MASTER EN SCIENCES INFORMATIQUES

Commutation numérique et temporelle
Tome Premier : Étude des fondements théoriques

Evrard, Marc; Vangaver, Rudy; Trigaux, Benoît

Award date:
1985

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR (BELGIQUE)

INSTITUT D'INFORMATIQUE

COMMUTATION NUMERIQUE ET TEMPORELLE

Tome Premier :

Etude des fondements théoriques

Promoteur : J. Brunin

Mémoire présenté par

Marc EVRARD
Rudy VANGAVER
Benoît TRIGAUX

en vue de l'obtention
du grade de

LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1984 - 1985

Nous tenons tout d'abord à remercier Monsieur le Professeur Jean Brunin qui a accepté de diriger ce mémoire et qui nous a permis, par ses conseils et critiques, de mener ce travail à son terme.

Nous exprimons également toute notre gratitude à Messieurs Schmerber de la société Ericsson (Bruxelles), J. Zandberg de Ericsson (Rijen, Pays-Bas) et A. van Dijck de Ericsson (Zwijndrecht, Pays-Bas) pour l'excellent accueil que nous avons reçu dans leurs implantations de Rijen et Zwijndrecht. Nous remercions aussi Monsieur J. Pierson des Facultés Universitaires de Namur pour l'aide qu'il nous a apportée dans le choix d'une implémentation adéquate, et Monsieur J. Davenport de la société Logica à Londres pour l'apport de documentation concernant les réseaux numériques.

Enfin, notre plus vive reconnaissance va à Monsieur Doumic Trigaux, Madame Dominique Vangaver-Gilles et Mademoiselle Catherine Brion pour l'aide irremplaçable qu'ils nous ont apportée dans la réalisation de ce mémoire.

TABLE DES MATIERES.

2. EVOLUTION DU R.T.F. VERS LE R.N.I.S.

Chapitre deuxième: Les commutateurs.

2. LES COMMUTATEURS.

2.1 Description générale T

2.1.1 Fonctionnalité

2.1.1.1 Introduction

2.1.1.2 Types de liaisons

2.1.1.3 Implantation des commutateurs
- A. Commutation urbaine
- B. Commutation interurbaine et internationale
- C. Commutation rurale

2.1.1.4 Impact du trafic

2.1.2 Caractéristiques

2.1.2.1 Réseau de connexion

2.1.2.2 Module de relation

2.1.2.3 Unité de commande
- A. Acquisition des événements
- B. Commande des équipements
- C. Gestion de la connexion
- D. Traduction
- E. Taxation
- F. Exploitation et maintenance

2.2 Composants

2.2.1 Réseau de connexion T

2.2.1.1 Introduction

2.2.1.2 Systèmes manuels

2.2.1.3 Systèmes automatiques
- A. Systèmes rotatifs
  - a.- système pas à pas
  - b.- système à enregistreur
- B. Systèmes crossbar
- C. Systèmes électroniques

a.- commutation spatiale
b.- commutation temporelle

2.2.1.4 AXE-10: le réseau de connexion numérique
A. Etablissement d'un appel local
a.- le sous-système d'abonnés
b.- le sous-système de commutation de groupe
b.1 module de commutation temporelle
b.2 module de commutation spatiale
B. Etablissement des autres types d'appel

2.2.1.5 ITT 1240: le réseau de connexion numérique
A. Principes et caractéristiques
a.- Topologie repliée
b.- Uniformité des modules de commutation
c.- Commmande à l'origine
d.- Performances et sécurité
B. Module de commutation numérique
C. Structure du RCN
a.- Première étape: configuration minimale
b.- Deuxième étape
c.- Troisième étape
d.- Quatrième étape
e.- Cinquième étape: configuration maximale
D. Etablissement d'un chemin dans le RCN
a.- Adresses-Réseau
b.- Algorithme de sélection
b.1 sélection de l'étage de réflexion
b.2 recherche libre
b.3 recherche dirigée
c.- Ordres

2.2.2 Organes de commande

2.2.2.1 Aspects T
A. Fonctions
a.- Fonctions de commutation
b.- Fonctions d'exploitation et de maintenance
B. Caractéristiques
a.- Contraintes
b.- Conséquences

2.2.2.2 Techniques
A. Application

A.1 Explorateur ou scanner T
A. composition de l'ensemble
1. Processeur
2. Explorateur
B. fonctions
C. types d'explorateurs
1. par sondages ou polling
1.a en mode synchrone et de façon continue
1.b en mode asynchrone ou synchrone arythmique
2. par interruptions

D. comparaison
1. sondage asynchrone
2. interruptions
3. conclusion
E. spécificité de l'exploration en commutation
1. fonctions et contraintes
2. principe et fonctionnement

A.2 Les Automates finis V
A.2.1 Définition
A.2.2 Les états internes d'un automate fini
A.2.3 Les fonctions de transition
A.2.4 Exemples d'automates
A.2.5 Application au commutateur

B. Système d'exploitation

B.1 Gestion du temps T
A. Introduction
B. Spécificité du caractère aléatoire
C. Horloge temps-réel et compteur d'intervalles

B.2 Gestion des processus V
B.2.1 Définition
B.2.2 La simultanéité
B.2.3 Le scheduling et le context switching
B.2.4 Gestion des processus dans le 1240
a. Fonction synchrone et asynchrone
b. Caractéristiques d'une FMM
c. Messages
d. Structure d'une FMM
e. Partie superviseur d'une FMM
f. Partie application d'une FMM

B.3 Gestion de la mémoire T
A. Introduction
B. Allocation dynamique de la mémoire
1. Réservation
2. Libération
3. Allocation de cadres ou pages
C. Allocation avec préemption

B.4 Communication et synchronisation entre processus E
A. Machines finies à messages
1. Rappel sur les automates
2. Automates en commutation
3. Signalisation
4. Contenu d'un message
B. Communication entre les processus
1. Combinaison automates et processus
2. Transmission de messages
3. Propriétés des messages et scheduling des tâches
4. Identification du processus et acheminement des messages
C. Communication entre modules de bas niveau

D. Communication dans 1240 et AXE
1. Types de blocs software
2. Supports de communication
(i) Architecture fonctionnelle et physique du software
(ii) Description des supports de communication
(iii) Méthodes d' acheminement
3. Déroulement d'une communication

B.5 Files d'attente E
A. Principes
B. Files d'attente et scheduling dans AXE10
C. Files d'attente et scheduling dans le 1240

C. Hardware

C.1 Les interruptions V
C.1.1 Principes généraux
a. Registre des interruptions
b. Interrupt table
c. Les états machine
d. Schéma général des interruptions
C.1.2 Gestion des interruptions en temps réel
a. Interrupt table
b. Traitement des appels à des modules de l'OSN
c. Traitement des interuptions provenant des device, clock, et erreur

C.2 Processeurs multiples E
a. Introduction
b. ITT 1240
1. Unités logicielles et processeurs physiques
2. Réseau de communication entre processeurs
3. Critique
c. AXE 10
1. Unités logicielles et processeurs physiques
2. Réseau de communication entre processeurs
3. Critique

D. Exemples d'interfonctionnement entre modules E

D.1 Interfonctionnement au niveau de l'application
a. Description générale du segment de procédure
b. Version 1240
b.1 Modules concernés
b.2 Sous-phases
c. Version AXE 10
c.1 Modules concernés
c.2 Sous-phases

D.2 Interfonctionnement entre application et OS
a. Situation envisagée pour l'exemple
b. Conception du schéma
c. Déroulement des événements

2.3 Architecture des commutateurs E

2.3.0 Introduction

2.3.1 Spécifications élémentaires d'un commutateur

2.3.2 Emplacement du système de contrôle

2.3.3 Principe de découpe architecturale

2.3.4 Les grandes fonctions

2.3.4.1 Contrôle des lignes d'abonnés

2.3.4.2 Contrôle des circuits de jonction

2.3.4.3 Connexion et commutation

2.3.4.4 Services hardware

2.3.4.5 Contrôle des communications

2.3.5 Configurations

Chapitre troisième: Le prototype

B. Notion de processus
C. Découpe du système en processus
D. Fonctionnement du système
E. Moniteur temps réel
E.1 moniteur temps réel
E.2 gestion des multiprocessus
F. Scheduler
F.1 Rôle du scheduler
F.2 Critères de sélection d'un processus

3.3.2.2 Gestion de la mémoire T
A. Introduction
B. Typologie des ressources mémoire
a. le bloc de message: BLC-MSG
b. le bloc de contexte: BLC-CTX
c. le bloc contrôleur de tâche: BLC-CTL-TCH
d. le bloc contrôleur de temporisation: BLC-CTL-TMP
C. Organisation de la mémoire libre
D. Primitives de traitement des BLC-MSG
E. Primitives de traitement des BLC-CTX
F. Primitives de traitement des BLC-CTL-TCH
G. Primitives de traitement des BLC-CTL-TMP
H. Procédures de traitement des files de bloc de mémoire libre

3.3.2.3 Gestion des messages E

3.3.2.4 Gérant du temps T
A. Introduction
B. Programmation de procédures périodiques
C. Contrôle de séquence et d'erreurs par temporisation
a.- typologie des demandes de temporisation
b.- création de temporisation
c.- gestion des temporisations
d.- expiration de temporisations
e.- annnulation de temporisation

3.3.3 Application

3.3.3.1 ILS (interface ligne/système) T
A. Scanner les lignes
B. Gérer les tables
C. Traiter les événements
D. Primitives ILS
E. Fonctions ILS
F. Procédures ILS

3.3.3.2 Colig V

3.3.3.3 Cocom V

3.3.3.4 Gestion de la taxation E
1. Module principal TAXAT
2. Procédures
3. Base de données
4. Contenu des messages

REMARQUE

Les lettres indiquent les responsabilités de telle personne sur telle partie du document.

- V : Van Gaver Rudy
- E : Evrard Marc
- T : Trigaux Benoît.

INTRODUCTION

Parmi les nombreuses branches de l'informatique moderne, l'étude des réseaux télématiques est certainement une voie d'avenir où beaucoup de recherches et d'améliorations restent à faire.

On assiste depuis plusieurs années à l'éclosion de réseaux à vocations diverses, transmission de données, télécommande, envoi de textes et d'images et bien d'autres encore. Quelques ancêtres comme le téléphone et le télégraphe tentent également de se moderniser pour répondre à l'évolution qualitative et quantitative des besoins actuels. De manière générale cependant, on constate un certain manque de coordination dans cette conjonction de mouvements.

Des organismes internationaux comme le C.C.I.T.T. essaient constamment d'établir des standards acceptés par le plus grand nombre, ce qui doit permettre une harmonisation des sytèmes, et de ce fait l'interconnexion et la collaboration entre réseaux.

Une norme particulièrement intéressante publiée par le C.C.I.T.T. à ce sujet est celle traitant du R.N.I.S. ou Réseau Numérique avec l'Intégration des Services. Dans sa phase finale, ce concept est celui d'un réseau unique et transparent, capable de transmettre n'importe quel type d'information, pourvu que celle-ci soit numérisée. Avec un tel réseau, il n'y aura plus de séparation entre réseaux à vocations différentes, mais un seul réseau combinant les services anciens comme le téléphone, et les services nouveaux comme la transmission de données.

Le premier chapitre de ce mémoire est consacré à cette évolution historique et technique des réseaux.

Dans le deuxième chapitre nous avons réduit notre champ d'investigation aux commutateurs numériques. Ceci comporte plusieurs apects:

1. Les commutateurs étant des systèmes en temps réel, nous avons dû envisagé les problèmes que cela soulevait, et les techniques utilisées pour y répondre.

2. La commutation elle-même se fait au moyen de la mise en connection de canaux numériques, et est contrôlée par la signalisation.

3. En temps que système informatique de contrôle de processus physique, les commutateurs ont des architectures très particulières, où le logiciel est étroitement lié au matériel.

Le troisième chapitre est une analyse complète d'un commutateur téléphonique numérique local écrit en langage C pour un micro-ordinateur basé sur un micro-processeur INTEL 8086. Toutes les notions théoriques vues au chapitre 2 sont reprises et appliquées.

"Si le commerce est le coeur d'une économie, on peut dire que les télécommunications sont pour une économie et une société leur véritable système nerveux."

Le président de la banque mondiale.

CHAPITRE 1

LES RESEAUX COMMUTES

1. LES RESEAUX COMMUTES.

1.1 Définition.

Nous désignons par réseau commuté l'ensemble des moyens mis-en-oeuvre pour permettre à des usagers distants d'échanger entre-eux des informations avec un délai aussi court que possible.

1.2 Types de réseau commuté.

Les types de réseau commuté sont classés en deux grandes familles.
D'une part, celle qui recouvre la transmission de la voix et d'autre part, celle qui recouvre la transmission des données.

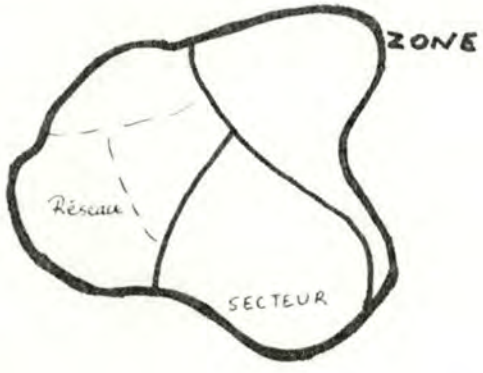
1.2.1 Première famille: transmission de la voix.

1.2.1.1 Le réseau téléphonique.

A. Architecture.

A.1 Division géographique.

La division géographique est hiérarchisée en zones, secteurs et réseaux.



La zone est caractérisée par le fait que tous les secteurs et réseaux passent par son centre pour établir la plupart des liaisons internes et toutes les communications extérieures à la zone.
Le secteur est une aire de taxation.
Le réseau est une aire de raccord ement de l'usager.

A.2 Organisation en étoile.

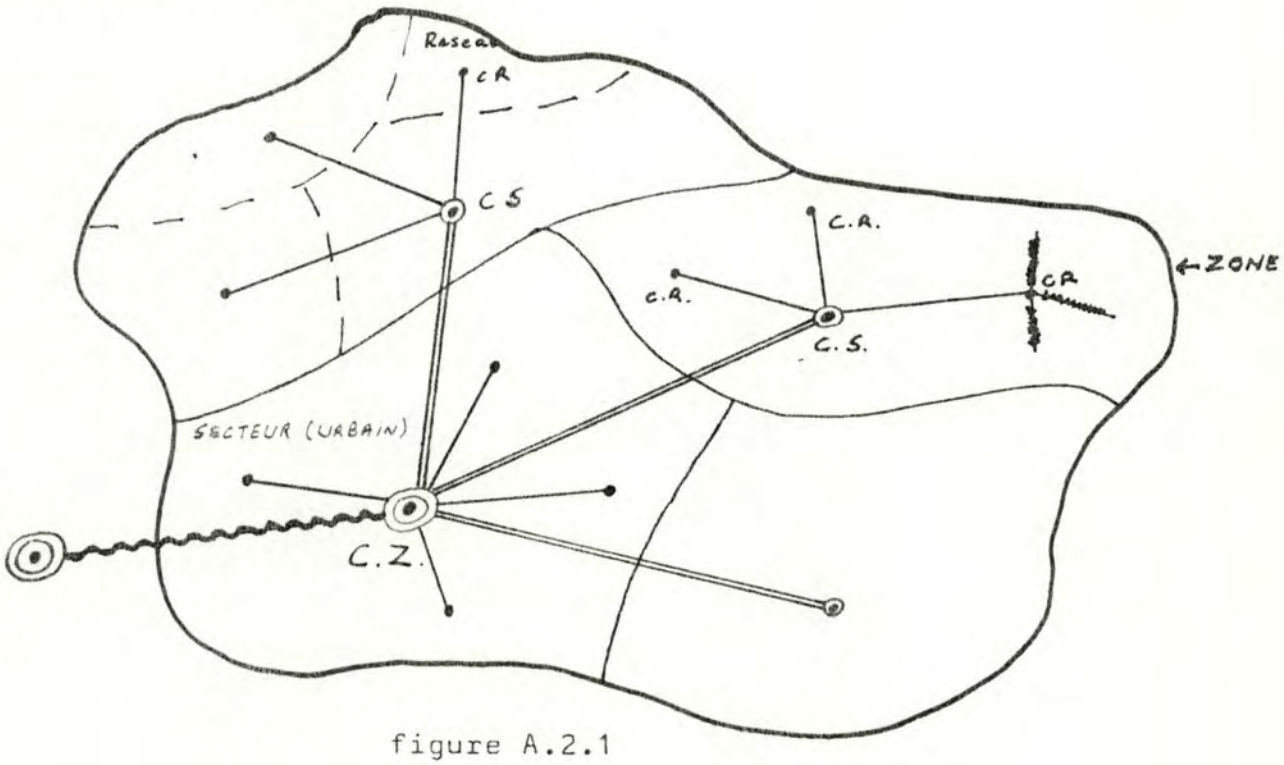


figure A.2.1

legende

- • : centre de raccordement.
- ◉ : centre de secteur.
- ◎ : centre de zone.
- : câbles locaux ou de raccordement.
- : jonctions urbaines.
- : câbles de jonctions.
- : câbles interurbains.

La figure (A.2.1) indique les principes de base pour l'organisation du réseau téléphonique. La cellule élémentaire étant la zone de distribution : dans chaque zone, le réseau téléphonique est étoilé autour d'un centre de zone. Des centres de secteur lui sont connectés. A ceux-ci, toujours en étoile, sont reliés les centres de raccordement.

B. Composants.

Matériellement, le réseau téléphonique est constitué

- des postes d'abonné qui assurent la conversion de la parole en signaux électriques et réciproquement.
- des commutateurs qui assurent la concentration du trafic des abonnés rattachés et l'aiguillage des communications.
- des supports de transmission.

B.1 Les commutateurs.

B.1.1 Types de commutateurs.

Il existe trois types de commutateur: traditionnel à processeur, numérique et temporel.

B.1.2 Commutateur traditionnel. (première génération)

Il est de nature électro-magnétique et mécanique à logique câblée.

Il est constitué de chercheurs en étages mettant en rapport l'abonné appelant avec un enregistreur qui lui transmet le ton d'invitation à envoyer les chiffres de l'appelé, reçoit les chiffres et transmet ceux-ci vers chacun des étages de sélection pour réaliser une liaison locale.

Dans le cas d'un appel régional ou interurbain il existe, en plus, une procédure de signalisation.

B.1.3 Commutateur à processeur. (seconde génération)

Ces commutateurs ont le même rôle que les commutateurs traditionnels. Cependant, ils constituent un premier pas vers la transmission numérique et l'intégration des organes.

En outre, il amène des avantages non-négligeables pour la téléphonie:

i. la réduction du matériel en passant de l'électro-magnétique à l'électronique.

ii. l'accroissement de la fiabilité des opérations.

iii. le développement des services fournis dû à la souplesse de la programmation par rapport au figeage des circuits câblés.

Les caractéristiques de cette commutation à processeur sont :

1. les liaisons sont réalisées à l'aide de matrices en grilles spatiales.
2. les circuits de commande sont électroniques et tendent de plus en plus vers l'intégration . Ils sont modulairement fonctionnels. La réduction de leur vitesse de fonctionnement permet d'en réduire le nombre.
3. le multiplexage de l'échantillonnage des circuits est possible. En d'autres termes, chaque organe est simultanément à la disposition de plusieurs appels.

B.1.4 Commutateur numérique et temporel (troisième génération)

Ce point sera détaillé dans le chapitre second de ce premier syllabus.

B.2 La transmission.

B.2.1 Les réseaux urbains.

Les lignes de rattachement d'abonné ou les circuits de jonction entre les commutateurs d'une même zone urbaine sont constitués le plus souvent par de simples paires de conducteurs métalliques. Sur le plan pratique, plusieurs pairs de conducteurs sont réunies entre-elles pour constituer un câble.

B.2.2 Le réseau interurbain.

A. Les câbles chargés.

La principale limitation du câble de type urbain est son affaiblissement inversément proportionnel à son diamètre.
Or pour des raisons économiques l'on a intérêt à utiliser des câbles de faible diamètre. Le câble

doit alors être chargé, c'est à dire que l'on amplifie le signal tous les X mètres .

B. Principe du multiplexage en fréquence.

Chaque signal téléphonique occupe une bande de fréquences nominalement fixée à (0-4000) HZ.
De nombreux supports filaires et radio-électriques permettent en fait de transmettre dans une bande beaucoup plus étendue (plusieurs mégahertz).
Il était donc tentant de mettre en oeuvre des systèmes utilisant le principe du multiplexage en fréquence : sur un même support physique, on transmet en parallèle plusieurs communications téléphoniques, chacune d'elles se voyant allouer à l'intérieur du spectre de fréquence transmis une plage de fréquence différente.

C. Application au multiplexage téléphonique.

On constitue d'abord le groupe primaire de base qui contient 12 voies dans la bande 60 à 108 KHZ.
Le groupe secondaire est constitué de 5 groupes primaires (312-552 KHZ).
Les groupes secondaires sont regroupés par paquets de 5 dans la bande 812-2044 KHZ pour constituer le groupe tertiaire de base. Enfin, on obtient le groupe quaternaire de base en regroupant 3 groupes tertiaires (8516-12.388 KHZ).

D. Supports utilisables par les multiplex analogiques.

D.1 Câble à paires symétriques.

- diamètre assez important.
- bande passante de 500 KHZ.
- la diaphonie est àretenir comme un désavantage à l'utilisation de ce support.

D.2 Câble coaxial.

- le plus utilisé en transmission à longue distance.

- bande passante de 6 MHZ à 60 MHZ

D.3 Faisceaux hertziens.

- emploi accru dans les dernières années
- bande passante de 4 GHZ et de 6 GHZ, cependant des systèmes à 7 GHZ et 11 GHZ sont en développement pour pallier à la saturation progressive des deux autres systèmes.

E. Transmission numérique et multiplexage temporel.

Outre la représentation analogique d'un signal téléphonique,il existe une autre façon de représenter-donc de transmettre et de commuter- sous la forme d'une suite d'éléments binaires.
On démontre ,en effet, qu'un signal à bande limitée peut être représenté par la suite de ses échantillons prélevés à une fréquence de 2 W au moins.
La voie téléphonique peut être reconstituée à partir d'échantillons prélevés tous les 1/2 W
$1/2\ W = 1/2x4x10^{3} = 125$ micro-secondes.
Comme l'illustre la figure (V.1) , l'amplitude de chaque échantillon est codée par un mot de 8 bits, ce qui permet la distinction entre 256 niveaux.
La suite des symboles binaires à transmettre a un débit de 64 Kbit/s. Le bruit de quantification est l'erreur que l'on commet en assimilant l'échantillon au niveau le plus proche parmi les 256 possibles.
En ce qui concerne la transmission,on multiplexe dans le temps les différentes voies (fig.V.2): entre deux échantillons successifs d'une même voie, on dispose de 125 micro-secondes pour transmettre les mots de 8 bits qui représentent les échantillons pour les autres voies.
Le multiplexage consiste donc à affecter,à l'intérieur d'une trame, un intervalle de temps à chaque signal unitaire; les différents échantillons entrelacés sont émis successivement, la voie i se présentant toutes les 125 micro-secondes.

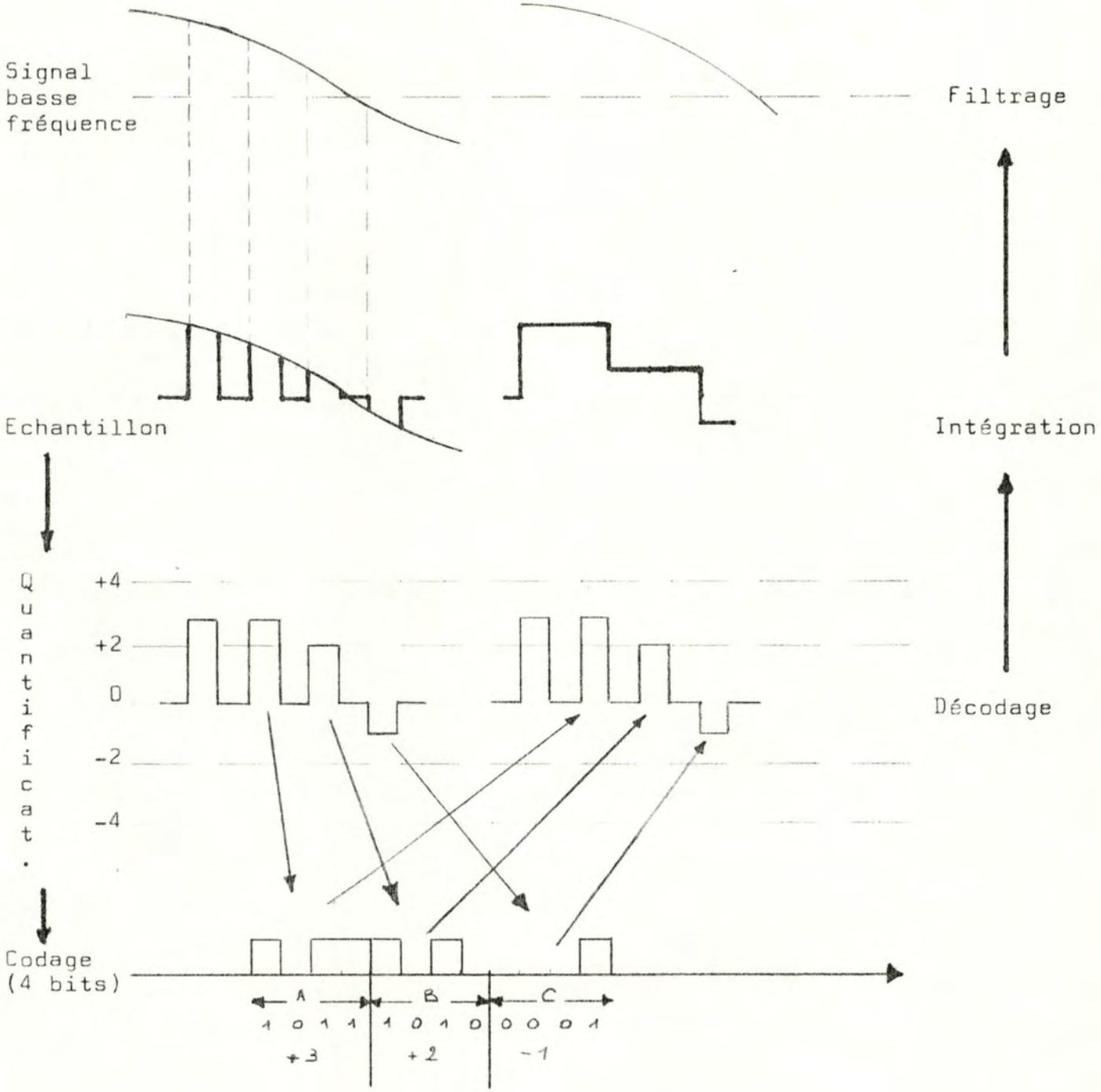


FIGURE V.1 : Principe de la modulation par impulsion et codage (MIC).

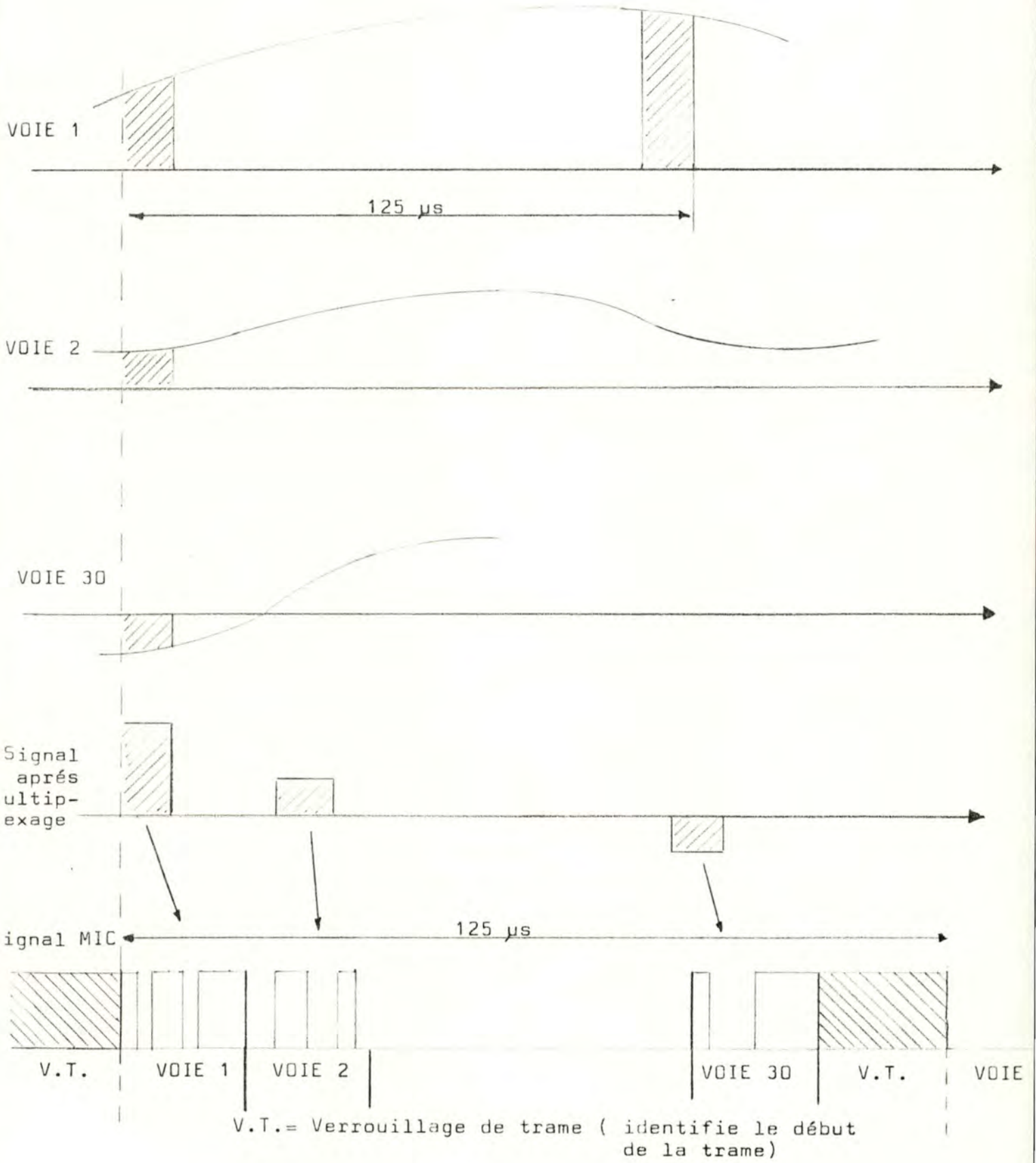


FIGURE V.2 : Principe du multipexage temporel MIC.

F. Hiérarchie et systèmes de transmission numérique.

Le multiplex primaire à 30 voies est regroupé par ensembles de 4 pour fournir un multiplex du second ordre de 120 voies à 8,448 Mbit/s. Le multiplex tertiaire (1920) comprend 16 signaux à 8 Mbit/s multiplexés à 140 Mbit/s.

