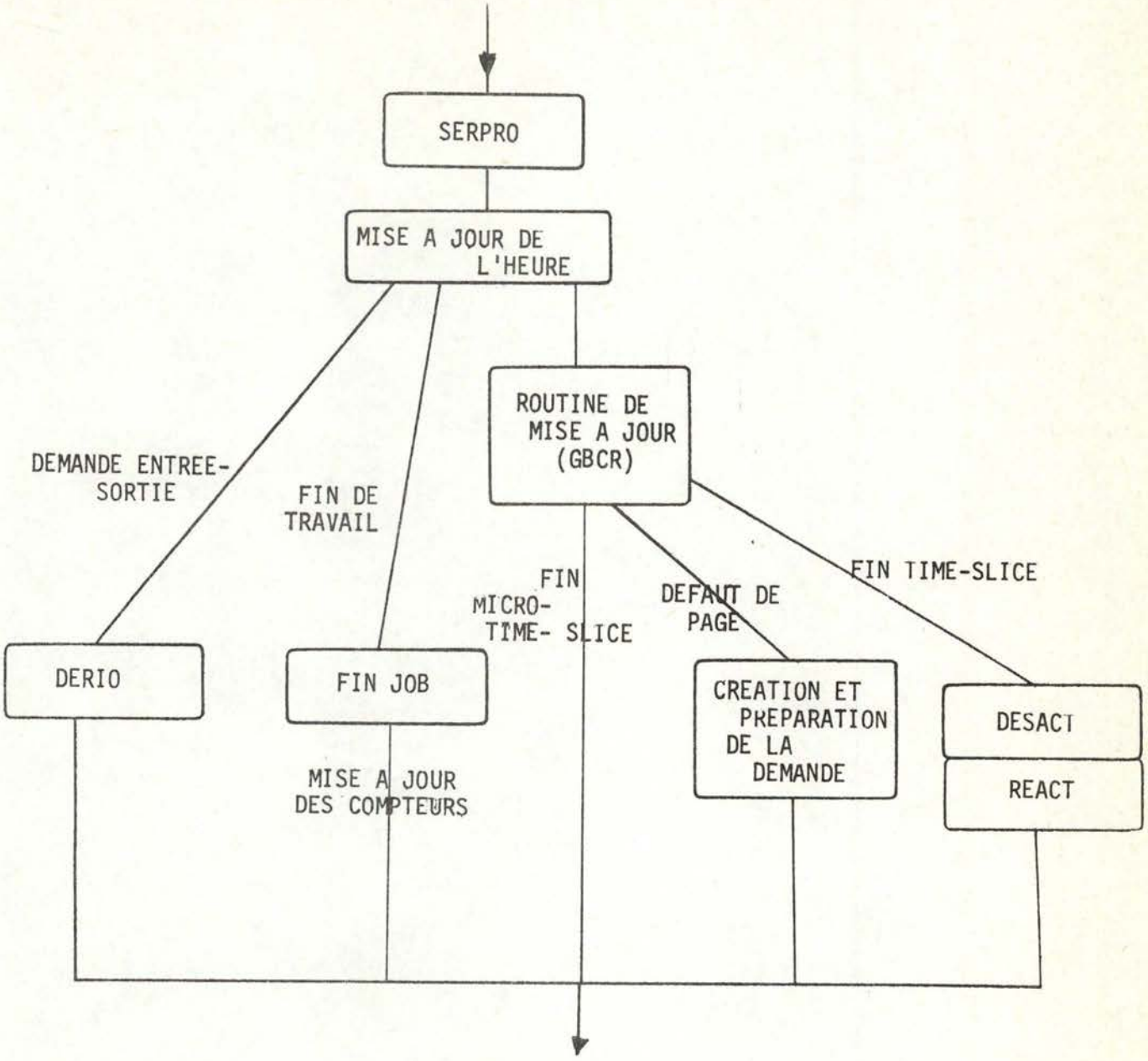
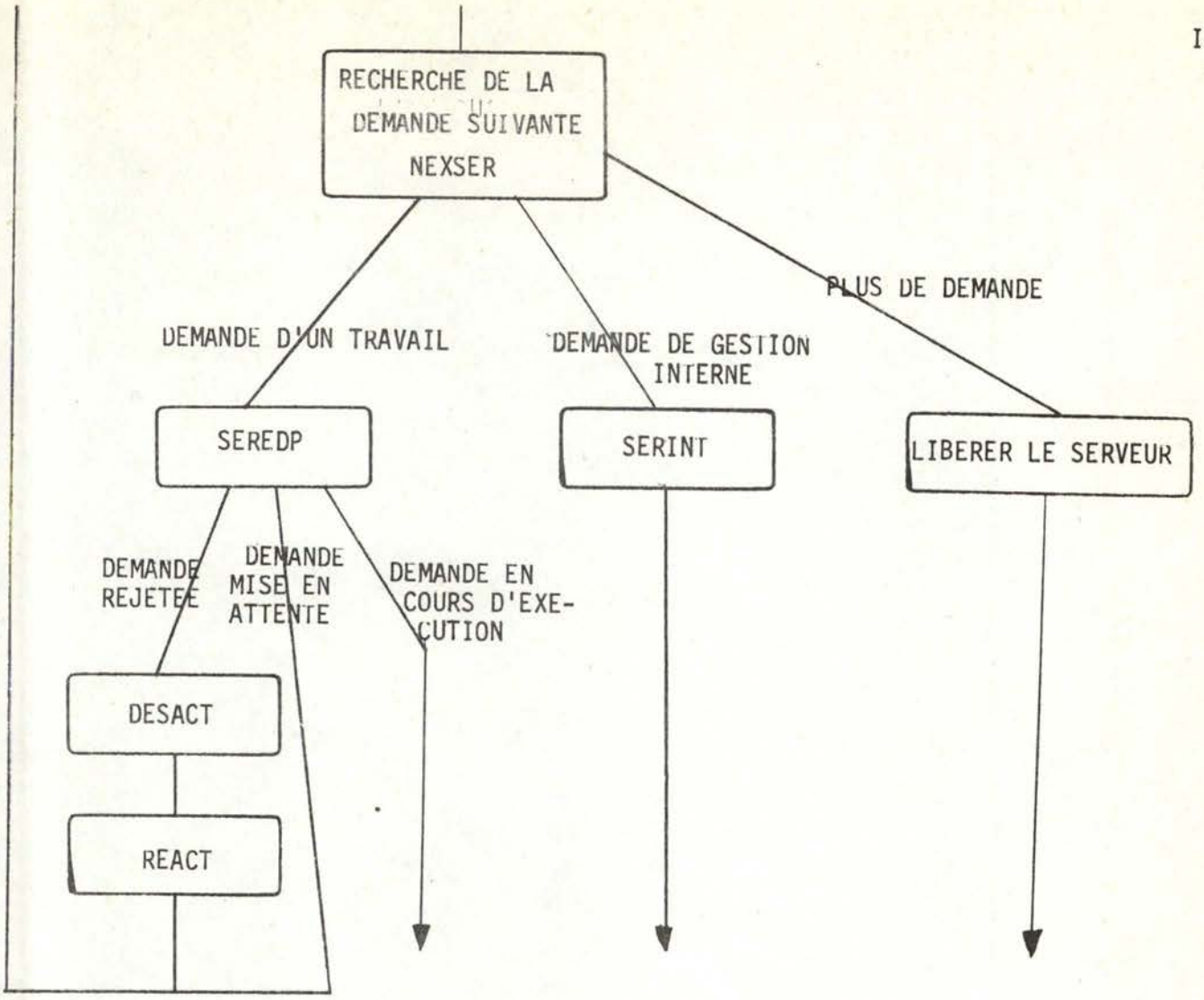
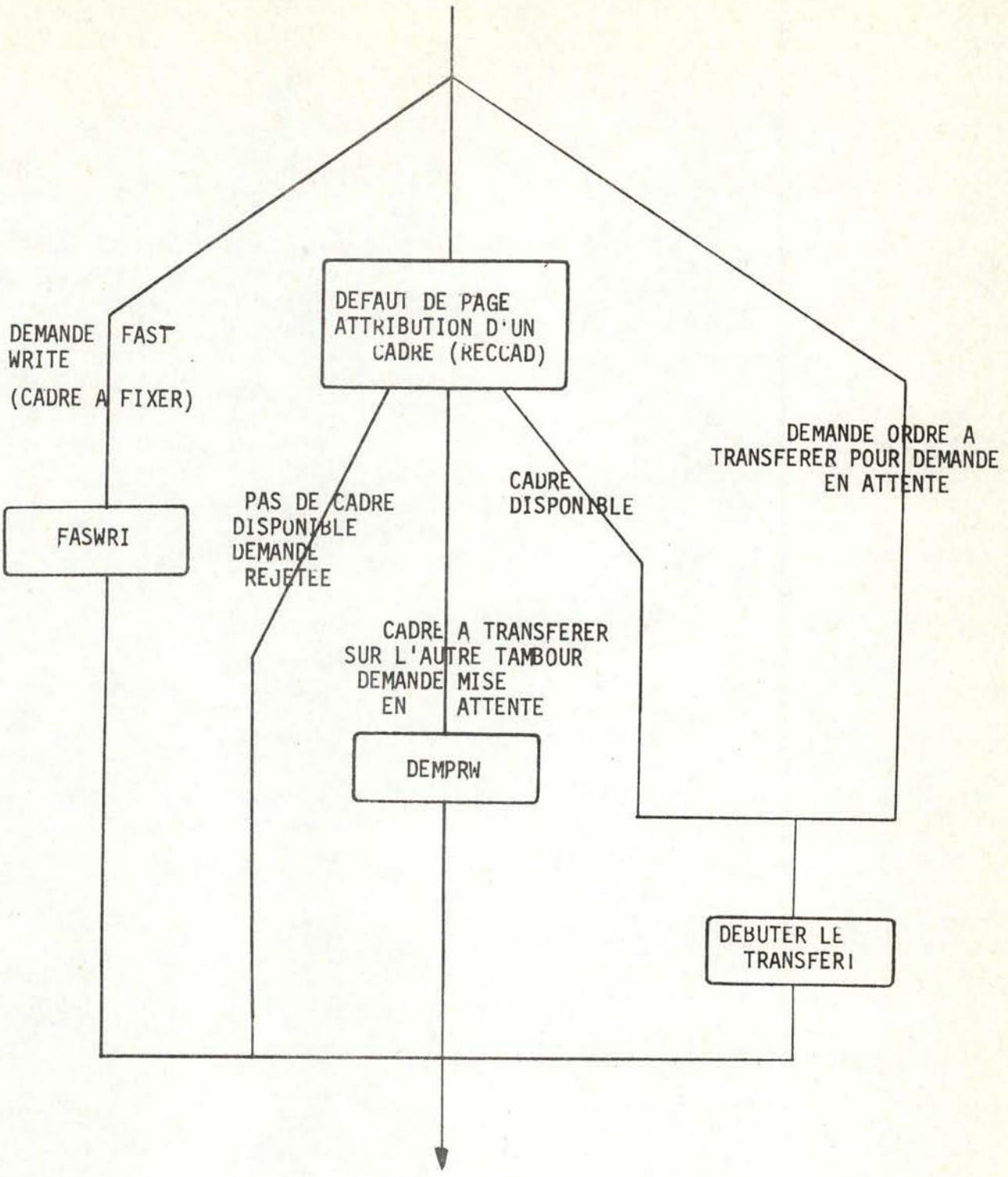



FIG. : MODELE DU PROCESSEUR ET DE SES ACTIVITES (PROCES)




GERANT DES DEMANDES DE MEMOIRE (GERDEM)




ANALYSE DE LA DEMANDE (SEREDP)

. DATA1 : Lecture des données initiales sur fichier pour la première simulation du banc de simulation (nombre, taille et durée des travaux, distribution des données liées au comportement des programmes)

. DATA2 : Lecture des données initiales sur fichier pour les simulations autres que la première du banc. Elle récupère les données sauvées par SAVE à la fin de la simulation précédente.

. DEBIO : Si un serveur d'entrée-sortie est libre, et s'il se crée une demande, elle est traitée, le serveur est occupé.

. DEMPRW :En cas d'allocation à une demande de page d'un cadre a transférer sur le second tambour, la demande est mise en attente, une autre demande prioritaire est générée pour accomplir le transfert du contenu du cadre.

. DESACT :Désactivation d'un travail, perte des cadres et placement dans la file d'attente des travaux désactivés.

. ENDSTA :Traitement de clôture des statistiques

. FASWRI :Demande d'un cadre à fixer en mémoire centrale pour le système ici erreur entraînant la fin de la simulation

. FINIO : Traitement final d'une entrée-sortie, réinsertion du travail dans les travaux prêts; recherche d'une autre entrée-sortie pour ce serveur.

. FINJOB: Retour des cadres dans les cadres libres, élimination du travail

. FINSER: Traitement final d'une demande de mémoire

. GBCR : Routine de mise à jour de l'état des cadres alloués aux travaux; élimination des cadres trop vieux.

. INISER: Dans le cadre de la première simulation, recherche des travaux prêts à occuper le processeur et initialisation des serveurs

. NEXSER: Recherche de la demande de mémoire suivante pour le serveur indiqué

. PRINTR: Impression des résultats

. REACT : Réactivation du travail, attribution d'un numéro d'activation; réinsertion parmi les travaux prêts

. RECCAD: Attribution d'un cadre en réponse à un défaut de page; gestion des tables

. SAVE : Sauvetage des résultats de la simulation sur un fichier

. SERINT: Gestion des demandes internes à la mémoire (transfert de cadres libres à recopier; mise à zéro de cadres de la mémoire auxiliaire)

. SERPRO: Simulateur du processeur central; gère l'écoulement du programme; génère les entrée-sortie et les références à la mémoire; s'arrête en cas de défaut de page, d'entrée-sortie, de micro-time-slice ou de time-slice, et de fin de travail.

# CONCLUSIONS

Depuis quelques années, de nombreux systèmes d'exploitation d'ordinateurs disposent d'une mémoire virtuelle, qui permet d'augmenter le nombre de programmes actifs à un moment donné. Pour fonctionner avec efficacité, un tel système doit avoir une gestion des mémoires centrale et auxiliaire et des transferts qui ne crée pas une charge trop grande pour le processeur central et qui ne ralentisse pas trop les travaux.

Nous avons décrit au chapitre I, les fonctions nécessaires à un tel système de gestion, et, dans le chapitre II, montré comment différents systèmes utilisaient ces fonctions : Siemens BS2000, IBM VS1 et MVS. On a pu constater que les solutions étaient assez différentes; bien que chaque fonction existe dans l'un ou l'autre système, leurs contenus et leurs interactions varient fortement.

Dès lors, pour analyser au mieux l'efficacité d'un système au point de vue de sa gestion de la mémoire, la création d'un simulateur s'imposait. Ce simulateur devrait partir d'une suite de références de page faite par chacun des travaux.

Il devrait simuler le chemin complet des travaux, en insistant évidemment sur la partie liée à la mémoire.

Il devrait en outre être fortement modularisé pour réduire l'impact d'un changement dans l'algorithme de gestion et faciliter ainsi l'évaluation de l'effet d'un tel changement.

Cette évaluation demande nécessairement la disposition de données permettant d'analyser le changement, et de juger de son intérêt.

Le chapitre III explique les solutions qui ont été retenues et donne la structure globale du simulateur; le texte intégral se trouve en annexe.

Faute de données précises sur le système Siemens BS2000, nous n'avons pas pu valider le modèle qui en a été fait, et donc nous ne pouvons pas juger avec précision de la valeur des résultats obtenus.

Le problème fondamental qui s'est posé tout au long de ce mémoire est celui de l'obtention de données, sûres, précises et détaillées, sur la structure des pro-

grammes (structure de localité) sur le fonctionnement réel du système (entrée-sortie, taille de la mémoire paginable, rapport entre temps passé pour des programmes ou des routines résidentes ou non).

Cela a amené une imprécision pour certaines décisions et un modèle descriptif trop rigide pour d'autres; la structure de localité adoptée en est un exemple.

Tout travail supplémentaire nécessiterait donc des études approfondies en ces domaines.

°
° °

# REFERENCES

Pour les comportements de programme

. Characteristic of program localities,
A.W. Madison, A.P. Batson
University of Virginia, Communications of the A.C.M., May 1976

. Comportement des programmes dans leur espace d'adresse.
Application à la gestion des mémoires hiérarchisées
J. Lenfant
Thèse présentée devant l'Université de Rennes, 1974

. Observations on programs for feeding an operating system simulator
M. Noirhomme-Fraiture
Computer Performance Evaluation, Online Conferences Limited,
Uxbridge, England, 1976