1.2.2. Seconde famille : transmission des données.

1.2.2.1. Le réseau télex.

La télégraphie est une technique de transmission d'informations écrites par les moyens de télécommunications: le réseau télégraphique commuté,ou réseau télex, assure en effet la connexion temporaire entre deux téléimprimeurs qui sont des machines à écrire télécommandables.

A. Caractéristiques.

De même qu'en téléphonie, la partie du réseau la plus proche de l'abonné est constitué de paires métalliques. La vitesse de transmission est de 50 bauds. Le codage utilisé est le C.C.I.T.T. à 5 moments.
La procédure X 20 régit la commutation, les échanges et la signalisation.

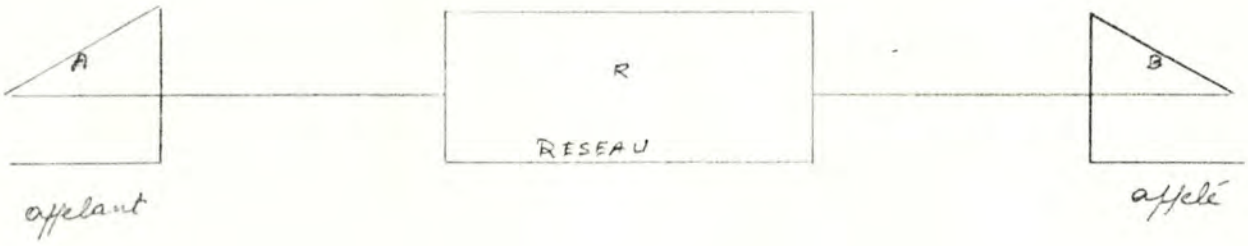
B. Types de commutateurs.

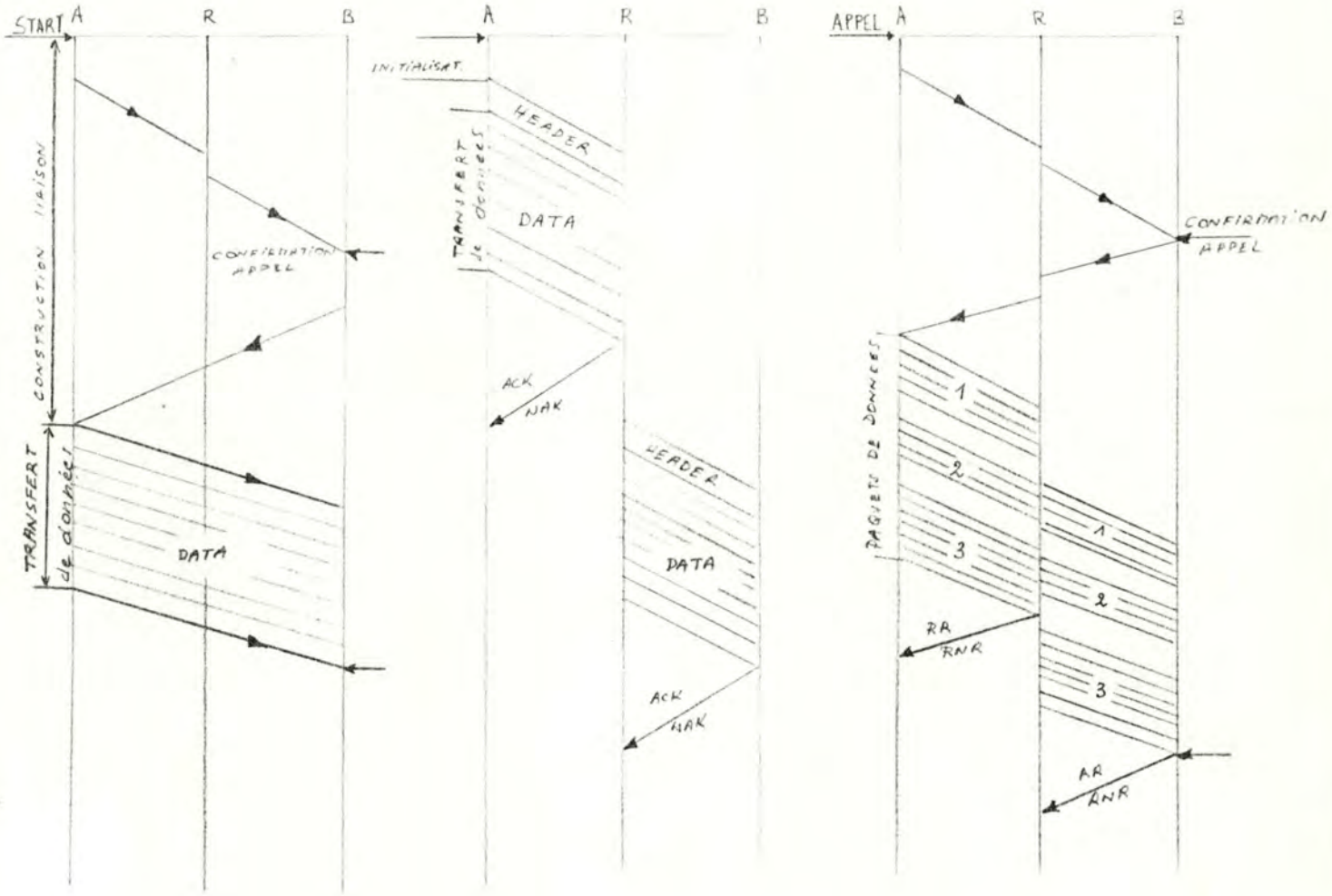
Ce réseau, à l'instar du réseau téléphonique, est formé de commutateurs d'architecture traditionnelle et à processeurs.
Les messages de données d'un abonné à un autre sont de nature alphanumérique, ils sont transmis d'un télétype à un autre et sont commutés vers l'adresse souhaitée en traversant un ou plusieurs commutateurs. Le réseau télex est international.
La tendance prochaine sera de substituer les commutateurs traditionnels par des commutateurs à architecture à processeurs.

1.2.2.2. Les réseaux de commutation de circuits, de messages et de paquets.

Pour ces trois types de réseaux commutés nous avons choisi une présentation comparative.

A. Leur technique de commutation.





CIRCUIT SWITCHING MESSAGE SWITCHING PACKET SWITCHING

B. Caractéristiques.

Ces trois types de réseau sont caractérisés par des procédures et/ou protocoles.

Réseau	Procédure	Protocole
R.C. Circuits	X21 (20)	-
R.C. Messages	-	BSC , HDLC
R.C. Paquets	X3, X28, X29	X25

L'architecture de ces trois réseaux est le "maillage".
Une caractéristique fondamentale du R.C.C. est la réservation, pour la durée de la communication, d'un chemin physique entre les deux abonnés.
Ceci n'est pas d'application dans les deux autres cas : pour le R.C.P., il existe un circuit VIRTUEL entre les abonnés; pour le R.C.M., le message est remis au réseau qui se chargera de l'amener à l'abonné appelé.
On peut aussi les caractériser sur base de certains critères.

Critère \ Services		Circuits	Paquets	Messages
Délai	Initial	qqs sec.	0.1 à qqs*	qqs s.* à
	Communication établie	± 0	0.1 à qqs s*	qqs h.*
Communication bidirectionnelle		oui	oui	
conversions possibles - codes - vitesses			oui	oui
Asservissement à la vitesse destinataire			oui	
Emission vers destinataire indisponible			envisa-geable	oui
Rendement élevé des lignes		selon nature du trafic		
Mémoire nécessaire		0	faible, rapide	importante, lente

2. EVOLUTION DU R.T.F. VERS LE R.N.I.S.

2.1. Introduction.

C'est dans le courant des années 70, que furent introduits séparément la transmission et la commutation numérique. Au cours de la présente décennie, elles ont fusionné pour donner naissance au réseau numérique intégré (R.N.I.) dans lequel les connexions établies par les commutateurs numériques seront utilisées pour la transmission de signaux numériques relatifs à un seul service, tel que la téléphonie.
Dans le même temps, de nombreaux réseaux privés, pour données, ont été réalisés au moyen de circuits loués pour servir les grandes entreprises.
Plus récemment, on a vu la création du réseau à commutation de circuits et du réseau à commutation de paquets, cette fois pour servir toutes les entreprises.
Prochainement, l'expansion des services de données touchera les entreprises individuelles et les abonnés résidentiels.
Cette expansion peut être réalisée de deux manières.
Premièrement, les réseaux de données peuvent être développés et assurer une large couverture comme le RTF
Secondement, les services téléphoniques et non téléphoniques peuvent être assurés par un réseau unique qui serait dérivé du réseau téléphonique existant.
Cette seconde solution est appelée le Réseau Numérique avec Intégration des Services.(voir figure V.3)

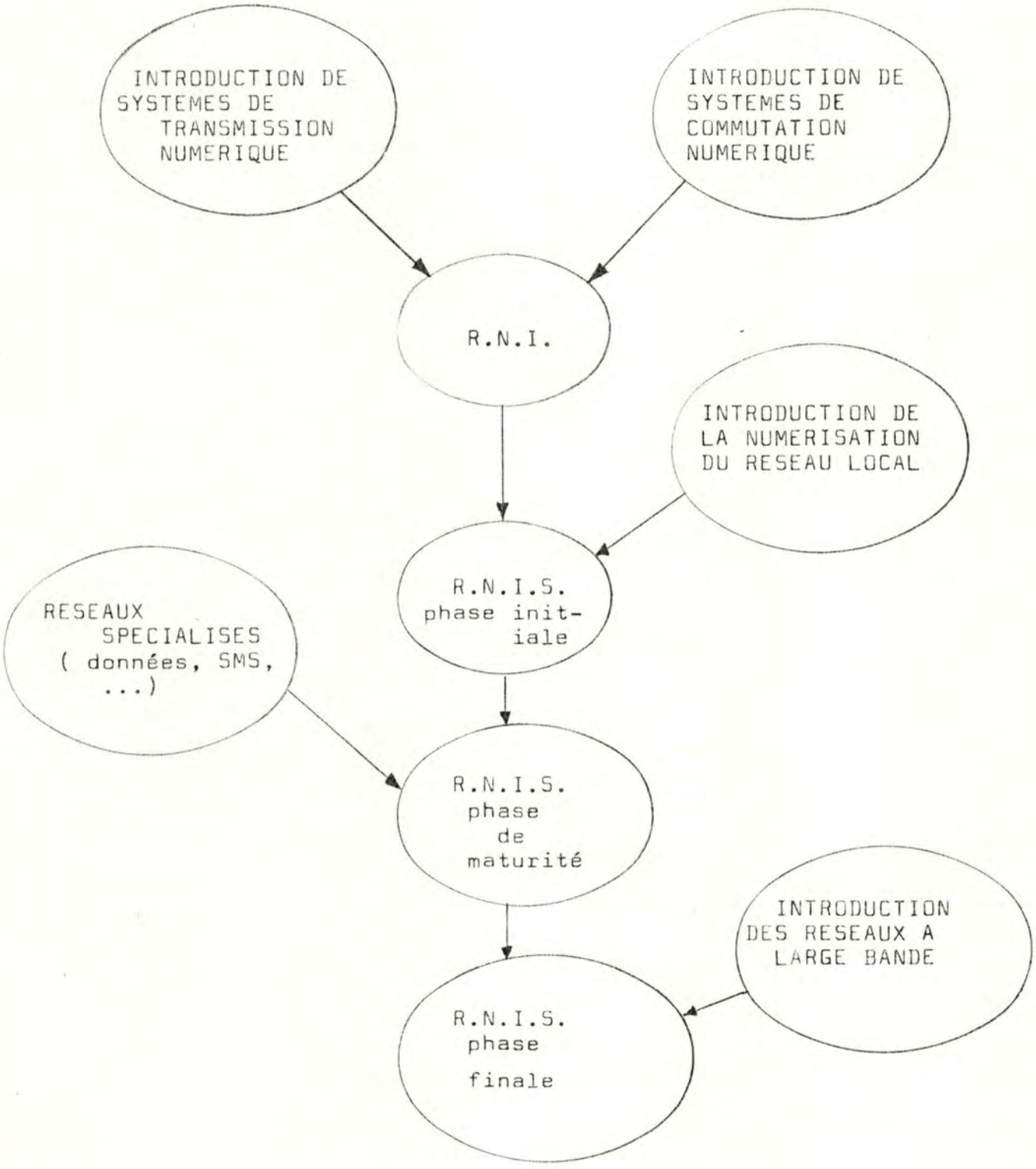


FIGURE V.3 : Scénario d'évolution du R.N.I. vers le R.N.I.S.

Cependant les performances du réseau analogique tendent à restreindre l'utilisation de ces nouveaux services.(tableau V.T.1)
Le R.N.I., par contre, permet une utilisation tout à fait satisfaisante de ces services grâce à son débit de 64 Kbit/s.(tableau V.T.2)
En outre, les communications à grande vitesse entre ordinateurs et certaines communications d'images (visiophone et visioconférence) nécessïtant une bande passante beaucoup plus grande devront être mises en oeuvre dans les réseaux à large bande.

Tableau V.T.1 - Limitation de vitesse d'un réseau analogique.

Service	Débit binaire du réseau analogique (Kbit/s)	Temps de transmission pour 1 page (secondes)	Temps de transmission souhaitable (secondes)	Débit nécessaire (Kbit/s)
Videotex	1,2	10	1	9,6
Télétex	2,4	10	2	9,6
Télécopie	2,4	120	5	64

Tableau V.T.2 - Services candidats à l'intégration

Bande passante	Service			
	Téléphonie	Données	Texte	Image
Parole numérique (64 Kbit/s)	Téléphone	Données à commutation de circuits	Télex	
		Données à commutation par paquets		
	Liaisons spécialisées	Liaisons spécialisées	Liaisons spécialisées	
	Recherche d'informations (par analyse et synthèse de la parole)	Télémesures	Videotex	
		Transferts financiers	Télécopie	
		Recherche d'informations	Recherche d'informations	Recherche d'informations
		Boîte aux lettres	Boîte aux lettres	Surveillance
		Courrier électronique	Courrier électronique	
		Alarmes		
Large bande (64 Kbit/s)	Musique	Communications à grande vitesse		Téléconférence Visiophone Télédistribution

2.2. Le réseau numérique intégré.

L'évolution vers le R.N.I.S. trouve sa première étape dans la réalisation du réseau numérique intégré.
La transmission et la commutation numériques sont intégrés, la commande s'effectue par programme enregistré. Le R.N.I. dispose d'une signalisation sur voie commune et d'une transmission à 64 Kbit/s de "central local origine" à "central local terminal".
A remarquer que la transmission entre abonné et central local est toujours analogique.(figure V.4)
L'étape suivante vers le R.N.I.S. est la possibilité d'utilisation directe des 64 Kbit/s par l'usager.
La ligne de raccordement sera, par conséquent, numérique. La signalisation locale sera plus évoluée et on pourra procéder à l'intégration des services dans le R.N.I.(figure V.5)

2.3. Evolution des services.

2.3.1. Les nouveaux services.

Des services de données existent déjà, on y trouve les réservations, opérations financières, ou encore la consultation de banques de données.
De nouveaux services sont déjà prévus.
Le service de télécopie qui consiste en la transmission et la reproduction, par un terminal éloigné, de toutes sortes de graphismes,manuscrits ou imprimés.
Le service de télétex qui permet aux terminaux d'abonnés d'échanger des correspondances de façon entièrement automatique. Ces terminaux offrent toutes les facilités de manipulation de la correspondance : mise en page,impression...
Le service de videotex permet la recherche d'informations par un dialogue avec une banque de données en utilisant des récepteurs de télévision ordinaires ou des terminaux spécialisés.

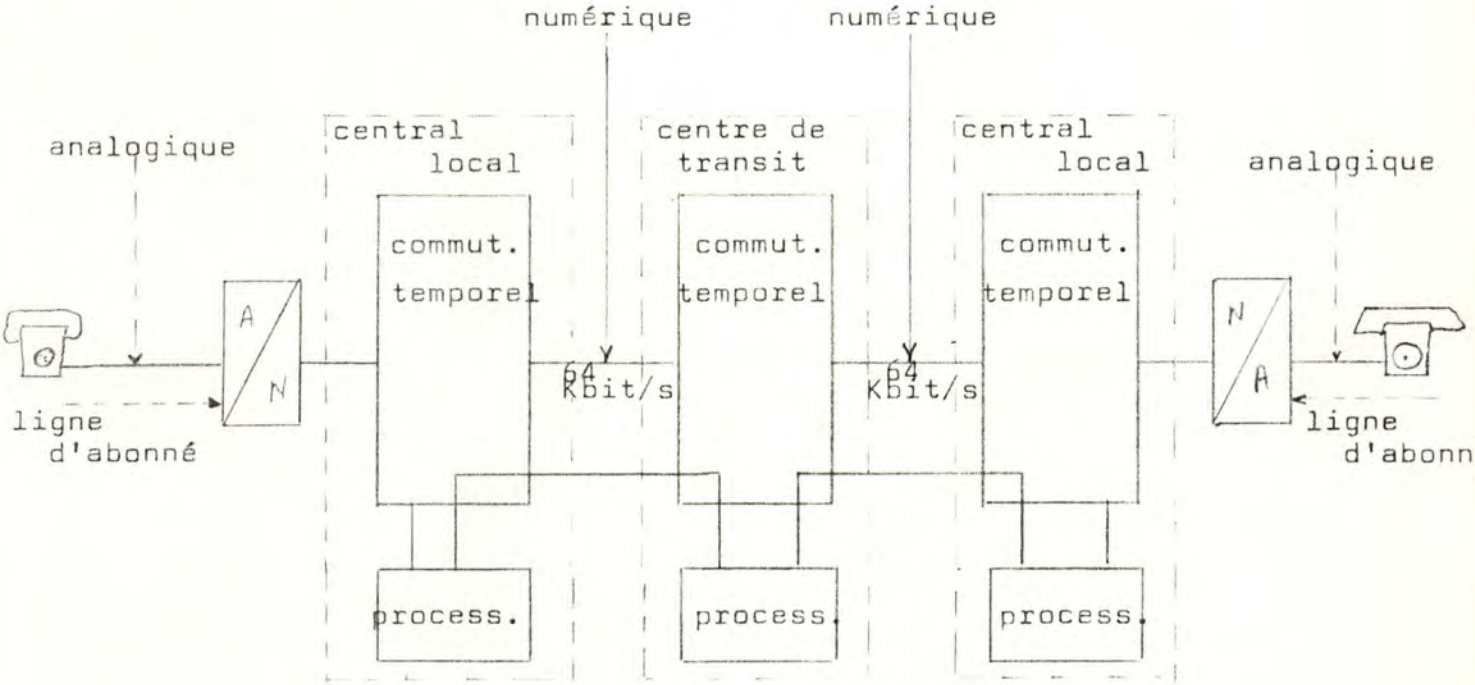


FIGURE V.4 : Schéma du R.N.I.

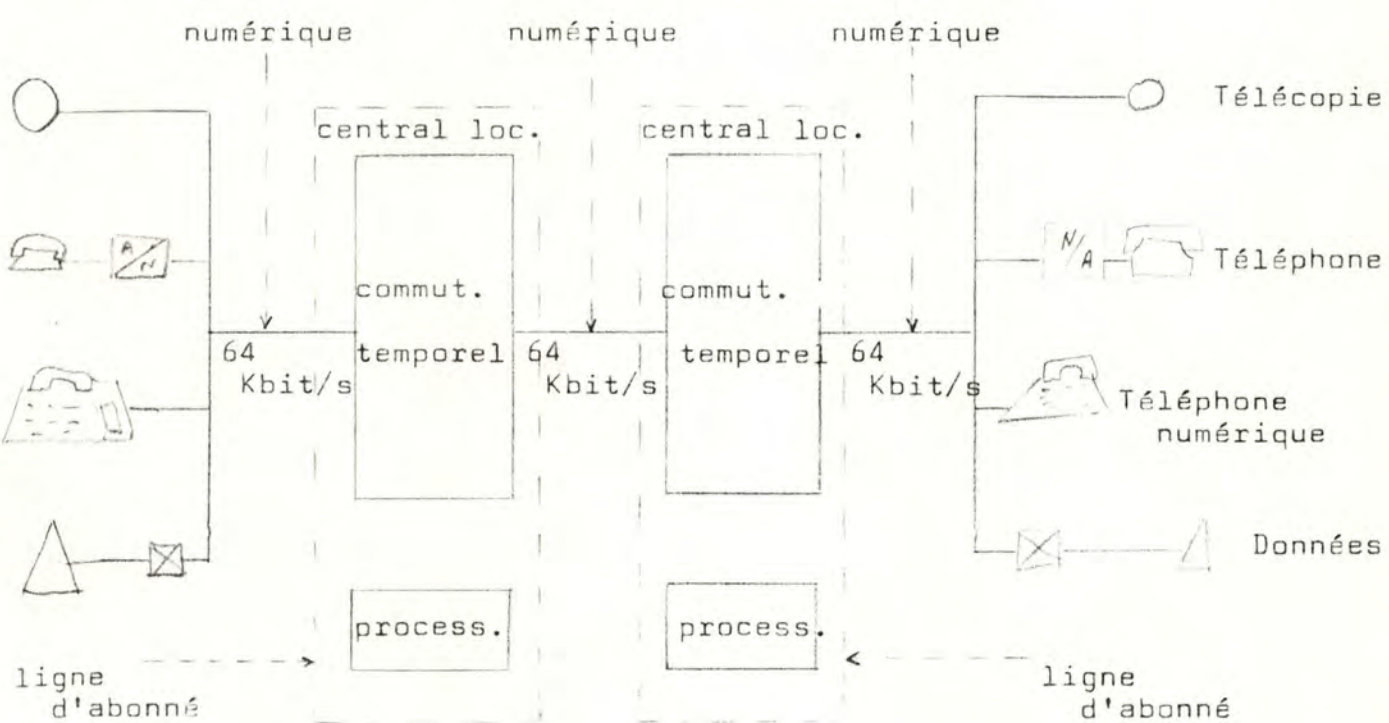


FIGURE V.5 : Schéma du R.N.I.S.

2.3.2. Caractéristiques des réseaux multi-services.

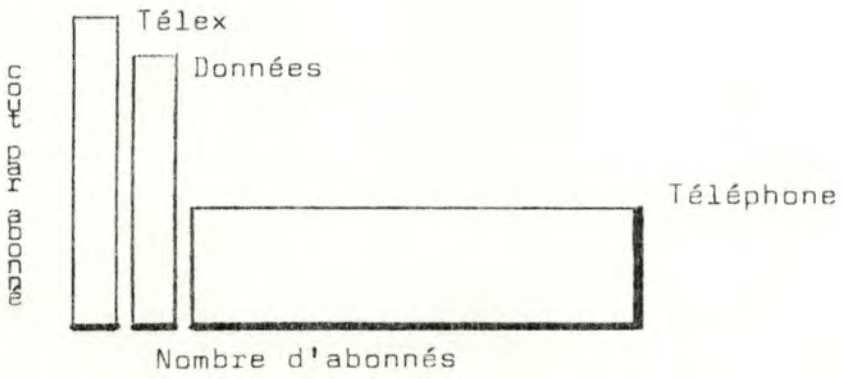
1. Gestion simplifiée pour l'administration.
   Il va de soi que la gestion d'un seul réseau est plus simple que la gestion d'une pluralité de réseaux aux caractéristiques différentes.
2. Pour l'abonné, surtout l'abonné résidentiel, il y aura des avantages de coût et d'usage puisque une seule ligne d'accès sera nécessaire au lieu de lignes séparées pour le téléphone, le télex et les données. En outre, l'intérêt d'un service croît avec le nombre de ses abonnés et le réseau R.N.I.S. offre le maximum de connexités pour tous les abonnés.
3. Les investissements seront mieux utilisés. (figure V.6)

   La figure V.6a montre les investissements nécessités par un réseau téléphonique sous la forme d'une surface rectangulaire T déterminée par le nombre d'abonnés téléphoniques et le coût par abonné. Les réseaux de données nécessitent des investissements plus faibles D_1 , D_2 parce qu'ils désservent seulement un petit nombre d'abonnés, à un coût supérieur par ligne.

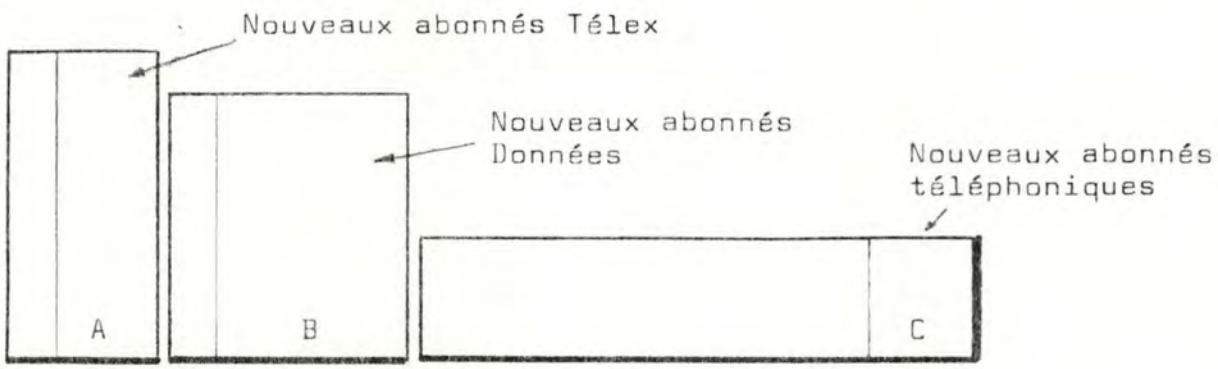
   La figure V.6b montre la situation lorsqu'il y a un grand nombre d'abonnés aux services de données; les nouveaux investissements dans les réseaux de données, indiqués par les rectangles A et B, deviennent comparables aux investissements dans le réseau téléphonique.

   La figure V.6c représente le cas du R.N.I.S. : la croissance des réseaux de données est arrêtée et les nouveaux abonnés, non téléphoniques, sont connectés au R.N.I.S. (rectangle A').
   On ne peut affirmer à priori qu'il n'y aura pas de coût supplémentaire B' pour les abonnés téléphoniques ordinaires.

a. Investissements actuels dans le réseau



b. Investissements futurs avec réseaux séparés



c. Investissements futurs avec réseau intégré

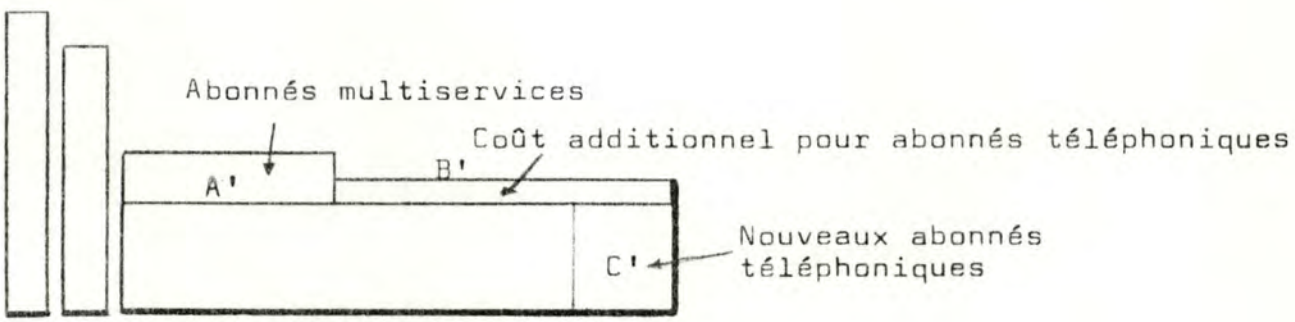


FIGURE V.6

Ce coût doit être le plus faible possible pour accroître le succès économique du concept de R.N.I.S.
Les investissements dans le R.N.I.S. (A'+B'+C') sont beaucoup plus faibles que dans des réseaux séparés (A+B+C)

2.4. Evolution du réseau.

La réalisation du R.N.I.S. passe nécessairement par la réalisation du réseau numérique intégré. Ensuite il évoluera à partir de celui-ci pendant une période de transition qui durera une ou deux décennies.
Les nouveaux services doivent donc être compatibles avec les communications à 64 Kbit/s.
Durant la période de transition, l'interfonctionnement entre les services amenés par d'autres réseaux doit être prévu. C'est la raison pour laquelle les concepts de catégories de fonctions de réseau et de degré d'intégration ont été introduits.
Deux catégories de fonctions de réseau sont envisagées:

- les fonctions de base relatives à des communications commutées en mode circuit à 64 Kbit/s qui seront offertes par tous les centres locaux; cette catégorie permet le service de téléphonie et éventuellement des services de données à commutation de circuit.
- les fonctions additionnelles relatives aux services qui nécessitent des ressources qui seront disponibles uniquement en certains noeuds du réseau ou dans des sous-réseaux spécialisés.

Trois étapes principales sont envisagées dans l'évolution du R.N.I. vers le R.N.I.S.

- Intégration minimale.
  La procédure normale est utilisée pour établir un appel téléphonique direct mais une connexion immédiate est utilisée dans le central local pour l'établissement d'un appel de données.

Par cette connexion les terminaux de données ont accès aux fonctions du réseau qui contrôlent l'établissement des appels de données en accord avec les Avis existant du C.C.I.T.T. (par exemple: X21, X25)

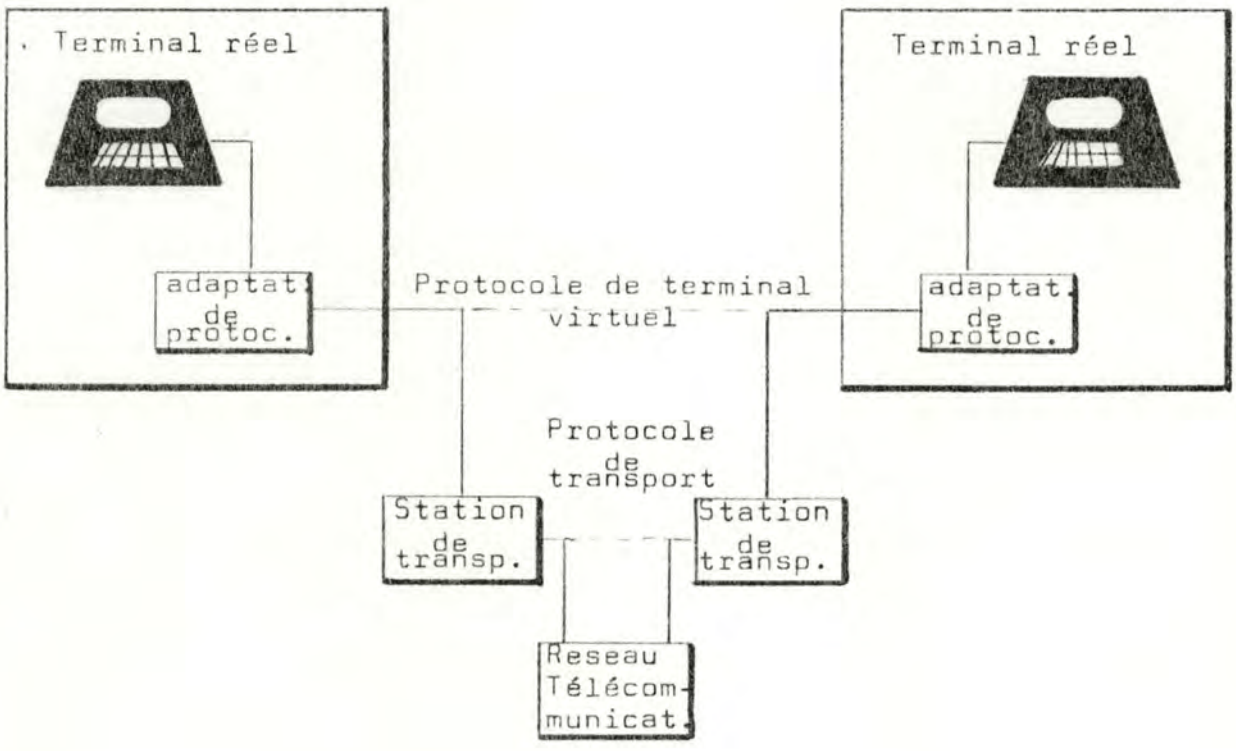
- Intégration moyenne.
  Dans ce cas, un protocole commun est utilisé pour établir un chemin servant à la fois à la téléphonie et aux données. Ce protocole contient des indicateurs de services qui permettent au central de traiter les fonctions de commande d'appel selon les nécessités d'un service donné et de diriger cet appel vers le réseau spécialisé correspondant.

- Intégration totale.
  Un protocole commun et une procédure commune de commande des appels sont utilisés pour établir l'appel et commander les services supplémentaires qui ont un objet commun. Dans l'idéal, les plans de numérotage et d'acheminement seraient communs. Seules les facilités qui sont applicables à un service spécifique seront traitées séparément.

Lorsque l'intégration totale sera réalisée, les abonnés pourront utiliser toutes sortes de terminaux spécialisés ou multi-fonctions connectés au central local par une seule ligne d'abonnés numérique.

2.5. Les terminaux.

Les caractéristiques hétérogènes des terminaux réels obligent la définition du concept de terminal virtuel qui est un modèle définissant un terminal abstrait en termes de fonctions logiques qui peuvent être réalisées différemment sur différents terminaux réels; par conséquent un terminal virtuel est un terminal réel muni d'un adaptateur de procédures. Les informations sont échangées entre terminaux virtuels en utilisant une procédure de terminal virtuel qui assure par conséquent la compatibilité entre les terminaux réels.



Deux types de terminaux sont intéressant pour l'évolution du réseau: le poste téléphonique numérique et le terminal multi-fonctions. L'évolution technologique est telle que les terminaux d'abonnés disposeront de plus en plus de puissance de traitement en vue de satisfaire à tous les nouveaux services.

Tableau V.T3 - Fonctions possibles des terminaux multi-services professionnels et résidentiels

Terminaux professionnels :

- Machines à dicter qui affichent les messages sur un écran
- Distribution de courrier électronique avec la possibilité de voir les messages sur un écran et d'imprimer seulement les plus importants
- Archives électroniques
- Systèmes d'entrée de données (claviers, analyseur d'images)
- Terminaux informatiques

Terminaux résidentiels :

- Communication de personne à personne (téléphonie)
- Distraction (jeux video, service, videotex)
- Publication : impression chez l'abonné de journaux et de magazines
- Système de gestion et de commande des appareils ménagers, ordinateur individuel pour le budget familial et les calculs d'impôts
- Enseignement : terminal de calculateur combiné avec un système d'affichage
- Correspondance : le service videotex avec boîte à lettres de messages pouvant remplacer la correspondance écrite utilise un clavier et des possibilités graphiques (télécopie ou terminal de télécriture)
- Système de transactions permettant les réservations et les achats ainsi que les mouvements financiers correspondants.

2.6. La transmission.

L'utilisation partagée de liaisons de transmission par plus d'un service, en utilisant le multiplexage temporel, est réalisée en premier dans l'évolution des R.N.I.

Les abonnés connectés au R.N.I.S. peuvent disposer des types suivants d'accès:

- un accès réservé à un service particulier
- un accès alternatif à deux ou plusieurs services
- un accès simultané à plusieurs services indépendants.

Pour offrir tous ces services, une liaison numérique allant jusque chez l'abonné sera nécessaire.

Trois techniques principales existent pour la transmission numérique entre le central et l'abonné sur une seule paire:

- transmission ping-pong ou par salves qui consiste en une séparation temporelle de chaque direction de transmission entre le central et l'abonné.
- séparation en fréquence
- suppression d'écho.

EN CONCLUSION :

L'évolution actuelle des caractéristiques des commutateurs téléphoniques associée à l'évolution de leurs lignes de transmission devrait déboucher vers la création d'un réseau polyvalent valable pour la téléphonie et les données.

On comprend, dès lors, l'intérêt du regroupement de ces services dans un seul réseau, que ce soit pour des arguments économiques ou pour des avantages plus qualitatifs.

Cependant, il n'est pas impossible que l'on tende vers le pluralisme des réseaux.

CHAPITRE 2

LES COMMUTATEURS

2 COMMUTATEURS

2.1 DESCRIPTION GENERALE

2.1.1 Fonctionnalité

2.1.1.1 Introduction

Les commutateurs, comme nous avons pu le voir dans le chapitre précédent, font partie intégrante de tout réseau de télécommunication.

Imaginons un instant un réseau de n abonnés démuni de tout système de commutation. Pour interconnecter nos n abonnés deux à deux, il nous faudrait établir pas moins de n(n-1)/2 liaisons directes. A partir du moment où n prend une dimension réaliste, l'absence de commutateur ne l'est plus du tout. Celui-Ci suppose un nombre de liaisons tel qu'il suffit de raccorder chaque abonné au commutateur, c'est-à-dire n liaisons dans notre exemple.

Le commutateur a donc pour rôle d'assurer la concentration du trafic des abonnés rattachés et d'aiguiller des communications.

Plus précisément, l'ensemble des commutateurs dans le réseau aura comme fonction d'établir à l'aide des informations fournies par l'abonné demandeur une connexion temporaire entre ce demandeur et le demandé désigné par la numérotation.

On peut remarquer que certaines lignes peuvent être spécialisées pour l'écoulement du trafic arrivée ou départ, une ligne ordinaire étant dite mixte. Il est de plus possible de réaliser un faisceau de lignes groupées pouvant toutes être atteintes sous le même numéro d'appel, l'aiguillage étant effectué automatiquement vers une des lignes disponibles.

2.1.1.2 Types de Liaisons

Les types de liaisons temporaires assurés par un commutateur sont les suivants :

- liaison entre deux lignes d'abonné raccordées au même commutateur. On parlera alors de communication locale ;

- liaison entre une ligne d'abonné et une jonction vers un autre commutateur. Si le choix de la jonction peut dépendre du numéro d'appel formé par l'abonné, le commutateur est dit "à autonomie d'acheminement".

- liaison entre une jonction provenant d'un autre commutateur et une ligne d'abonné ;

- liaison entre deux jonctions provenant de deux commutateurs distincts (figure T1).

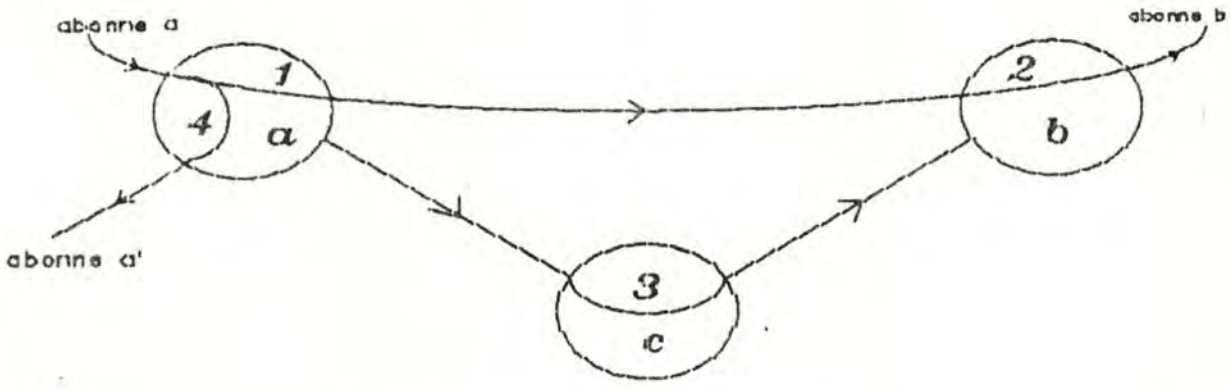


figure T1 : types de liaison

1) appel sortant de a
2) appel entrant dans b
3) appel de tranzit dans a
4) appel local dans a

Cette distinction est basée sur la différence de densité de trafic que supportent une ligne d'abonné (1/10 d'heure chargée) et une jonction entre commutateurs (7/10 d'heure chargée).

Des différents types de liaisons que nous venons de voir, découlent les définitions ci-après :

- un commutateur universel assure les quatre types de liaison ;

- un commutateur de rattachement d'abonnés assure les trois premiers types de liaison c'est-à-dire ceux où au moins une ligne d'abonné est impliquée ;

- un commutateur de transit assure uniquement la liaison de commutateur à commutateur.

A présent que nous disposons de commutateurs dont les caractéristiques sont connues, restent les problèmes de leur utilisation concrète au coeur de zones d'abonnés plus ou moins denses et dont le trafic répond à différents critères.

2.1.1.3 Implantation des Commutateurs

A) Commutation Urbaine

A l'origine, la population des abonnés urbains se caractérisait par un nombre relativement faible mais très concentré d'abonnés, ce qui permettait l'utilisation d'un seul commutateur. L'expansion des villes et l'engouement pour l'utilisation de certains réseaux à amener à devoir utiliser plusieurs commutateurs. Leur nombre croissant, ceux-ci ne furent pas reliés deux à deux mais bien connectés à un commutateur de transit. Ceci donnait une configuration en étoile (figure T2).

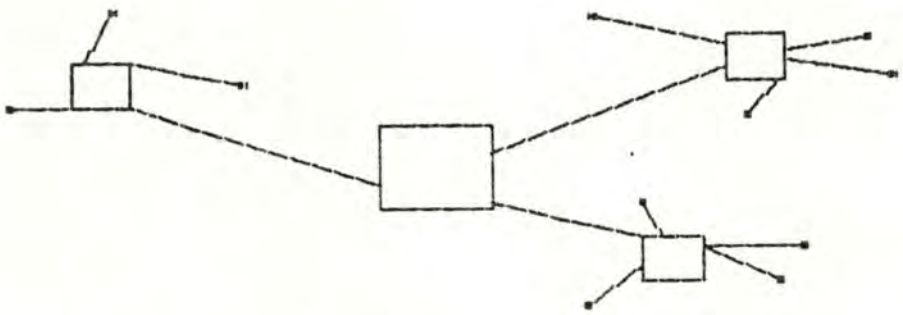


figure T2 : reseau en etoile

B) Commutation Interurbaine et Internationale

Pour mettre en relation différentes villes, il suffit de relier deux à deux les commutateurs de transit que l'on trouve dans les centres urbains. La configuration du réseau devient maillée (figure T3).

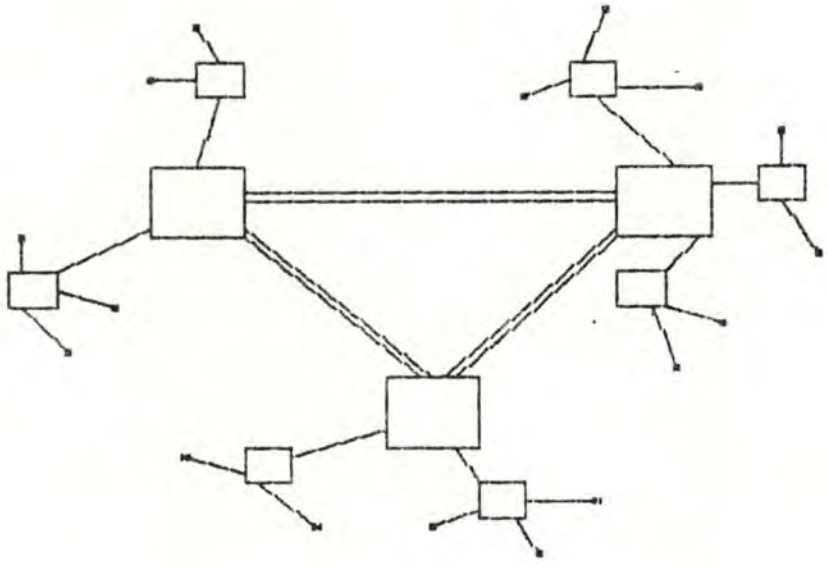


figure T3 : reseau maille

Pour les communications internationales à longues distances est prévu un niveau supérieur de commutation regroupant toutes les communications d'un pays dans un centre de transit international vers un autre centre de transit international.

C) Commutation Rurale

Le problème de la mise en relation de centres comprenant un faible nombre d'abonnés, relativement concentrés et éloignés de tout autre centre d'importance, est d'ordre plus économique - voire sociologique- que technique. En général, seront utilisés soit des petits commutateurs éloignés assurant les communications locales et délégant ses autres fonctions à un centre plus important auquel il est relié, soit un simple concentrateur reliant les abonnés distant au moyen d'une seule ligne au commutateur principal (figure T4).

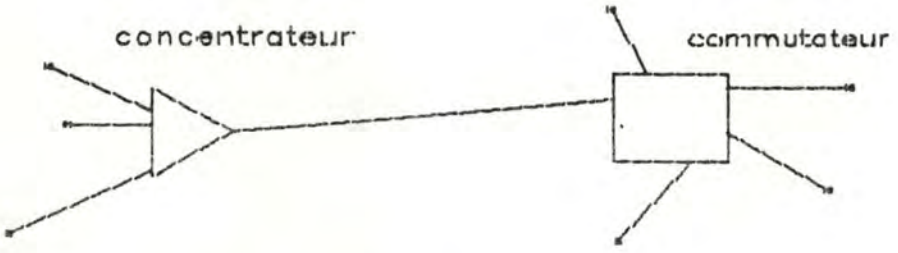


figure T.4 : reseau avec concentrateur

2.1.1.4 Impact du Trafic

Les performances d'un commutateur influent directement sur la qualité du service que l'on veut garantir. Or, ces performances pourront être ajustées en fonction du trafic en faisant varier deux paramètres fondamentaux déterminant le dimensionnement d'un commutateur : le nombre de lignes raccordées et le nombre maximum de communications admises simultanément.

La mesure du trafic quant à elle, sera basée sur deux principes régissant les liaisons temporaires sur un réseau commuté :

- les appels apparaissent à des instants quelconques et indépendants les uns des autres ;

- les communications ont des durées variables dont la durée moyenne est connue.

2.1.2 Caractéristiques

Les fonctions spécifiées au paragraphe précédent seront réalisées par les modules fonctionnels qui suivent :

2.1.2.1 Réseau de Connexion

Une liaison physique n'est possible que via la transmission de courant porteur d'information au sein du commutateur.

Connecter une voie entrante à une voie sortante de telle façon que les signaux portés par la première se retrouvent identiquement sur la seconde est réalisé par des équipements appelés globalement réseau de connexion.

2.1.2.2 Module de Relation

Un commutateur agit en réponse à des demandes faites à distance. L'établissement des communications nécessite donc des échanges d'informations pour lesquelles le commutateur doit disposer de fonctions de dialogue avec l'extérieur : ce sont les fonctions de relation.

L'ensemble des procédures qui régissent ces échanges est appelé signalisation.

2.1.2.3 Unité de Commande

Celle-ci traite des informations de signalisation de manière à assurer les différentes opérations de commutation. Ces traitements peuvent être regroupés en grandes fonctions.

A) Acquisition des Evénements

Les différents événements sont détectés par l'intermédiaire d'un explorateur activé à une cadence dépendant de la durée des événements à détecter.

B) Commande des Equipements

L'UC peut positionner dans différents états les équipements supportant les ressources du système.

C) Gestion de la Connexion

L'UC va déterminer et établir pour un appel un itinéraire dans le réseau de connexion.

D) Traduction

Le numéro de l'abonné demandé ne peut être utilisé directement pour commander les opérations de commutation et doit, pour ce faire, être préalablement traduit.

E) Taxation

Le service rendu par le réseau de télécommunication doit être facturé aux abonnés selon différents principes. Les commutateurs doivent donc permettre l'élaboration de données nécessaires à la facturation des communications qu'ils établissent.

F) Exploitation et Maintenance

Le commutateur doit assurer les deux impératifs suivant :

- gérer ses équipements en fonction de la demande et de l'évolution du trafic et du réseau ;

- maintenir la qualité du service offert à un niveau satisfaisant (figure T5).

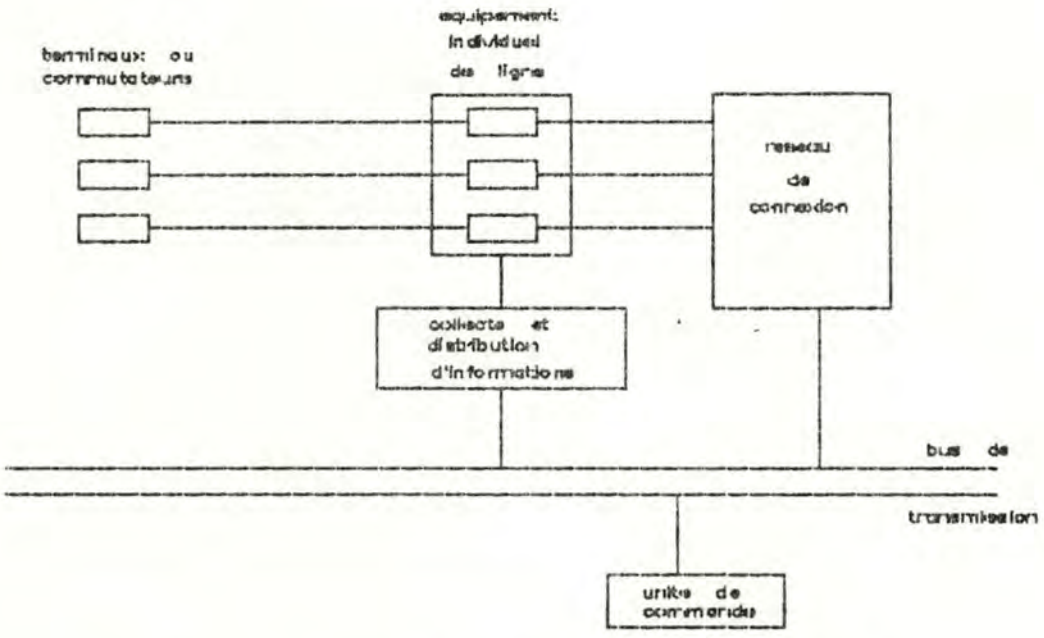


figure T5 : blocs fonctionnels d'un commutateur

Les fonctions de connexion et de commande seront développées dans les paragraphes suivant tandis que la fonction de relation fera l'objet de travaux annexes.

Alors qu'en commutation électro-mécanique classique les fonctions de commande étaient réalisées en logique cablée, elles sont actuellement centralisées dans un ordinateur sous forme de programme enregistré dans une mémoire.

En effet, ces fonctions sont d'analyser des événements extérieurs émanant d'une multitude de sources indépendantes, de les interpréter comme demandes de communication déterminées et d'en déduire les actions à entreprendre. Tout ceci peut être décrit au moyen d'une succession de choix élementaires dans un programme à déroulement conditionnel.

Avec l'apparition d'ordinateurs dotés d'une très grande puissance de traitement, la centralisation à outrance des fonctions de commande semble favorisée. Deux difficultés majeures tempèrent cette tendance :

- la dimension du logiciel rendrait celui-ci difficile à maîtriser ;

- en cas de défaillance de l'ordinateur unique, la panne serait complète.

Ces arguments donnèrent naissance à trois types de structure de commutateur relatifs à l'importance accordée à chacun des arguments.

a.- Organisation Dispersée

Les étages de sélection de lignes peuvent être placés à distance et constituer des centres satellites, le centre principal gardant la commande centralisée.

b.- Structure à deux Niveaux

Les centres satellites (locaux) sont munis d'organes de commande spécialisés d'une puissance limitée aux fonctions d'établissement des appels.

c.- Commande Décentralisée

Partage des fonctions entre des processeurs spécifiques et un processeur de contrôle à l'intérieur d'un même commutateur.

2.2 COMPOSANTS

Dans le commutateur, on distingue deux ensembles :

- les organes qui assurent effectivement la liaison temporaire entre deux abonnés et qui constituent le réseau (ou matrice) de connexion;

- les équipements communs qui aident à la mise en place du réseau de connexion et qui ne sont occupés que pendant une partie du temps de la communication et qui appartiennent à l'unité de commande.

2.2.1 Réseau de Connexion

2.2.1.1 Introduction

L'évolution chronologique et technologique des commutateurs a débuté en 1878 à New Haven aux Etats-Unis avec un premier système manuel de commutation.

Depuis lors, l'évolution des commutateurs vers l'électronique, parallèle aux développements des systèmes de transmission, fut beaucoup moins rapide que ces derniers.

A l'heure actuelle, le débat sur la commutation se développe autour des deux pôles que sont l'électronisation et l'informatisation.

Le dernier né de cette communion est la commutation temporelle qui permet une symbiose de plus en plus poussée entre la transmission et la commutation de par l'identité entre les signaux utilisés dans les deux applications.

Ces développements, pour être utilisables à l'échelon international, ont fait l'objet de nombreuses nomenclatures.

2.2.1.2 systèmes manuels

Une opératrice se tenant devant un tableau répond aux appels des abonnés, établit les communications demandées à l'aide d'un cordon et les libère quand la communication est terminée.

2.2.1.3 systèmes automatiques

A) Systèmes rotatifs

a.- Systèmes pas à pas

Le composant élémentaire de ce type de système est le sélecteur. Celui-ci comprend dix niveaux. A chaque niveau, se trouvent dix sorties vers un autre sélecteur ou vers un abonné.

Quand ce dernier compose le premier chiffre du numéro, chaque impulsion émise par le cadran provoque l'ascension d'un niveau du sélecteur relié à l'abonné. Pendant que le cadran revient au repos, le sélecteur tourne autour de son axe vertical et s'arrête sur la première sortie libre. L'abonné compose alors son deuxième chiffre qui provoque de même la montée du sélecteur relié à la sortie choisie précédemment, et ensuite, la recherche d'un niveau libre. Ainsi de suite jusqu'à la connexion à l'abonné demandé (figure T6).

Un avantage de ce système est qu'un étage déjà mis en place est transparent à la numérotation ultérieure de l'abonné.

Par contre, le plan de numérotation est dépendant de la structure du réseau, ce qui a conduit à la conception du système à enregistreur.

b.- Système à enregistreur

Le numéro de l'abonné est reçu directement dans un enregistreur. Celui-ci fait appel à un traducteur qui lui donne les éléments pour acheminer l'appel à travers les sélecteurs.

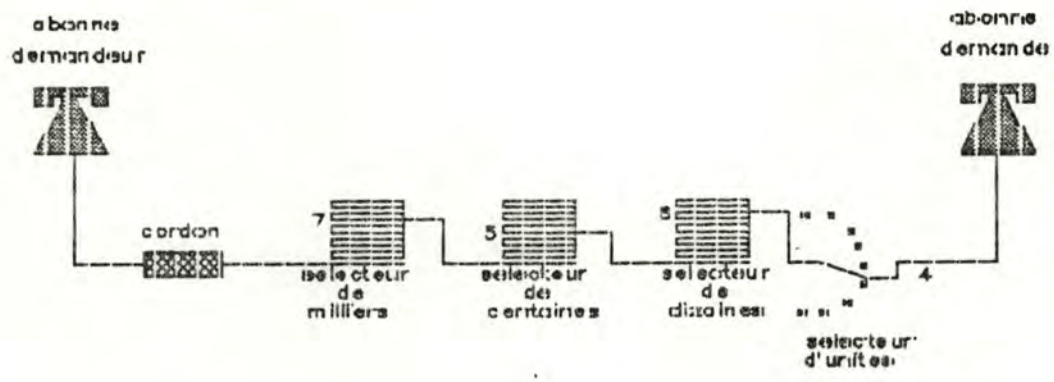


figure T6 : principe d'un commutateur pas à pas

L'abonné demandé a le numéro 75845.

On remarque déjà ici la distinction entre le "réseau de connexion" représenté par les étages de sélection et l'"unité de commande" représentée par l'enregistreur et le traducteur (figure T7).

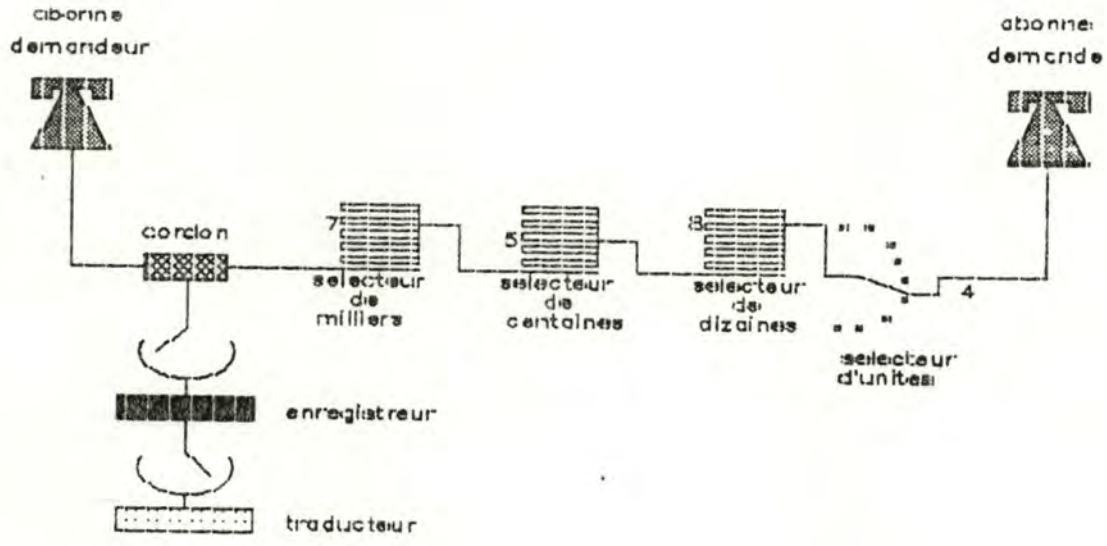


figure T7 : principe d'un commutateur à enregistreur

L'abonné demandé a le numéro 7584.

Le seul inconvénient reste que les contacts établissant les communications se déplacent sur des distances importantes. Les systèmes crossbar inventés par Bertulander en 1917 vont tenter d'y remédier.

B) systèmes crossbar

Ces systèmes sont basés sur des sélecteurs appelés "crossbar" car leur principe consiste à mettre en place, à l'aide d'électro-aimants, deux barres croisées, l'une horizontale, l'autre verticale, séparées au repos de quelques millimètres, à l'intersection desquelles un contact est établi.

Contrairement au système rotatif, le système crossbar offre l'avantage de la sélection conjuguée : une entrée et une sortie peuvent être connectées si il existe au moins un chemin libre entre elles.

C) Systèmes électroniques

a.- Commutation spatiale

Le principe régissant la commutation spatiale consiste à établir physiquement un chemin continu entre la ligne entrante et la ligne sortante. La recherche d'itinéraire permet d'obtenir un chemin de proche en proche en connectant bout à bout des éléments de voie au moyen de points de connexion à deux états : passant ou bloqué.

Lorsque le chemin continu est établi, il permet le passage direct des signaux de la ligne entrante à la ligne sortante et est consacré à une seule communication pendant toute la durée de celle-ci.

Les points de connexion sont disposés comme les lignes et colonnes d'un tableau (matrice de connexion) pour que chaque ligne puisse être connectée à chaque colonne par un point de connexion commun.

Les matrices sont rangées en étages adjacents connectés entre eux. Cette technique est très voisine du système crossbar utilisé en commutation électro-mécanique. Elle en diffère de par la nature des sélecteurs utilisés, de par la taille des matrices et de par l'arrangement des étages de sélection.

b.- Commutation temporelle

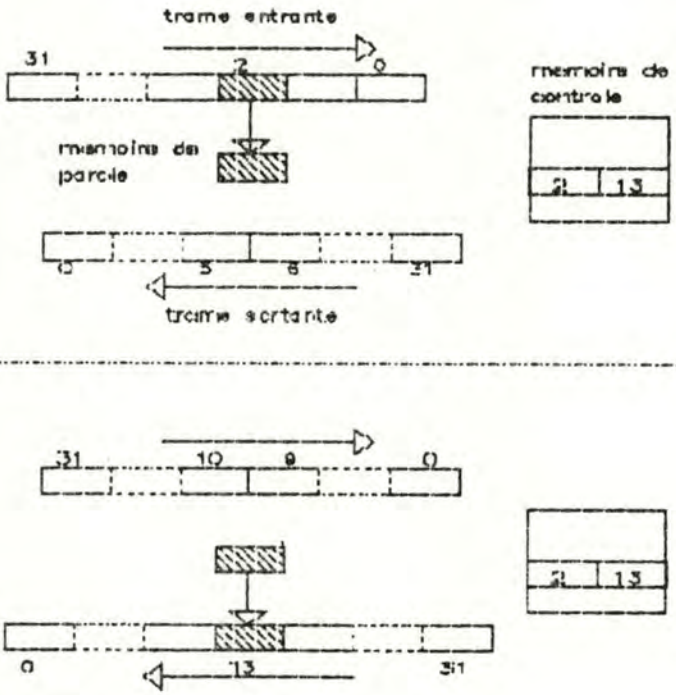


figure T8 : opération T

Au temps i,l'information de l'IT 2 de la voie entrante est transférée dans la mémoire de parole.
Au temps i+7,5 , l'information de la mémoire de parole est transférée dans l'IT 13 de la voie sortante.
Les voies entrante et sortante ne sont ni alignées, ni synchronisées.

La commutation temporelle s'applique, non plus à des lignes entrantes et sortantes individualisées (correspondant à une voie ou une communication), mais bien à des lignes où circulent des trames de 32 intervalles de temps (IT), 30 de ces intervalles correspondant à 30 voies multiplexées dans le temps.

Le problème consiste à transférer les informations véhiculées dans l'IT de la trame entrante qui correspond à la voie entrante indiquée dans un IT donné d'une trame sortante donnée.

Une table de correspondance maintient la relation entre les positions temporelles dans la trame entrante et dans la trame sortante. Il s'agit de la mémoire de contrôle.

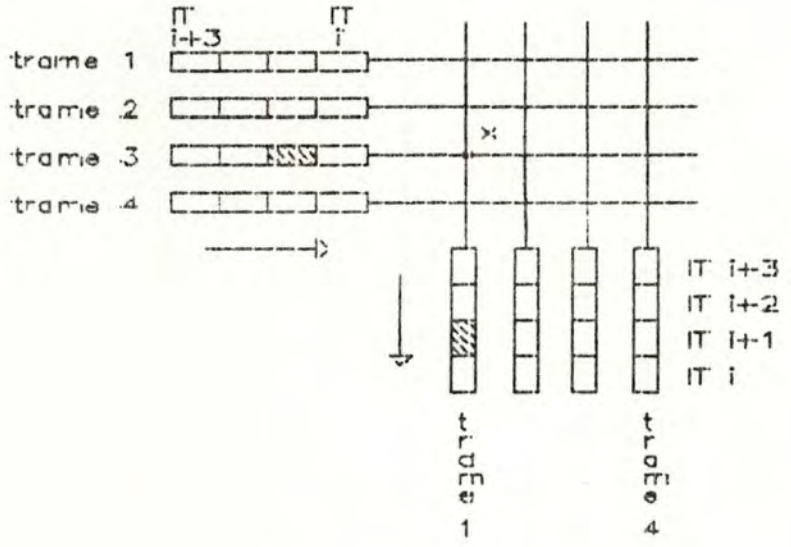


Figure 19 : operation S

Les liaisons numériques étant synchronisées et alignées, il reste à transférer l'information de l'IT i+2 de la trame 3 à la trame 1. Pour ce faire, au temps i+2 le point de connexion x devient passant.