Pour le Siemens BS2000

VM029 : narrative description of central control system

Pour IBM

Introduction to virtual storage in systems/370 GR20-4260
OS/VS1 - Planning and use guide
OS/VS1 - Supervisor Logic

```
      PROGRAM SIMULV
C
C
C
C
C     VALEUR DES COMMON
C     -----------------
C
C
C
C
C
C
C     TEMPS
C           TIME EPOQUE OU EST ARRICE LA SIMULATION
C           TIMRET(TSNN) RETIENS HEURE DE DEBUT POUR LES STATISTIQUES
C
C
C     FILEDP  FILE D'ATTENTE DU SERVEUR MEMOIRE
C           FILED1(2)  FILE D'ATTENTE DES TRAVAUX POUR LE SERVEUR 1(2)
C     RANGES EN ORDRE CROISSANT SUIVANT PARAMETRE CONTENU DANS COLONNE DEYX
C     NBFIE1(2) NOMBRE DE TRAVAUX EN ATTENTE DANS LA FILE FILED1(2)
C
C
C     SERV  DONNEES LIEES AUX SERVEURS MEMOIRE
C     USER  COLONNE 1 OCCUPANT DU SERVEUR
C           COLONNE 2 TEMPS DE FIN DE SERVICE
C     CEDP(SERV) INDIQUE SI LE SERVEUR EST OCCUPE(1) OU NON(0)
C     FLAG(SERV) =1 IL EXISTE UNE DEMANDE POUR CE SERVEUR ET LE SERVEUR EST LIBI
C     PPNW(SERV) INDIQUE LE CADRE ENGAGE DANS LE TRANSFERT EN COURS(PP OU PW)
C
C
C     USERIO  DONNEES LIEES AUX SERVEURS I/O
C     USERIO  INDIQUE QUI EST EN COIRS DE SERCICE(COLONNE1) ET
C     QUAND FINIT LE SERVICE"(COLONNE 2)
C     FILEIO(I,J) INDIQUE POUR CHAQUE SERVEUR(J),LES TRAVAUX EN ATTENTE(I) ORDR)
C     D'INTRODUCTION FIFO
C     OCCIO(SERV) =1 SI SERV OCCUPE 0 SINON
C     NBFIIO(SERV) NOMBRE DE TRAVAUX EN ATTENTE PAR SERVEUR I/O
C
C
C     PROC DONNEES LIEEES AU PROCESSEUR
C     FILEPR(20,2) COLONNE1 NUMERO DU TRAVAIL EN ATTENTE DU PROCESSEUR
C           COLONNE 2  ORDRE DE PRIORITE(ORDRE CROISANT)
C
C
C     ACTC
C     NACT DERNIER NUMERO D'ACTIVATION ATTRIBUE(PREMIER=999) ORDRE DECROISSANT
C     NJOBTA  NOMBRE DE TRAVAUX DANS LE SYSTEME
C
C
C     TSN  DONNEES LIEES AUX TRAVAUX
C     COLONNE          1  FIN DE LA CHAINE DES CADRES
C     COLONNE          2  DEBUT DE LA CHAINE DES CADRES
C     COLONNE          3 NOMBRE DE CADRES ALLOUES
C     COLONNE          4 CADRE A UTILISER EN CAS DE REMPLACEMENT
C     COLONNE          5 NOMBRE MAXIMUM DE CARRES ALLOUABLES
C     COLONNE          6 TEMPS DE VIE RESTANT AU TRAVAIL(EN TIME-SLICE)
C     COLONNE          7 TEMPS RESTANT DANS LE TIME-SLICE(EN MICRO-TIME-SLICE)
C     COLONNE          8 NUMERO DACTIVATION ATTRIBUE AU TRAVAIL
C     COLONNE          9  NOMBRE DE PAGES DU TRAVAIL
```

```
C      COLONNE          10  NUMERO DE LA DISTRIBUTION DES REFERENCES AUX PAGES
C      COLONNE          11  CENTRE DE LA LOCALITE
C      COLONNE          12 TAILLE DE LA LOCALITE
C      COLONNE          13 NUMERO DE PRIORITE DE LA DEMANDE  MEOIRE
C      COLONNE          14  TYPE DE LA DEMANDE(2,3,)
C      COLONNE          15 NUMERO DE LA PAGE DEMANDEEE
C      COLONNE          16 TAMBOUR OU EST SITUE CETTE PAGE
C      COLONNE          17 TYPE SERVICE ACCOMPLI(1  PW  2 AUTRES)
C
C
C      COMPT   COMPTEUR D'OCCURENCE 40 ENTRRES ENTREES 22-40 NON UTILISEES
C      COMP=  1  /2 NOMBRE I/O SERVEUR 1/2
C      COMP=  3/4 NOMBRE DE DEMANDES PW SUR LES DEUX SERVEURS
C      COMP=  5 NOMBRE DE FAST WRITE
C      COMP=  6/7  FIN DE SERVICE POUR PW
C      COMP=  8/9 NOMBRE DE DEFAUTS DE PAGE DE TYPR RO(PAS DE RECOPIE)
C      COMP=  10/11  NOMBRE DD DEVAUTS DE PAGE RW(CADRE A RECOPIET SUR LE TAMBOUI
C      COMP=  12/13 POSTPAGE RECOPIE DES CADRES LIBRES SUR LE TAMBOUR
C      COMP=  14/15 MISE A ZERO DES CADRES MEMOIRE AUXILIAIRE(SW)
C      COMP=  16 NOMBRE DESACTIVATION
C      COMP=  17  NOMBRE PAGES VOLEES PAR GBCR
C      COMP=  18 NOMBRE DE PAGES DEMANDEES PAR TRAVAUS
C      COMP=  19  NOMBRE PAGES RECUPERREES PAR TRAVAUX(FAUS DEFAUT DE PAGE)
C      COMP=  20  NOMBRE PAGES PERDUES PAR TRAVAUX
C      COMP=  21       NOMBRE FAUX DEFAUT DE PAGE
C
C
C      STAT1 A 9
C      TABLES STATISTIQUES CONTENANT LE NOMBRE D'OCCURENES D'UN EVENEMENT
C      DONNE PAR CLASSE(TABST) LE NOMBRE DE CLASSE(NBCLA) LIMITE A 20
C      RESTA CONTIENT LES DOONNEES POUR CALCULER LA MOYENNE
C      VALEURS DES YABLES
C      TABLE  1TEMPS ENTRE DEFAUT DE PAGE TAMBOIR 1
C      TABLE  2 TEMPS ENTRE DEFAUTS DE PAGE TAMBOUR 2
C      TABLE  3  TEMPS ENTRE DEFAUT DE PAGE  TOUS EERVEURS
C      TABLE  4 TEMPS D'ATTENTE POUR OBTENIR SERVEUR MEMOIRE 1
C      TABLE  4 TEMPS D'ATTENTE POUR OBTENIR SERVEUR MEMOIRE 1
C      TABLE  5  TEMPS D'ATTENTE POUR OBTENIR SERVEUR MEMOIRE 2
C      TABLE  6  TENPS D'ATTENTE POUR OBTENIR SERVEUR MEMOIRE 1 OU 2
C      TABLE  7  TEMPS ATTENTE ET SERVICE POUR ENTREE-SORTIE
C      TABLE  8 TEMPS D'UTILISATION DU PROCESSEUR
C      TABLE  9       TEMPS SERVICE ET ATTENTE SERVEUR MEMOIRE
C
C
C      SLOT DONNEES CONCERNANT LA MEMOIRE AUXILIAIRE
C             NSLT  NOMBRE DE CADRES DISPONIMLES SUR LE SERVEUR
C             NSLV(SERV) NOMBRE DE CADRE A MZTTRE A ZERO SUR LE SERVEUR
C
C
C      L DONNNEES LIEES AU TEMPS ET AUX TRAVAUX
C      TIMLOC  RESTE DE DUREE DANS LA LOCALITE
C      TIMOCC TEMPS D'OCCUPATION DU PROCESSEUR
C      TIMMTS RESTE DU TEMPS DANS LE MICRO-TIME-SLICE
C      TIMIO RESTE DU TEMPS AVANT PROCHAINE I/O
C      TIMDP  TEMPS DEPUIS DERNIER EEFAUT DE PAGE
C
C
C      CADRE  DONNEES LIEES AUX CADRES
C      PPAT(PPN,J)  DONNEES POUR CAAQUE CADRE(PPN)
C             COLONNE 1 PAGE QUI OCCUPE LE CADRE
```

```
C          COLONNE 2 TAMBOUR OU EST SITUEE LA PAGE
C          COLONNE 3 CHAINE A LAQUELLE LE CADRE APPARTIENT (1,2,3,5 OU9)
C          COLONNE 4 CADRE UTILISE DEPUIS DERNIER PASSAGE GBCR
C          COLONNE 5  CADRE AETE ECRIT
C          COLONNE 6 LIEN VERS CADRE SUIVANT DANS LA CHAINE
C          COLONNE 7  LIEN VERS LE CADRE PRECEDENT DE LA CHAINE
C          COLONNE 8 NUMERO DU TSN A LAQUELLE LE CADRE APPARTIENT
C
C
C     A  DONNEES LIEES AUX PAGES  EQUIVAUT A LA TABLE DES PABES
C          VPNTMC(TSNN,VPN)  INDIQUE POUR CHAQUE PAGE DE CHAQUE
C          TRAVAIL LE CADRE OU LA PAGE EST PEESENTE
C          VPNTDR  NUMERO DE TAMBOUR DE LA PAGE
C
C
C     G  TABLE DE DISTRIBUTION
C     COLONNE1
C     NUMERO 1TENPS DANS LOCALITE
C     NUMERO 2 TAILLE DE LOCALITE
C     NUMERO ,4,5  TABLE DE REFERENCES AUX PAGES DANS UNE LOCALITE
C     NUMERO 6  TABLE DE DISTRIBUTION DES ENTREE-SORTIE
C
C
C     H  NOMBRE D'ENTREE UTILISEES DANS LES TAVLES DE DISTRIBUTIONS
C
C
C     J  VALEURS DES NOMBRES ALEATOIRES
C
C
C
C*************************************************************************
C
C
C     GERANT DE LA SIMULATION DETECTE LES FINS DE SERVICES
C     SI AUCUNE LANCE LE PROCESSEUR
C     SI PLUS RIEN A FAIRE
C     CLOTURE LES STATISTIQUES  IMPRIME LES RESILTATS
C
C
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /USERIO/USERIO(2,2),FILEIO(20,2),OCCIO(2),NBFIIO(2)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      COMMON /ACT/NACT,NJOBTA
      INTEGER *2 FILED1,FILED2,NBFIE1,NBFIE2,EDP,FLAG,PPNW
      INTEGER *2 FILEIO,NBFIIO,OCCIO,FILEPR,NBFIPR
      INTEGER TIME,TIMRET,USER,USERIO,TMAXSI,FIREVE
      INTEGER *2 NSIM,NJOBTA,NACT,TSNN,RESULT,SERV,CANAL
      READ(1,9200)NSIM,TMAXSI
C
C     INITIALISATION DE LA SIMULATION
C     SI NSIM=1 PART D'UN FICHIER EXTERIEUR A CREER
C           SI=2 PART DU RESULTAT DE LA SIMULATION PRECEDENTE
C
      IF(NSIM .NE.1) GOTO 20
      CALL DATA1(NJOBTA)
      CALL INISER(NJOBTA)
      CALL INIALE(NJOBTA)
      NBFIPR=NJOBTA
```

```
      GOTO 90
20    CALL DATA2(NJOBTA)
90    TMAXSI=TMAXSI+TIME
100   IF(TIME .GE. TMAXSI) GOTO 8000
C
C     DEMANDE POUR LA MEMOIRE EXISTANTE AVEC SERVEUR LIBRE
C
      DO 110 SERV=1,2
      IF(FLAG(SERV) .EQ. 0) GOTO 110
      FLAG(SERV)=0
      CALL GERDEM(SERV,RESULT)
      GOTO(110,8000),RESULT
110   CONTINUE
C
C     FIN DE SERVICE ENTREE-SORTIE
C
      DO 210 CANAL=1,2
      IF(USERIO(CANAL,2) .LE.TIME) CALL FINIO(CANAL)
210   CONTINUE
C
C     FIN DE SERVICE MEMOIRE
C
300   DO 350 SERV=1,2
      IF(USER(SERV,2) .GT. TIME) GOTO 350
      CALL FINSER(SERV)
      CALL GERDEM(SERV,RESULT)
      GOTO(350,8000),RESULT
350   CONTINUE
C
C     RECHERCHE DU PROCHAIN EVENEMENT
C
      FIREVE=TMAXSI
      DO 400 CANAL =1,2
      IF(USERIO(CANAL,2) .GT. FIREVE) GOTO 400
      FIREVE=USERIO(CANAL,2)
400   CONTINUE
      DO 500 SERV=1,2
      IF(USER(SERV,2) .GT. FIREVE ) GOTO 500
      FIREVE=USER(SERV,2)
500   CONTINUE
      IF(NBFIPR .NE. 0) GOTO 1100
C
C     PLUS DE TRAVAUX EN ATTENTE TIME VA JUSQU'AU PREMIER EVENEMENT
C
      TIME=FIREVE
      GOTO 100
C
C     LANCEMENT DU PROCESSEUR
C
1100  CALL PROCES(SERV,TSNN,FIREVE)
      GOTO 100
C
C     CLOTURE C
C
8000  CALL SAVE(TIME)
      CALL PRINTR(TIME)
      CALL ENDSTA(TIME)
      STOP
9200  FORMAT(I4,I10)
      END
```

```
      SUBROUTINE INISER(NJOBTA)
C
C
C     INITIALISATION DES SERVEURS
C     REMPLISSAGE DE LA FILE D'ATTENTE DU PROCESSEUR
C
C
      COMMON /B/TSN(20,17)
      COMMON /USERIO/USERIO(2,2),FILEIO(20,2),OCCIO(2),NBFIIO(2)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /ACT/NACT,NJOBTA
      INTEGER *2 TSNN1,NACT,NJOBTA,RET,TRET,MIN,TSNN,SERV
      INTEGER *2 TSN,FILEPR,NBFIPR,EDP,FLAG,PPNW,FILEIO,OCCIO,NBFIIO
      INTEGER USER,USERIO
      RET=1000
      MIN=0
      TRET=0
      DO 100 TSNN1=1,NJOBTA
      DO 50 TSNN=1,20
      IF(TSN(TSNN,8) .LE. MIN) GOTO 50
      IF(TSN(TSNN,8) .GE. RET) GOTO 50
      RET=TSN(TSNN,8)
      TRET=TSNN
50    CONTINUE
      MIN=RET
      FILEPR(TSNN1,1)=TRET
      FILEPR(TSNN1,2)=TSN(TRET,8)
      TRET=0
      RET=1000
100   CONTINUE
C
C     SERVEURS MEMOIRE ET ENTREE-SORTIE LIBRES
C
      DO 200 SERV=1,2
      USER(SERV,2)=999999999
200   CONTINUE
      DO 300 SERV=1,2
      USERIO(SERV,2)=999999999
300   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DEBIO(TSNN,CANAL)
C
C
C     DEBUT D'ENTREE-SORTIE SI LE SERVEUR EST LIBRE
C
C
      COMMON /USERIO/USERIO(2,2),FILEIO(20,2),OCCIO(2),NBFIIO(2)
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /COMPT/COMP(40)
      INTEGER *2 FILEIO,OCCIO,NBFIIO
      INTEGER USERIO,TIME,TIMRET,COMP
      INTEGER *2 TSNN,CANAL
      COMP(CANAL)=COMP(CANAL)+1
      TIMRET(TSNN)=TIME
      IF(OCCIO(CANAL) .EQ. 0) GOTO 100
      NBFIIO(CANAL)=NBFIIO(CANAL)+1
      FILEIO(NBFIIO(CANAL),CANAL)=TSNN
      GOTO 1000
100   OCCIO(CANAL)=1
      USERIO(CANAL,1)=TSNN
      USERIO(CANAL,2)=TIME+32000
1000  RETURN
      END
```

```
      SUBROUTINE FINIO(SERV)
C
C
C     FIN DE SERVICE POUR LES ENTREE-SORIEC
C
C
      COMMON /STAT7/TABST7(20,2),NBCLA7,RESTA7(3)
      COMMON /B/TSN(20,17)
      COMMON /USERIO/USERIO(2,2),FILEIO(20,2),OCCIO(2),NBFIIO(2)
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      COMMON /COMPT/COMP(40)
      INTEGER *2 TSN,FILEIO,OCCIO,TSNN,SERV,FILEPR,NBFIPR,ATT,NBFIIO
      INTEGER TABST7,NBCLA7
      REAL RESTA7
      INTEGER TIME,TIMRET,COMP,USERIO,DUREE