Les trames n'étant pas nécessairement synchrones, un changement d'IT (opération T) est nécessaire. Cette opération réalise le déphasage voulu en faisant séjourner dans une mémoire temporaire (mémoire de parole) le temps convenable l'information transportée par la voie entrante de manière à la synchroniser et à l'aligner avec l'IT correspondant à la voie sortante (figure T8).

Le commutateur temporel fournit l'accessibilité totale, sans blocage. Malheureusement, il est limité en capacité par la technologie disponible. Les gros réseaux devront être des réseaux à étages réalisés par assemblage d'opérations T et d'opérations S.

Cette dernière est une commutation spatiale. Une fois les IT synchronisés et alignés, il suffit, dans une matrice, de rendre passant le point de connexion mettant en relation les trames des lignes entrantes et sortantes (figure T9).

2.2.1.4 AXE-10 : Le Réseau de Connexion Numérique

La version du Réseau de Connexion (RC) réalisée dans l'AXE 10 est du type commutation temporelle. Nous étudierons ici le sous-système de commutation de groupes qui contient le réseau de commutation de groupes. Nous verrons également les tenants et aboutissants de celui-ci au travers des sous-systèmes d'abonnés et de jonctions (figure T10).

figure T10 : sous-systèmes de connexion et de commutation

Les développements pré-cités se reposent sur l'explication du déroulement des quatre types d'appels évoqués déjà précédemment : l'appel local, l'appel sortant, l'appel entrant et l'appel de transit (figure T11).

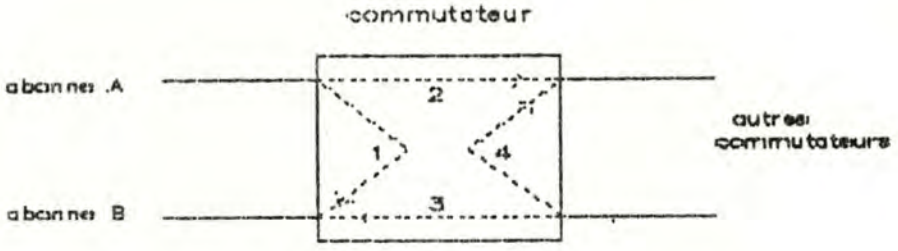


figure T11 : les 4 types d'appel

1.— appel local
2.— appel sortant
3.— appel entrant
4.— appel de transit

A) Etablissement d'un Appel local

a.- le Sous-système d'Abonnés

Chaque ligne d'abonné est reliée au commutateur via un module de "joncteur d'abonné" (LIC). Ce module réalise des fonctions comme le courant de ligne, la détection du décrochage, la connexion de sonnerie ou la conversion analogique/numérique grâce à un CODEC.

Quand l'appel est détecté par le module LIC, le système de commande ordonne l'allocation d'un "Intervalle de Temps" (IT) dans la liaison numérique à 32 voies qui connecte le sous-système d'abonnés au sous-système de commutation de groupes (figure T12).

figure T12 : le joncteur d'abonné LIC

Pendant l'heure chargée, une ligne d'abonné n'est utilisée en moyenne que de 6 à 10 % du temps. Sous ces conditions, il n'est pas économiquement possible d'allouer à chaque abonné une voie de parole sur la liaison MIC.

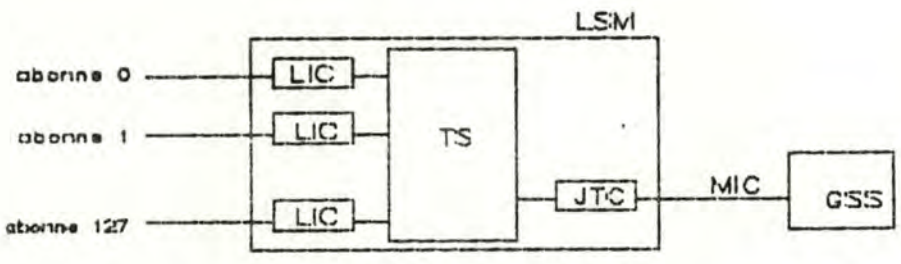


figure T13 : un module de lignes (LSM)

Ces voies MIC peuvent être mieux utilisées si un grand nombre d'abonnés partagent un nombre relativement faible de liaisons numériques. Cela est réalisé grâce à une commutation temporelle entre le module LIC de l'abonné et le "Joncteur Interne" (JTC) utilisé pour connecter un IT d'abonné avec une voie de parole libre. 128 abonnés peuvent être reliés à un module de commutation spatiale TS et un joncteur interne, formant ainsi un "Module de Lignes" (LSM (figure T13).

Un nombre maximum de 16 modules LSM peuvent être interconnectés au travers d'un bus commun (TSB) pour former une unité de 2048 abonnés appelée "Groupe de Modules d'Extension" (EMG). Dans ce cas, le nombre de liaisons MIC reliant le module EMG à GSS dépendra du trafic. Habituellement, les liaisons MIC ne sont connectées qu'à quelques uns des modules LSM. Un abonné connecté à un module LSM n'ayant pas de liaisons MIC se verra connecté via le bus TSB à un module LSM bénéficiant d'une liaison MIC.

De la même manière, un abonné peut être connecté à une autre liaison de module LSM si son propre LSM a une liaison MIC dont toutes les voies sont occupées (figure T14).

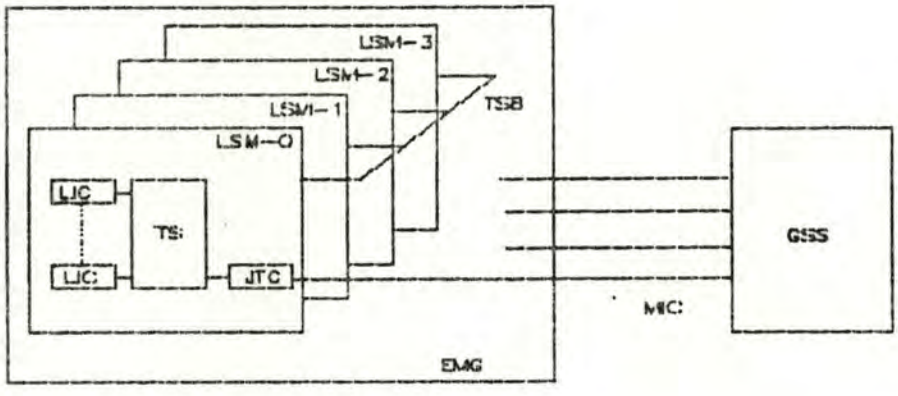


figure T14 : groupe de modules d'extension (EMG)

Un ou plusieurs modules EMG formeront le sous-système d'abonnés.

Dans le cas d'un appel local, une fois l'abonné appelé déterminé, de la même manière que pour l'abonné appelant, une voie dans une des liaisons MIC du module EMG de l'appelé est réservée pour la suite de l'échange, à partir du GSS vers le module LSM.

b.- Le Sous-système de Commutation de Groupe

Jusqu'à présent,le chemin n'a pas encore été établi entre les voies numériques des deux abonnés. Le module GSS sélectionne, connecte et déconnecte les chemins de parole. Ce module est réalisé à partir de Modules de Commutation Temporelle (TS) qui contiennent des mémoires pour les échantillons de parole contenus dans les voies entrantes et sortantes, ainsi que des Modules de Commutation Spatiale (SP) qui consistent en matrices 32 * 32. Le module TS est divisé en deux parties identiques : le "Module Entrant" (ME) et le "Module Sortant" (MS).

b.1 MODULE DE COMMUTATION TEMPORELLE

Chaque partie comporte une mémoire de parole (SS) où les échantillons MIC sont entreposés dans des intervalles de temps choisis arbitrairement à l'intérieur d'une trame, au moyen d'une "Mémoire de Commande" (CS). Les mémoires SS et CS contiennent chacune 512 cellules correspondant au nombre de voies entrantes dans le module TS, à savoir 16 jonctions de 32 voies chacune. Pendant chaque IT, la cellule indiquée dans la mémoire de commande est lue dans la mémoire de parole de ME (figure T15) et écrite dans la mémoire de parole de MS.

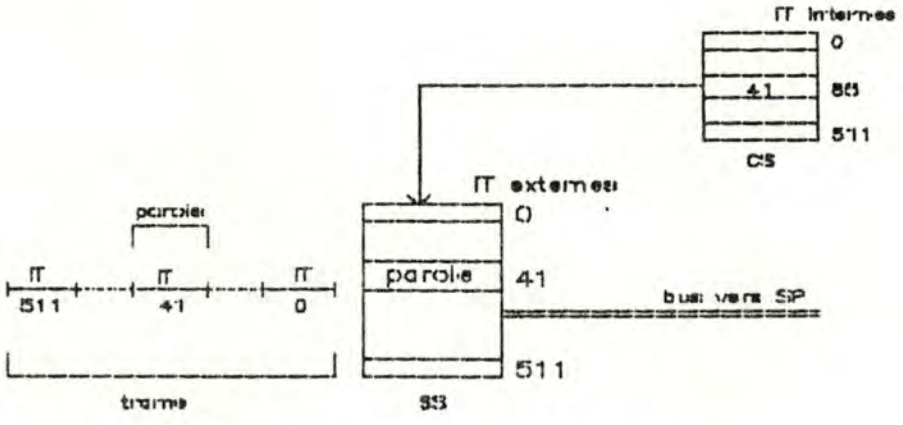


Figure T15 : module TS

Connexion de la voie dans l'IT externe 41 vers le module SP pendant l'IT interne 85.

b.2 MODULE DE COMMUTATION SPATIALE

A un module SP peuvent être connectés 32 modules TS, ce qui représente un nombre d'abonnés pouvant être servis de 16.348.

Chacun de ces modules TS est connecté au module SP via un bus entrant (BE) pour ME et un bus sortant (BS) pour MS, une connexion étant unidirectionnelle.

Le module SP est utilisé pour commuter les bus entrant vers les bus sortant. A chaque colonne de points de connexion correspondant à un bus entrant est assignée une mémoire de commande (CSC) ayant 512 cellules. Chaque cellule correspond à un IT interne et contient l'adresse du point de connexion sur la colonne qui doit devenir actif pendant l'IT spécifié, l'IT interne de la voie entrante étant nécessairement le même que celui de la voie sortante.

La recherche des chemins libres dans le réseau de connexion est effectuée par le processeur central. Le réseau est à accessibilité totale entre toute entrée et toute sortie (figure T16).

Aussi longtemps que les informations contenues dans les mémoires de commande sont maintenues, la même séquence de commutation temporelle et spatiale sera exécutée cycliquement, trame après trame.

Le chemin de parole inverse, de l'abonné appelé à l'abonné appelant, est établi par la "méthode antiphase". Dès qu'un chemin libre de l'appelant à l'appelé est trouvé pendant un IT interne, un chemin libre de l'appelé à l'appelant est garanti une demi trame plus tard, c'est-à-dire 256 IT plus tard.

On peut également interconnecter 16 modules SP en une matrice 4 * 4 permettant de connecter ainsi 128 modules TS, ce qui représente 65.536 voies.

B) Etablissement des autres Types d'Appel

En fonction du type d'appel, la nature des modules auxquels seront reliées les voies entrante et sortante sera différente. L'ensemble des modules périphériques au GSS et autres que ceux du

sous-système d'abonnés font partie du "Sous-système de Jonctions" (TSS).

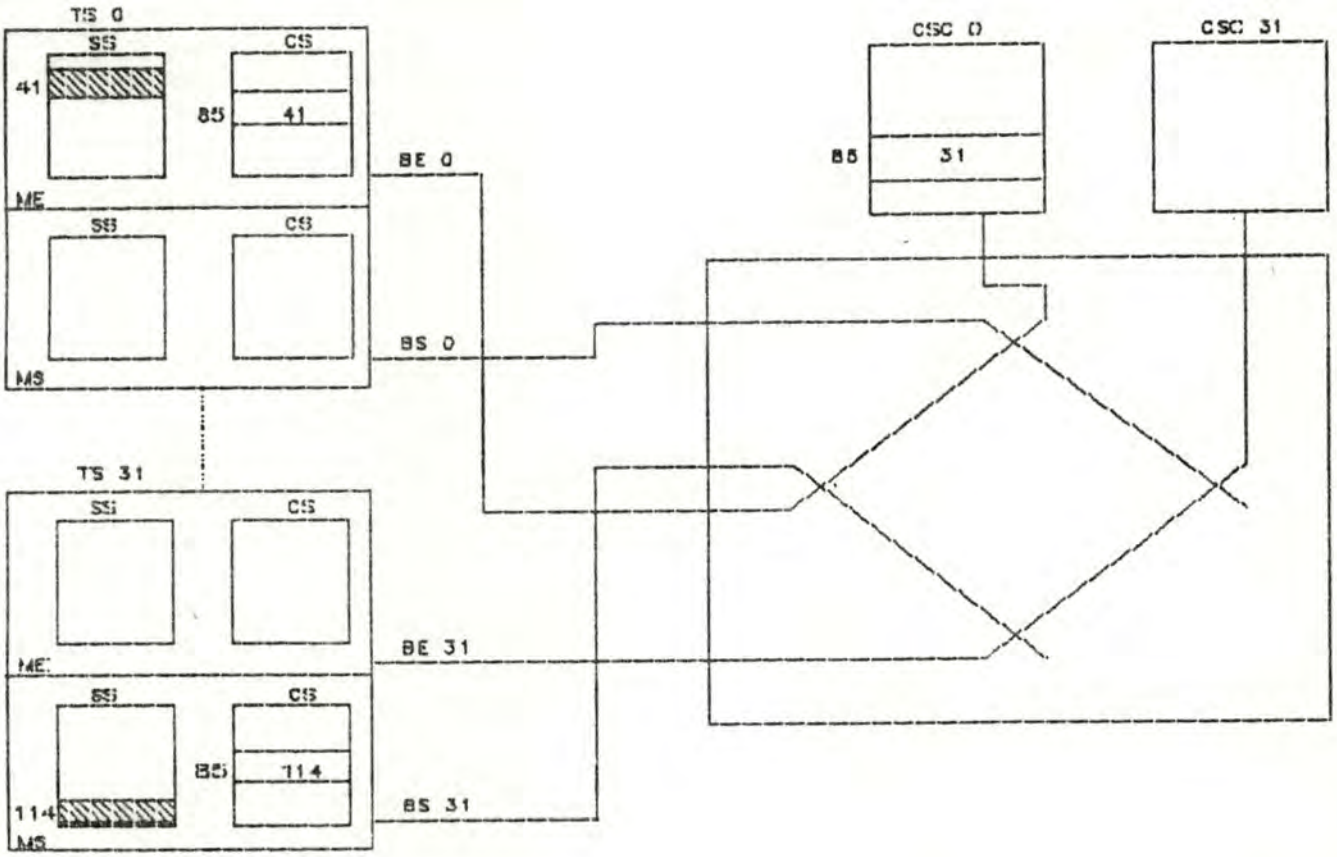


figure T16 : module SP

Connexion entre la voie entrante 41 du module TS 0 vers la voie sortante 114 du module TS 31. Le chemin est établi pendant l'IT interne 85.

Essentiellement trois types de joncteurs sont utilisés dans le module TSS. Les communications en provenance de ou vers un autre commutateur seront connectées, si la jonction est numérique, à un "Joncteur d'Echange" (ETC) capable de transmettre simultanément 30 conversations.

Si la jonction est analogique et que nous sommes en présence d'un appel sortant, un "Module Joncteur Sortant" (OTC) sera saisi. Si, par contre, l'appel est entrant, un "Joncteur Entrant" (ITC) sera utilisé. Dans ces deux derniers cas, un CODEC servira d'interface entre les joncteurs OTC et ITC et le module de commutation de groupes GSS.

L'appel de transit peut être considéré comme une succession d'un appel entrant et d'un appel sortant et utilisera les modules correspondant à la nature analogique ou digitale des jonctions entrante et sortante (figure T17).

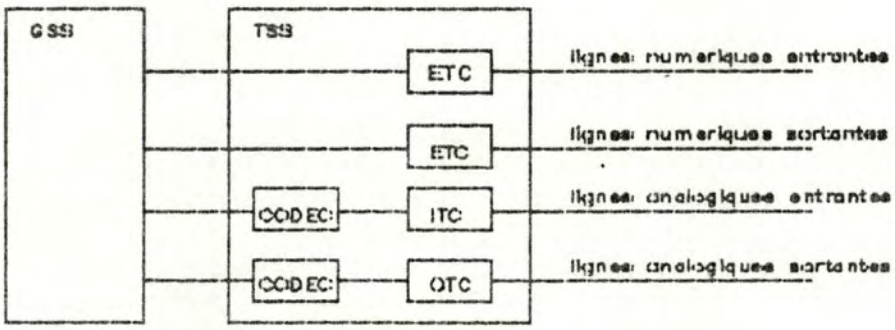


Figure T17 : le sous-système de jonctions

2.2.1.4 ITT-1240 : Le Réseau de Connexion Numérique

A) Principes et Caractéristiques

a.- Topologie repliée

Cette topologie consiste à structurer le Réseau de Connexion Numérique (RCN) en étages. Cette configuration possède deux atouts majeurs. D'abord, la connexion utilise si possible un nombre minimum d'étages. Ceci permet de réaliser beaucoup plus efficacement les connexions locales sans pour autant freiner les autres (figure T18).

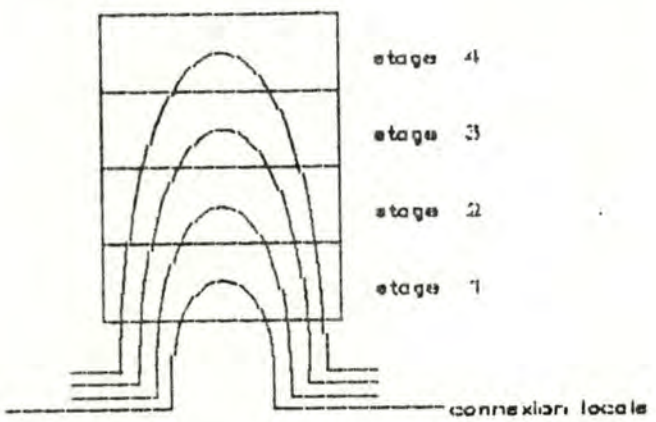


figure T18 : etages de commutation

De plus, tous les raccordements étant situés du même côté, l'extension du RCN est rendue possible par simple ajoût d'étages.

Ces deux caractéristiques, rapidité et souplesse, offrent au système 1240 un avantage par rapport à la commutation traditionnelle TST ou STS qui oblige toute connexion à traverser le réseau complet.

Cette structure repliée donne au système ITT une grande échelle d'adaptation étant donné la gamme de capacités qu'il peut supporter : de moins de 100 à plus de 100.000 modules terminaux connectables.

Un système traditionnel, tout en couvrant une gamme plus restreinte, doit être plannifié à l'avance pour une capacité maximale déterminée.

b.- Uniformité des Modules de Commutation

Le RCN est composé entièrement de petits modules de commutation numérique identiques. Chacun d'entre eux comporte de la commutation spatiale et temporelle. L'uniformité des composants de base assure au système une souplesse telle que la maintenance et la fiabilité, grâce à l'ajoùt et au retrait des modules selon les besoins, sans modification du reste de la structure, deviennent maximales.

c.- Commande à l'Origine

Dans les commutateurs traditionnels, un processeur central et des processeurs périphériques supervisent l'acheminement des messages à travers le réseau.

Dans le système 1240, la commande est totalement répartie ce qui signifie que les ordres viennent des modules demandeurs. Ces derniers déterminent les chemins à suivre dans le réseau et donnent au RCN les commandes qui permettent de l'établir. Le RCN ne fait qu'interpréter les ordres reçus et ne possède aucune intelligence autonome.

d.- Performances et Sécurité

De par la structure en plans du réseau, la probabilité de blocage est très faible, de l'ordre de 10-3 en charge normale. Une multiplicité de connexions entre deux modules donnés est possible, offrant ainsi une sécurité et une qualité de service exceptionnelle.

B) Module de Commutation Numérique

Ce module est l'unité fonctionnelle de base du réseau. Il consiste en une carte enfichable de circuit intégré portant 16 circuits d'accès interconnectés par un bus multiplexé dans le temps. Chaque circuit d'accès est connecté à une ligne entrante et une ligne sortante comprenant chacune 32 canaux MIC de 16 bits (figure T19).

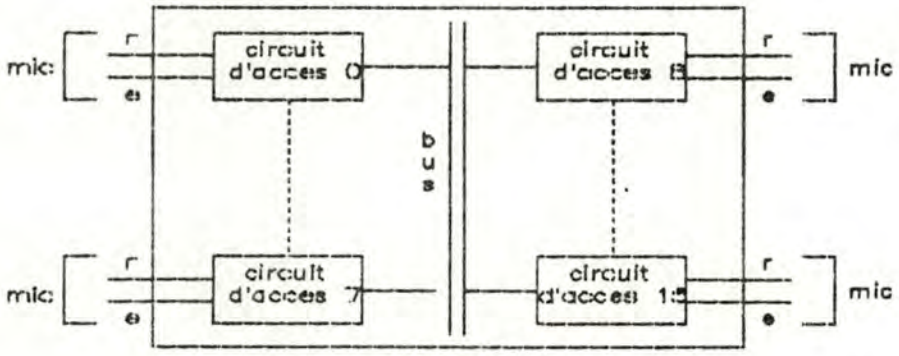


figure T19 : module de commutation numerique

Le Module de Commutation Numérique (MCN) est capable d'établir et de maintenir un chemin sur lequel on peut transférer des informations de n'importe quel canal du côté réception d'une quelconque des 16 jonctions MIC vers n'importe quel canal du côté transmission d'une quelconque des 16 jonctions MIC sortantes.

C) Structure du RCN

L'organisation du RCN du 1240 étant assez complexe, nous envisageons dans cette étude une description évolutive qui, partant d'une configuration minimale, arrive en cinq étapes à la configuration maximale. Cette évolution prendra corps par l'adjonction de modules de commutation numérique s'enchaînant à la structure existante et cela, pour augmenter la capacité de traitement du commutateur et donc de l'adapter aux changements de l'environnement.

a.- Première Etape : Configuration Minimale

Dans cette version, pourront être connectées au réseau RCN un nombre maximum de 60 lignes d'abonné. Ces lignes sont raccordées à un Module Terminal de Lignes (MTL) assurant l'interface intelligent entre les abonnés et le RCN.

Chaque MTL est connecté à une paire de Modules de Commutation Numérique (MCN) appelés Commutateurs d'Accès (CA). Sur chacune des deux jonctions circule une trame MIC de 32 canaux dont 30 voies de parole. Sur l'ensemble des deux jonctions, nous avons donc 60 voies de parole correspondant aux 60 lignes raccordées au MTL (figure T20).

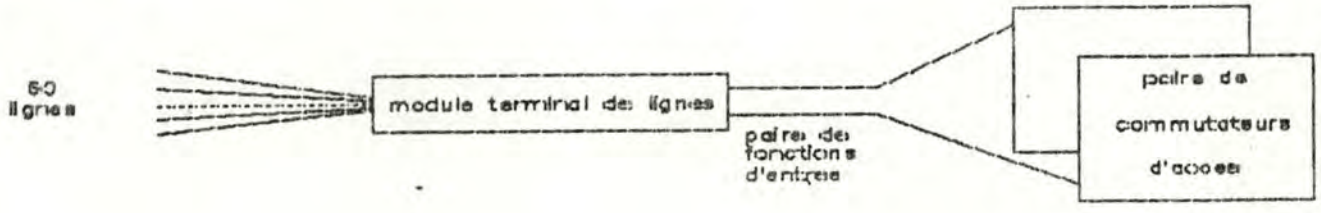


figure t20 : configuration minimale

Nous avons vu précédemment qu'un MCN possédait 16 circuits d'accès. Dans un commutateur d'accès, nous trouvons 12 circuits d'accès du côté du MTL et 4 du côté du réseau (figure T21).

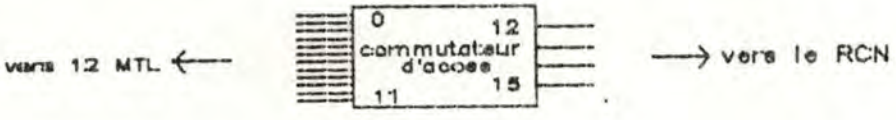


figure T21 : circuits d'accès du CA

Ainsi, avec notre seule paire de CA, nous pouvons raccorder 12 MTL, c'est-à-dire 720 lignes, les 4 circuits d'accès côté réseau restant provisoirement libres et inutilisés. Les informations en provenance des 12 MTL pourront être commutées vers un quelconque de ces 12 modules.

On perçoit déjà ici la notion de structure repliée car une connexion peut déjà avoir lieu au niveau des commutateurs d'accès qui sont, comme on le verra plus loin, le premier étage du RCN.

b.- Deuxième Etape

Ma paire de CA pouvant commuter 720 lignes risque de s'avérer insuffisante. Dans ce cas, de nouvelles paires de CA sont ajoutées. Mais ces paires, pour pouvoir accéder les unes aux autres, nécessite l'adjonction d'un deuxième étage de commutation. Nous entrons, avec ce deuxième étage, dans le commutateur de groupe auquel sont reliés les commutateurs d'accès (figure T22).

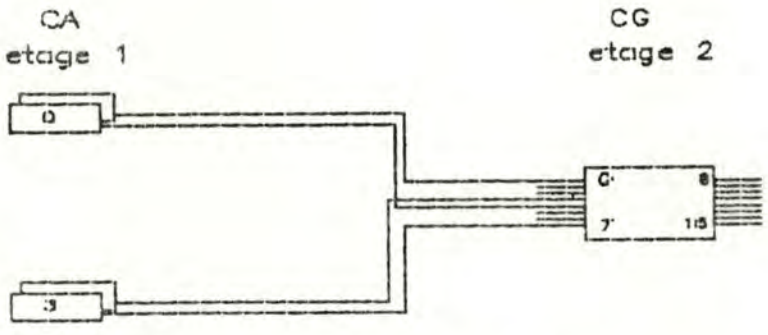


figure T.2.2 : adjonction du premier etage du commutateur de groupe

Il est important de noter que, suivant la topologie repliée, une connexion ne se fait via le Commutateur de Groupe (CG) que si les deux interlocuteurs sont reliés à des CA différents.

Un module de commutation numérique de l'étage deux a 8 circuits d'accès tournés vers les CA et 8 tournés vers les étages supérieurs du CG. On peut donc y connecter 4 paires de CA, ce qui représente 48 paires de jonctions d'entrée et donc 2.880 lignes.

c.- Troisième Etape

Lorsqu'un commutateur de groupe avec un seul MCN devient insuffisant, un étage numéro 3 est ajouté au CG. Un MCN de l'étage 3 posséde la même configuration (8 circuits d'accès de part et d'autre) qu'un MCN du deuxième étage. 8 de ces derniers, formant ainsi un groupe, pourraient être connectés à un MCN du troisième étage.

La solution retenue sera quelque peu différente pour répondre à des critères de sécurité et de performance. En effet, le nombre de lignes raccordées devenant fort important (23.040), un MCN d'étage 3 risque d'être débordé et de créer un blocage s'il est seul responsable d'un groupe d'étage deux.

Pour cette raison, 8 MCN d'étage 3 seront connectés à 8 MCN d'étage 2. La version maximale de cette étape correspond à un groupe (figure T23).

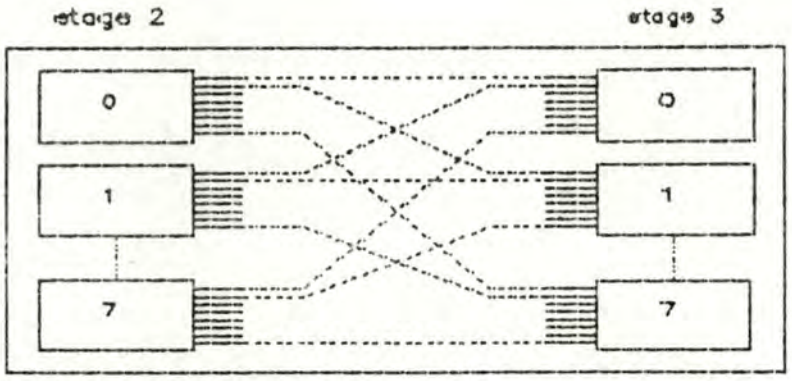


figure T23 : groupe au sein du CG

d.- Quatrième Etape

Lorsqu'un groupe (étages 2 et 3) arrive à saturation, un quatrième et dernier étage également composé de MCN est ajouté à la structure en place. Les MCN d'étage 4 ont leurs 16 circuits d'accès tournés vers l'étage 3 (figure T24).

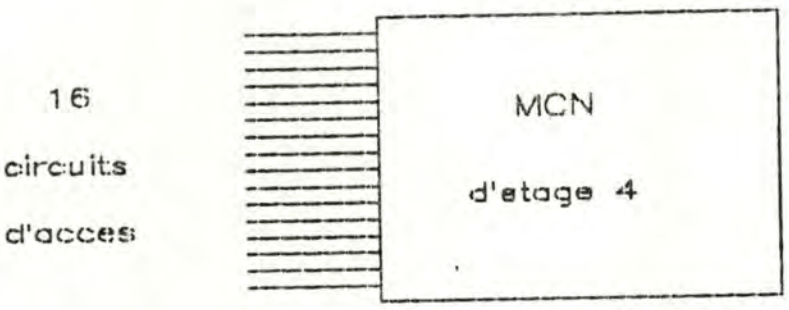


figure T24 : circuits d'acces d'un MCN d'etage 4

Un groupe de 8 MCN d'étage 4, c'est-à-dire 128 circuits d'accès, pourront donc supporter deux groupes d'étage 2/3 offrant le même nombre de circuits d'accès.

Pratiquement, dans la configuration maximale, nous trouvons 16 groupes d'étage 2/3 supportés par 8 groupes d'étage 4.

Dans ce cas, le circuit d'accès X du MCN Y du groupe Z d'étage 2/3 sera raccordé au circuit d'accès Z du MCN X du groupe Y d'étage 4.

Le commutateur de groupe qui reprend à présent les étages 2,3 et 4 et le commutateur d'accès qui reprend les paires de MCN décrites en 3.1 forment la configuration maximale de la quatrième étape (figure T25).

e.- Cinquième Etape : Configuration Maximale

Malgré la multiplicité des chemins offerts par la structure décrite dans le paragraphe précédent, une telle configuration peut être facilement débordée si le trafic devient important.

On pourra donc doubler, tripler ou quadrupler un plan de commutation quel qu'il soit. Un plan correspond à l'une quelconque des étapes décrites dans les paragraphes précédents. Un plan de groupe correspond à la figure 8.