      TSNN=USERIO(SERV,1)
      DUREE=TIME-TIMRET(TSNN)
      CALL STAT(TABST7,NBCLA7,DUREE,RESTA7)
      CALL INSERT(FILEPR,NBFIPR,TSN(TSNN,8),TSNN)
      IF(FILEIO(1,SERV) .NE. 0) GOTO 100
C
C     PLUS DE NOUVELLES DEMANDES SERVEUR LIBRE
C
      USERIO(SERV,1)=0
      USERIO(SERV,2)=999999999
      OCCIO(SERV)=0
      GOTO 1000
C
C     NOUVELLE DEMANDE PRISE EN CHARGE
C     SERVEUR OCCUPE FILE D'ATTENTE MISE A JOUR
C
100   USERIO(SERV,1)=FILEIO(1,SERV)
      USERIO(SERV,1)=TIME+32000
      OCCIO(SERV)=1
      DO 200 ATT=1,19
      FILEIO(ATT,SERV)=FILEIO(ATT+1,SERV)
200   CONTINUE
      FILEIO(20,SERV)=0
      NBFIIO(SERV)=NBFIIO(SERV)-1
1000  RETURN
      END
```

```
      SUBROUTINE INSERT(TAB,NB,PAR,ELEM)
C
C
C     INSERTION D'UN ELEMENT DANS UNE FILE D'ATTENTE(TAB) SUIVANT
C     UN ORDRE CROISSANT(PAR)
C
C
      INTEGER *2 TAB(20,2),PAR,ELEM,NB,LIMINF,K,POS
      POS=0
      IF(NB .EQ. 0) GOTO 105
      DO 100 POS=1,NB
      IF(PAR .LT. TAB(POS,2)) GOTO 110
100   CONTINUE
105   POS=NB+1
      GOTO 300
110   LIMINF=NB-POS+1
      DO 200 K=1,LIMINF
      TAB(NB+2-K,1)=TAB(NB+1-K,1)
      TAB(NB+2-K,2)=TAB(NB+1-K,2)
200   CONTINUE
300   TAB(POS,1)=ELEM
      TAB(POS,2)=PAR
      NB=NB+1
      RETURN
      END
```

```
      SUBROUTINE ERASE(TAB,NB,ELEM)
C
C
C     DESTRUCTION DE L'ELEMENT ELEM DANS LA FILE D'ATTENTE TAB
C
C
      INTEGER *2 TAB(20,2),NB,ELEM,POS
      DO 100 POS=1,20
      IF(TAB(POS,1) .NE. ELEM) GOTO 100
      DO 50 I=POS,19
      TAB(I,2)=TAB(I+1,2)
      TAB(I,1)=TAB(I+1,1)
50    CONTINUE
      TAB(20,1)=0
      TAB(20,2)=0
      NB=NB-1
      GOTO 110
100   CONTINUE
110   RETURN
      END
```

```
      SUBROUTINE NEXSER(SERV,TSNN)
C
C
C     RECHERCHE DE LA PROCHAINE DEMANDE DES TRAVAUX POUR LA MEMOIRE
C
C
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      INTEGER *2 FILED1,FILED2,NBFIE1,NBFIE2,SERV,TSNN
      IF(SERV .EQ. 2) GOTO 2000
      TSNN=FILED1(1,1)
      IF(TSNN .EQ. 0) GOTO 4000
      CALL ERASE(FILED1,NBFIE1,TSNN)
      GOTO 4000
2000  TSNN=FILED2(1,1)
      IF(TSNN .EQ. 0) GOTO 4000
      CALL ERASE(FILED2,NBFIE2,TSNN)
4000  RETURN
      END
```

```
      SUBROUTINE FINSER(SERV)
C
C
C      TRAITEMENT A EFFECTUER LORS DE LA FIN D'UN SERVICE POUR
C     LA MEMOIRE
C
C
      COMMON /STAT9/TABST9(20,2),NBCLA9,RESTA9(3)
      COMMON /B/TSN(20,17)
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /SLOT/NSLT(2),NSLV(2)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      INTEGER TABST9,NBCLA9
      REAL RESTA9
      INTEGER *2 FILEPR,NBFIPR,TSN,NSLT,NSLV,EDP,FLAG,PPNW
      INTEGER *2 TSNN,SERV,QDEP,QARR,CHOIX
      INTEGER USER,TIME,TIMRET,DUREE
      TSNN=USER(SERV,1)
      IF(TSNN .EQ. 1000) GOTO 300
      IF(TSNN .EQ. 1500) GOTO 400
      CHOIX=TSN(TSNN,17)
      GOTO(100,200),CHOIX
C
C     FIN DE TRAITEMENT PRIORITY WRITE
C
100   CALL FINPW(TSNN,SERV)
      GOTO 5000
C
C     FIN DE TRAITEMENT DE DEFAUT DE PAGE LE TRAVAIL RETOURNE
C     DANS LA FILE D'ATTENTE DU PROCESSEUR
C
200   CALL INSERT(FILEPR,NBFIPR,TSN(TSNN,8),TSNN)
      DUREE=TIME-TIMRET(TSNN)
      CALL STAT(TABST9,NBCLA9,DUREE,RESTA9)
      GOTO 5000
C
C     FIN DE POSTPAGE LE CADRE RECOPIE ENTRE DANS LA FILE 1
C
300   QDEP=5
      QARR=1
      CALL MQEQ(QDEP,QARR,PPNW(SERV))
      GOTO 5000
C
C     FIN DE SHORT WRITE REMISE A JOUR DES INDICATEURS DE CADRES DE
C     MEMOIRE AUXILIAIRE OCCUPES ET LIBRES
C
400   NSLT(SERV)=NSLT(SERV)+1
      NSLV(SERV)=NSLV(SERV)-1
5000  RETURN
      END
```

```
      SUBROUTINE FINPW(TSNN,SERV)
C
C
C     FIN DU TRAITEMENT D'UNE DEMANDE PRIORITY WRITE
C     LE CADRE CONCERNE EST REMIS EN ROQ
C     LA DEMANDE EN ATTENTE EST MISE EN FILE D'ATTENTE NORMALE
C
C
      COMMON /B/TSN(20,17)
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      INTEGER *2 TSN,FILED1,FILED2,NBFIE1,NBFIE2,EDP,FLAG,PPNW
      INTEGER *2 QDEP,QARR,TSNN,SERV,PPN,PRIOR
      INTEGER USER
C
C     RETOUR DU CADRE DANS LA FILE 1
C
      PPN=PPNW(SERV)
      QDEP=5
      QARR=1
      CALL MQEQ(QDEP,QARR,PPN)
C
C     REMISE EN ETAT DE LA DEMANDE EN ATTENTE
C
      TSN(TSNN,14)=3
      PRIOR=3000+TSN(TSNN,8)
      TSN(TSNN,13)=PRIOR
      GOTO(100,200),SERV
200   CALL INSERT(FILED1,NBFIE1,PRIOR,TSNN)
      GOTO 300
100   CALL INSERT(FILED2,NBFIE2,PRIOR,TSNN)
300   IF(EDP(3-SERV) .EQ. 0) FLAG(3-SERV)=1
      RETURN
      END
```

```
      SUBROUTINE DEMPRW(TSNN,PPN,SERV)
C
C
C     PRISE EN CHARGE D'UNE DEMANDE PRIORITY WRITE
C     LE CADRE CONCERNE EST MIS EN FILE 5
C     LA DEMANDE  EST MISE EM ATTENTE
C     UNE DEMANDE PRIORITY WRITE EST CREEE ET MISE EN FILE
C
C
      COMMON /B/TSN(20,17)
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      INTEGER *2 FILED1,FILED2,NBFIE1,NBFIE2,TSN,EDP,FLAG,PPNW
      INTEGER USER
      INTEGER *2 TSNN,PPN,SERV,QDEP,QARR,PRIOR
C
C     CADRE VA EN FILE 5
C
      QARR=5
      QDEP=4-SERV
      CALL MQEQ(QDEP,QARR,PPN)
C
C     DEMANDE P.W. CREEE
C
      PRIOR=2000+TSN(TSNN,8)
      TSN(TSNN,13)=PRIOR
      TSN(TSNN,14)=2
      PPNW(SERV)=PPN
      GOTO(1000,2000),SERV
1000  CALL INSERT(FILED2,NBFIE2,PRIOR,TSNN)
      GOTO 3000
2000  CALL INSERT(FILED1,NBFIE1,PRIOR,TSNN)
3000  IF(EDP(3-SERV) .EQ. 0) FLAG(3-SERV)=1
      RETURN
      END
```

```
      SUBROUTINE SERINT(SERV,PARAM)
C
C
C     GESTION DES DEMANDES INTERNES A LA MEMOIRE:RECOPIE AUTOMATIQUE
C     DE CADRES SI LE SERVEUR EST LIBRE(POSTPAGE) OU MISE A ZERO
C     DE CADREX DE LA MEMOIRE AUXILIAIRE(SHORT WRITE)
C
C
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /SLOT/NSLT(2),NSLV(2)
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /COMPT/COMP(40)
      INTEGER *2 EDP,FLAG,PPNW,PPAT,NQ,PTDQ,PTFQ,NSLT,NSLV
      INTEGER TIME,TIMRET,USER,COMP,TIMINT
      INTEGER *2 SERV,QDEP,QARR,PARAM
C
C     RECHERCHE D'UN CADRE A RECOPIER DANS LA FILE :HATTENTE DU SERVEUR
C
      IF(NQ(SERV+1) .EQ. 0) GOTO 1000
      QDEP=1+SERV
      QARR=5
      COMP(11+SERV)=COMP(11+SERV)+1
      CALL HQEQ(QDEP,QARR,PPNW(SERV))
      USER(SERV,1)=1000
      GOTO 2000
C
C     RECHERCHE D'UN CADRE A VIDER SUR LA MEMOIRE AUXILIAIRE
C
1000  IF(NSLV(SERV) .EQ. 0) GOTO 3000
      COMP(13+SERV)=COMP(13+SERV)+1
      USER(SERV,1)=1500
2000  USER(SERV,2)=TIME+20000
      PARAM=1
      GOTO 4000
C
C     PARAM=1  SI UNE DEMANDE A ETE COMMENCEE
C
3000  PARAM=2
4000  RETURN
      END
```

```
      SUBROUTINE FASWRI(TSNN)
C
C
C     TRAITEMENT D'UNE DEMANDE FAST WRITE  CAS NON PREVU DANS
C     LE SIMULATEUR ERREUR GAAVE
C
C
      COMMON /COMPT/COMP(40)
      INTEGER *2 TSNN
      INTEGER COMP
      COMP(5)=COMP(5)+1
      WRITE(20,9000) TSNN
      RETURN
9000  FORMAT(1X,///,'ERREUR POUR JOB NB ',I4,'FAST WRITE DEMANDE',9(/))
      END
```

```
      SUBROUTINE REACT(TSNN)
C
C
C     REACTIVATION D'UN TRAVAUL ATTRIBUTION D'UN NUMERO
C     D'ACTIVATION ET MISE EN FILE D'ATENTE DU PROCESSEUR
C
C
      COMMON /ACT/NACT,NJOBTA
      COMMON /B/TSN(20,17)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      INTEGER *2 NACT,NJOBTA,FILEPR,NBFIPR,TSN,TSNN
      NACT=NACT-1
      TSN(TSNN,8)=NACT
      CALL INSERT(FILEPR,NBFIPR,NACT,TSNN)
      RETURN
      END
```

```
      SUBROUTINE SEREDP(SERV,TSNN,PARAM)
C
C
C     GESTION DE LA MEMOIRE  ALLOCATION DE CADRE ET REPONSE
C           AUX DEMANDES DE MEMOIRE
C
C
      COMMON /STAT4/TABST4(20,2),NBCLA4,RESTA4(3)
      COMMON /STAT5/TABST5(20,2),NBCLA5,RESTA5(3)
      COMMON /STAT6/TABST6(20,2),NBCLA6,RESTA6(3)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /B/TSN(20,17)
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /COMPT/COMP(40)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      INTEGER TABST4,TABST5,TABST6,NBCLA4,NBCLA5,NBCLA6
      REAL RESTA4,RESTA5,RESTA6
      INTEGER *2 EDP,FLAG,PPNW,FILED1,FILED2,NBFIE1,NBFIE2,TSN,FILEPR
      INTEGER TIME,TIMRET,USER,COMP,DELAI
      INTEGER *2 SERV,CHOIX,TSNN,PARAM,VPN,DRN,REPONS,PPN,NBFIPR
      EDP(SERV)=1
3110  CHOIX=TSN(TSNN,14)
      GOTO(3200,3300,3400),CHOIX
C
C     FAST WRITE  DEMANDE POUR FIXER UN CADRE EN MEMOIRE
C     CENTRALE NON REPRIS DANS CE SIMULATEUR ERREUR GRAVE
C
3200  CALL FASWRI(TSNN)
      PARAM=2
      GOTO 4000
C
C     TRAITEMENT D'UNE DEMANDE FRIORITY WRITE
C
3300  TSN(TSNN,17)=1
      COMP(5+SERV)=COMP(5+SERV)+1
      DELAI=20000
      GOTO 3950
C
C     TRAITEMENT DUN DEFAUT DE PAGE
C     RECHERCHE D'UN CADRE(RECCAD)
C     RESULTAT DE LA RECHERCHE DANS REPONS
C     REPONS=        1  CADRE ATTRIBUE ET LIBRE
C                    2,3  CADRE ATTRIBUE A RECOPIER
C                    5 CADRE A VIDER PAR PRIORITY WRITE
C                    8 PAS DE CADRE ATTRIBUE DEMANDE REJETEE
C                    4,6,7, ERREUR DANS LA GESTION STOPPER SIMULATIONC
C
3400  VPN=TSN(TSNN,15)
      DRN=TSN(TSNN,16)
      IF(DRN .NE. SERV) GOTO 3740
      CALL RECCAD(PPN,TSNN,REPONS,DRN,VPN,NBFIPR)
      GOTO(3500,3600,3600,3700,3800,3700,3700,3900),REPONS
3500  DELAI=20000
      COMP(7+SERV)=COMP(7+SERV)+1
      GOTO 3650
3600  DELAI=40000
      COMP(9+SERV)=COMP(9+SERV)+1
3650  TSN(TSNN,17)=2
      GOTO 3950
```

```
3800   CALL DEMPRW(TSNN,PPN,SERV)
       COMP(2+SERV)=COMP(2+SERV)+1
       PARAM=3
       GOTO 4000
3900   PARAM =4
       GOTO 4000
3740   WRITE(20,9500)TSNN,DRN,SERV
       PARAM=2
       GOTO 4000
3700   WRITE(20,9600) TSNN,REPONS
       PARAM=2
       GOTO 4000
3950   USER(SERV,1)=TSNN
       USER(SERV,2)=TIME+DELAI
       DUREE=TIME-TIMRET(TSNN)
       CALL STAT(TABST6,NBCLA6,DUREE,RESTA6)
       GOTO (3960,3970),SERV
3960   CALL STAT(TABST4,NBCLA4,DUREE,RESTA4)
       GOTO 3980
3970   CALL STAT(TABST5,NBCLA5,DUREE,RESTA5)
3980   PARAM=1
4000   RETURN
9500   FORMAT(1X,///,'ERREUR JOB',I4,'DEMANDE DRUM',I4,'TRAITE PAR',I4)
9600   FORMAT(1X,///,'ERREUR DRUM',I4,' REPONSE=',I4,'TYPE INCONNUE')
       END
```

```
      SUBROUTINE PROCES(SERV,TSNN,FIREVE)
C
C
C     GESTION DU PROCESSEUR CENTRAL:ACTIVATION DE CELUI-CI ET
C     TRAITEMENT DU RESULTAT OBTENU
C
C
      COMMON /STAT1/TABST1(20,2),NBCLA1,RESTA1(3)
      COMMON /STAT2/TABST2(20,2),NBCLA2,RESTA2(3)
      COMMON /STAT3/TABST3(20,2),NBCLA3,RESTA3(3)
      COMMON /STAT8/TABST8(20,2),NBCLA8,RESTA8(3)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /B/TSN(20,17)
      COMMON /L/TIMLOC(20),TIMOCC(20),TIMMTS(20),TIMIO(20),TIMDP(20)
      INTEGER TABST1,TABST2,TABST3,TABST8,NBCLA1,NBCLA2,NBCLA3,NBCLA8
      REAL RESTA1,RESTA2,RESTA3,RESTA8
      INTEGER TIMIO,TIMDP,TIMLOC,TIMOCC,TIMMTS,DT,TIME,TIMEX,DUREE,USER
      INTEGER TIMRET,FIREVE
      INTEGER *2 FILEPR,NBFIPR,FILED1,FILED2,NBFIE1,NBFIE2
      INTEGER *2 EDP,FLAG,PPNW,TSN,PRIOR
      INTEGER *2 TSNN,INDIC,VPN,SERV,DISQUE,SW4
      TSNN=FILEPR(1,1)
C
C     RESULTAT DU TRAVAIL AU PROCESSEUR DANS INDIC
C           INDIC=  1  FIN DE TRAVAIL
C                   2  FIN DE MICRO-TIME SLICE
C                   3  DEFAUT DE PAGE
C                   4 FINDE TIME-SLICE ET DESACTIVATION
C                   5 DEMANDE D'ENTREE-SORTIE
C                   6  INTERRUPTION DU PROCESSEUR SUITE A UNE FIN DE
C                     SERVICE D'UN DES SERVEURS(MEMOIRE OU I/O)
C
      CALL SERPRO(TSNN,INDIC,DISQUE,VPN,SERV,TIMEX,FIREVE)
      CALL STAT(TABST8,NBCLA8,TIMEX,RESTA8)
      TIMDP(TSNN)=TIMDP(TSNN)+TIMEX
      TIMOCC(TSNN)=TIMOCC(TSNN)+TIMEX
      IF(INDIC .EQ. 6) GOTO 1000
      IF(INDIC .EQ. 2) GOTO 50
      CALL ERASE(FILEPR,NBFIPR,TSNN)
50    IF(INDIC .EQ. 5) GOTO 400
C
C     PASSAGE DE LA ROUTINE DE MISE A JOUR DES CADRES
C
      DT=TIMOCC(TSNN)
      SW4=0
      CALL GBCR(TSNN,SW4,DT,SERV)
      TIMOCC(TSNN)=0
      GOTO(100,1000,300,400),INDIC
C
C     TRAVAIL ACHEVE
C
100   CALL FINJOB(TSNN)
      GOTO 1000
C
C     DEMANDE DE PAGE EMISE PAR LE TRAVAIL
C     ETABLISSEMENT DE LA DEMANDE ET MISE EM FILE D'ATTENTE
C
```

```
200   PRIOR=3000+TSN(TSNN,8)
      TSN(TSNN,13)=PRIOR
      TSN(TSNN,14)=3
      TSN(TSNN,15)=VPN
      TSN(TSNN,16)=SERV
      TIMRET(TSNN)=TIME
      DUREE=TIMDP(TSNN)
      CALL STAT(TABST3,NBCLA3,DUREE,RESTA3)
      IF(EDP(SERV) .EQ. 0) FLAG(SERV)=1
      GOTO(210,220),SERV
210   CALL INSERT(FILED1,NBFIE1,PRIOR,TSNN)
      CALL STAT(TABST1,NBCLA1,DUREE,RESTA1)
      GOTO 250
220   CALL INSERT(FILED2,NBFIE2,PRIOR,TSNN)
      CALL STAT(TABST2,NBCLA2,DUREE,RESTA2)
250   TIMDP(TSNN)=0
      GOTO 1000
C
C     DESACTIVATION ET REACTIVATION DE TRAVAIL
C
300   CALL DESACT(TSNN)
      CALL REACT(TSNN)
      GOTO 1000
C
C     DEMANDE D'ENTREE-SORTIE
C
400   CALL DEBIO(TSNN,DISQUE)
1000  RETURN
      END
```

```
      SUBROUTINE GERDEM(SERV,RESULT)
C
C
C     GESTION DES DEMANDES
C
C
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      INTEGER *2 EDP,FLAG,PPNW,SERV,RESULT,PARAM,TSNN
      INTEGER USER
C
C     RECHERCHE DE LA DEMANDE SUIVANTE SI PLUS DE DEMANDES,TSNN=0
C
2000  CALL NEXSER(SERV,TSNN)
      IF(TSNN .NE. 0) GOTO 1000
C
C     TRAITEMENT DES DEMANDES INTERNES,POSTPAGE ET SHORT WRITE
C
      CALL SERINT(SERV,PARAM)
      IF(PARAM .EQ. 1) GOTO 7000
C
C     PAS DE DEMANDES INTERNES SERVEUR LIBERE
C
      USER(SERV,1)=0
      USER(SERV,2)=999999999
      EDP(SERV)=0
      GOTO 7000
C
C     APPEL A ROUTINE DE TRAITEMENT D'UNE DEMANDE
C     RESULTATS PARAM=1 TRAITEMENT COMMENCE
C             =2     ERREUR GRAVE
C             =3   DEMANDE EN ATTENTE CHERCHER AUTRE DEMANDE
C             =4   DEMANDE REJETEE DESACTIVER LE TRAVAILC
C
1000  CALL SEREDP(SERV,TSNN,PARAM)
      GOTO(7000,8000,2000,3000),PARAM
C
C     DESACTIVATION ET RECATIVATION DU TRAVAIL
C
3000  CALL DESACT(TSNN)
      CALL REACT(TSNN)
      GOTO 2000
7000  RESULT=1
      GOTO 9000
8000  RESULT=2
9000  RETURN
      END
```

```
      SUBROUTINE RECCAD(PPN,TSNN,REPONS,SERV,VPN,NBFIPR)
C
C
C     RECHERCHE D'UN CADRE  RESULTAT DANS REPONS
C     MISE A JOUR DE L'ETAT DU CADRE
C
C
      INTEGER *2 NQ,TSNN,REPONS,SERV,VPN,PPN,NBFIPR,SW2,SW3,PTDQ,PTFQ
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      INTEGER *2 TSN,PPAT,VPNTMC,VPNTDR,SW4,VPN1
      INTEGER DT,TIMINT
      SW2=1
C
C     SI NOMBRE CADRES ALLOUES < NOMBRE DE CADRES ALLOUABLES
C     ALLER A EXTENSION(SW3=0)  SINON REMPLACER(SW3=1)
C
      IF( TSN(TSNN,5) .GT. TSN(TSNN,3) )GOTO 100
      SW3 = 1
      GOTO 2000
100   SW3 = 0
C
C     EXTENSION  RECHERCHE D'UN CADRE DANS UNE DES FILES DE CADRES
C     LIBRES
C
5000  IF( NQ(1) .EQ. 0 )GOTO 5300
      REPONS = 1
      GOTO 5400
5300  IF( NQ(SERV + 1) .EQ. 0 )GOTO 5600
      REPONS=1+SERV
5400  CALL HQET(REPONS,TSNN,PPN)
      IF(SW3 .EQ. 1) TSN(TSNN,5) = TSN(TSNN,5)+1
      GOTO 3850
5600  IF(NQ(4-SERV) .EQ. 0) GOTO 6000
      REPONS = 5
      PPN=PTDQ(4-SERV)
      GOTO 8990
C
C     ECHEC DE L'EXTENSION SI PREMIER PASSAGE REMPLACER SINON
C     ALGORITHME D'URGENCE
C
6000  IF( SW2 .EQ. 0 )GOTO 6200
      SW2 = 0
      GOTO 2000
C
C     CAS D'URGENCE  SI UN SEUL TRAVAIL EN ETAT D'OCCUPER LE PROCESSEUR
C     ALORS ENLEVER UNE DES PAGES AU HASARD
C     SINON DESACTIVER TRAVAIL ET REFUSER DEMANDE
C
6200  IF( NBFIPR .GT. 1 )GOTO 6500
      DT=0
      SW4=1
      CALL GBCR(TSNN,SW4,DT,SERV)
C
C
C     REMPLACEMENT D'UN CADRE PAR CELUI QUE LA GBCR A DESIGNEC
C     SI PRIORITY WRITE,REPONS=5
C     SINON REPONS=1,2 ET CADRE A SON CONTENU MIS A JOUR
C
```

```
      IF(TSN(TSNN,4) .EQ. 0) GOTO 6500
      GOTO 2000
6500  REPONS = 8
      TSN(TSNN,5)=TSN(TSNN,5)+3
      GOTO 8990
2000  IF(TSN(TSNN,4) .EQ. 0) GOTO 4000
      PPN = TSN(TSNN,4)
      PPAT(PPN,4)=1
      TSN(TSNN,4)=0
      TIMINT(PPN) = 0
      IF( PPAT(PPN,5) .EQ. 0 )GOTO  3800
      PPAT(PPN,5) = 0
      IF( PPAT(PPN,2) .EQ. SERV )GOTO  3700
      REPONS = 5
      GOTO 8990
3700  REPONS = 2
      GOTO 3850
3800  REPONS = 1
C
C     CADRE OCCUPE PAR LA NOUVELLE PAGE
C
3850  VPN1=PPAT(PPN,1)
      IF(VPN1 .NE. 0) VPNTMC(TSNN,VPN1)=0
      VPNTMC(TSNN,VPN)=PPN
      PPAT(PPN,1) = VPN
      PPAT(PPN,2) = SERV
      GOTO 8990
C
C     SI ECHEC DU REMPLACEMENT ALLER A EXTENSION SI PREMIER PASSAGE
C     SINON REFUS DE DEMANDE(REPONS=8)
C
4000  IF(SW2 .EQ. 0) GOTO 6200
      SW2 = 0
      GOTO 5000
8990  RETURN
      END
```

```
      SUBROUTINE GBCR(TSNN,SW4,DT,SERV)
C
C
C     ROUTINE DE MISE A JOUR DE L'ETAT DES CADRES
C     ENLEVE CADRES VIEUX DE PLUS DE 120 MSEC
C     METS A JOUR LE COMPTEUR D'INUTILISATION
C     SI INUTILISE DEPUIS MOINS DE 3000 SECONDES NE FAIT RIEN
C     INDIQUE POUR LE REMPLACEMENT (TSN(TSNN,4)) LE CADRE LE PLUS
C     VIEUX COMPRIS ENTRE 30ET 120 MSEC
C
C
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /COMPT/COMP(40)
      INTEGER *2 TSNN,RET,TRET,PPN,FIN,QARR
      INTEGER *2 VPNTMC,VPNTDR,TSN,SW4,PTDQ,PTFQ,PPAT,NEXPPN
      INTEGER *2 NQ,FLAG,PPNW,EDP
      INTEGER DT,COMP,TIMINT,TMAX,TMIN,USER
      RET=0
      TRET=0
      IF(TSN(TSNN,3) .EQ. 0)GOTO 8000
      TMIN=24000
      TMAX=96000
C
C     SW4=1  ON PRENDS LA DERNIERE PAGE INUTILISESS DE LA CHAINE DES CADRES
C     DU TRAVAIL
C
      IF(SW4 .EQ.1)TMIN=0
      PPN=TSN(TSNN,2)
      FIN=TSN(TSNN,1)
1000  NEXPPN=PPAT(PPN,6)
      IF(PPAT(PPN,4) .EQ. 1) GOTO 4000
      TIMINT(PPN)=TIMINT(PPN)+DT
      IF(TIMINT(PPN) .LT. TMIN) GOTO 6000
      IF(TIMINT(PPN) .GT. TMAX) GOTO 3000
      IF(TIMINT(PPN) .LE. TRET) GOTO 6000
      RET=PPN
      TRET=TIMINT(PPN)
      GOTO 6000
C
C     VOL D'UNE PAGE TROP VIEILLE
C
3000  TSN(TSNN,5) = TSN(TSNN,5) - 1
      COMP(17)=COMP(17)+1
      QARR=1
      IF(PPAT(PPN,5) .EQ. 0) GOTO 3100
      IF(EDP(SERV) .EQ. 0) FLAG(SERV)=1
      QARR=PPAT(PPN,2)+1
3100  CALL MTEQ(TSNN,QARR,PPN)
      GOTO 6000
C
C     PAGE UTILISEE DEPUIS LE DERNIER PASSAGE DE LA ROUTINE
C
4000  TIMINT(PPN)=0
      PPAT(PPN,4)=0
6000  IF(NEXPPN .EQ. 0) GOTO 8000
      PPN=NEXPPN
      GOTO 1000
```

```
C
C       INDICATION DE LA PAGE LA PLUS VIEILLE
C
8000    TSN(TSNN,4) = RET
        RETURN
        END
```

```
      SUBROUTINE PREPTA(TAB,NB)
C
C
C     PREPARATION DES TABLES DE TRANSFORMATION D'UN NOMBRE ALEATOIRE
C     DE DISTRIBUTION UNIFORME EN UN NOMBRE SUIVANR UNE DISTRIBUTION FIXEE
C     C
C
      DIMENSION TAB(20,4)
      INTEGER *2 NB,I
      REAL X1,TAB
      DO 10 I=2,NB
      X1=TAB(I-1,1)-TAB(I,1)
      TAB(I,3)=(TAB(I-1,2)-TAB(I,2))/X1
      TAB(I,4)=(TAB(I,2)*TAB(I-1,1)-TAB(I-1,2)*TAB(I,1))/X1
10    CONTINUE
      TAB(1,3)=0
      TAB(1,4)=0
      RETURN
      END
```

```
      SUBROUTINE FINJOB(TSNN)
C
C
C     FIN DE TRVAIL
C     RETOUR DES CADRES A LA FILE DES CADRES LIBRES
C     MISE A ZERO DES DONNEES LIEES AU TRAVAIL
C
C
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /SLOT/NSLT(2),NSLV(2)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      INTEGER *2 FIRPPN,LASPPN,NQ,PTDQ,PTFQ,NSLT,NSLV
      INTEGER *2 VPNTMC,VPNTDR,TSN,PPAT,TSNN,I,NSL(2),NBOPAG
      INTEGER TIMINT,USER
      INTEGER *2 SERV,EDP,FLAG,PPNW
      NSL(1)=0
      NSL(2)=0
      NBOPAG=TSN(TSNN,9)
      FIRPPN=TSN(TSNN,2)
      LASPPN=TSN(TSNN,1)
      IF(FIRPPN .EQ. 0) GOTO 9
      IF(NQ(1) .EQ. 0) GOTO 5
      PPAT(PTFQ(1),6)=FIRPPN
      PPAT(LASPPN,6)=0
      PPAT(FIRPPN,7)=PTFQ(1)
5     NQ(1)=NQ(1)+TSN(TSNN,3)
6     DO 7 I=1,5
      PPAT(FIRPPN,I)=0
7     CONTINUE
      PPAT(FIRPPN,3)=1
      PPAT(FIRPPN,8)=0
      LASPPN=PPAT(FIRPPN,6)
      IF(LASPPN .EQ. 0) GOTO 9
      FIRPPN=LASPPN
      GOTO 6
9     DO 10 I=1,15
      TSN(TSNN,I) = 0
10    CONTINUE
      DO 20 I=1,NBOPAG
      IF(VPNTDR(TSNN,I) .EQ. 1) NSL(1)=NSL(1)+1
      VPNTMC(TSNN,I)=0
      VPNTDR(TSNN,I)=0
20    CONTINUE
C
C     MISE A JOUR DU NOMBRE DE CADRES  DE LA MEMOIRE AUXILIAIRE A VIDER
C
      NSL(2)=NBOPAG-NSL(1)
      DO 100 SERV=1,2
      IF(NSL(SERV) .EQ. 0) GOTO 100
      NSLV(SERV)=NSLV(SERV)+NSL(SERV)
      IF(EDP(SERV).EQ. 0) FLAG(SERV)=1
100   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE SAVE(BIDON)
C
C
C     SAUVETAGE DES DONNEES EN FIN DE SIMULATION SUR UN FICHIER(DSET25)
C     LES RESULTATS PEUVENT ETRE UTILISES COMME DEBUT DE
C     LA SIMULATION SUIVANTE PAR DATA2
C
C
      COMMON /STAT1/TABST1(20,2),NBCLA1,RESTA1(3)
      COMMON /STAT2/TABST2(20,2),NBCLA2,RESTA2(3)
      COMMON /STAT3/TABST3(20,2),NBCLA3,RESTA3(3)
      COMMON /STAT4/TABST4(20,2),NBCLA4,RESTA4(3)
      COMMON /STAT5/TABST5(20,2),NBCLA5,RESTA5(3)
      COMMON /STAT6/TABST6(20,2),NBCLA6,RESTA6(3)
      COMMON /STAT7/TABST7(20,2),NBCLA7,RESTA7(3)
      COMMON /STAT8/TABST8(20,2),NBCLA8,RESTA8(3)
      COMMON /STAT9/TABST9(20,2),NBCLA9,RESTA9(3)
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /G/TABD1(20,4),TABD2(20,4),TABD3(20,4),TABD4(20,4),TABD5(
     920,4),TABD6(20,4)
      COMMON /H/NB1,NB2,NB3,NB4,NB5,NB6
      COMMON /J/ ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      COMMON /L/TIMLOC(20),TIMOCC(20),TIMMTS(20),TIMIO(20),TIMDP(20)
      COMMON /USERIO/USERIO(2,2),FILEIO(20,2),OCCIO(2),NBFIIO(2)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON/PROC/FILEPR(20,2),NBFIPR
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /SLOT/NSLT(2),NSLV(2)
      COMMON /ACT/NACT,NJOBTA
      INTEGER TABST1,TABST2,TABST3,TABST4,TABST5,TABST6,TABST7
      INTEGER NBCLA1,NBCLA2,NBCLA3,NBCLA4,NBCLA5,NBCLA6,NBCLA7
      REAL RESTA1,RESTA2,RESTA3,RESTA4,RESTA5,RESTA6,RESTA7
      INTEGER TABST8,TABST9,NBCLA8,NBCLA9
      REAL RESTA8,RESTA9
      INTEGER TIMLOC,TIMOCC,TIMDP,TIMMTS,TIMIO
      INTEGER USERIO,USER,TIME,TIMRET
      INTEGER *2 FLAG,PPNW,EDP,FILEIO,FILED1,FILED2,FILEPR
      INTEGER *2 NBFIE1,NBFIE2,NBFIIO,OCCIO,NBFIPR,NSLT,NSLV,NJOBTA