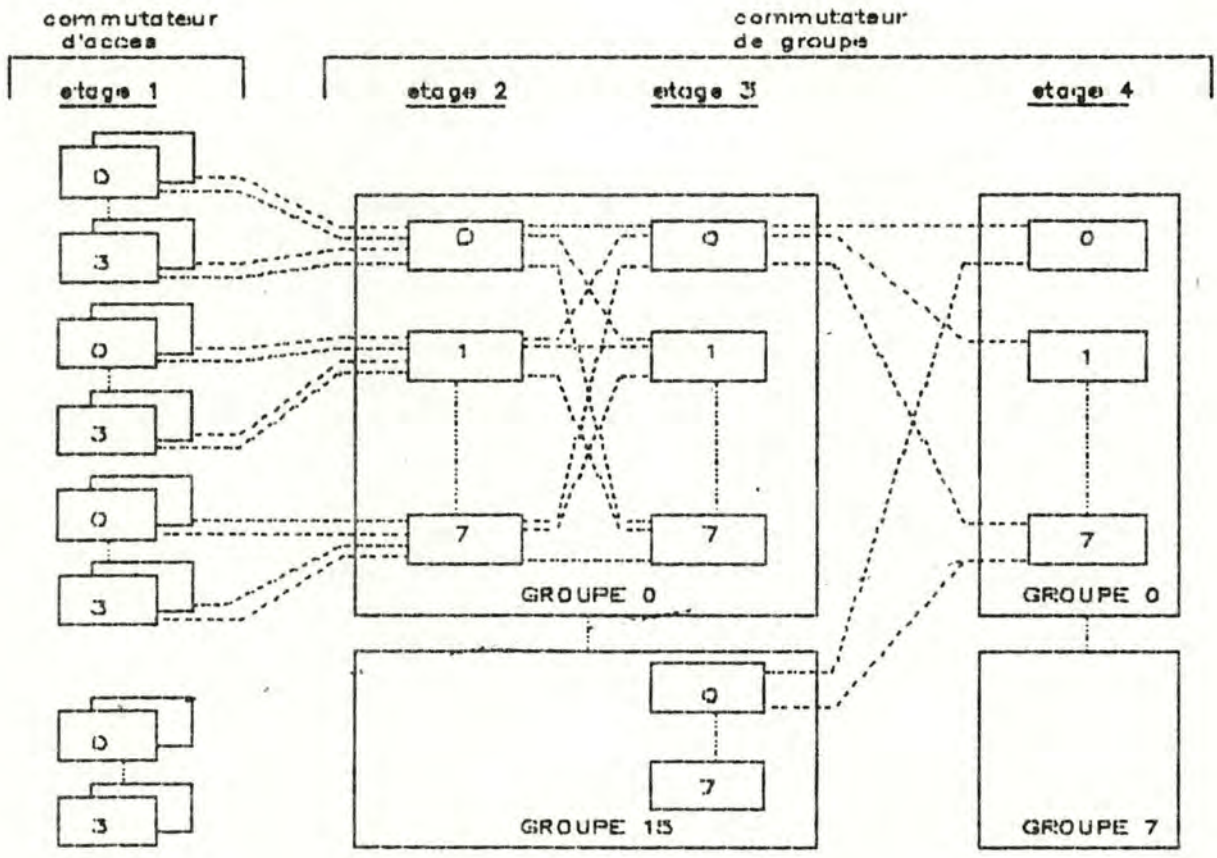


figure T25 : plan de groupe

Ces plans de commutation ne sont pas interconnectés entre eux et sont reliés directement aux commutateurs d'accès. Une connexion peut donc se faire indépendemment via l'un ou l'autre de ces plans, ce qui accroît considérablement la capacité de trafic du RCN (figure T26).

La configuration maximale avec 4 étages complets et 4 plans est caractérisée par les chiffres suivants :

- chaque plan de commutateur de groupe comporte 320 modules;
- le commutateur d'accès comporte 512 paires de MCN;
- l'ensemble du RCN comporte ((4*320)+(2*512)) 2.304 MCN;
- le commutateur d'accès offre 6.144 paires de jonctions d'entrée et donc autant de modules terminaux de lignes;

- la capacité dépasse 100.000 lignes ou 60.000 circuits.

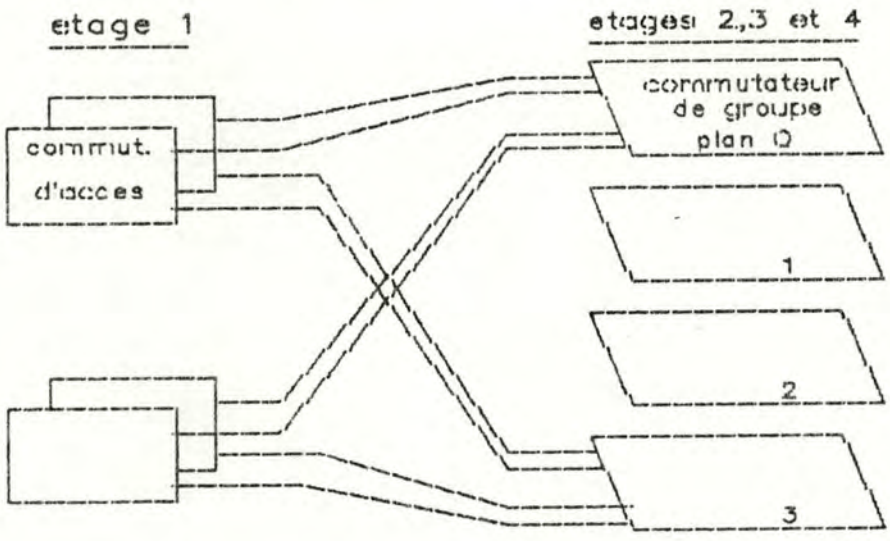


figure 128 : configuration avec 4 plans de commutateurs de groupe

D) Etablissement d'un Chemin dans le RCN

a.- Adresses-Réseau

Tous les modules terminaux de lignes (MTL) sont identifiés par une adresse ABCD.

A : sélection d'un MTL (0-11) à partir d'un commutateur d'accès.

B : sélection d'une paire de CA (0-3) à partir d'un MCN d'étage 2.

C : sélection d'un MCN d'étage 2 (0-7) à partir d'un MCN d'étage 3.

D : sélection d'un groupe d'étage 3 (0-15) à partir d'un MCN d'étage 4.

b.- Algorithme de Sélection

L'algorithme de sélection doit répondre à deux critères :

- être rapide;
- ne pas aller plus loin que nécessaire dans les étages du RCN.

b.1 SELECTION DE L'ETAGE DE REFLEXION

Les adresses ABCD de l'émetteur et A'B'C'D' du récepteur sont comparées (figure T27).

Si	D≠D'	Alors	etage 4
Si et	D=D' C≠C'	Alors	etage 3
Si et et	D=D' C=C' B≠B'	Alors	etage 2
Si et et	D=D' C=C' B=B'	Alors	etage 1

figure T27 : selection de l'etage de reflexion

b.2 RECHERCHE LIBRE

Un chemin est établi à partir du MTL appelant via des MCN quelconques sur des jonctions quelconques jusqu'à un MCN de l'étage de réflexion. Ce MCN est appelé point de réflexion.

b.3 RECHERCHE DIRIGEE

Un chemin est établi à partir du point de réflexion via des MCN quelconques sur des jonctions déterminées par l'adresse ABCD de l'appelé, jusqu'à ce MTL appelé.

c.- Ordres

Quatre ordres de sélection de chemin sont utilisés :

X : recherche libre dans le CA vers l'un des 4 plans;

Y : recherche libre dans le MCN d'étage 2 (3) vers l'un des 8 MCN d'étage 3 (4);

N : sélection dirigée d'une des 8 jonctions possibles sur base de A,C et D;

NZ : sélection dirigée d'une des 2 jonctions possibles sur base de B.

2.2.2 : Organes de Commande

2.2.2.1 : Aspects

A) Fonctions

Dans un commutateur, on désigne sous le vocable "organes de commande" l'ensemble des équipements qui exécutent les fonctions les plus "intelligentes". C'est là que se trouve la "logique" plus ou moins complexe du système, là que se prennent les décisions. (figure T5)

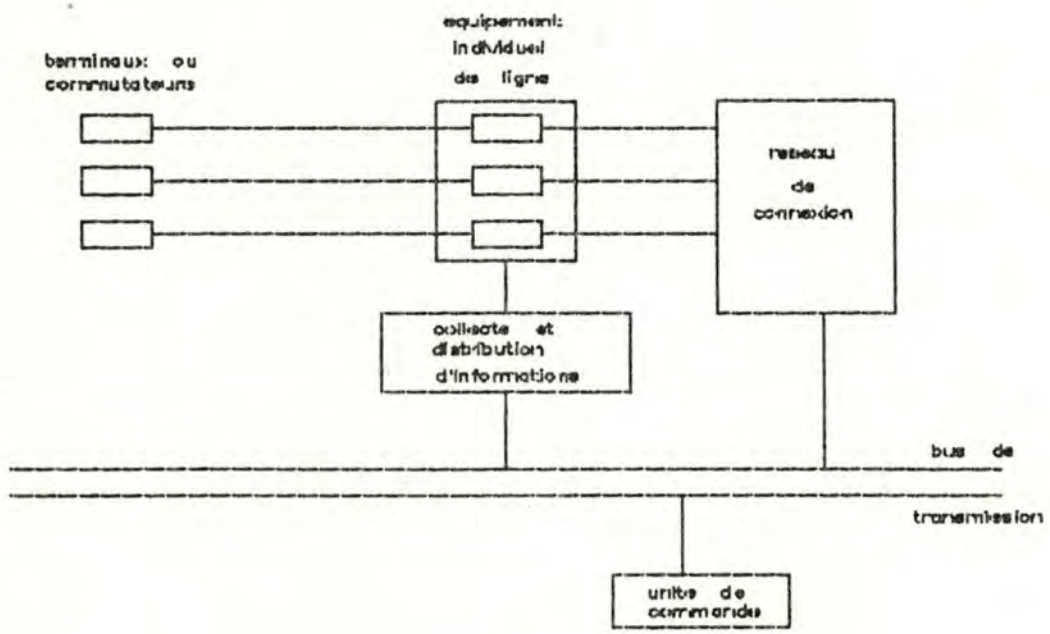


figure T5 : blocs fonctionnels d'un commutateur

Par nature, les fonctions de commande reposent d'abord sur l'existence d'une fonction de mémorisation qui contient à chaque instant les données caractérisant l'état de tous les éléments matériels ou logiques, du système.

Le passage d'un état à un autre de chacun de ces éléments est le résultat d'actions conditionnelles d'automates qui analysent et modifient l'état en fonction d'événements extérieurs.

En effet, la mise en oeuvre des fonctions de commutation nécessite le transfert d'informations entre un abonné et un commutateur, ou entre deux ou plusieurs commutateurs. Ces informations échangées au moment de l'établissement et de la rupture des connexions constituent la signalisation. Ces signaux correspondent en fait aux différentes phases de la communication.

Une ligne d'abonné diffère d'une jonction entre commutateurs par le mode d'échange des informations c'est-à-dire par la signalisation.

a.- Fonctions de Commutation

Les fonctions "traitements des appels" sont d'abord des "fonctions d'acquisition" des événements se produisant sur les lignes d'abonné et les jonctions entre commutateurs à travers des explorateurs ou scanners.

Ce sont ensuite des "fonctions de distribution" pour commander l'émission de signaux vers ces lignes ou jonctions.

Entre les deux se placent des fonctions internes. Ces dernières ont pour but l'"analyse des signaux" reçus, comme la traduction, les "décisions" concernant la progression des appels comme l'acheminement et la "fonction de connexion" qui assure la sélection, la mise en place et le relâchement des itinéraires dans le réseau de connexion.

Comme nous le verrons plus loin, à toutes ces fonctions s'appliquent des contraintes de temps réel de l'ordre de quelques dizaines ou centaines de msec, soit en raison de la nature même des signaux, soit en raison de normes de qualité de services à assurer (durée de la présélection par exemple). Leur éxécution nécessite la mémorisation de l'état instantané des éléments mis en jeu, de leurs interconnexions permanentes, de la situation de chaque appel en cours, ainsi que des informations permettant d'interpréter la numérotation et les signaux reçus (tableau de traduction, catégories des abonnés etc...)

Enfin, un haut degré de parallèlisme est requis, puisque le nombre d'appels en cours de traitement peut atteindre plusieurs centaines. La puissance, en commutation, des organes de commande se mesure d'ailleurs par le nombre d'appels qu'ils sont capables de traiter par unité de temps.

b.- Fonctions d'exploitation et de maintenance

L'exploitation et la maintenance désignent les fonctions qui permettent

- d'une part, de gérer, c'est-à-dire d'ajouter,de modifier, de supprimer les éléments matériels ou logiques constituant le commutateur et ses accès, de surveiller son fonctionnement et de mesurer son trafic ;
- d'autre part, de détecter et de localiser les défaillances des équipements.

Ces fonctions "accessoires" de support représentent maintenant typiquement plus des deux tiers du volume des automates logiques.

B. Caractéristiques

L' unité de commande, après avoir été réalisée dans un premier temps en logique cablée, se trouve, dans les commutateurs modernes, sous la forme de programmes enregistrés.

Mais, ces programmes, de par la nature même des fonctions à réaliser, doivent supporter des contraintes sévères leur imposant une conception spécifique.

a.- Contraintes

a.1

L'apparition d' un nombre aléatoire d'événements à des moments aléatoires, nécessite une "puissance de traitement en temps réel" qui peut être représentée par le "nombre maximal d'appels à l' heure chargée" que le système de commande est susceptible de traiter avec des normes fixées de qualité de service.

On peut distinguer plusieurs niveaux de temps réels :

- Les contraintes en temps réel les plus sévères sont liées au traitement de la signalisation : la précision exigée sur certains signaux est de l'ordre de dix msec ;

- Un deuxième niveau concerne le reste des programmes de traitement des appels. Les délais de réponse aux actions des abonnés doivent rester inférieurs à des valeurs de l'ordre de la seconde pour que le service offert soit jugé satisfaisant ;

- Le troisième niveau correspond aux tâches d'exploitation et de maintenance, pour lesquelles les délais de réponse requis peuvent varier entre plusieurs secondes et plusieurs dizaines de minutes ou plus.

a.2

La permanence du service peut être représentée par la disponibilité D du système :

$$D = \frac{TMF}{TMF = TMR}$$

avec TMF = temps moyen de fonctionnement entre pannes;
TMR = temps moyen de réparation;

Les pannes pouvant survenir proviennent de défaillances du matériel ou du logiciel et la disponibilité peut être améliorée de la manière suivante :

- en augmentant TMF
  par une plus grande qualité de composants utilisés ou en apportant une redondance aux équipements par une duplication par exemple;

- en diminuant TMR
  par une amélioration de la maintenance du système du point de vue Hardware grâce à des programmes de test et de localisation automatique des défaillances et du point de vue Software grâce à des programmes de reconfiguration automatique.

Mais plus fondamentalement, dans un commutateur, alors qu'un certain nombre d'appels mal traités est tolérable, l'arrêt complet du système

c'est-à-dire l'impossibilité pour tout appel d'aboutir, est considéré comme catastrophique.

C'est ainsi que la maintenance et l'extension de capacité d'un commutateur doivent être possibles sans arrêter le traitement des appels (ce qui est le cas d'un centre de calcul) mais avec éventuellement une certaine dégradation du service.

a.3

L'ordinateur de commande fonctionne en multiprogrammation car plusieurs tâches doivent pouvoir être entreprises simultanément.

Les tâches les plus nombreuses concernent le traitement des appels avec un processus pour chaque communication en phase de conversation ou en cours d'établissement ou de rupture. Il est donc nécessaire de conserver en mémoire les contextes de ces processus dont un grand nombre, en attente d'un évènement extérieur, peut être réactivé à tout instant.

Chaque fois qu'un événement est attendu pour un appel, la tâche correspondante est désactivée et d'autres relatives à d'autres appels sont exécutées entretemps.

Lorsque l'événement attendu survient, la tâche initiale est réactivée et le traitement de l'appel se poursuit.

a.4

De nombreuses données doivent être maintenues et traitées par le logiciel opérationnel. Ces données peuvent être classées en deux grandes catégories :

- Les "données permanentes", peu fréquemment modifiées, décrivent le matériel du commutateur et son environnement.

- Les "données temporaires" , dont la durée de vie est celle d'une transaction, décrivent l'état des ressources, des liens temporaires entre ces ressources et les informations particulières à la transaction.

b. Conséquences

b.1

Les programmes de commutation sont organisés en plusieurs niveaux de priorité et sont optimisés pour répondre aux contraintes de temps réel tout en assurant la coexistence dans un même organe des fonctions de commutation, d'exploitation et de maintenance.

b.2

Un logiciel de défense chargé de faire face aux fautes matérielles et logicielles est introduit pour répondre à la contrainte de permanence du service.

b.3

Le logiciel devra procurer des facilités d'exploitation tant en ce qui concerne le logiciel opérationnel proprement dit que le logiciel de support.

b.4

La gestion de la mémoire devra être optimisée de manière à diminuer les coûts, qu'il s'agisse de mémoire de programmes ou de données. Ces caractéristiques spécifiques au logiciel de commutation seront implémentées grâce aux outils fournis dans les paragraphes suivants.

2.2.2.2 Techniques

A. Application

A.1- Explorateur ou Scanner

A. COMPOSITION DE L'ENSEMBLE

1. PROCESSEUR

Celui-ci est à la disposition de x Unités (U_i ; i=1 à N) en pseudo-simultanéité.

2. EXPLORATEUR

L'explorateur proprement dit palpe cycliquement les bornes ou les tampons d'état et/ou de données des n unités, de façon à gérer les échanges entre le processeur et chacune de ces unités. L'explorateur peut être de nature spatiale (figure T28) ou de nature temporelle (figure T29). Dans ce dernier cas, deux explorateurs synchronisés sont utilisés et la trame d'une voie commune contient autant d'intervalles de temps IT que de bornes ou d'unités.

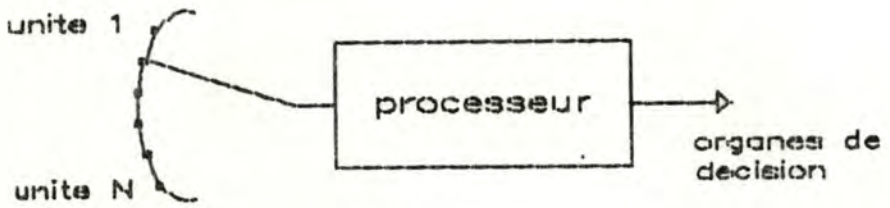


figure T28 : explorateur spatial

B. FONCTION

La fonction de scanning consiste à gérer les échanges de données en entrée ou en sortie, ceux-ci étant traités par le processeur.

figure T29 : explorateur temporel

Ces échanges peuvent être unidirectionnels, le plus souvent dans le sens des unités vers le processeur, pour permettre à ce dernier l'acquisition de ses données et la réalisation des traitements adaptés aux Organes de Décision tels que : Computer, Processus industriel ou autre; Enregistrement sur périphérique, Programme réentrant,...

Ces échanges peuvent être également bidirectionnels de telle sorte que l'on parlera :

- d'opération WRITE ou d'émission dans le sens processeur vers unités;

- d'opération READ ou de réception dans le sens unités vers processeur.

C. TYPES D'EXPLORATEUR

Les types d'explorateur sont ventilés selon la manière dont ils opèrent.

1. PAR SONDAGE OU POLLING

1.a en mode synchrone et de façon continue

L'explorateur teste successivement le niveau de potentiel (haut / bas représentant le 1/0 en logique positive et l'inverse en logique négative) de chaque borne (opération READ). Le processeur enregistre ces valeurs en mémoire et les compare aux valeurs enregistrées lors du passage précédent. Le résultat des comparaisons lui permettra de prendre une décision concrétisée par une action.

L'utilisation de ce mode est surtout adoptée pour le cas où des informations multiples sont contenues dans des impulsions ou des créneaux de potentiel c'est-à-dire des informations de type STIMULI.

Si t est le temps nécessaire pour tester chaque borne et si N est le nombre total de bornes à explorer, la période ou temps mis pour effectuer un cycle complet (t pour deux passages sur la même borne) vaudra N*t. Dans ces conditions, N*t devra être strictement inférieur à la durée du créneau le plus faible dans le spectre afin de ne pas manquer l'information qui y est contenue.

1.b en mode asynchrone ou synchrone arythmique

Dans ce mode, l'information relative à chaque unité est contenue dans un tampon T qui comportera au minimum un bit d'état (1/0 si T est vide/plein) ou un byte d'états si une analyse plus fine doit être faite.

La fonction essentielle de cet explorateur sera de tester le bit d'état et d'opérer un transfert de données selon l'algorithme de la figure T30.

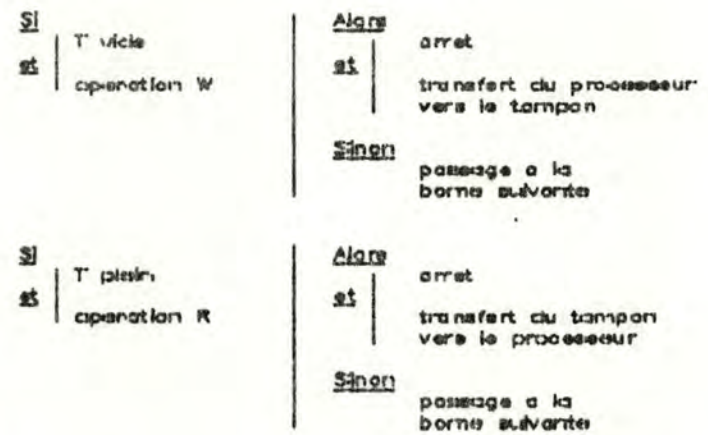


figure T30 : exploration des bornes

L'utilisation de ce mode d'exploration est adopté lorsque l'information est contenue dans des MESSAGES à un ou plusieurs bytes successifs et non plus dans des créneaux de potentiel étant donné que la période synchrone (t) des passages successifs

sur une borne quelconque est variable et ne peut donc calibrer l'information.

2. PAR INTERRUPTIONS

Une interruption :

- est générée si une opération READ c'est-à-dire un transfert d'une unité vers le processeur ou si une opération WRITE c'est-à-dire un transfert du processeur vers une des unités est demandée;

- est sollicitée par transfert de l'adresse de la borne à positionner pour une opération R/W sur l'une ou l'autre pile FIFO R ou W (temps de transfert sur la pile et mise à jour des pointeurs : t_j).

Entre les interruptions, l'une ou l'autre routine (R/W) effectuera les opérations successives, adresse après adresse, borne aprés borne (temps d'exécution : t_T).

Si le temps d'attente avant le traitement dépasse un temps limite tl (qui peut se traduire en une hauteur limite de la pile) il pourra mettre en péril, sinon le bon fonctionnement du système, du moins ses performances.

L'utilisation de ce mode d'exploration est adoptée lorsque l'information est contenue dans des MESSAGES, d'autant que pour le cas de plusieurs bytes, le traitement de l'interruption et le DMA sont automatiques.

D. COMPARAISON

Nous allons étudier les cas où il y aura lieu de choisir entre l'exploration par sondage asynchrone et l'exploration par interruptions.

1.SONDAGE ASYNCHRONE

Soient

N : le nombre total de lignes;
x : le nombre de lignes actives;
tt : temps de test d'une ligne;
tTRT : temps de traitement d'une ligne active;

Alors

le temps total pour une exploration est :

$t_t * N + t_{TRT} * x.$

2. INTERRUPTIONS

Soient

x : le nombre d'interruptions ou de lignes actives (en READ ou en WRITE);
ti : temps de positionnement de l'interruption et temps de traitement de celle-ci;

Alors

le temps total pour une exploration est :

$t_i * x.$

3. CONCLUSION

La figure T31 nous montre que le choix s'exprime en fonction d'un nombre de lignes actives.

L'exploration par INTERRUPTIONS sera préférée lorsque

- le temps de traitement d'une interruption ti est faible;
- le temps de test d'une ligne tt est élevé;
- le temps de traitement d'une ligne active tTRT est élevé.

L'exploration par INTERRUPTIONS sera toujours préférée lorsque

$$t_i \leq t_t + t_{TRT}.$$

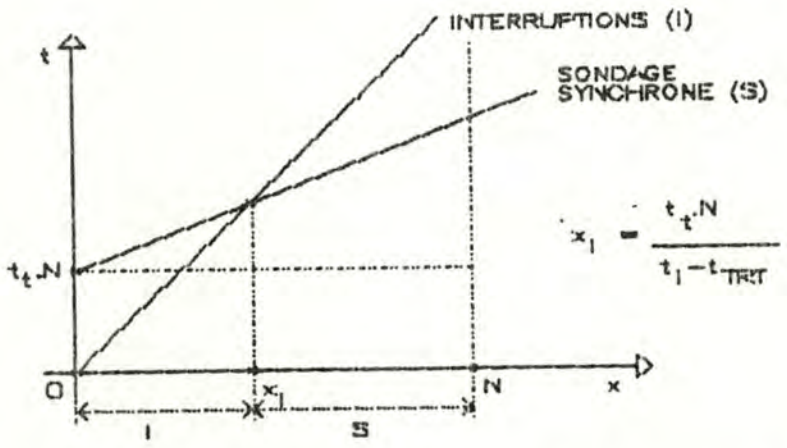


Figure T31 : comparaison des scanners par interruptions et par sondage asynchrone

L'exploration par sondage ASYNCHRONE sera préférée lorsque

- le temps de traitement d'une interruption ti est élevé;
- le temps de test d'une ligne tt est faible;
- le temps de traitement d'une ligne active tRT est faible.

L'exploration par sondage ASYNCHRONE sera toujours préférée lorsque

$t_t = 0$ et $t_{TRT} > t_i$.

D. SPECIFICITE DE L'EXPLORATION EN COMMUTATION

1. FONCTIONS ET CONTRAINTES

L'exploration est la fonction par laquelle l'Unité de Commande (UC) receuille les différents évènements générés par l'environnement du commutateur. Cet environnement constitué des périphériques que sont les ressources et les équipements de ligne, est en général passif, c'est pourquoi l'UC doit elle-même receuillir les différentes informations nécessaires aux différentes phases du traitement d'un appel.

Il s'agit donc de traduire en un signal logique utilisable par l'UC, un signal ou un code analogique ou numérique apparaissant sur une ligne d'abonné ou une ressource.

Les signaux receuillis associés éventuellement avec des informations complémentaires fournissent à l'UC l'état logique dans lequel se trouvent la ligne et la conversation.

Après avoir traité ces données, l'UC sera à même de réagir sur la périphérie par l'envoi d'ordres. Cette opération constitue la distribution.

En plus de cette fonction de base d'interface électrique entre l'équipement de ligne et l'UC, l'Explorateur de Lignes d'Abonnés (ELA) se voit confier des tâches nécessitant un certain degré d'autonomie comme la détection des changements d'états. Ce travail autonome est d'autant plus intéressant qu'il soulage l'UC d'une fonction très simple mais répétitive et, par la même, couteuse en temps. En fait, le nombre d'évènements arrivant pendant un cycle d'exploration est extrêmement faible.

2. PRINCIPE DE FONCTIONNEMENT

2.a

Le principe de fonctionnement de l'ELA est d'explorer un à un, cycliquement, tous les abonnés dont il a la charge. Lorsqu'un changement d'état est intervenu sur l'abonné examiné, il interrompt son exploration, en avertit l'UC, puis il reprend son exploration à l'endroit où il s'était arrêté.

Pratiquement, deux types de signalisation sont utilisés.

Dans la "signalisation par changement d'état", l'explorateur communique chaque changement d'état qu'il détecte à l'UC qui prendra les dispositions voulues. Après la conversion électrique des signaux réalisée à l'entrée du commutateur par des adaptateurs de signalisation, la détection des changements d'états est une opération logique - combinatoire pour le temporel- très simple à réaliser.

Dans la "signalisation par état", l'explorateur détecte l'état de la ligne qu'il communique à l'UC. Cette transmission sera renouvelée jusqu'à ce qu'une modification ait été apportée soit par l'abonné, soit par l'UC.

2.b

L'explorateur doit aussi pouvoir interrompre ce travail autonome pour répondre à une demande de l'UC concernant l'état d'un abonné dont elle lui fournit l'identificateur. Une fois la réponse fournie, l'exploration autonome reprend là où elle s'était arrêtée.

2.c

Le cadencement de l'exploration peut être adapté au type d'événement à superviser de façon à ne pas allonger outre mesure le travail de l'explorateur ou de l'UC.

La réception de la numérotation en téléphonie, par exemple, impose une période de temps relativement courte entre deux cycles d'exploration, si l'on ne veut pas perdre un quelconque des changements d'états qui peuvent être séparés de 30 ou 40 msec. On trouve donc pour ces points, des cycles d'exploration espacés de 5, 10 ou 20 msec suivant les systèmes.

Par contre, des événements tels que décrochage ou raccrochage d'abonnés ne nécessitent pas une exploration à cadence rapide puisque les fonctions qui s'ensuivent autorisent des temps de réponse relativement longs (quelques centaines de millisecondes). Des cadences de 80 ou de 100 msec sont suffisantes pour l'exploration de ce type d'événement.

A.2 Les Automates finis.

A.2.1. Définition.

L'automate fini est un être mathématique pour lequel la réponse à un événement extérieur dépend de cet événement et de l'état interne de l'automate. Un automate fini a un nombre fini d'états internes possibles. Les événements sont susceptibles de faire passer l'automate d'un état à un autre. L'automate fini est entièrement déterminé par la donnée de ses fonctions de transition qui fournissent le nouvel état et la réponse,ceci en fonction de l'ancien état et de l'événement reçu.

A.2.2. Les états internes d'un automate fini.

L'étude des automates finis permet de donner un support théorique à de nombreux développements liés à l'informatique: étude des circuits, des algorithmes, des langages formels, des techniques de compilation,...etc.

L'étude proposée se limite à voir un automate comme une boîte noire comportant des entrées et des sorties.

Nous représentons les entrées ou événements par la lettre E ; les sorties ou réponses par la lettre S.

Nous regardons l'automate à des instants successifs discrets : t, t+1 , t+2 , ...

Un événement E à l'instant t, noté E(t) produit une réponse S à l'instant t+1, notée S(t+1).

S(t+1) dépend de E(t) mais aussi d'une fonction du temps H(t).

$$S(t+1) = F\left[H(t)\ ,\ E(t)\right] \qquad (1)$$

Pour parvenir à définir les états internes ou états d'un automate, il faut que H(t) réponde à deux hypothèses :

- le première est que la fonction H(t) est complètement déterminée par l'histoire de l'automate antérieur à l'instant t.
- la seconde est que l'automate est constitué d'un nombre fini d'éléments, donc d'un nombre fini d'éléments de mémoire, de telle sorte qu'il ne peut se souvenir d'une histoire arbitrairement longue, ni faire des distinctions parmi un nombre arbitrairement grand d'histoires différentes.

Ceci veut dire que la fonction H(t) ne peut en réalité prendre qu'un nombre fini de valeurs distinctes qu'on appelle ETATS INTERNES ou ETATS de l'automate.

Les automates vérifiant ces hypothèses sont appelés AUTOMATES FINIS.