      REAL TABD1,TABD2,TABD3,TABD4,TABD5,TABD6
      INTEGER *2 NB1,NB2,NB3,NB4,NB5,NB6
      INTEGER ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      INTEGER *2 TSN,PPAT,I,J,BIDON,NQ
      INTEGER *2 VPNTMC,VPNTDR,PTDQ,PTFQ,NACT
      INTEGER TIMINT
      DO 10 I=1,20
      WRITE(25,200)(TSN(I,J),J=1,17)
10    CONTINUE
      WRITE(25,2000)(TIMLOC(I),I=1,10)
      WRITE(25,2000)(TIMLOC(I),I=11,20)
      WRITE(25,250)(TIMMTS(I),I=1,20)
      WRITE(25,250)(TIMOCC(I),I=1,20)
      WRITE(25,2000)(TIMIO(I),I=1,10)
      WRITE(25,2000)(TIMIO(I),I=11,20)
      WRITE(25,2000)(TIMDP(I),I=1,10)
      WRITE(25,2000)(TIMDP(I),I=11,20)
      DO 40 I=1,20
      WRITE(25,900)(VPNTMC(I,J),J=1,25)
```

```
      WRITE(25,900)(VPNTMC(I,J),J=26,50)
      WRITE(25,900)(VPNTMC(I,J),J=51,75)
      WRITE(25,900)(VPNTMC(I,J),J=76,100)
40    CONTINUE
      DO 50 I=1,20
      WRITE(25,800)(VPNTDR(I,J),J=1,50)
      WRITE(25,800)(VPNTDR(I,J),J=51,100)
50    CONTINUE
      DO 20 I=1,100
      WRITE(25,1050)(PPAT(I,J),J=1,8)
20    CONTINUE
      DO 25 I=1,10
      WRITE(25,2000)(TIMINT((I-1)*10+J),J=1,10)
25    CONTINUE
      WRITE(25,1300)(NQ(I),I=1,5),NACT
      WRITE(25,2000)(PTDQ(I),I=1,5),(PTFQ(I),I=1,5)
      WRITE(25,1700)((USERIO(I,J),I=1,2),J=1,2),TIME
      WRITE(25,1300)((USER(I,J),J=1,2),I=1,2),(FLAG(I),I=1,2)
      WRITE(25,1400)((FILEIO(I,J),J=1,2),I=1,20),(PPNW(I),I=1,2)
      WRITE(25,1300)(OCCIO(I),I=1,2),(NBFIIO(I),I=1,2),(EDP(I),I=1,2)
      WRITE(25,1500)((FILEPR(I,J),J=1,2),I=1,20),NBFIPR
      WRITE(25,1500)((FILED1(I,J),J=1,2),I=1,20),NBFIE1
      WRITE(25,1500)((FILED2(I,J),J=1,2),I=1,20),NBFIE2
      WRITE(25,1700)(NSLT(I),I=1,2),(NSLV(I),I=1,2),NJOBTA
      WRITE(25,1300) NB1,NB2,NB3,NB4,NB5,NB6
      DO 60 I=1,NB1
      WRITE(25,1200)(TABD1(I,J),J=1,4)
60    CONTINUE
      DO 70 I=1,NB2
      WRITE(25,1200)(TABD2(I,J),J=1,4)
70    CONTINUE
      DO 80 I=1,NB3
      WRITE(25,1200)(TABD3(I,J),J=1,4)
80    CONTINUE
      DO 90 I=1,NB4
      WRITE(25,1200)(TABD4(I,J),J=1,4)
90    CONTINUE
      DO 100 I=1,NB5
      WRITE(25,1200)(TABD5(I,J),J=1,4)
100   CONTINUE
      DO 110 I=1,NB6
      WRITE(25,1200)(TABD6(I,J),J=1,4)
110   CONTINUE
      WRITE(25,1050)ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,
     9ISEED8
      WRITE(25,2000)(TABST1(I,1),I=1,10)
      WRITE(25,2000)(TABST1(I,1),I=11,20)
      WRITE(25,2000)(TABST2(I,1),I=1,10)
      WRITE(25,2000)(TABST2(I,1),I=11,20)
      WRITE(25,2000)(TABST3(I,1),I=1,10)
      WRITE(25,2000)(TABST3(I,1),I=11,20)
      WRITE(25,2000)(TABST4(I,1),I=1,10)
      WRITE(25,2000)(TABST4(I,1),I=11,20)
      WRITE(25,2000)(TABST5(I,1),I=1,10)
      WRITE(25,2000)(TABST5(I,1),I=11,20)
      WRITE(25,2000)(TABST6(I,1),I=1,10)
      WRITE(25,2000)(TABST6(I,1),I=11,20)
      WRITE(25,2000)(TABST7(I,1),I=1,10)
      WRITE(25,2000)(TABST7(I,1),I=11,20)
      WRITE(25,2000)(TABST8(I,1),I=1,10)
```

```
      WRITE(25,2000)(TABST8(I,1),I=11,20)
      WRITE(25,2000)(TABST9(I,1),I=1,10)
      WRITE(25,2000)(TABST9(I,1),I=11,20)
      WRITE(25,1700)NBCLA1,NBCLA2,NBCLA3,NBCLA4,NBCLA5
      WRITE(25,2200)NBCLA6,NBCLA7,NBCLA8,NBCLA9
      WRITE(25,2000)(TIMRET(I),I=1,10)
      WRITE(25,2000)(TIMRET(I),I=11,20)
      RETURN
200   FORMAT(17I5)
250   FORMAT(20I5)
800   FORMAT(50I2)
900   FORMAT(25I3)
1050  FORMAT(8I10)
1200  FORMAT(4F15.5)
1300  FORMAT(6I10)
1400  FORMAT(42I3)
1500  FORMAT(40I4,I3)
1700  FORMAT(5I10)
2000  FORMAT(10I10)
2200  FORMAT(4I5)
      END
```

```
      SUBROUTINE MQET(TSNN,QDEP,PPN)
C
C
C     ENLEVE UN CADRE DONNE DANS UNE DES CHAINES DE CADRES
C     PLACE LE CADRE  A LA FIN DE LA CHEINE DES CADRES DU TRAVAIL
C
C
      INTEGER *2 TSNN,QDEP,PPN,PREC,SUIV
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /COMPT/COMP(40)
      INTEGER *2 TSN,PPAT,PTDQ,PTFQ,NQ
      INTEGER TIMINT,COMP
      NQ(QDEP)=NQ(QDEP)-1
      PREC=PPAT(PPN,7)
      SUIV=PPAT(PPN,6)
      IF( PREC .EQ. SUIV )GOTO 650
      IF( PREC .EQ. 0 )GOTO 600
      IF( SUIV .EQ. 0 )GOTO 500
      PPAT(PREC,6) = SUIV
      PPAT(SUIV,7) = PREC
      GOTO 700
500   PTFQ(QDEP)=PREC
      PPAT(PREC,6) = 0
      GOTO 700
600   PTDQ(QDEP)=SUIV
      PPAT(SUIV,7) = 0
      GOTO 700
650   PTDQ(QDEP)=0
      PTFQ(QDEP)=0
700   IF( TSN(TSNN,2) .EQ. 0 )GOTO 750
      PREC = TSN(TSNN,1)
      PPAT(PREC,6)=PPN
      GOTO 790
750   TSN(TSNN,2)=PPN
      PREC = 0
790   PPAT(PPN,6)=0
      PPAT(PPN,4)=1
      PPAT(PPN,7)=PREC
      TSN(TSNN,1)=PPN
      PPAT(PPN,3)=9
      PPAT(PPN,8)=TSNN
      TSN(TSNN,3)=TSN(TSNN,3)+1
      COMP(19)=COMP(19)+1
      RETURN
      END
```

```
      SUBROUTINE HQET(QDEP,TSNN,PPN)
C
C
C     ENLEVE PREMIER CADRE D'UNE CHAINE DE CADRES
C     LE PLACE EN FIN DE CHAINE DE CADRES D'UN TRAVAIL
C
C
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      COMMON /COMPT/COMP(40)
      INTEGER *2 QDEP,TSNN,NQ,PPN,SUIV,PREC
      INTEGER *2 TSN,PPAT,PTDQ,PTFQ
      INTEGER *2 VPNTMC,VPNTDR
      INTEGER TIMINT,COMP
      PPN=PTDQ(QDEP)
      IF( NQ(QDEP) .EQ. 1 )GOTO 500
      SUIV=PPAT(PPN,6)
      PTDQ(QDEP) = SUIV
      PPAT(SUIV,7) = 0
      GOTO 2000
500   PTDQ(QDEP) = 0
      PTFQ(QDEP) = 0
2000  NQ(QDEP) = NQ(QDEP) -1
      IF( TSN(TSNN,2) .EQ. 0 )GOTO 2500
      PREC = TSN(TSNN,1)
      PPAT(PREC,6)=PPN
      GOTO 3000
2500   TSN(TSNN,2)=PPN
      PREC = 0
3000   PPAT(PPN,6)=0
      PPAT(PPN,4)=1
      PPAT(PPN,5)=0
      PPAT(PPN,7)=PREC
      TSN(TSNN,1)=PPN
      PPAT(PPN,3)=9
      PPAT(PPN,8)=TSNN
      TSN(TSNN,3)=TSN(TSNN,3)+1
      COMP(18)=COMP(18)+1
      RETURN
      END
```

```
      SUBROUTINE DESACT(TSNN)
C
C
C     DESACTIVATION D'UN TRAVAIL
C     TOUS LES CADRES ALLOUES SONT REMIS DANS LA FILE DES CADRES (1,2,OU 3)
C
C
C
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /COMPT/COMP(40)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      INTEGER *2 NQ,TSNN,PPN,NEXPPN,QARR
      INTEGER *2 TSN,PPAT,PTDQ,PTFQ
      INTEGER *2 FLAG,PPNW,EDP
      INTEGER TIMINT,COMP,USER
      COMP(16)=COMP(16)+1
      IF(TSN(TSNN,3) .EQ. 0) GOTO 3500
      PPN = TSN(TSNN,2)
10    NEXPPN=PPAT(PPN,6)
      QARR=1
      IF(PPAT(PPN,5) .EQ. 0) GOTO 3100
      IF(EDP(SERV) .EQ. 0) FLAG(SERV)=1
      QARR=PPAT(PPN,2)+1
3100  CALL MTEQ(TSNN,QARR,PPN)
      IF(NEXPPN .EQ.0) GOTO 3500
      PPN=NEXPPN
      GOTO 10
3500  RETURN
      END
```

```
      SUBROUTINE HQEQ(QDEP,QARR,PPN)
C
C     ENLEVE PREMIER CADRE DE LA CHAINE(QDEP)C
C     LE PLACE A LA FIN D'UNE AUTRE CHAINE(QARR)
C
C
      INTEGER *2 QDEP,QARR,NQ,PPN,SUIV,PREC
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      INTEGER *2 TSN,PPAT,PTDQ,PTFQ
      INTEGER TIMINT
      PPN=PTDQ(QDEP)
      IF(NQ(QDEP) .EQ. 1) GOTO 500
      SUIV=PPAT(PPN,6)
      PTDQ(QDEP)=SUIV
      PPAT(SUIV,7) = 0
      GOTO 2000
500   PTDQ(QDEP)=0
      PTFQ(QDEP)=0
2000  NQ(QDEP)=NQ(QDEP)-1
      IF(PTDQ(QARR) .EQ. 0) GOTO 2500
      PREC=PTFQ(QARR)
      PPAT(PREC,6)=PPN
      GOTO 2800
2500  PTDQ(QARR)=PPN
      PREC=0
2800   PPAT(PPN,6)=0
      PPAT(PPN,7)=PREC
      PTFQ(QARR)=PPN
      PPAT(PPN,3)=QARR
      PPAT(PPN,5)=0
      NQ(QARR)=NQ(QARR)+1
      RETURN
      END
```

```
      SUBROUTINE MQEQ(QDEP,QARR,PPN)
C
C
C     ENLEVE UN CADRE DONNE(PPN) DE LA CHAINE DE CADRE QDEP POUR LE PLACER
C     EN FIN DE CHAINE QARR
C
C
      INTEGER *2 QDEP,QARR,NQ,PPN,PREC,SUIV
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      INTEGER *2 PPAT,PTDQ,PTFQ
      INTEGER TIMINT
      PREC=PPAT(PPN,7)
      SUIV=PPAT(PPN,6)
      IF(PREC .EQ. SUIV) GOTO 800
      IF( PREC .EQ. 0 )GOTO 700
      IF( SUIV .EQ. 0 )GOTO 600
      PPAT(PREC,6) = SUIV
      PPAT(SUIV,7) = PREC
      GOTO 1100
600   PTFQ(QDEP) = PREC
      PPAT(PREC,6) = 0
      GOTO 1100
700   PTDQ(QDEP) = SUIV
      PPAT(SUIV,7) = 0
      GOTO 1100
800   PTDQ(QDEP) = 0
      PTFQ(QDEP) = 0
1100  NQ(QDEP) = NQ(QDEP) - 1
      IF( NQ(QARR) .EQ. 0 )GOTO 1200
      PREC = PTFQ(QARR)
      PPAT(PREC,6)=PPN
      PPAT(PPN,7)=PREC
      PTFQ(QARR)=PPN
      GOTO 1500
1200  PTFQ(QARR)=PPN
      PTDQ(QARR)=PPN
      PPAT(PPN,7)=0
1500  PPAT(PPN,3)=QARR
      PPAT(PPN,5)=0
      PPAT(PPN,6)=0
      NQ(QARR) = NQ(QARR) + 1
      RETURN
      END
```

```
      SUBROUTINE DATA1(NJOBTA)
C
C
C     ROUTINE D'INTRODUCTION DES DONNEES POUR LA PREMIERE SIMULATION
C     LECTURE SUR FICHIER(DSET50)
C
C
      COMMON /STAT1/TABST1(20,2),NBCLA1,RESTA1(3)
      COMMON /STAT2/TABST2(20,2),NBCLA2,RESTA2(3)
      COMMON /STAT3/TABST3(20,2),NBCLA3,RESTA3(3)
      COMMON /STAT4/TABST4(20,2),NBCLA4,RESTA4(3)
      COMMON /STAT5/TABST5(20,2),NBCLA5,RESTA5(3)
      COMMON /STAT6/TABST6(20,2),NBCLA6,RESTA6(3)
      COMMON /STAT7/TABST7(20,2),NBCLA7,RESTA7(3)
      COMMON /STAT8/TABST8(20,2),NBCLA8,RESTA8(3)
      COMMON /STAT9/TABST9(20,2),NBCLA9,RESTA9(3)
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /ACT/NACT,NJOBTA
      COMMON /G/TABD1(20,4),TABD2(20,4),TABD3(20,4),TABD4(20,4),TABD5(
     920,4),TABD6(20,4)
      COMMON /H/NB1,NB2,NB3,NB4,NB5,NB6
      COMMON /J/ ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      COMMON /L/TIMLOC(20),TIMOCC(20),TIMMTS(20),TIMIO(20),TIMDP(20)
      COMMON /SLOT/NSLT(2),NSLV(2)
      INTEGER TIMLOC,TIMOCC,TIMDP,TIMMTS,TIMIO
      REAL TABD1,TABD2,TABD3,TABD4,TABD5,TABD6
      INTEGER TABST1,TABST2,TABST3,TABST4,TABST5,TABST6,TABST7
      INTEGER NBCLA1,NBCLA2,NBCLA3,NBCLA4,NBCLA5,NBCLA6,NBCLA7
      REAL RESTA1,RESTA2,RESTA3,RESTA4,RESTA5,RESTA6,RESTA7
      INTEGER TABST8,TABST9,NBCLA8,NBCLA9
      REAL RESTA8,RESTA9
      INTEGER *2 NB1,NB2,NB3,NB4,NB5,NB6
      INTEGER ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      INTEGER TIMINT,BORINF,LONCLA
      INTEGER *2LIMIT,RTT,NSLT,NSLV
      INTEGER *2 VPNTMC,VPNTDR,DRN,NDR,X,J
      INTEGER *2 TSN,I,NBOPAG,PPN,VPN,TSNN
      INTEGER *2 PREC,SUIV,PTDQ,PTFQ,PPAT,NQ,NJOBTA,NACT
C
C     LECTURE DU NOMBRE DE TRAVAUX(NJOBTA) ET DE LA TAILLE EN PAGE DE
C     LA MEMOIRE CENTRALE(NQ(1))
C
      READ(50,200) NJOBTA,NQ(1)
      DRN=1
      NDR=2
C
C     REMPLISSAGE DE LA CHAINE DES CADRES NUMERO 1 AVEC TOUS LES CADRES
C
      PREC = 0
      SUIV = 2
      LIMIT=NQ(1)-1
      DO 50 PPN=1,LIMIT
      PPAT(PPN,3)=1
      PPAT(PPN,6) = SUIV
      PPAT(PPN,7) = PREC
      SUIV=SUIV+1
      PREC = PPN
50    CONTINUE
```

```
      PPAT(NQ(1),6)=0
      PPAT(NQ(1),7)=LIMIT
      PPAT(NQ(1),3)=1
      PTDQ(1) = 1
      PTFQ(1)=NQ(1)
      NSLT(1)=500
      NSLT(2)=500
      NACT = 999
      DO 10 TSNN=1,NJOBTA
C
CC    LECTURE DES DONNEES CONCERNANT LES TRAVAUX
CC          NBOPAG  NOMBRE DE PAGES DY TRAVAIL
CC          TEMPS DE VIE DU TRAVAIL EXPRIME EN TIME-SLICE
C