A.2.3. Les fonctions de transition.

Soit Q(t) : l'état d'un automate fini à l'instant t, l'équation (1) s'écrit :

$$S(t+1) = F\left[Q(t)\ ,\ E(t)\right] \qquad (2)$$

ce qui exprime que la réponse d'un événement E(t) ne dépend que de l'état à l'instant t et non de l'histoire antérieure.

De même, l'état Q(t+1) de l'automate à l'instant t+1 ne peut dépendre que de son état antérieur Q(t) et de ce qu'il reçoit de l'extérieur à savoir E(t). Sinon, l'automate pourrait distinguer dans son comportement ultérieur deux histoires ayant abouti au même état Q(t) , ceci étant contraire à l'hypothèse. De là :

$$Q(t+1) = G\left[Q(t)\ ,\ E(t)\right] \qquad (3)$$

Les équations (2) et (3) permettent de calculer étape par étape le comportement d'un automate fini pour n'importe quelle suite d'événements E(t), E(t+1), E(t+2), ... qu'il peut recevoir à partir de l'instant t.

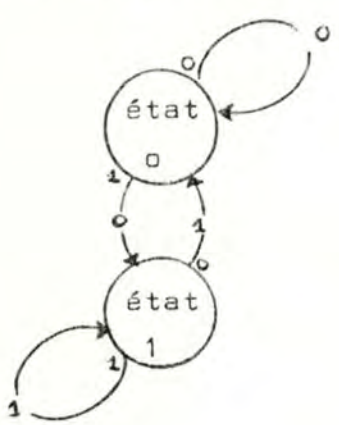
Les fonctions F et G sont les FONCTIONS DE TRANSITION et se représentent sous forme de tables de transition ou sous forme de diagrammes de transition.

A.2.4. Exemple d'automate fini.

Considérons un automate à deux états possibles (o ou 1) ; en fonction de ces états, il produit une sortie de même valeur que l'état modifié suite à l'arrivée d'un événement (o ou 1).
En voici la table de transition :

événement	état(t)	état(t+1)	sortie(t+1)
o	o	o	o
o	1	o	o
1	o	1	1
1	1	1	1

En voici le diagramme de transition :



Les flèches représentent les transitions entre états. Au départ de chaque flèche, une première information indique les valeurs d'entrée qui ont entraîné la transition; au milieu de la flèche, une deuxième information indique les valeurs de sortie produites.
Par exemple, la flèche supérieure du diagramme indique que s'il est à l'état o et reçoit une entrée o, l'automate retourne à l'état o en produisant une sortie o.

A.2.5. Application aux commutateurs.

La notion d'automate fini est très utile à la conception d'un commutateur numérique et temporel. En effet, ce type de commutateur est programmé, ce qui signifie que l'on a besoin d'un outil informatique particulier(l'automate) pour prendre en compte et répondre aux événements survenant de la ligne de l'abonné, détectés par scanning. La réponse (suite d'instructions à exécuter) se rapporte à l'état du commutateur après la détection de l'événement. Une large illustration est donnée dans le second syllabus où l'on détaille les analyses fonctionnelles et organiques pour une séquence téléphonique. Ces analysent contiennent, en outre, deux automates.

B) Système d'Exploitation

a.- Gestion du Temps

A. INTRODUCTION

Dans des applications classiques, l'utilisation du temps peut habituellement être organisée en détail lors de la conception des programmes. Ce n'est pas le cas des systèmes en temps réel pour lesquels des événements arrivent à des moments qui ne peuvent être connu exactement a priori. Par exemple, les durées de flux d'entrée/sortie, et en particulier des références à des fichiers à accès aléatoire, ne sont pas connus exactement. Les messages des opérateurs humains sont entrés à des temps dépendant des événements humains. La séquence des types de messages ou des fonctions à réaliser peut être imprévisible.

A cause de ce manque de prévisibilité, l'ordinateur ne peut suivre un cycle d'événements répétitif mais doit allouer le temps aux différentes unités selon les besoins. Cette allocation de temps peut être faite de manière simple, périodique ou peut être très complexe avec un haut degré de multiprogrammation et de recouvrement d'opérations. Dans certains systèmes pour lesquelles l'utilisation de l'unité de traitement ne nécessite pas un timing sévère, il n'est pas nécessaire d'élaborer le timing des opérations en détail. Néanmoins, sur d'autres systèmes comme les commutateurs, le timing est extrêmement précis et important, ce qui nécessite une attention accrue aux problèmes d'allocation du temps.

B. SPECIFICITE DU CARACTERE ALEATOIRE

Comme nous l'avons évoqué, pour des applications informatiques conventionnelles, le programmeur a défini l'utilisation du temps (et de la mémoire) en détail lors de la conception des programmes. Sur la plupart des systèmes on/line, il ne peut le faire à cause du caractère aléatoire des événements.

Ainsi, les temps suivants sont imprévisibles :

B.1
Les moments auxquels les événements externes se produisent. Par exemple, le moment où un abonné téléphonique décroche son combiné.

B.2
Les moments où ont lieu les interruptions d'horloge. Une horloge peut être initialisée pour que l'ordinateur exécute une action à un moment donné et elle peut interrompre le programme en cours en un point quelconque.

B.3
Les temps de réalisation d'instructions d'entrée/sortie, en particulier et pour rappel, les accès à des fichiers organisés aléatoirement.

Sur un système en temps réel, les messages arrivant à l'ordinateur en provenance d'opérateurs humains ou d'appareils automatiques nécessitent une réponse rapide. L'allocation du temps, de la mémoire centrale, des canaux, des périphériques et autres ressources doivent, pour ce faire, être contrôlés rapidement par le système d'exploitation.

C. HORLOGE TEMPS-REEL ET COMPTEUR D'INTERVALLES

La majorité des systèmes en temps réel et beaucoup de systèmes on/line mais pas temps réel ont une horloge temps réel ou un compteur d'intervalles, tout en sachant que certains de ces systèmes n'en ont pas besoin. Certains ordinateurs, par contre, ont une horloge temps réel et un compteur d'intervalles séparé. D'autres ont même plusieurs compteurs d'intervalles.

De tels mécanismes de timing fonctionnent normalement, indépendemment de ce que fait l'ordinateur. L'horloge temps réel peut être "lue" par l'ordinateur, ce qui permet à celui-ci de connaître le temps écoulé depuis la dernière initialisation de l'horloge. Le compteur d'intervalles peut être initialisé par l'ordinateur pour calculer, attendre un intervalle de temps donné, au terme duquel, le compteur d'intervalles interrompt l'ordinateur.

Ces serveurs sont utilisés pour quatre types de fonctions :

C.1 donner l'heure de la journée.

Ceci permet à l'ordinateur d'indiquer l'heure sur certains messages, par exemple.

C.2 calculer un intervalle de temps donné.

L'ordinateur peut devoir exécuter une certaine action si un événement n'apparait pas pendant un certain intervalle de temps, par exemple, si un abonné téléphonique, après avoir décroché son téléphone, ne forme pas un chiffre dans les 30 secondes. Dans un système en temps partagé, il peut être nécessaire de rendre le contrôle au système d'exploitation toutes les 200 msec et assurer ainsi des temps de réponse normaux à chaque utilisateur.

C.3 lancer des actions à des moments donnés.

L'ordinateur peut avoir à envoyer un certain message à une heure du jour particulière. Périodiquement, il va explorer une table d'actions à exécuter en fonction d'une heure précise. De même, l'ordinateur peut décider, pendant son traitement "si jeudi prochain, à 16 heures, tel ou tel événement n'a pas été reçu, un message d'erreur doit être envoyé". Il doit alors mettre ce nouvel événement dans la table. Parfois, une interruption d'horloge est utilisée pour lancer une action, à laquelle une haute priorité sera donnée par le système d'exploitation.

C.4 comme "chien de garde".

Pour protéger l'ordinateur de mauvais fonctionnements, surtout de la part des programmes, un timer "chien de garde" est nécessaire, aussi bien d'ailleurs en ce qui concerne la protection de la mémoire. C'est particulièrement le cas des systèmes multi-tâches. Si un des programmes commençait à boucler, cela pourrait empêcher l'exécution des autres tâches en attente. Un compteur d'intervalles peut alors être initialisé, comme dans la figure T32, chaque fois que la routine de scheduling est appelée. Il calcule un temps donné, disons 500 msec, à moins qu'il ne soit réinitialisé avant l'écoulement de ces 500 msec par un retour au scheduler. Si 500 msec s'écoulent sans que le contrôle ait été rendu au scheduler, soit quelque chose ne va pas, soit un programmeur a désobéi aux règles, et une interruption

est lancée. Comme pour une violation de protection mémoire, le contrôle est alors donné à un programme prioritaire qui traitera le problème.

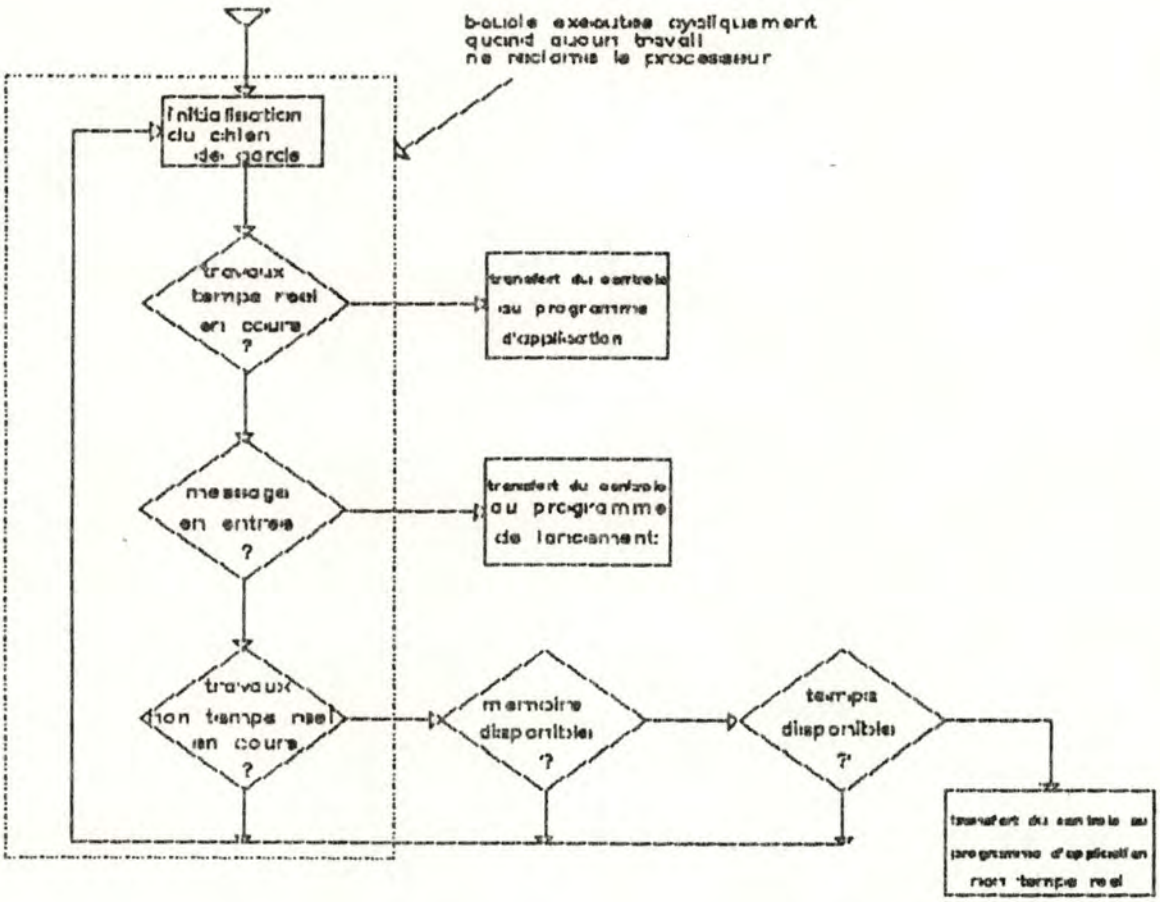


figure 7.3.2 : utilisation du chien de garde

Si la machine a un compteur d'intervalles, il est possible de programmer son utilisation de manière à réaliser toutes les opérations mentionnées ci-dessus. Il est plus facile, néanmoins, d'utiliser une horloge et un compteur d'intervalles séparé. Quand plusieurs périodes de temps doivent être calculées en parallèle, plusieurs compteurs doivent être utilisés. Si un seul compteur est disponible, le programme du système d'exploitation appelé Gérant du Temps contrôlera les différents timers en parallèle.

B.2 Gestion des processus.

B.2.1 Définition.

Nous désignons par processus toute suite d'actions obtenues par l'exécution d'une séquence d'instructions (un programme) dont le résultat constitue une des fonctions de l'operating system ou une des fonctions de l'utilisateur.

Remarques :

- un processus peut entraîner l'exécution de plusieurs programmes
- un programme ou module peut être impliqué dans plusieurs processus
- le processus étant une suite d'actions, il est essentiellement dynamique.
- le programme étant une suite d'instructions il est essentiellement statique.

B.2.2 La simultanéité.

La survenance d'événements aléatoires provenant de la ligne d'abonné entraîne la nécessité de réponses rapides de la part du système. Cette vitesse de réponse ne peut être obtenue que par la simultanéité dans l'exécution des processus représentatifs des réponses. Cette simultanéité consiste en une exécution en parallèle des processus. Elle n'est réelle que lorsque l'on dispose d'un nombre de processeurs supérieur ou égal au nombre de processus. sinon, on parle de pseudo-simultanéité. Dans notre étude, nous sommes confrontés au second cas, ne disposant que d'un seul processeur pour implémenter notre maquette.

Cette pseudo-simultanéité peut être obtenue en basculant le(s) processeur(s) d'un processus à l'autre. Si le basculement est réalisé à des intervalles suffisamment courts, le système donne l'illusion de la simultanéité.

Le système d'exploitation doit posséder des fonctions de sélection de processus (après le basculement) et de sauvetage du contexte dans lequel le processus se trouvait lorsqu'on a basculé.

B.2.3 Le scheduling et le context switching.

Le context switching et le scheduling consistent à interrompre le processus courant, en sauvant suffisamment d'informations qui permettera à ce processus d'être réactivé par après, et activer un autre processus sélectionné dans le souci de respecter le principe de simultanéité.
Le système d'exploitation doit utiliser une structure de données organisée, la table des processus pour réaliser ces deux fonctions. Cette table contient toutes les informations concernant les processus qui permettent à l'operaying system d'effectuer les opérations de context switching et de scheduling.
Elle peut être vue comme une liste chaînée de descripteurs de processus. Un descrpteur de processus contient:

- l'état du processus qui peut être ACTIF s'il est exécuté sur le processeur;
  ACTIVABLE si un processeur peut lui être alloué;
  NON ACTIVABLE dans les autres cas.
- la priorité du processus.
  En effet, il est intéressant de pouvoir fournir pour un processus une priorité qui valorise l'intérêt que l'on porte à voir l'exécution d'un processus plutôt qu'un autre.
- l'environnement volatile.
  Le processus avant d'être interrompu a positionné les registres de la machine

à certaines valeurs. Si on veut pouvoir le réactiver à partir de l'endroit de l'interruption, il est nécessaire de stocker ces registres.

B.2.4 Gestion des processus dans le 1240.

Dans l'implémentation de notre maquette, nous appliquons les principes d'une gestion de processus en monoprocesseur.
Il nous paraît intéressant de présenter une gestion de processus en multi-processeur comme celle qui est implémentées dans le commutateur 1240.

A. Fonctions synchrone et asynchrone.

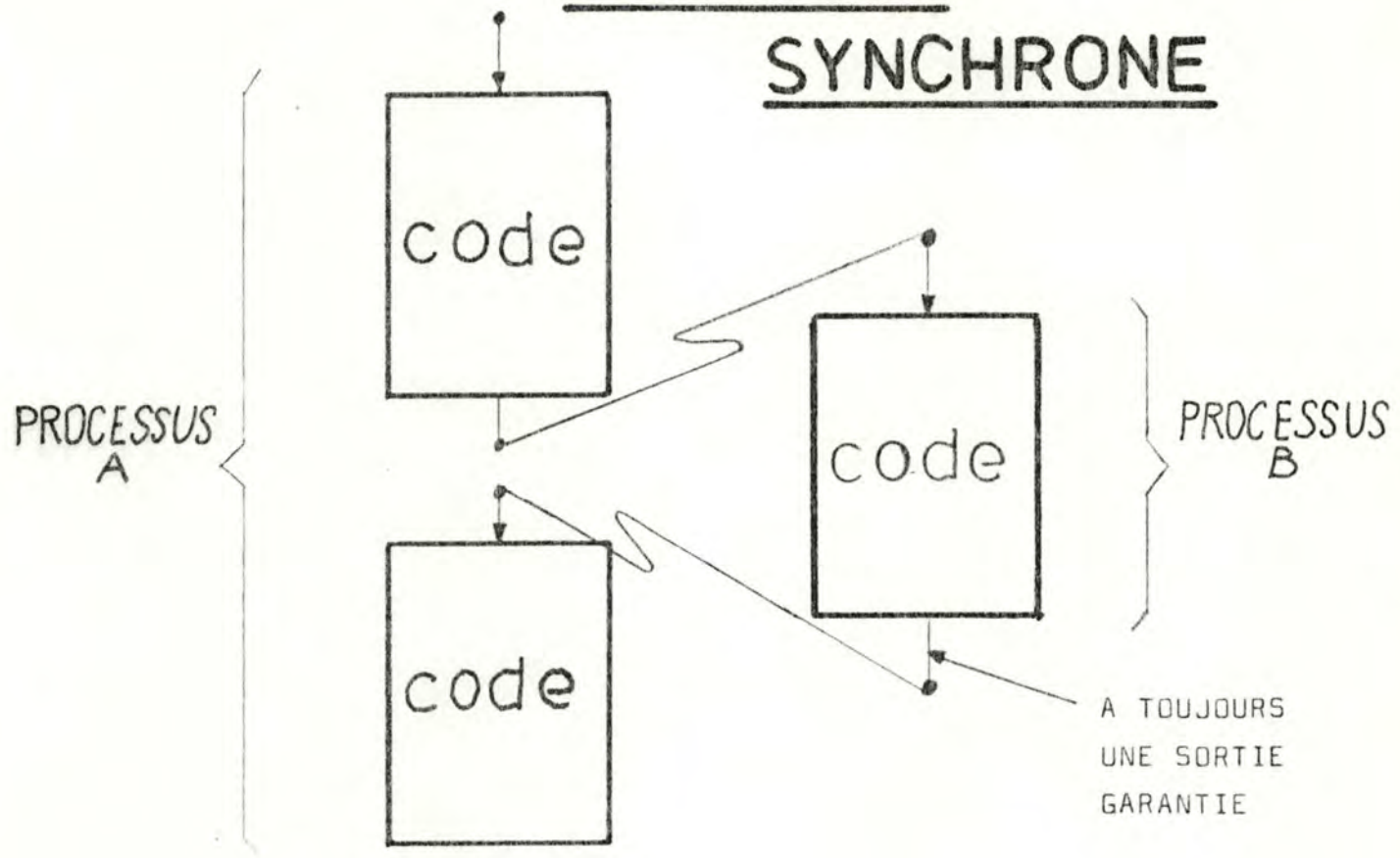
Toutes les fonctions exécutées par un système quelconque peuvent être séparées en deux catégories principales, soit:

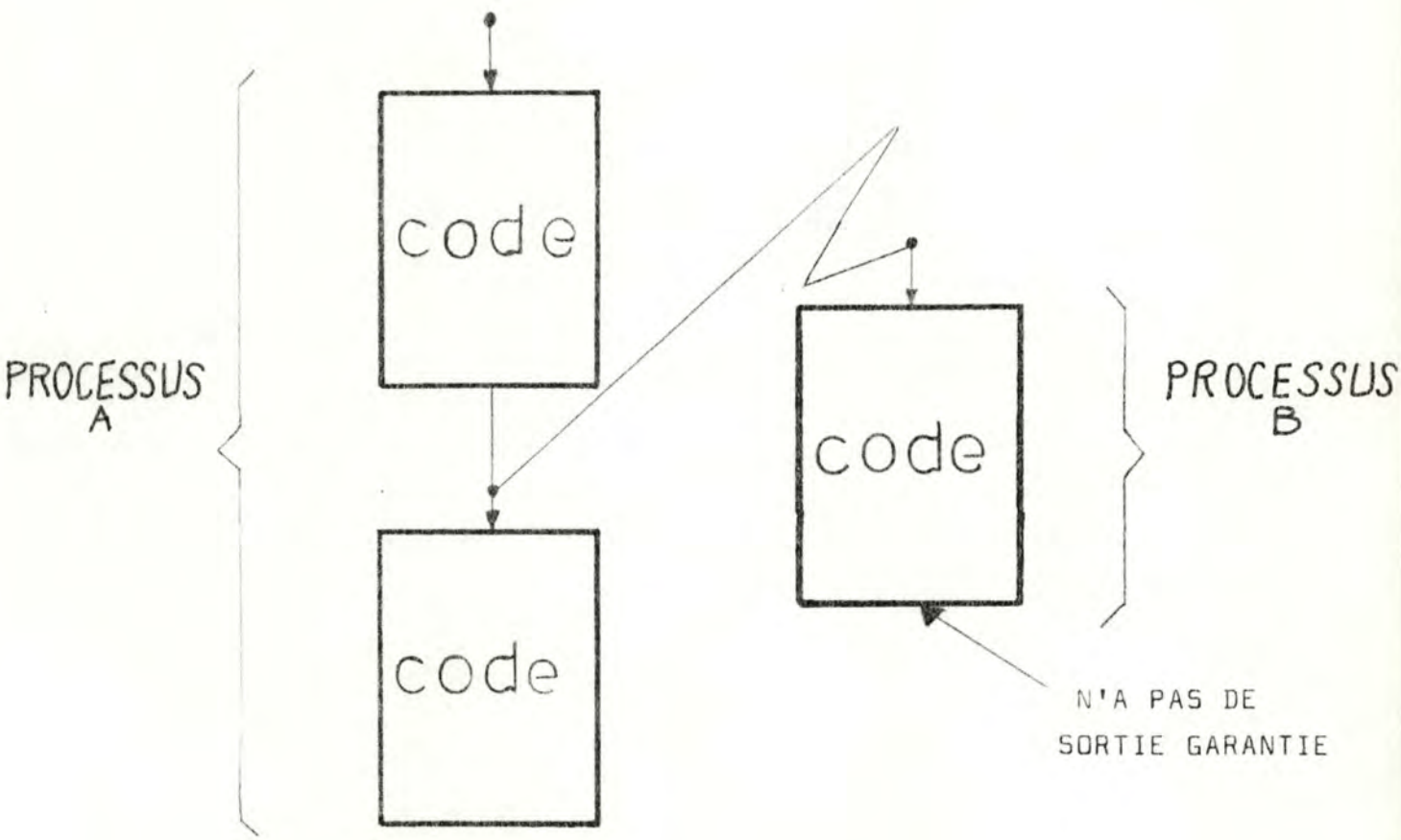
- les fonctions synchrones ou de nature séquentielle;
- les fonctions asynchrones ou de nature simultanée.

Un processus synchrone, lorsqu'il est activé, provoque un arrêt des activités du processus en cours, processus qui l'a activé. Cet arrêt est maintenu jusqu'à ce que le processus synchrone ait terminé son exécution. A ce moment le processus renvoie la commande du processus d'activation.
Contrairement au processus synchrones, lorsqu'il est activé,le processus asynchrone est capable de poursuivre son exécution en parallèle avec le processus d'activation.
Un conséquence immédiate évidente, dérivant d'une structure software fournissant la possibilité d'une exécution parallèle, est que les processus activé et activant peuvent résider dans des processeurs physiquement séparés. Il sera évident qu'un résultat important, dérivant de l'aptitude à utiliser plus d'un processeur pour une tâche

donnée, est la possibilité d'amélioration du rendement du système.
Pour des processeurs synchrones, on voit qu'il n'y a pas de gain de rendement en plaçant le processus A et le processus B dans des processeurs différents. Pour des processus asynchrones par contre, il pourrait y avoir des avantages significatifs.
Les processus de fonctions asynchrones dans un système sont indépendants de tous les autres processus dans ce système quoiqu'ils doivent naturellement communiquer avec d'autres processus.
Un processus asynchrone dans un système peut être activé par un processus n'importe où dans le système. Pour ce faire, le processus d'activation doit connaître de façon explicite l'idendité du processus asynchrone.

FONCTION SYNCHRONE





FONCTION ASYNCHRONE

B. Caractéristiques d'un FMM

Une FMM:

- est une entité asynchrone indépendante.
- est un "black box" vu de l'extérieur.
- doit être activée pour fonctionner.
- peut nécessiter des paramètres d'entrée en relation avec son activation.
- possède un ensemble fini non vide de messages d'entrée.
- possède un ensemble fini non vide de messages de sortie.
- répond à tout message d'entrée en période d'activation uniquement.
- peut être activé indéfiniment.
- peut uniquement être désactivée contrainte et non fonctionnelle(la désactivation peut se produire par défaut du système.
- peut activer d'autres FMM.
- le comportement fonctionnel est complètement spécifié en termes de séquences de messages (reçus et émis).

C. Messages

Un message est un moyen d'intercommunication entre objets asynchrones.

Un message:

- peut porter de l'information.
- est créé par une opération de transmission de messages.
- cesse d'exister sans aucune réponse, si dirigé vers une FMM non active ou non existante.
- existe jusqu'à l'acceptation explicite par la FMM de destination s'il est actif.
- peut être mis en file d'attente.
- peut être prioritaire par rapport à d'autres messages.

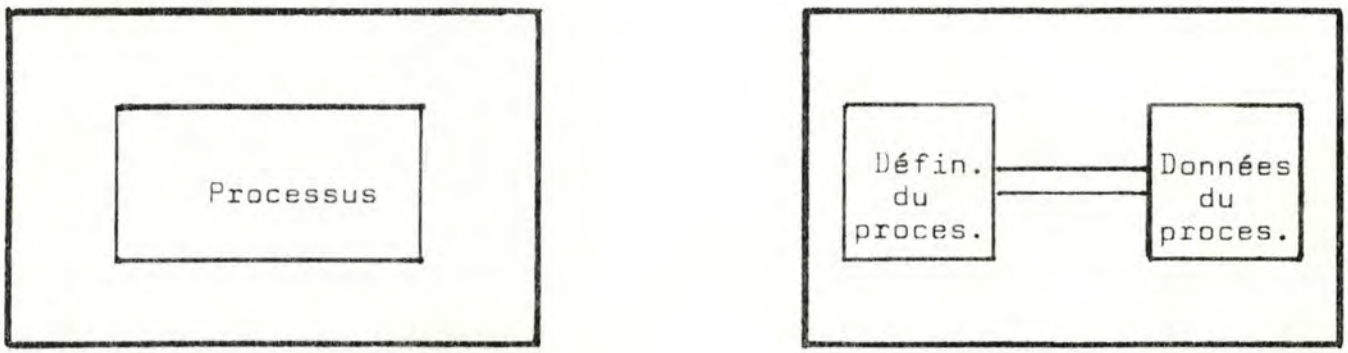
D. Structure d'une FMM.

Dans des systèmes à temps réel un grand nombre d'appareils identiques fonctionent en parallèle et de façon asynchrone, par exemple des jonctions, des récepteurs de chiffres, des appareils de signalisation, des unités de commande de mémoires auxiliaires. On devra souvent faire exécuter par le système un certain nombred'actions pendant la même période de temps mais pas nécessairement au même moment, par exemple, le chargement.
Une FMM peut être conçue pour la logique qui fonctionne sur l'un des appareils ci-dessus et qui est utilisée pour effectuer les opérations en parallèle sur un certain nombre de ces mêmes appareils. Néanmoins, chaque appareil séparé nécessitera des données différentes.
La structure la plus logique à utiliser pour un tel système est une structure dont le programme commun définit la séquence logique des opérations. Une aire de données est associée à ce programme pour contenir les données nécessaires à son exécution. Un programme commun et son aire de données est appelé un processus lorsqu'il est exécuté.

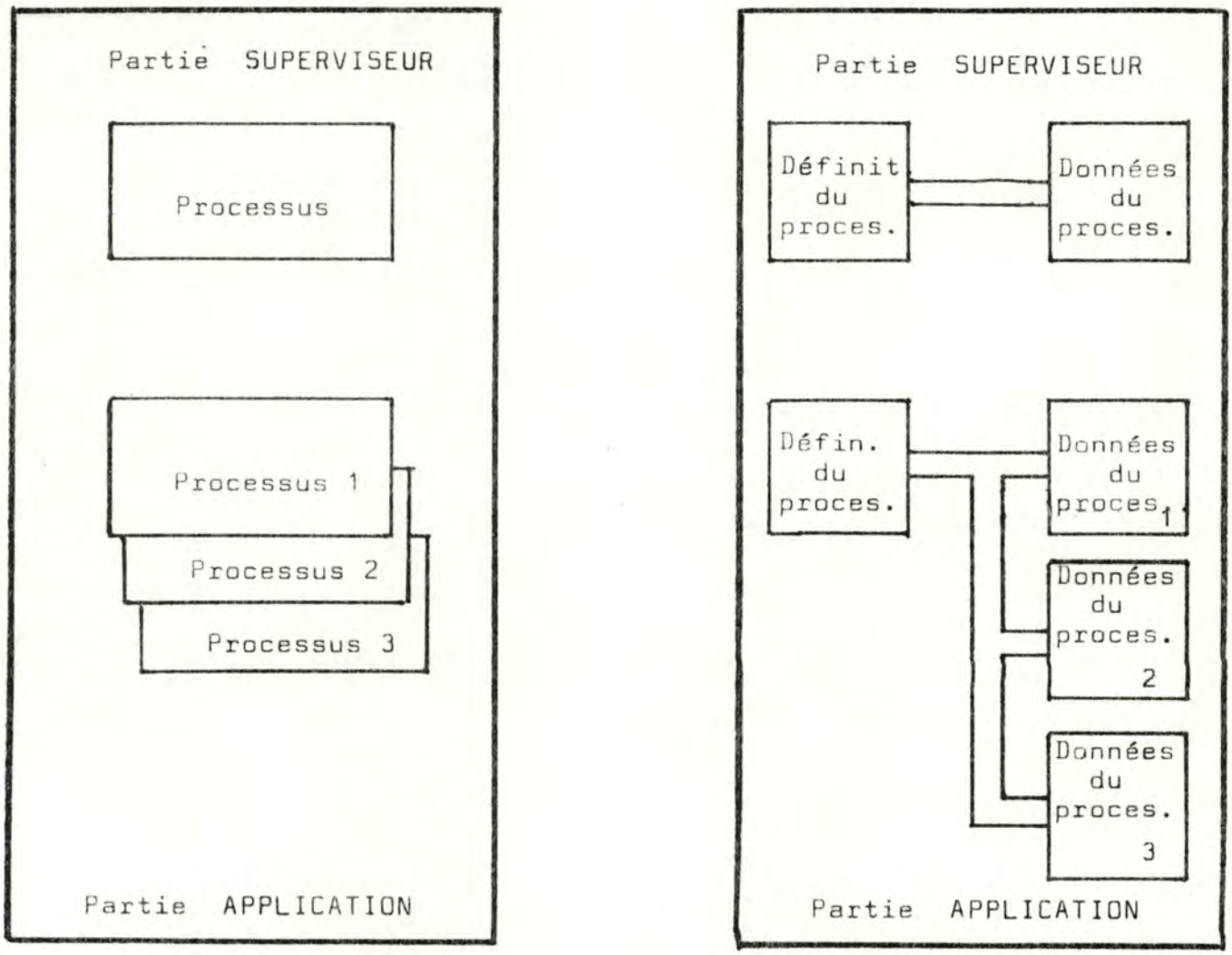
Une FMM construite de cette façon est appelée un multiprocessus FMM et la logique est divisée en deux parties:

- une partie superviseur
- une partie application.

Deux multiprocessus peuvent être basés sur la partie application; un processus simple est basé sur la partie supervision dont la fonction est de contrôler les processus d'application.
Les processus individuels d'une FMM multiprocessus fonctionnent de façon asynchrone les uns par rapport aux autres.
D'autres FMM dont la fonction est telle qu'elle ne requiert pas de processus multiples sont considérées comme des FMM monoprocessus.



MONO PROCESSUS FMM



MULTI PROCESSUS FMM

E. Partie superviseur d'une FMM

La partie superviseur a les caractéristiques suivantes:

- elle consiste uniquement en un processus simple.
- elle a un point d'entrée défini au moment de la production du software. Ce point d'entrée est utilisé par l'operating system pour la réception du message d'initialisation.
- elle peut envoyer et recevoir des messages.

Elle a les fonctions suivantes:

- elle reçoit tous les messages de base envoyés à la FMM et soit les absorbe, soit les transforme en messages dirigés vers les processus d'application nouvellement créés.
- si nécessaire, elle commande la création de processus d'application dans le FMM; à cette fin, elle gère tous les aspects de la création de la partie application y compris, en cas de nécessité, la détection de surcharge et la réponse à donner à l'enregistrement et le compte rendu des statistiques de création de processus.
- elle effectue la gestion des ressources appartenant intrinséquement à la FMM.

F. Partie application d'une FMM

La partie application d'une FMM a les caractéristiques suivantes:

- ces processus doivent être créés à la demande de la partie superviseur. La limite supérieure du nombre statique ou dynamique des processus qu'elle comprend est modifiable.
- ces processus peuvent envoyer et recevoir des messages dirigés.
- ces processus peuvent envoyer mais non recevoir des messages de base.
- un processus peut se terminer.

B.3- Gestion de la Mémoire

A. INTRODUCTION

Dès l'instant où le programme à exécuter par un ordinateur n'est pas entièrement et définitivement chargé dans la mémoire pour toute la durée de l'exploitation, se pose le problème de gérer les demandes d'allocation et de désallocation des places en mémoire.

Néanmoins, la logique et la complexité de la gestion de la mémoire découlent des objectifs généraux assignés au système d'exploitation, objectifs déterminés en fonction des exigences de l'application.

En commutation, ces exigences ont été longuement développées dans les points précédents et ont conduit naturellement à des choix quant à la politique de gestion de la mémoire, quant à l'allocation des blocs mémoire.

En effet, certaines politiques peuvent être rejetées immédiatement de par la nature radicalement opposée aux exigences basées sur le temps réel. Ainsi, l'allocation unique et statique par processus d'une part de mémoire préalablement définie est inadéquat. Cette méthode suppose que le découpage de la mémoire défini a priori par rapport aux processus convienne suffisamment. Or, en commutation, les demandes des processus sont imprévisibles et fréquentes, rendant cette méthode difficile à maîtriser. De même, une allocation permanente, c'est-à-dire sans préemption, où le processus garde le ou les blocs de mémoire reçus jusqu'à ce qu'il n'en ait plus besoin, n'est pas envisageable.

En conclusion, nous étudierons deux grandes classes de systèmes de gestion de la mémoire compatibles avec notre type d'application. Ces classes sont :

- les systèmes à allocation dynamique de la mémoire;
- les systèmes à allocation avec préemption.

B. ALLOCATION DYNAMIQUE DE LA MEMOIRE

Nous avons besoin d'algorithmes de réservation et de libération de blocs de mémoire de taille variable à partir d'une plus grande zone de stockage, où ces blocs consistent en cellules mémoire contiguës. Ces techniques sont généralement appelées des algorithmes d'allocation dynamique de mémoire.

1. RESERVATION

Supposons l'état de la mémoire partitionnée en blocs réservés, c'est-à-dire utilisés, et en blocs libres ou disponibles (figure T33).

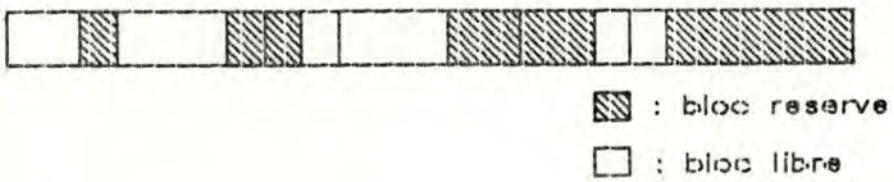


figure T33 : etat de la memoire

Le problème est :

a) comment représenter l'espace libre partitionné ?

b) étant donné une certaine représentation des espaces libres, comment trouver un bloc de n cellules libres contiguës et le réserver ?

La réponse à la question (a) est, bien sûr, d'établir une liste de l'espace disponible. Les blocs disponibles sont liés entre eux : le premier mot de chaque bloc contient un flag indiquant s'il est libre ou non, la taille du bloc, un pointeur vers le bloc suivant dans la liste libre et, éventuellement, un pointeur vers le bloc précédent (figure T34).

Les blocs libres peuvent être reliés par ordre de taille croissant ou décroissant, par ordre d'adresse en mémoire ou en ordre aléatoire.

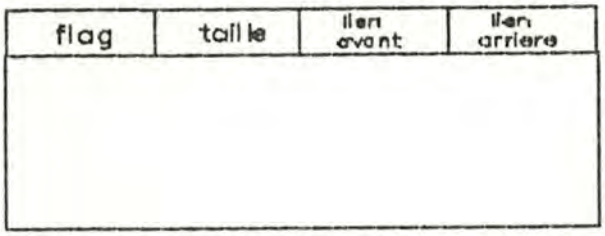


figure T34 : bloc libre

Pour la question (b), si nous voulons n cellules contiguës, nous devons repérer un bloc de m = n cellules libres et réduire sa taille à m - n. Il peut y avoir plusieurs blocs de n cellules ou plus et la question devient quel bloc choisir ?

Deux méthodes apparaissent tout de suite : la première (first-fit algorithm) consiste à rechercher le premier vide convenable c'est-à-dire un bloc libre de taille m supérieure à la demande n; la seconde (best-fit algorithm) consiste à rechercher le vide qui convienne le mieux à la demande c'est-à-dire le plus petit bloc libre de taille m1 supérieure à la demande n.

$$n <= m_1 <= m_2 <= \ldots <= m_j$$

Cette dernière méthode nécessite une recherche dans toute la liste des blocs libres avant qu'une décision soit prise.

Elle apparait naturellement comme une bonne politique étant donné qu'elle garde disponible les plus grands blocs possibles pour une utilisation future. Néanmoins, cette méthode soulève plusieurs objections :

- elle est relativement lente, lenteur dûe à une longue recherche;
- elle tend à augmenter le nombre de blocs très petits, ce qui n'est évidemment pas souhaitable.

2. LIBERATION

Considérons maintenant le problème inverse : comment retourner des blocs vers l'espace libre losqu'on n'en a plus besoin ?

Nous allons tenter de résoudre ce problème par la technique du "Garbage Collector". Cette technique utilise dans chaque bloc un flag d'état (figure T34). Quand un bloc est libéré, on met simplement à jour le flag d'état. Tant que de l'espace libre peut être alloué aux demandes, on ne

fait rien. Quand ce n'est plus le cas, le Garbage Collector renvoie vers la liste des blocs libres tous les blocs occupés dont le flag est dans l'état libre.

Cette technique apporte beaucoup de lenteur lorsque la mémoire est presque toute occupée. De plus le phénomène suivant en est une conséquence. Supposons deux blocs adjacents disponibles dont l'un d'eux n'est pas encore dans la liste libre (figure T35.a).

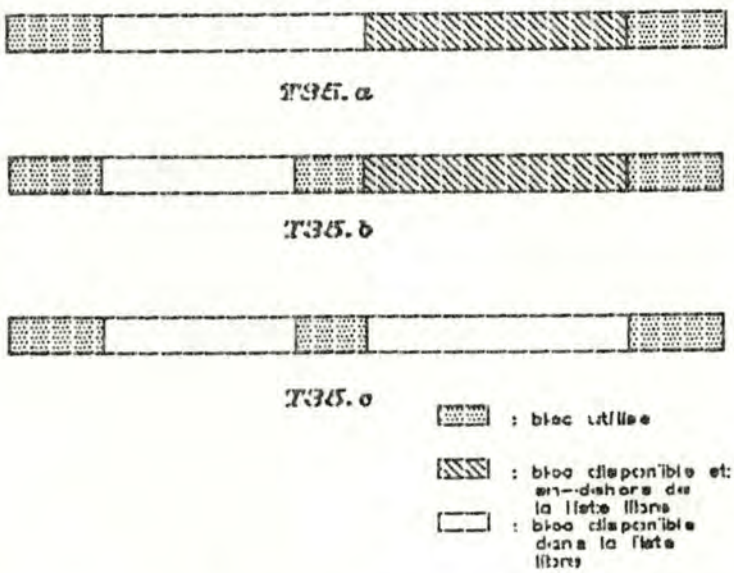


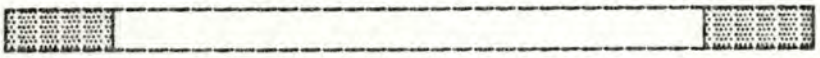
figure T35 : conséquence du Garbage Collector

Si l'on réserve une section du bloc disponible de la liste libre, la situation risque de devenir telle qu'illustrée par la figure T35.b. Si maintenant, le Garbage Collector intervient, nous aurons deux blocs libres séparés comme le montre la figure T35.c. Les frontières entre les zones disponibles et réservées ont tendance à se perpétuer d'elles-mêmes, et la situation empire avec le temps.

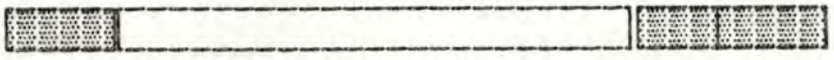
Par contre, si nous retournons dans la liste libre les blocs dès qu'ils sont libérés et si nous rassemblons les zones libres adjacentes, nous aurions obtenu la situation de la figure T36.

Pour contourner cette difficulté, il est possible d'utiliser en même temps que le Garbage Collector, un tassement de mémoire qui consiste à rassembler l'un à la suite de l'autre les blocs

réservés, de sorte que tous les blocs disponibles se retrouvent ensemble pour ne plus former qu'un seul grand bloc lorsque s'exécute le Garbage Collector.



T36.a



T36.b

: bloc utilisé

: bloc disponible dans la liste libre

figure T36 : solution au problème du Garbage Collector

3. ALLOCATION DE CADRES OU PAGES

La mémoire est allouable par morceaux de taille fixe, appelés cadres ou pages. Toute demande de mémoire est formulée en nombre de cadres.

Pour que cette méthode ait un avantage fondamental sur les précédentes, il faut que l'emplacement des cadres alloués soit indifférent, de telle sorte qu'on puisse allouer des cadres n'importe où, et notamment, des cadres non-adjacents.

La gestion de la mémoire est réduite à une comptabilisation des cadres libres et occupés au moyen d'une table de booléens. Malgré la simplicité de cette méthode, un problème reste cependant à résoudre : celui de la taille du cadre. Choisir une petite taille de cadre est intéressant car on alloue dans ce cas que ce qui est nécessaire à l'avancement du processus. Mais cela entraîne une lourdeur dans la gestion de la correspondance entre référence et adresse, et de nombreux transferts entre la mémoire principale et secondaire. Il faut donc trouver un compromis.

Toute fragmentation de la mémoire n'a pas disparu. En effet, le besoin en mémoire du processus doit être exprimé en nombre entier de cadres : le dernier cadre n'est donc que rarement rempli. En moyenne, le dernier cadre de toute demande est à moitié utilisé : à tout moment la perte vaut

ND * TC * 1/2

où ND est le nombre de demandes satisfaites,
TC est la taille du cadre.

Tout en gardant les principaux avantages du système d'allocation de pages, le problème du choix de la taille du cadre peut être résolu par la méthode des jumeaux.

4. METHODE DES JUMEAUX

Cette autre approche d'allocation dynamique de mémoire est très rapide et particulièrement adaptée aux applications en temps réel, comme c'est le cas pour la commutation.

Un bloc ne pourra avoir une taille que de 2k cellules, k étant un entier. Si une demande ne correspond pas à une puissance de 2 cellules, un bloc de la puissance de 2 juste supérieure à la demande sera allouée, fournissant ainsi un espace supplémentaire non utilisé.

L'idée de cette méthode est de garder une liste libre différente pour chaque taille de bloc 2k, $0 \leq k \leq m$. L'ensemble de la mémoire a une taille de 2m cellules. Au démarrage du système, le bloc entier de 2m cellules est disponible.

Lorsqu'un bloc de 2^k mots est demandé, et si aucun bloc de cette taille n'est disponible, le plus petit bloc de taille $\geq 2k+1$ sera partagé en deux blocs de même longueur. Cette opération sera répétée jusqu'à l'obtention d'un bloc de longueur 2k. Quand un bloc est partagé en deux blocs égaux, ceux-ci sont appelés des "blocs jumeaux".

L'efficacité et la rapidité de ce système viennent du fait que si l'on connait l'adresse d'un bloc (c'est-à-dire l'adresse de son premier mot), on connait l'adresse de son jumeau par une simple opération combinatoire sur la représentation binaire des adresses.

De plus, cette méthode offre une garantie contre les recherches dans une longue liste libre.

C. ALLOCATION AVEC PREEMPTION

Deux classes de systèmes ont été envisagées :

- sans préemption : le processus garde la ou les parts reçues jusqu'à ce qu'il n'en ait plus besoin;
- avec préemption : les parts reçues par un processus peuvent lui être enlevées au profit d'un processus prioritaire.

Dans ce dernier cas, une question doit être posée : le processus perdant une partie ou toute son allocation de mémoire est-il de ce fait tué ou pourra-t-il ultérieurement se poursuivre ?

Cette dernière solution est celle réalisée dans les systèmes en temps réel : lorsqu'un processus est bloqué en attente d'un événement, la mémoire qu'il a acquise est remise à la disposition de tout autre processus jugé prioritaire.

Dans ce systèmes, les processus suspendus doivent conserver le contenu des parts de mémoire qui leur sont enlevées : aussi, le gérant avant de réallouer ces zones, recopie-t-il leur contenu sur une mémoire de réserve ou mémoire secondaire.

Tout système à préemption sans destruction de processus est composé de deux niveaux de mémoire :

- une mémoire principale où sont chargés les éléments les plus importants pour l'exécution des processus les plus susceptibles d'être exécutés;
- une mémoire secondaire où se trouvent les copies des zones perdues en mémoire principale par les processus jugés les moins prioritaires.

Ce système de va-et-vient entre les deux mémoires (swap-in, swap-out)existe avec des méthodes d'allocation à parts variables et à pagination.

2.2.2.2.B.4. Communication et synchronisation entre processus .

Combinée avec la technique des automates, la communication entre processus est un outil fondamental pour la **logique** du commutateur . Elle est le vecteur principal de la **signalisation** .

Le fonctionnement interne global du commutateur fera l'objet d'une prochaine section (D) . On se limitera ici à voir comment la communication complète les notions d'automates et de processus pour réaliser la signalisation .

Les points suivants seront traités dans la suite :

a) Les automates vus comme **machines finies à messages** . On se place au niveau de l'application qui est de réaliser la signalisation.

b) La communication entre **processus** se place au niveau de l'O.S. Pour des raisons d'efficacité (contraintes du temps réel),il est intéressant d'implémenter les modules software sous forme de processus (ou de tâches : cf B.2.).

c) La communication entre les modules de bas niveau : cas spécial.

d) La communication dans les systèmes AXE10 et 1240.

a) Machines Finies à Messages.

a.1) Rappel sur les automates.

Un automate dans sa version générale contrôle un **système** qui peut se trouver dans des **états** différents. L'automate tient à jour cet état et d'autres informations sur le système contrôlé. Dans chaque état , des **événements** peuvent survenir qui provoquent une réaction de l'automate.

Cette réaction se fait en deux temps :

(1) Exécution d'un certain nombre d'actions .
(2) Passage à un autre état.

état présent	événement	action	état suivant
1	x	y	2
2	z	t	n
3	:	:	:
:	:	:	:
n	:	:	:
:	:	:	:

Fig E1. automate général

a.2) Automates en commutation.

En commutation, la notion d'automate convient particulièrement bien pour le contrôle des abonnés, des circuits interurbains, et des communications elles-mêmes entre interlocuteurs. En effet ces "objets" ont bien un certain nombre d'états et le passage entre états se fait suivant une logique bien définie.

Cette logique est la signalisation. Dans ce contexte, les automates subissent deux adaptations importantes :

-Les événements sont des réceptions de messages en provenance d'autres modules.

-Les actions comprennent non seulement des mises-à-jour d'informations sur le système contrôlé, mais aussi et surtout des envois de messages à d'autres modules.

De plus, ces automates ont un nombre fini d'états.

De la sorte, un automate de commutation devient une machine finie à messages. Il a la forme suivante pour chaque état:

-Liste des messages pouvant être reçus.
-mises-à-jour des informations sur le système contrôlé. Ces informations proviennent partiellement des informations contenues dans le message.
-Liste des messages à envoyer.
-Etat suivant de l'automate.

état présent	message reçu	action	message envoyé	état suivant
1	m1	...	m5	2
	m2	...	m4,m23	3
2	m7	...	...	4
3	m8	...	...	4
4	...	...	...	...
...	...	...	...	...
n	...	...	...	...

Fig E2. Machine finie à messages.

a.3) Signalisation.

La logique de signalisation devient donc une séquence d'enclenchements d'automates. Cette séquence comporte à chaque étape un ou plusieures branches alternatives prévues et spécifiées dans une procédure de signalisation.

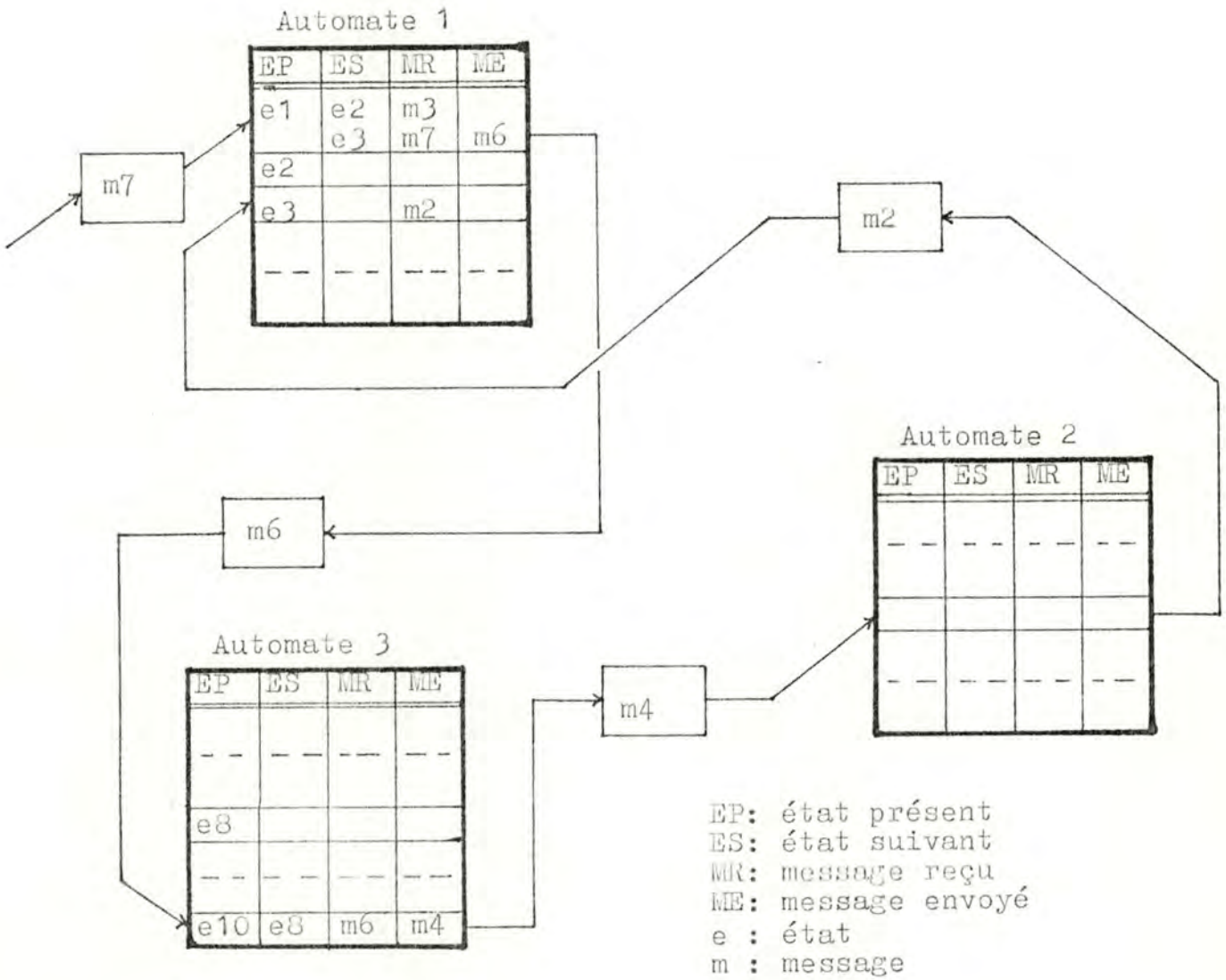


Fig E3. Signalisation et automates.

NOTE : Il faut spécifier la destination du message. Cela peut être indiqué soit dans le message lui-même soit dans l'ordre d'envoi. Ce point sera examiné plus tard.

Un exemple de signalisation est donné au point D.

a.4) Contenu d'un message.

En ce qui concerne les automates (niveau application), les informations contenues dans un message sont les suivantes :

-Numéro de message (celui-ci peut implicitement désigner un destinataire).
-indication de l'abonné appelant ou appelé concerné.
-divers renseignements sur les abonnés.
-heure et date.
-catégories,...

b) Communication entre processus.

b.1) Combinaison automates et processus.

On a vu en B.2 l'utilité des processus pour l'exécution des modules de commutateur.

On vient également de voir comment les automates s'échangeaient des messages qui étaient autant d'événements modifiant leur état.

Dès lors si chaque automate est exécuté sous forme d'un processus indépendant, l'échange de messages entre automates devient une communication entre processus.

b.2) Transmission de messages.

Dans l'O.S en temps réel, on dispose de primitives de gestion des échanges de messages entre processus. Les deux principales sont l'envoi de messages, et la réception de messages.

On va extraire ici l'essentiel du mécanisme d'échange. Deux systèmes particuliers seront examinés par après (ITT,ERICSSON).

L'utilisation de deux primitives au lieu d'une seule pour transmettre un message vient du fait que les processus ne sont pas tous en cours d'exécution au même moment et qu'ils travaillent indépendamment l'une de l'autre.

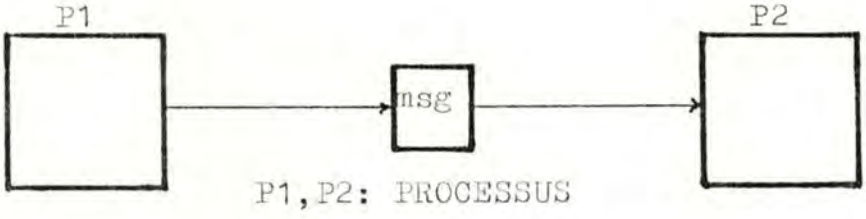


Fig E4. Transmission immédiate d'un message.

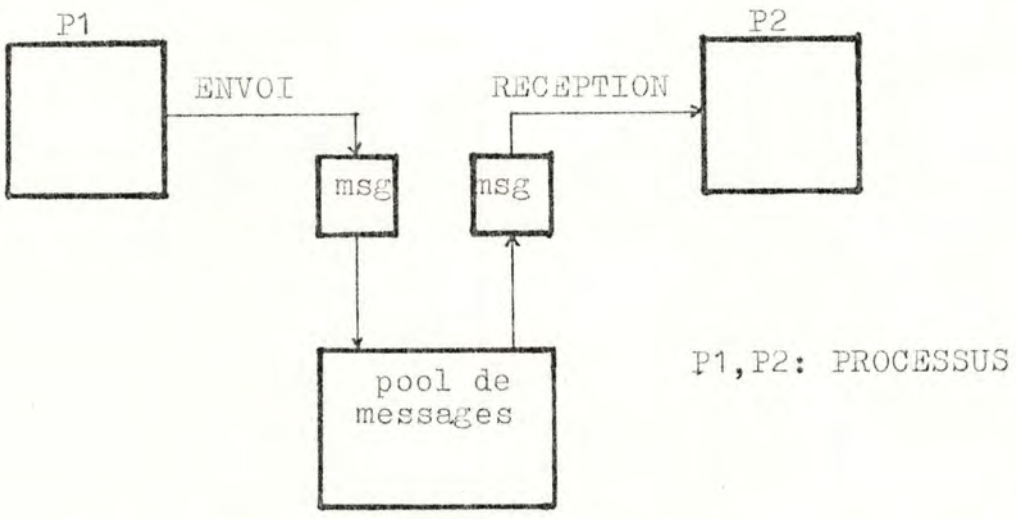


Fig E5. Transmission désynchronisée d'un message. L'opération se passe en deux temps avec un délai variable entre envoi et réception. Le pool de messages est la zone intermédiaire où les messages envoyés attendent d'être reçus.

(1) Envoi d'un message.

(i) Le processus émetteur doit tout d'abord disposer d'un bloc de mémoire dans lequel il pourra placer son message. Il obtient ce bloc par appel au module O.S gérant la mémoire.(cfB.3).

(ii) Le processus place ensuite les informations composant le message dans le bloc mémoire mis à sa disposition.

(iii) Le processus envoie le bloc message dans le pool des messages où il attendra d'être reçu par le processus destinataire.

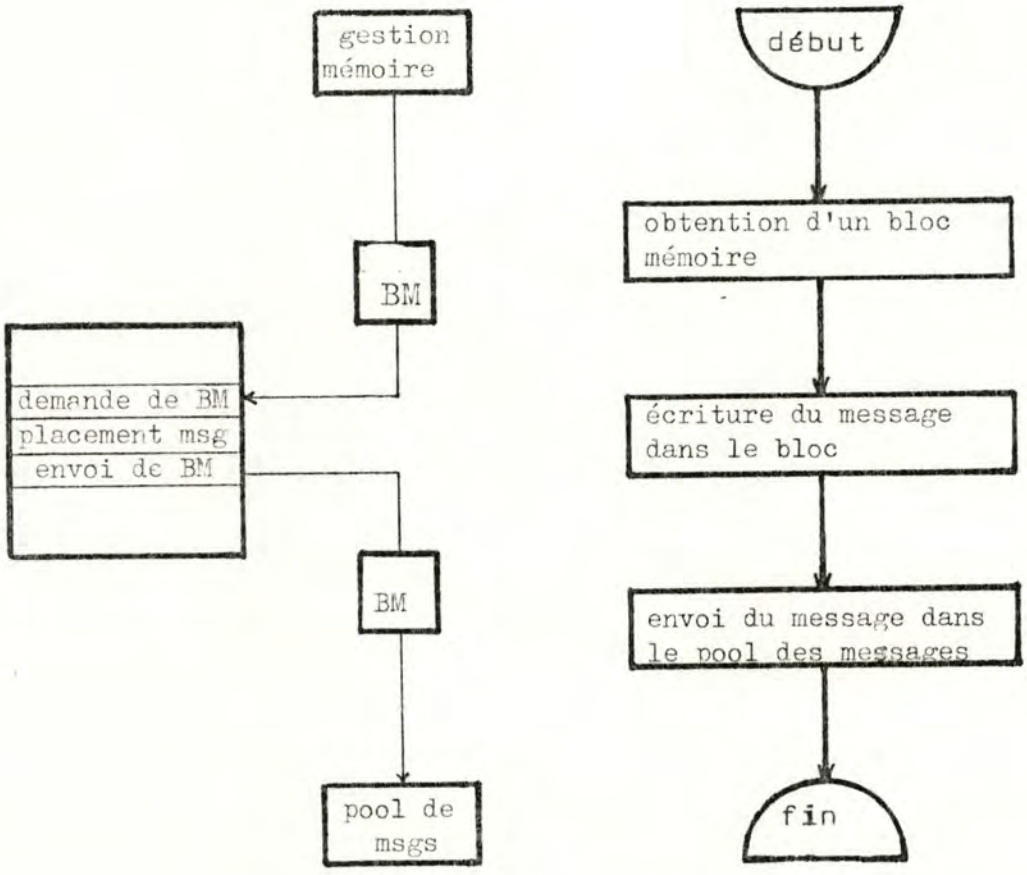


Fig E6. Envoi de messages.

NOTE : Selon les systèmes, il peut y avoir une seule primitive réalisant la séquence complète, ou alors plusieurs opérations successives.

(2) Réception d'un message.

(i) Le processus récepteur se met en attente d'un message. (cf fonctionnement des automates : réception d'un message = événement déclencheur).

(ii) Chaque fois que l'O.S doit lancer une nouvelle tâche, il prend un message dans le pool (voir les problèmes de priorités et les tâches ne consistant pas en des automates). Et il le met à disposition du processus destinataire.

(iii) Le processus est lancé (enclenchement d'un automate.)

(iiii) Le bloc mémoire est libéré et rendu au gestionnaire de la mémoire.