      READ(50,200) NBOPAG,RTT
      TSN(TSNN,8) = NACT
      TSN(TSNN,9) = NBOPAG
      TSN(TSNN,5)=8
      TSN(TSNN,6)=RTT
      TSN(TSNN,7)=5
      NACT = NACT - 1
C
CC    REMPLISSAGE ALTERNE DE LA MEMOIRE AUXILIAIRE
C
      DO 10 VPN=1,NBOPAG
      VPNTDR(TSNN,VPN)=DRN
      X=DRN
      DRN=NDR
      NDR=X
      IF(DRN .EQ. 1) NSLT(1)=NSLT(1)-1
      IF(DRN .EQ. 2) NSLT(2)=NSLT(2)-1
10    CONTINUE
C
CC    LECTURE DES DONNEES LIEES AUX DISTRIBUTIONS
CC    LA COLONNE 1 COMPREND LES POINTS DE REPERES DE LA DISTRIBUTION UNIFORME
CC    LA COLONNE DEUX CEUX DE LA DISTRIBUTION CHOISIE
C
      READ(50,250) NB1,NB2,NB3,NB4,NB5,NB6
      DO 70 I=1,NB1
      READ(50,300)(TABD1(I,J),J=1,2)
70    CONTINUE
      CALL PREPTA(TABD1,NB1)
      DO 80 I=1,NB2
      READ(50,300)(TABD2(I,J),J=1,2)
80    CONTINUE
      CALL PREPTA(TABD2,NB2)
      DO 90 I=1,NB3
      READ(50,300)(TABD3(I,J),J=1,2)
90    CONTINUE
      CALL PREPTA(TABD3,NB3)
      DO 92 I=1,NB4
      READ(50,300)(TABD4(I,J),J=1,2)
92    CONTINUE
      CALL PREPTA(TABD4,NB4)
      DO 95 I=1,NB5
      READ(50,300)(TABD5(I,J),J=1,2)
95    CONTINUE
      CALL PREPTA(TABD5,NB5)
      DO 97 I=1,NB6
      READ(50,300)(TABD6(I,J),J=1,2)
```

```
97      CONTINUE
        CALL PREPTA(TABD6,NB6)
        ISEED1=15080706
        ISEED2=14942317
        ISEED3=16062212
        ISEED4=15151512
        ISEED5=15120806
        ISEED6=15181711
        ISEED7=16254378
        ISEED8=16429739
C
C       INITIALISATION DES DEMANDES ENTREE-SORTIE
C
        DO 110 TSNN=1,NJOBTA
        RESULT=RANDOM(ISEED7)
        CALL DISTRI(TABD6,NB6,RESULT)
        TIMIO(TSNN)=RESULT*1600000
110     CONTINUE
C
C       LECTURE DES DONNEES CONCERNANT LES STATISTIQUES
C             NBCLA  NOMBRE DE CLASSES DANS LE TABLEAU STATISTIQUE
C             (NON COMPRIS LA CLASSE AU DESSUS DE LA DERNIERER LIMITE
C             LA PREMIERE LIMITE DE CLASSE EST TOUJOURS 0 COMME BORNE INFERIEURE)
C             BORUNF  BORNE SUPERIEURE DE LA PREMIERE CLASSE
C             LONCLA  LONGUEUR DE LA CLASSE
C
        READ(50,2000)NBCLA1,BORINF,LONCLA
        CALL PRETST(TABST1,NBCLA1,BORINF,LONCLA)
        READ(50,2000)NBCLA2,BORINF,LONCLA
        CALL PRETST(TABST2,NBCLA2,BORINF,LONCLA)
        READ(50,2000)NBCLA3,BORINF,LONCLA
        CALL PRETST(TABST3,NBCLA3,BORINF,LONCLA)
        READ(50,2000)NBCLA4,BORINF,LONCLA
        CALL PRETST(TABST4,NBCLA4,BORINF,LONCLA)
        READ(50,2000)NBCLA5,BORINF,LONCLA
        CALL PRETST(TABST5,NBCLA5,BORINF,LONCLA)
        READ(50,2000)NBCLA6,BORINF,LONCLA
        CALL PRETST(TABST6,NBCLA6,BORINF,LONCLA)
        READ(50,2000)NBCLA7,BORINF,LONCLA
        CALL PRETST(TABST7,NBCLA7,BORINF,LONCLA)
        READ(50,2000)NBCLA8,BORINF,LONCLA
        CALL PRETST(TABST8,NBCLA8,BORINF,LONCLA)
        READ(50,2000)NBCLA9,BORINF,LONCLA
        CALL PRETST(TABST9,NBCLA9,BORINF,LONCLA)
        RETURN
200     FORMAT(2I4)
250     FORMAT(6I4)
300     FORMAT(2F14.5)
2000    FORMAT(3I10)
        END
```

```
      SUBROUTINE MTEQ(TSNN,TYWR,PPN)
C
C
C     ENLEVE UN CADRE DONNE(PPN) DE LA CHAINE DES CADRES D'UN TRAVAIL
C     ON LE PLACE EN FIN DE FILE QDEP
C
C
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /COMPT/COMP(40)
      INTEGER *2 TSNN,TYWR,PPN,NQ,PREC,SUIV
      INTEGER *2 TSN,PPAT,PTDQ,PTFQ
      INTEGER TIMINT,COMP
      IF( TSN(TSNN,3) .EQ. 1 )GOTO 5000
      IF( PPN .EQ. TSN(TSNN,2) )GOTO 4000
      IF( PPN .EQ. TSN(TSNN,1) )GOTO 3000
      PREC = PPAT(PPN,7)
      SUIV = PPAT(PPN,6)
      PPAT(PREC,6) = SUIV
      PPAT(SUIV,7) = PREC
      GOTO 6000
3000  PREC = PPAT(PPN,7)
      TSN(TSNN,1) = PREC
      PPAT(PREC,6) = 0
      GOTO 6000
4000  SUIV = PPAT(PPN,6)
      TSN(TSNN,2) = SUIV
      PPAT(SUIV,7) = 0
      GOTO 6000
5000  TSN(TSNN,1) = 0
      TSN(TSNN,2) = 0
6000  TIMINT(PPN) = 0
      PPAT(PPN,4) = 0
      TSN(TSNN,3) = TSN(TSNN,3) - 1
      PREC=PTFQ(TYWR)
      IF(PREC .EQ.0) GOTO 8000
      PTFQ(TYWR)=PPN
      PPAT(PREC,6)=PPN
      GOTO 9000
8000  PTDQ(TYWR) = PPN
      PTFQ(TYWR) = PPN
9000  PPAT(PPN,6) = 0
      PPAT(PPN,7)=PREC
      PPAT(PPN,3)=TYWR
      NQ(TYWR) = NQ(TYWR) + 1
      COMP(20)=COMP(20)+1
      RETURN
      END
```

```
      SUBROUTINE SERPRO(TSNN,INDIC,DISQUE,VPN,DRN,TIMEX,FIREVE)
C
C
C     ROUTINE GENERANT LA SUITE DES REFERENCES AUX PAGES ET GERANT LES EVENEMEN\
C     LIES AU PROCESSEUR
C
C
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /COMPT/COMP(40)
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /G/TABD1(20,4),TABD2(20,4),TABD3(20,4),TABD4(20,4),TABD5(
     920,4),TABD6(20,4)
      COMMON /H/NB1,NB2,NB3,NB4,NB5,NB6
      COMMON /J/ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      COMMON /L/TIMLOC(20),TIMOCC(20),TIMMTS(20),TIMIO(20),TIMDP(20)
      REAL TABD1,TABD2,TABD3,TABD4,TABD5,TABD6
      INTEGER *2 NB1,NB2,NB3,NB4,NB5,NB6
      INTEGER ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      INTEGER *2 NBOPAG,BLISIZ,INDIC,DRN,TSNN
      INTEGER *2 TSN,VPN,PPN,QDEP,PPAT,PTDQ,PTFQ,DISQUE,NQ
      INTEGER *2 VPNTDR,VPNTMC
      INTEGER TIMLOC,TIMOCC,TIMDP,TIMMTS,TIMIO,TIMINT,TIMEX
      INTEGER TIME,TIMRET,FIREVE,COMP
      REAL RESULT
      TIMEX=0
      NBOPAG=TSN(TSNN,9)
C
C     STOPPE PROCESSEUR SI TIME=PREMIER EVENEMENT A VENIR(INTERRUPTION
C     DUE A FIN D'I/O DISQUE OU TAMBOUR
C
1     IF(FIREVE .EQ. TIME) GOTO 850
      TIME=TIME+1
      TIMLOC(TSNN)=TIMLOC(TSNN)-1
      TIMMTS(TSNN)=TIMMTS(TSNN)+1
      TIMIO(TSNN)=TIMIO(TSNN)-1
      TIMEX=TIMEX+1
      IF(TIMLOC(TSNN).GT. 0) GOTO 20
C
C     FIN DE LOCALITE C
C     RECHERCHE DES CARACTERISTISUES DE LA NOUVELLE LOCALITE
C     TAILLE,CENTRE ET DUREE
C
      RESULT=RANDOM(ISEED1)
      CALL DISTRI(TABD1,NB1,RESULT)
      TIMLOC(TSNN)=RESULT*800
      RESULT=RANDOM(ISEED2)
      CALL DISTRI(TABD2,NB2,RESULT)
      BLISIZ=RESULT*NBOPAG+1
      IF(BLISIZ .GT. 20) BLISIZ=20
      TSN(TSNN,12)=BLISIZ
      RESULT=RANDOM(ISEED3)
      TSN(TSNN,10)=INT(RESULT*3)-1
      RESULT=RANDOM(ISEED5)
      TSN(TSNN,11)=RESULT*NBOPAG
20    IF(TIMIO(TSNN) .GT. 0) GOTO 50
C
C     DEMANDE D'ENTREE-SORTIE RECHERCHE DU DISQUE ET DE LA PROCHAINE DEMANDE
C     POUR CE TRAVAIL
```

```
C
      RESULT=RANDOM(ISEED7)
      CALL DISTRI(TABD6,NB6,RESULT)
      TIMIO(TSNN)=RESULT*1600000
      DISQUE=RANDOM(ISEED8)*2 +1
      INDIC=5
      GOTO 900
50    IF(TIMMTS(TSNN) .LT. 40000) GOTO 500
      TIMMTS(TSNN)=0
      TSN(TSNN,7)=TSN(TSNN,7)-1
      IF(TSN(TSNN,7) .LE. 0) GOTO 300
C
C     C       FIN DE MICRO-TIME-SLICE
C
      INDIC=2
      GOTO 900
300   TSN(TSNN,6)=TSN(TSNN,6)-1
      TSN(TSNN,7)=5
      TSN(TSNN,5)=TSN(TSNN,3)
      IF(TSN(TSNN,6) .LT. 0) GOTO 400
C
C     FIN DE TIME-SLICE
C
      INDIC=4
      GOTO 900
C
C     FIN DE TRAVAIL
C
400   INDIC=1
      GOTO 900
C
C     RECHERCHE DE LA PROCHAINE PAGE DEMANDEE PAR LE TRAVAILC
C
500   RESULT=RANDOM(ISEED4)
      IF(TSN(TSNN,10)) 600,610,620
600   CALL DISTRI(TABD3,NB3,RESULT)
      GOTO 700
610   CALL DISTRI(TABD4,NB4,RESULT)
      GOTO 700
620   CALL DISTRI(TABD5,NB5,RESULT)
700   VPN=RESULT*TSN(TSNN,12)+1+TSN(TSNN,11)
      IF(VPN .GT. NBOPAG) VPN=VPN - NBOPAG
      PPN=VPNTMC(TSNN,VPN)
      IF(PPN .EQ. 0) GOTO 800
C
C     PAGE DEJA EN MEMOIRE CENTRALE
C
      PPAT(PPN,4)=1
      QDEP=PPAT(PPN,3)
      IF(QDEP .EQ. 9) GOTO 750
C
C     CADRE N'EST PLUS DANS LA CHAINE DU TRAVAIK ON L'Y RAMENE(FAUX
      DEFAUT DE PAGE)
C
      IF(TSN(TSNN,3) .EQ. TSN(TSNN,5)) TSN(TSNN,5)=TSN(TSNN,5)+1
      COMP(21)=COMP(21)+1
      CALL MQET(TSNN,QDEP,PPN)
750   RESULT=RANDOM(ISEED6)
      IF(RESULT .GT. 0.8) PPAT(PPN,5)=1
      GOTO 1
```

```
C
C       DEFAUT DE PAGE
C
800     INDIC=3
        DRN=VPNTDR(TSNN,VPN)
        GOTO 900
850     INDIC=6
900     RETURN
        END
```

```
      SUBROUTINE DATA2(NJOBTA)
C
C
C     ROUTINE DE LECTURE DES DONNEES(DSET50) POUR LES SIMULATIONS AUTRES QUE LA
C     PREMIERE
C
C
      COMMON /STAT1/TABST1(20,2),NBCLA1,RESTA1(3)
      COMMON /STAT2/TABST2(20,2),NBCLA2,RESTA2(3)
      COMMON /STAT3/TABST3(20,2),NBCLA3,RESTA3(3)
      COMMON /STAT4/TABST4(20,2),NBCLA4,RESTA4(3)
      COMMON /STAT5/TABST5(20,2),NBCLA5,RESTA5(3)
      COMMON /STAT6/TABST6(20,2),NBCLA6,RESTA6(3)
      COMMON /STAT7/TABST7(20,2),NBCLA7,RESTA7(3)
      COMMON /STAT8/TABST8(20,2),NBCLA8,RESTA8(3)
      COMMON /STAT9/TABST9(20,2),NBCLA9,RESTA9(3)
      COMMON /A/VPNTMC(20,100),VPNTDR(20,100)
      COMMON /B/TSN(20,17)
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      COMMON /ACT/NACT,NJOBTA
      COMMON /G/TABD1(20,4),TABD2(20,4),TABD3(20,4),TABD4(20,4),TABD5(
     920,4),TABD6(20,4)
      COMMON /H/NB1,NB2,NB3,NB4,NB5,NB6
      COMMON /J/ ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      COMMON /L/TIMLOC(20),TIMOCC(20),TIMMTS(20),TIMIO(20),TIMDP(20)
      COMMON /USERIO/USERIO(2,2),FILEIO(20,2),OCCIO(2),NBFIIO(2)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON/PROC/FILEPR(20,2),NBFIPR
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /SLOT/NSLT(2),NSLV(2)
      INTEGER TABST1,TABST2,TABST3,TABST4,TABST5,TABST6,TABST7
      INTEGER NBCLA1,NBCLA2,NBCLA3,NBCLA4,NBCLA5,NBCLA6,NBCLA7
      REAL RESTA1,RESTA2,RESTA3,RESTA4,RESTA5,RESTA6,RESTA7
      INTEGER TABST8,TABST9,NBCLA8,NBCLA9
      REAL RESTA8,RESTA9
      INTEGER *2 TSN,PPAT,PTDQ,PTFQ,NQ,NACT,VPNTDR,VPNTMC
      INTEGER *2 TSNN,NJOBTA,I,J,PPN
      REAL TABD1,TABD2,TABD3,TABD4,TABD5,TABD6
      INTEGER *2NB1,NB2,NB3,NB4,NB5,NB6
      INTEGER ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      INTEGER TIMINT,TIMLOC,TIMIO,TIMOCC,TIMDP,TIMMTS
      INTEGER USERIO,USER,TIME,TIMRET
      INTEGER *2 FLAG,PPNW,EDP,FILEIO,FILED1,FILED2,FILEPR
      INTEGER *2 NBFIE1,NBFIE2,NBFIIO,OCCIO,NBFIPR,NSLT,NSLV
      DO 10 TSNN=1,20
      READ(50,100)(TSN(TSNN,I),I=1,17)
10    CONTINUE
      READ(50,2000)(TIMLOC(I),I=1,10)
      READ(50,2000)(TIMLOC(I),I=11,20)
      READ(50,250)(TIMMTS(I),I=1,20)
      READ(50,250)(TIMOCC(I),I=1,20)
      READ(50,2000)(TIMIO(I),I=1,10)
      READ(50,2000)(TIMIO(I),I=11,20)
      READ(50,2000)(TIMDP(I),I=1,10)
      READ(50,2000)(TIMDP(I),I=11,20)
      DO 11 TSNN=1,20
      READ(50,500)(VPNTMC(TSNN,I),I=1,25)
      READ(50,500)(VPNTMC(TSNN,I),I=26,50)
      READ(50,500)(VPNTMC(TSNN,I),I=51,75)
```

```
      READ(50,500)(VPNTMC(TSNN,I),I=76,100)
11    CONTINUE
      DO 12 TSNN=1,20
      READ(50,600)(VPNTDR(TSNN,I),I=1,50)
      READ(50,600)(VPNTDR(TSNN,I),I=51,100)
12    CONTINUE
      DO 20 PPN=1,100
      READ(50,1050)(PPAT(PPN,I),I=1,8)
20    CONTINUE
      DO 25 I=1,10
      READ(50,2000)(TIMINT((I-1)*10+J),J=1,10)
25    CONTINUE
      READ(50,1300)(NQ(I),I=1,5),NACT
      READ(50,2000)(PTDQ(I),I=1,5),(PTFQ(I),I=1,5)
      READ(50,1700)((USERIO(I,J),I=1,2),J=1,2),TIME
      READ(50,1300)((USER(I,J),J=1,2),I=1,2),(FLAG(I),I=1,2)
      READ(50,1400)((FILEIO(I,J),J=1,2),I=1,20),(PPNW(I),I=1,2)
      READ(50,1300)(OCCIO(I),I=1,2),(NBFIIO(I),I=1,2),(EDP(I),I=1,2)
      READ(50,1500)((FILEPR(I,J),J=1,2),I=1,20),NBFIPR
      READ(50,1500)((FILED1(I,J),J=1,2),I=1,20),NBFIE1
      READ(50,1500)((FILED2(I,J),J=1,2),I=1,20),NBFIE2
      READ(50,1700)(NSLT(I),I=1,2),(NSLV(I),I=1,2),NJOBTA
      READ(50,1300)NB1,NB2,NB3,NB4,NB5,NB6
      DO 60 I=1,NB1
      READ(50,1200)(TABD1(I,J),J=1,4)
60    CONTINUE
      DO 70 I=1,NB2
      READ(50,1200)(TABD2(I,J),J=1,4)
70    CONTINUE
      DO 80 I=1,NB3
      READ(50,1200)(TABD3(I,J),J=1,4)
80    CONTINUE
      DO 90 I=1,NB4
      READ(50,1200)(TABD4(I,J),J=1,4)
90    CONTINUE
      DO 95 I=1,NB5
      READ(50,1200)(TABD5(I,J),J=1,4)
95    CONTINUE
      DO 110 I=1,NB6
      READ(50,1200)(TABD6(I,J),J=1,4)
110   CONTINUE
      READ(50,1050)ISEED1,ISEED2,ISEED3,ISEED4,ISEED5,ISEED6,ISEED7,
     9ISEED8
      READ(50,2000)(TABST1(I,1),I=1,10)
      READ(50,2000)(TABST1(I,1),I=11,20)
      READ(50,2000)(TABST2(I,1),I=1,10)
      READ(50,2000)(TABST2(I,1),I=11,20)
      READ(50,2000)(TABST3(I,1),I=1,10)
      READ(50,2000)(TABST3(I,1),I=11,20)
      READ(50,2000)(TABST4(I,1),I=1,10)
      READ(50,2000)(TABST4(I,1),I=11,20)
      READ(50,2000)(TABST5(I,1),I=1,10)
      READ(50,2000)(TABST5(I,1),I=11,20)
      READ(50,2000)(TABST6(I,1),I=1,10)
      READ(50,2000)(TABST6(I,1),I=11,20)
      READ(50,2000)(TABST7(I,1),I=1,10)
      READ(50,2000)(TABST7(I,1),I=11,20)
      READ(50,2000)(TABST8(I,1),I=1,10)
      READ(50,2000)(TABST8(I,1),I=11,20)
      READ(50,2000)(TABST9(I,1),I=1,10)
```

```
      READ(50,2000)(TABST9(I,1),I=11,20)
      READ(50,1700)NBCLA1,NBCLA2,NBCLA3,NBCLA4,NBCLA5
      READ(50,2200)NBCLA6,NBCLA7,NBCLA8,NBCLA9
      READ(50,2000)(TIMRET(I),I=1,10)
      READ(50,2000)(TIMRET(I),I=11,20)
      RETURN
100   FORMAT(17I5)
250   FORMAT(20I5)
500   FORMAT(25I3)
600   FORMAT(50I2)
1050  FORMAT(8I10)
1200  FORMAT(4F15.5)
1300  FORMAT(6I10)
1400  FORMAT(42I3)
1500  FORMAT(40I4,I3)
1700  FORMAT(5I10)
2000  FORMAT(10I10)
2200  FORMAT(4I5)
      END
```

```
      SUBROUTINE DISTRI(TAB,NB,RESULT)
C
C
C     TRANSFORME UN NOMBRE ALEATOIRE DE DISTRIBUTION UNIFORME
C     EN NOMBRE ALEATOIRE SUIVANT DISTRIBUTION CONNUE
C     INTERPOLATION ENTRE LES POINTS DE LA TABLE LINEAIRE
C
C
      DIMENSION TAB(20,4)
      REAL TAB,RESULT
      INTEGER *2 NB
      RESULT=RESULT*TAB(NB,1)
      DO 100 I=1,NB
      IF (RESULT .GT. TAB(I,1)) GOTO 100
      RESULT=RESULT*TAB(I,3)+TAB(I,4)
      GOTO 200
100   CONTINUE
200   RETURN
      END
```

```
      SUBROUTINE STAT(TAB,NBTAB,VALEUR,RESTA)
C
C     CALCUL DE SOMME ET SOMME DES CARRES DES LEMENTS
C     CALCUL DU NOMBRE DE ENTREES DANS LA TABLE
C     LA DERNIERE ENTREE CONTRENT LE NOMBRE DE VALEURS
C     DEPASSANT LA BORNE SUPERIEURE DE LA DERNIEREE CLASSE
C
C
C     ROUTINE D'INSERTION DES DONNES DANS LES TABLES DE STATISTIQUES
C     ON CHERCHE DANS QUELLE CLASSE DE LA TABLE TAB LA VALEUR SE TROUVE
C     ON INCREMENTE LE NOMBRE D'OCCURENCE D'UNE UNITE
C     CHAQUE OCCURENCE EST STOCKEE DANS LA LIGNE CORRESPONDANTE A LA
C     CLASSE DANS LA SECONDE COLONNE
C     LA VALEUR INDIQUE DANS LA PREMIERE COLONNE INDIQUE LA BORNE INFERIEURE DE
C     LA CLASSE
      INTEGER TAB(20,2),NBTAB,VALEUR,PAS,POS,MULT
      REAL RESTA(3)
      IF(NBTAB .LE.16) GOTO 10
      PAS=16
      POS=16
      GOTO 20
10    PAS=8
      POS=8
20    MULT=-1
      IF(VALEUR .GE. TAB(POS,1)) MULT=1
100   IF(PAS .EQ. 1) GOTO 2000
      PAS=PAS/2
      POS=POS+MULT*PAS
      IF(POS .LE.NBTAB) GOTO 20
      MULT=-1
      GOTO 100
2000  IF(MULT) 4000,3000,3000
3000  TAB(POS,2)=TAB(POS,2)+1
      GOTO 5000
4000  TAB(POS-1,2)=TAB(POS-1,2)+1
5000  RESTA(1)=RESTA(1)+VALEUR
      RESTA(2)=RESTA(2)+VALEUR*VALEUR
      RESTA(3)=RESTA(3)+1
      RETURN
      END
```

```
      SUBROUTINE CLASTA(TAB,NBCLA,RESTA,NUMERO)
C
C
C     CLOTURE DES STATISTIQUES TABLE PAR TABLE
C
C
      INTEGER TAB(20,2),NBCLA,NUMERO
      REAL RESTA(3)
      REAL MOY,VAR,ECART,LIMINF,LIMSUP,VALRET
      REAL PROP,PROPCU,NB
      WRITE(20,100) NUMERO
C
C     CALCUL DE LA MOYENNE DE LA VARIANCE ET DE L'INTERVALLE DE
C     CONFIANCE(DONNEES INDEOENDANTES -VALEUR A 95 %)
C
C
      NB=RESTA(3)
      IF(NB .LE. 0) GOTO 90
      MOY=RESTA(1)/NB
      VAR=(RESTA(2)/NB)-MOY*MOY
      WRITE(20,200)MOY,VAR
      IF(VAR .LE. 0) GOTO 4
      ECART=SQRT(VAR/NB)*1.96
      LIMINF=MOY-ECART
      LIMSUP=MOY+ECART
      WRITE(20,300) LIMINF,MOY,LIMSUP
4     IF(MOY .LE. 0) GOTO 5
      VALRET=ECART/MOY
5     WRITE(20,400) VALRET
C
C     IMPRESSION POUR CHAQUE TABLE DES BORNES,NOMBRE D'OCCURENCES
C     PROPORTION DANS CHAQUE CLASSE ET PROPORTION CUMULEE
C
      WRITE(20,500)
      PROPCU=0
      DO 10 I=1,NBCLA
      PROP=TAB(I,2)/NB
      PROPCU=PROPCU+PROP
      WRITE(20,600)TAB(I,1),TAB(I,2),PROP,PROPCU
10    CONTINUE
      GOTO 95
90    WRITE(20,700)
95    RETURN
100   FORMAT(1X,//,'RESULTATS STATISTIQUES POUR LA TABLE',I3)
200   FORMAT(1X,/,'MOYENNE = ',F14.5,' VARIANCE = ',F14.5)
300   FORMAT(1X,/,'INTERV. CONFIANCE :',F14.5,' <',F14.5,' <',F14.5)
400   FORMAT(1X,/,'VALEUR RELATIVE DU DEMI-INTERVALLE : ',F14.5)
500   FORMAT(1X,///,6X,' CLASSE * OCCURENCE ','*PROPORTION * CUMUL')
600   FORMAT(1X,2X,I8,'    * ',2X,I6,'  * ',F10.6,' * ',F9.6,' *')
700   FORMAT(1X,'PAS DE RESULTATS POUR CETTE TABLE',///)
      END
```

```
      SUBROUTINE ENDSTA(TIME)
C
C
C     CLOTURE DES STATISTIQUES   APPEL DE LA ROUTIME DE TRAITEMENT POUR CHAQUE
C     TABLE
C
C
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /STAT1/TABST1(20,2),NBCLA1,RESTA1(3)
      COMMON /STAT2/TABST2(20,2),NBCLA2,RESTA2(3)
      COMMON /STAT3/TABST3(20,2),NBCLA3,RESTA3(3)
      COMMON /STAT4/TABST4(20,2),NBCLA4,RESTA4(3)
      COMMON /STAT5/TABST5(20,2),NBCLA5,RESTA5(3)
      COMMON /STAT6/TABST6(20,2),NBCLA6,RESTA6(3)
      COMMON /STAT7/TABST7(20,2),NBCLA7,RESTA7(3)
      COMMON /STAT8/TABST8(20,2),NBCLA8,RESTA8(3)
      COMMON /STAT9/TABST9(20,2),NBCLA9,RESTA9(3)
      INTEGER  TABST1,TABST2,TABST3,TABST4,TABST5,TABST6,TABST7
      INTEGER  NBCLA1,NBCLA2,NBCLA3,NBCLA4,NBCLA5,NBCLA6,NBCLA7
      INTEGER TABST8,TABST9,NBCLA8,NBCLA9
      REAL  RESTA1,RESTA2,RESTA3,RESTA4,RESTA5,RESTA6,RESTA7
      REAL RESTA8,RESTA9
      INTEGER TIME,TIMRET,NB
      WRITE(20,100) TIME
      NB=1
      CALL CLASTA(TABST1,NBCLA1,RESTA1,NB)
      NB=2
      CALL CLASTA(TABST2,NBCLA2,RESTA2,NB)
      NB=3
      CALL CLASTA(TABST3,NBCLA3,RESTA3,NB)
      NB=4
      CALL CLASTA(TABST4,NBCLA4,RESTA4,NB)
      NB=5
      CALL CLASTA(TABST5,NBCLA5,RESTA5,NB)
      NB=6
      CALL CLASTA(TABST6,NBCLA6,RESTA6,NB)
      NB=7
      CALL CLASTA(TABST7,NBCLA7,RESTA7,NB)
      NB=8
      CALL CLASTA(TABST8,NBCLA8,RESTA8,NB)
      NB=9
      CALL CLASTA(TABST9,NBCLA9,RESTA9,NB)
      RETURN
100   FORMAT(1X,/////,'STATISTIQUES CLOTUREES APRES ',I10,'UNITES')
200   FORMAT(1X,9I10)
      END
```

```
      SUBROUTINE PRETST(TAB,NBCLA,BORINF,LONCLA)
C
C
C     CALCUL DES BORNES DES CLASSES DES TABLES STATISTIQUES
C
C
      INTEGER TAB(20,2),NBCLA,BORINF,LONCLA,I
      LIMITE=BORINF
      NBCLA=NBCLA+1
      TAB(1,1)=0
      DO 10 I=2,NBCLA
      TAB(I,1)=LIMITE
      LIMITE=LIMITE+LONCLA
10    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE PRINTR(TIME)
C
C
C     IMPRESSION DES RESULTATS
C
C
      COMMON /TEMPS/TIME,TIMRET(20)
      COMMON /SERV/USER(2,2),EDP(2),FLAG(2),PPNW(2)
      COMMON /USERIO/USERIO(2,2),FILEIO(20,2),OCCIO(2),NBFIIO(2)
      COMMON /FILEDP/FILED1(20,2),FILED2(20,2),NBFIE1,NBFIE2
      COMMON /COMPT/COMP(40)
      COMMON /B/TSN(20,17)
      COMMON /PROC/FILEPR(20,2),NBFIPR
      COMMON /CADRE/PPAT(100,8),TIMINT(100),NQ(5),PTDQ(5),PTFQ(5)
      INTEGER *2 EDP,FLAG,PPNW,FILEIO,NBFIIO,OCCIO,FILED1,FILED2
      INTEGER *2 NBFIE1,NBFIE2,FILEPR,NBFIPR,PPAT,NQ,PTDQ,PTFQ,TSN
      INTEGER TIME,TIMRET,TIMINT,USER,USERIO,COMP
      WRITE(20,100) TIME
C
C     COMPTEUR D'OCCURENCE
C
      DO 10 I=1,40
      WRITE(20,200) I,COMP(I)
10    CONTINUE
C
C     CONTENU DES FILES D'ATTENTE
C
      WRITE(20,300)
      WRITE(20,400)(FILEPR(I,1),I=1,NBFIPR)
      WRITE(20,400)(FILED1(I,1),I=1,NBFIE1)
      WRITE(20,400)(FILED2(I,1),I=1,NBFIE2)
C
C     CONTENU DES SERVEURS
C
      WRITE(20,500)
      WRITE(20,600)(USER(I,1),I=1,2)
      WRITE(20,600)(USERIO(I,1),I=1,2)
C
C     CADRES LIBRES ET CADRS ALLOURS
C
      WRITE(20,700)
      WRITE(20,800)(NQ(I),I=1,5)
      WRITE(20,900)(TSN(I,3),I=1,20)
      RETURN
100   FORMAT(1X,'FIN DE SIMULATION  TEMPS = ',I10)
200   FORMAT(1X,6X,'COMPTEUR NUMERO',I4,'VALEUR = ',I4)
300   FORMAT(1X,//,'CONTENU DES FILES DATTENTE')
400   FORMAT(1X,/,'TRAVAUX EN ATTENTE ; ',2014)
500   FORMAT(1X,////,'CONTENU DES SERVEURS MEMOIRE ET IO')
600   FORMAT(1X,//,'TRAVAUX EN ATTENTE DE FIN DE TRANSFERT  ',2I4)
700   FORMAT(1X,///,'NOMBRE DE CADRES LIBRES ET ATTRIBUES AUX TRAVAUX')
800   FORMAT(1X,/,'CADRES LIBRES : ',5I5)
900   FORMAT(1X,/,'CACADRES ALLOUES AUX TRAVAUX : ',2014)
      END
```

```
      SUBROUTINE INIALE(BIDON)
C
C
C     INITIALISATION DES NOMBRES ALEATOIRES
C
C
      COMMON /J/ISEED1,ISEED2,ISED3,ISEED4,ISED5,ISEED6,ISEED7,ISEED8
      INTEGER ISEED1,ISEED2,ISED3,ISEED4,ISEED5,ISEED6,ISEED7,ISEED8
      DO 10 I=1,100
      CALL RANDOM(ISEED1)
10    CONTINUE
      DO 20 I=1,100
      CALL RANDOM(ISEED2)
20    CONTINUE
      DO 30 I=1,100
      CALL RANDOM(ISEED3)
30    CONTINUE
      DO 40 I=1,100
      CALL RANDOM(ISEED4)
40    CONTINUE
      DO 50 I=1,100
      CALL RANDOM(ISEED5)
50    CONTINUE
      DO 60 I=1,100
      CALL RANDOM(ISEED6)
60    CONTINUE
      DO 70 I=1,100
      CALL RANDOM(ISEED7)
70    CONTINUE
      DO 80 I=1,100
      CALL RANDOM(ISEED8)
80    CONTINUE
      RETURN
      END
```

```
RANDOM   CSECT
         USING                 *,15
         STM                   2,5,28(13)
         L                     2,0(1)
         L                     5,A
         M                     4,0(2)
         D                     4,P
         ST                    4,0(2)
         SRL                   4,7
         A                     4,CHAR
         ST              4,ZONE
          LE             0,ZONE
         LM                    2,5,28(13)
         MVI                   12(13),X'FF'
         BR                    14
CHAR          DC               F'1073741824'
A         DC                   F'16807'
P         DC                   F'2147403647'
ZONE      DC          F'0'
         END
```