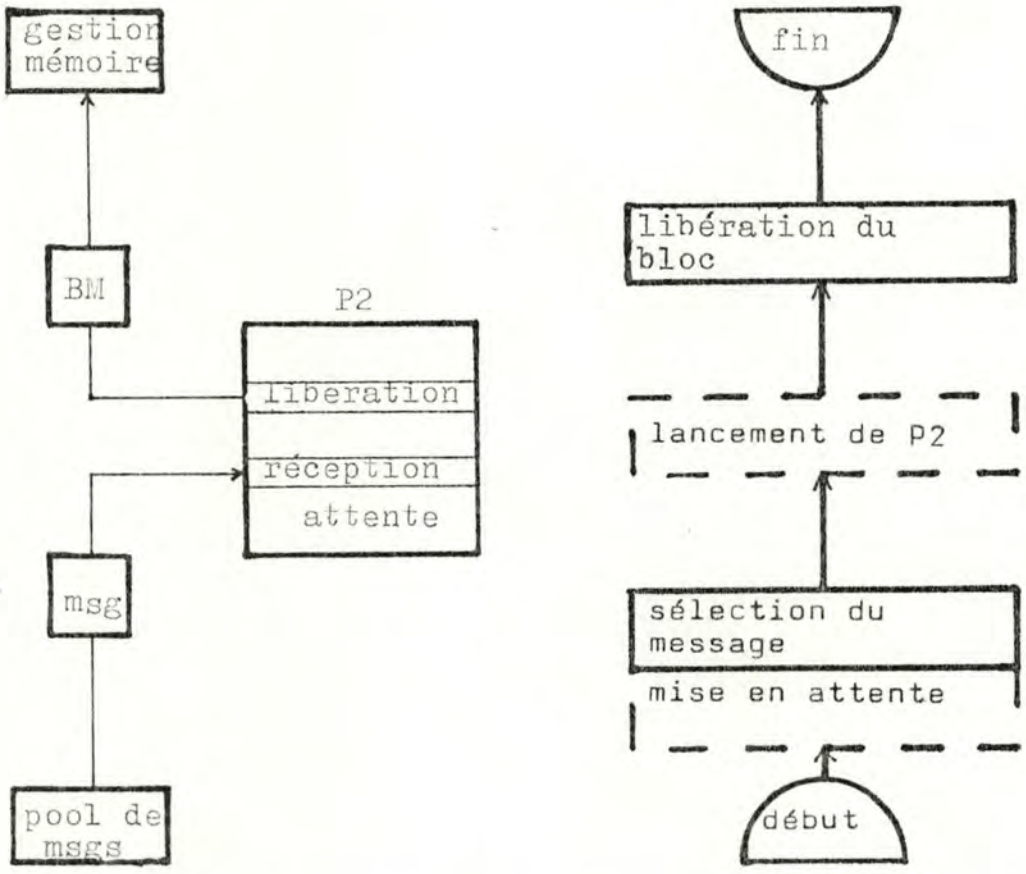


Fig E7. Réception d'un message et lancement du processus récepteur.

Remarques :

(i) La mise en attente est provoquée par P2 lui-même systématiquement après un changement d'état de l'automate.

(ii) La réception du message et le lancement de P2 est faite par le moniteur aidé du scheduler.

(iii) La libération du bloc et la restitution au gestionnaire de la mémoire se fait normalement automatiquement par le moniteur, mais ce n'est pas indispensable.

b.3) Priorités des messages et scheduling des tâches.

Ces notions ont été abordées en B.2 mais il est utile d'en rappeler ici les points fondamentaux.

Lorsque plusieurs processus sont en attente de messages et que tous ces messages attendus sont présents dans le pool, prêts à être reçus, le scheduler doit décider quel message acheminer à son destinataire, et par conséquence quel processus activer, puisqu'un processus est activé lorsqu'il reçoit un message.

On voit donc à quel point l'envoi de message et le scheduling de tâches sont étroitement reliés. Pour cette raison, les messages seront affectés de priorités différentes établies en fonction de la priorité de la tâche destinatrice.

A ces différentes priorités correspondront en fait plusieurs files d'attentes de messages (voir B.5 à ce sujet).

b.4) Identification du processus et acheminement des messages.

Les processus ont un identificateur (ID). Cet identificateur peut être hiérarchisé de manière à préciser par exemple :

- L'organe sur lequel le processus tourne.
- Le module software constituant le code exécuté par le processus.
- L'identification du processus lui-même.

Différents types de messages et de méthodes d'acheminement peuvent ainsi être dégagés. On verra plus loin comment ce problème est traité dans 1240 ET AXE.

c) Communication entre modules de bas niveau.

Bien que cela ne soit pas à proprement parler de la communication entre processus indépendants, il faut cependant mentionner ici un autre type d'échange de données ou commandes entre modules.

En effet, les modules automates tels que décrits plus haut ne se justifient que pour implémenter des fonctions de contrôle de haut niveau.

Ils sont par contre inadéquats et peu performants pour des fonctions de bas niveau qui sont routinières et doivent s'exécuter sans délai. Le principal exemple de ce genre de fonctions est le scanning des lignes et circuits (voir A.1). De manière générale, tous les modules software de bas niveau très proche du hardware tombent dans cette catégorie .

Les communications avec ces modules se réduisent donc à des mécanismes beaucoup plus rudimentaires : appels de procédures, interruptions.

On verra à la section suivante comment ITT et ERICSSON ont réalisé ces fonctions. ITT nomme ces modules "machines de support de systèmes" (SSM) tandis que ERICSSON les considère comme du "software régional".

d) Communication dans 1240 et AXE.

La description générale de la communication entre blocs software telle qu'elle a été donnée ci-dessus se rapproche fortement de la méthode employée dans le 1240, et aussi dans le prototype COLO/S. AXE 10 utilise des notions voisines, mais néanmoins différentes en ce qui concerne notamment les automates.

L'accent est mis sur les principes communs au-delà des différences relativement superficielles de vocabulaire ou de représentations.

Les points suivants seront abordés:

1. Types de blocs software.
2. Support de communication .
3. Déroulement d'une communication.

d.1)Types de blocs software.

(i) Les modules de haut niveau.

Les modules de contrôle de haut niveau réalisent des fonctions complexes et non routinière (ex: analyse de numérotation).

* Du point de vue application, ces modules sont des suites de petites séquences d'instructions, chaque séquence traitant un cas particulier de contrôle.

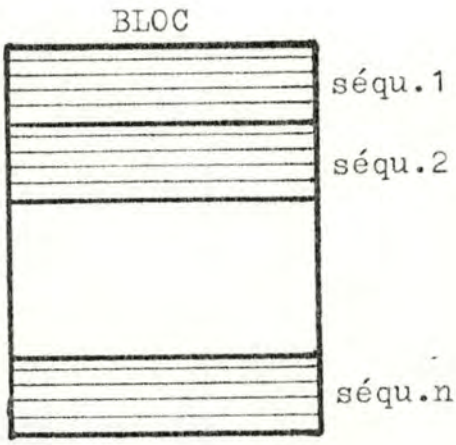


Fig E8. Composition d'un bloc de haut niveau.

Les séquences ont le format général suivant :

- Réception d'une communication.
- Exécution de certaines opérations (par exemple : mises-à-jour de BD).
- Envoi d'une communication.

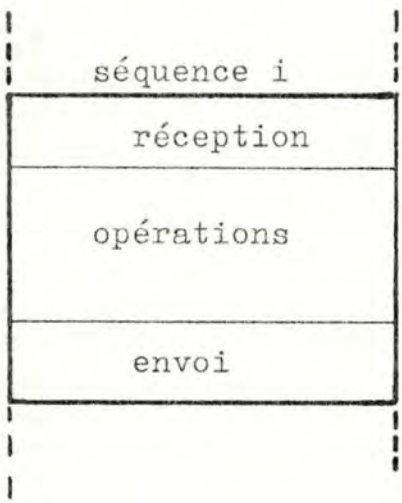


Fig E9. Composition d'une séquence.

Une différence importante existe ici entre 1240 et AXE. Les blocs 1240 sont des automates (FMM) tels que décrits plus haut . Cela signifie qu'à tout moment l'automate est dans un état déterminé. A chaque état de l'automate correspond une et une seule séquence d'instructions. Et à la fin de chaque séquence, on spécifie quel nouvel état l'automate doit prendre. Les blocs AXE, sont par contre de simples suites de séquences non enchaînées entre elles.

Ce détail a une importance que l'on explicitera plus bas : lorsqu'une communication arrive à un bloc 1240, l'état présent de l'automate détermine automatiquement quelle séquence doit être activée. Par contre, dans AXE, la communication doit spécifier elle-même à quelle séquence du bloc elle s'adresse.

* du point de vue O.S, ces modules doivent être exécutés pour des communications simultanées entre de nombreux abonnés. C'est la raison de l'utilisation de processus. Bien qu'exprimées de manière différente, les méthodes 1240 et AXE sont très semblables.

-1240 parle de processus d'application contrôlés par un superviseur. Il y a un superviseur par automate, et un processus d'application pour chaque exécution particulière de cet automate (voir B.2).

-AXE parle d'une unité software globale comprenant un code commun, des données connues et des données individuelles propres à chaque tâche particulière de contrôle. C'est dans ces données individuelles que l'on retrouve implicitement la notion de processus.

(ii) Modules de bas niveau.

Les modules de bas niveau implémentent des fonctions routinières rapides et relativement simples (par exemple : scanning de lignes automatiques conçus de manière différente). Ils sont des modules de haut niveau (aussi bien au niveau hardware que software) pour des raisons de clarté et d'efficacité.

-1240 les appelle SSM : machines de support au système.

-AXE les classe dans le software "régional" c'est-à-dire affectation particulière à chaque unité hardware.

Ces modules sont en fait des ensembles de petites procédures qui peuvent être enclenchées très rapidement. Le système de communication est donc différent de celui utilisé pour les modules de haut niveau (voir plus loin).

d.2 Supports de communication.

On a parlé plus haut de "messages" envoyés entre modules de haut niveau, et de "moyens" de communication rapides entre modules de bas niveau et modules de haut niveau. Nous allons voir comment ces modules ont été conçus dans 1240 et AXE . Des références seront faites à l'architecture de ces systèmes. Pour plus de renseignements ,voir 2.3.

(i) Architectures fonctionnelle et physique du software.

* 1240

- Les blocs fonctionnels principaux du 1240 sont appelés "éléments de contrôle" (CE). Ils se composent d'une combinaison variable de trois types de modules :

  -SSM ou machines de support de système .
  -FMM ou machines à nombre fini de messages .
  -bases de données .

  . Les SSM contiennent des procédures activables par interruptions software, ce qui est un moyen rapide de communication.

  . Les FMM sont des automates que l'on peut activer en leur envoyant des messages.

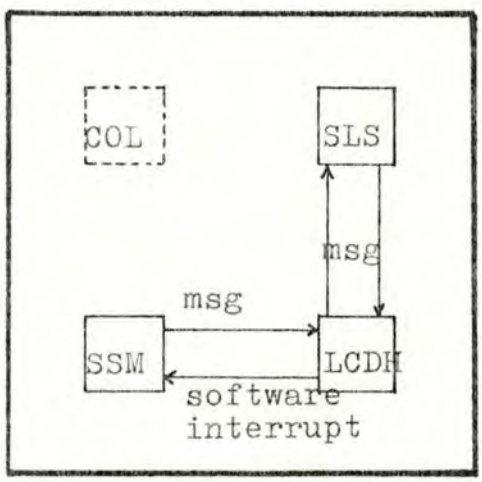


Fig E10. exemple de CE 1240 : contrôle des lignes d'abonnés .

- Les blocs physiques du 1240 correspondent exactement aux blocs fonctionnels. Il faut cependant préciser que chaque bloc physique tourne sur un processeur indépendant (architecture fortement répartie). Tous les EC et leur processeur sont connectés à un même réseau de connection numérique (DSN).

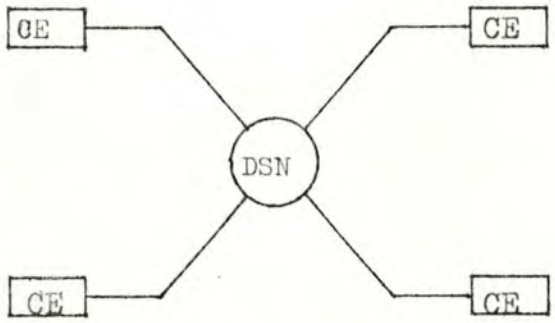


Fig E11. conception architecturale du 1240.

De plus les commutateurs sont reliés entre eux via des circuits .

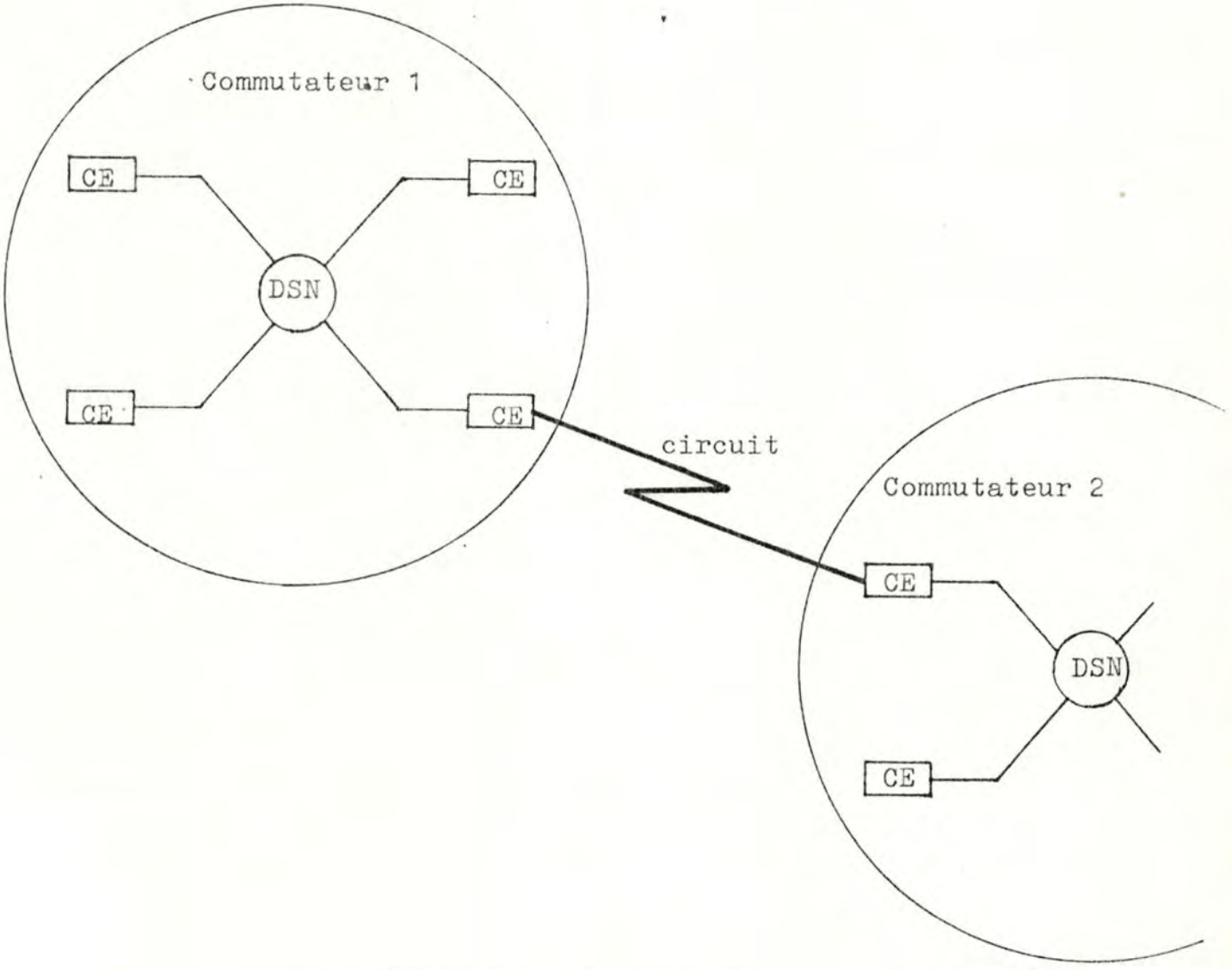


Fig E12. interconnection de commutateurs 1240.

Il y a par conséquent un problème d'acheminement des messages suivant qu'ils connectent deux modules d'un même EC, de deux EC différents appartenant au même commutateur, ou de deux EC appartenent à des commutateurs différents. Cette question est examinée en (ii).

* AXE.

- Les blocs fonctionnels de AXE (appelés tels quels par ERICSSON) sont composés de trois parties :

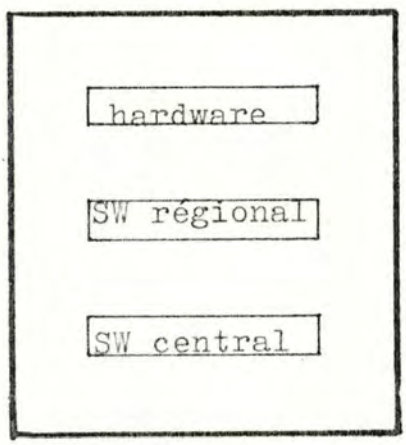


Fig E13. conception architecturale de AXE au niveau fonctionnel.

Le software central contient les modules de haut niveau (FMM du 1240).

Le software régional contient des routines de bas niveau (SSM du 1240) de contrôle d'unités hardware.

Le hardware, le software régional et software central sont appelés "unités fonctionnelles".

- Les blocs physiques de AXE sont très différents des blocs fonctionnels.

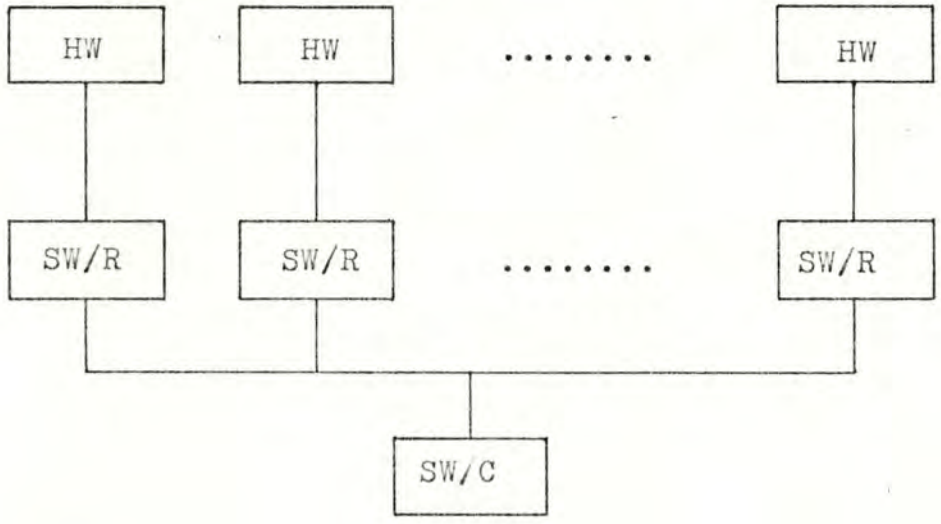


Fig E14. conception architecturale de AXE au niveau physique.

Le software central est en fait physiquement situé sur un seul processeur central. Par contre , chaque unité de software régional dispose de son propre processeur régional.

On voit donc qu'à le différence du 1240, tous les messages de haut niveau s'échangent à l'intérieur d'un même processeur. On les appelle cependant "externes" car ils s'échangent entre blocs fontionnels différents. Les signaux de bas niveaux entre software régional et software central sont appelés quant à eux "internes" car ils s'échangent à l'intérieur d'un même bloc fontionnel, même si physiquement, ils sont transmis entre deux processeurs différents.

(ii) Description des supports de communication.

* 1240
- L'activation des SSM se fait par interruption software.Le mécanisme sera explicité en D.1.Rappelons seulement qu'il permet un changement de contexte et une activation de procédure très rapides et prioritaires.
- Une communication adressée à une FMM se fait au moyen de messages. Les messages comportent les renseignements principaux suivants :

NOTE : pour le paramètre "type de message", voir la section suivante à propos de l'acheminement.

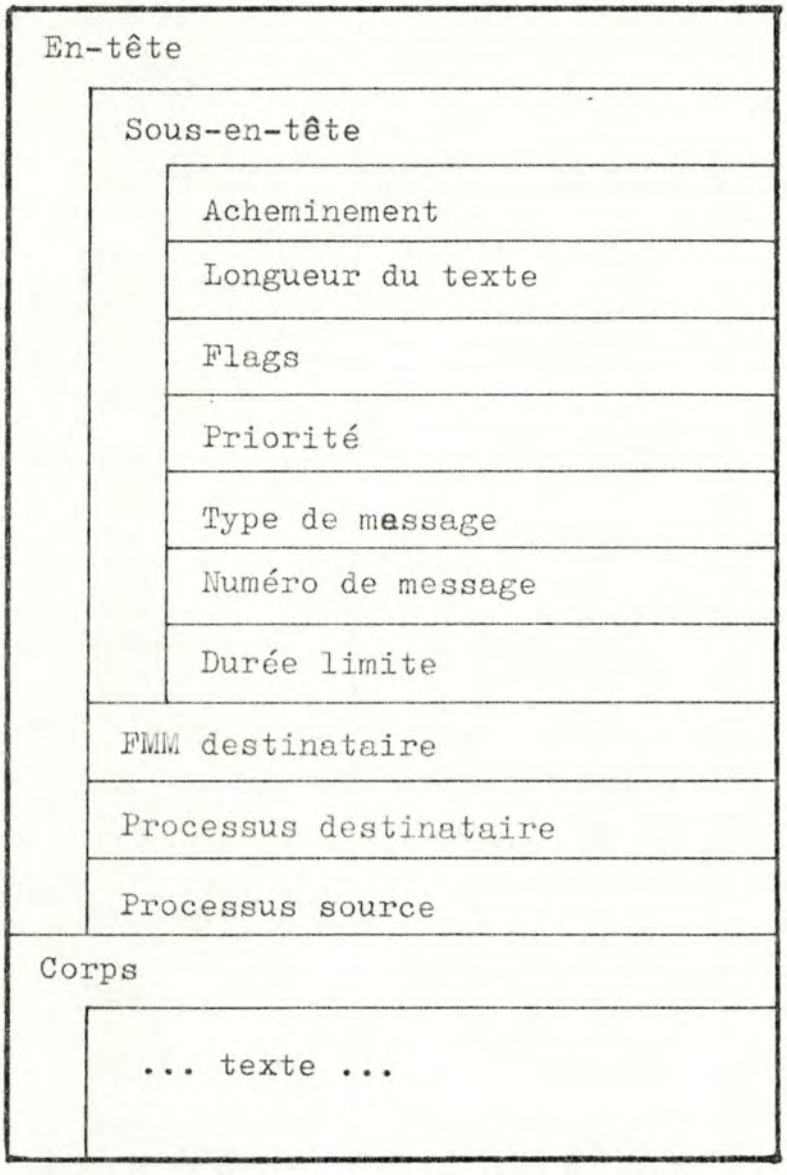


Fig E15. structure d'un message dans le 1240.

Il faut mettre en évidence le fait que le messager ne spécifie pas quelle séquence d'instructions de la FMM doit être exécutée . En effet , la FFM maintient elle-même l'état de l'automate et sait par conséquent quelle séquence exécuter.

* AXE

- Les messages envoyés aux unités régionales comportent principalement : . un numéro de bloc (BN).
  . une référence à la fonction à exécuter.

- Les signaux envoyés aux unités centrales comportent une en-tête : . un numéro de bloc (BN) référençant biunivoquement un bloc.
  . une localisation de signal (SL) permettant de situer la séquence d'instruction à exécuter (voir d.3)
  . un nom de signal permettant de préciser sa priorité (voir B.5) et un corps comprenant des données.

(iii) Méthodes d'acheminement.

* 1240

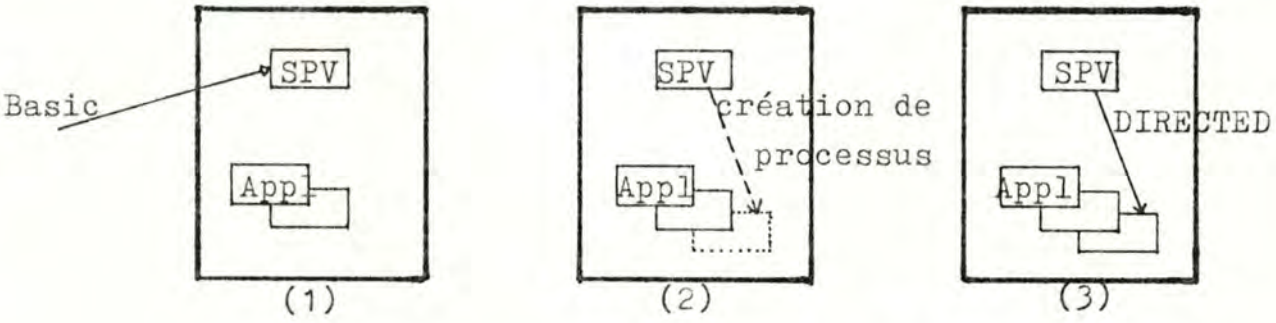
a) Sans entrer dans trop de détails, on peut distinguer deux paramètres réglant l'acheminement :

-La nature BASIC et DIRECTED du message.

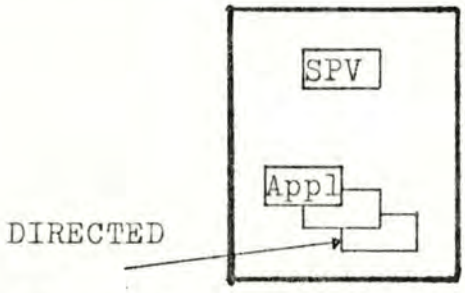
On a expliqué en A.2 et B.3 qu'un automate peut être "instancié" pour chaque tâche différente auquel il s'applique. Dans ces conditions, il faut lui adjoindre un superviseur capable de créer des processus d'application et de canaliser les messages vers les processus d'application destinataires.

. BASIC : lorsqu'il n'y a pas encore de processus d'application créé pour une tâche donnée, ce message est envoyé au superviseur . Celui-ci se charge alors de créer un nouveau processus d'application et de lui transmettre le message.

. DIRECTED : lorsqu'il existe un processus d'application pour la tâche donnée, on peut s'y adresser directement en faisant référence au numéro de processus.



(a) envoi de message pour la première fois.



(b) envoi de message par la suite.

Fig E16. message BASIC/DIRECTED.

-les tables et paramètres d'acheminement.

Il existe une série de tables gérées par le module de gestion des systèmes : message number T., type/FCB T., discriminator T., etc.

Suivant l'acheminement désiré, le message sera envoyé avec des paramètres VIA, FOR, INTO, ONTO, ou des combinaisons de ceux-ci.

b) Les voies d'acheminement.

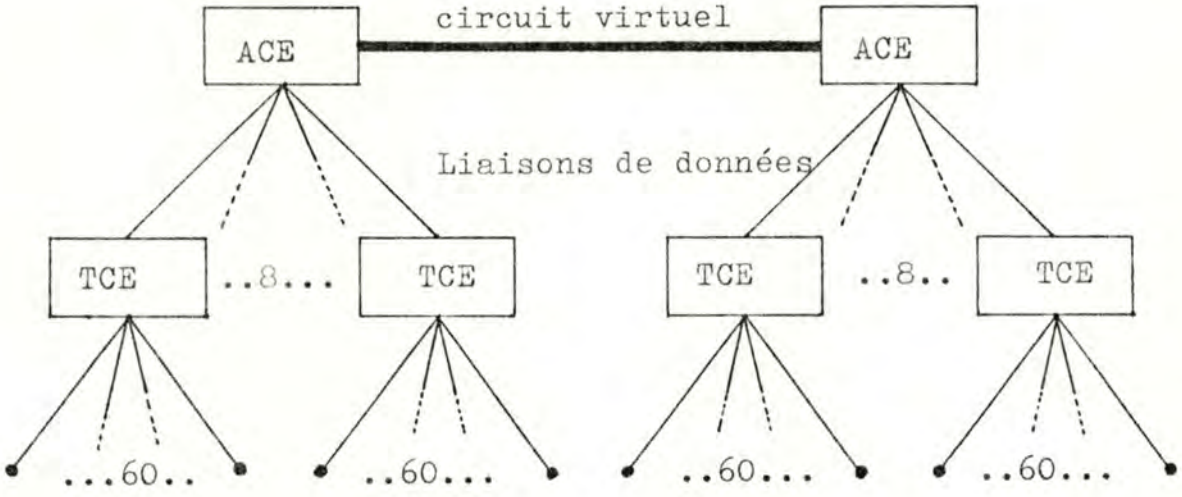


Fig E17. Voies d'acheminement de messages.

Il y a deux sortes de noeuds de commutation dans le 1240 :

- Les TCE : éléments de contrôle terminal, auxquels on peut connecter jusqu'à 60 lignes d'abonnés.

- Les ACE : éléments de contrôle auxiliaire, auquels on peut connecter jusqu'à 8 TCE, c'est-à-dire 480 lignes d'abonnés.

Entre chaque ACE et les TCE qui y sont connectés, une liaison de données est établie de manière permanente à l'initialisation du système.

Entre deux ACE, on peut établir à la demande des chemins virtuels, ne subsistant que pour la durée d'une communication.

Il n'y a pas de lien direct entre deux TCE.

Les différentes possibilités d'acheminement sont donc les suivantes :

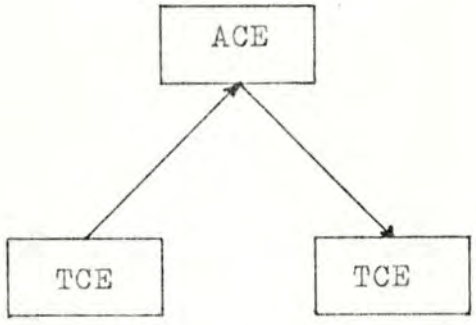


Fig E18. local à un même ACE.

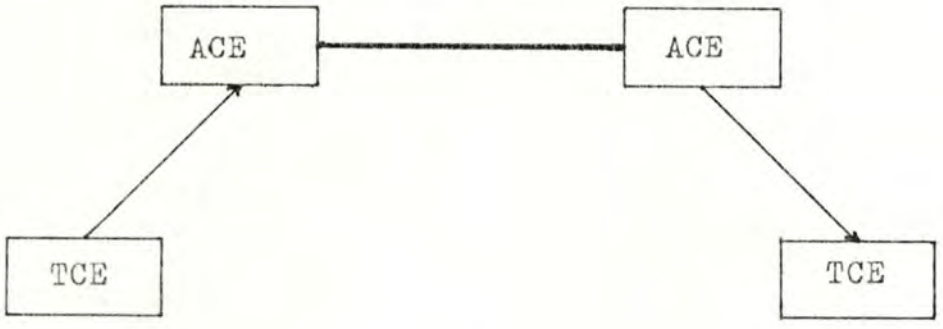


Fig E19. en tandem via les ACE.

* AXE

Etant donné que tout le software central est situé dans le même processeur, les problèmes d'acheminement ne se posent pas. Tous les signaux sont envoyés sans problème aux blocs destinataires via leur BN.

L'envoi à une unité régionale est le même à ceci près que l'on passe par le bus des processeurs régionaux (voir C.2.).

d.3) Déroulement d'une communication.

(i) 1240

Ainsi que cela a été dit plus haut, il y a deux primitives principales qui s'appellent ici : TRANSMIT (envoi de message) et MSG-WAIT (attente de réception de message).

* TRANSMIT se base sur une procédure en langage assembleur appelée MSG-SEND.

MSG-SEND commence par sauver tous les registres courants de la machine plus un code (61) qui est la référence de la primitive à exécuter. Ensuite , une interruption sotfware est lancée. Son numéro est 48, cela indique qu'il s'agit d'une interruption pour exécution d'une primitive.

La primitive fait appel aux trois procédures suivantes :

- SEND-MSG-BUF pour envoyer un buffer de message.
- MHC-ROUTE-MSG pour effectuer l'acheminement du message.
- Q-ON-READY pour placer le buffer de message sur la file "prête"(cf B.5).

D'autre part, lorsque le message voyage entre deux éléments de contrôle différents, le NH (getion du réseau) fait appel à la procédure message-received.

Enfin, il faut signaler qu'il est fait appel à la procédure (du gérant de mémoire) BM-MB-Q-OFF, pour extraire un buffer de message de la file des buffers, en vue d'y placer le message à envoyer.

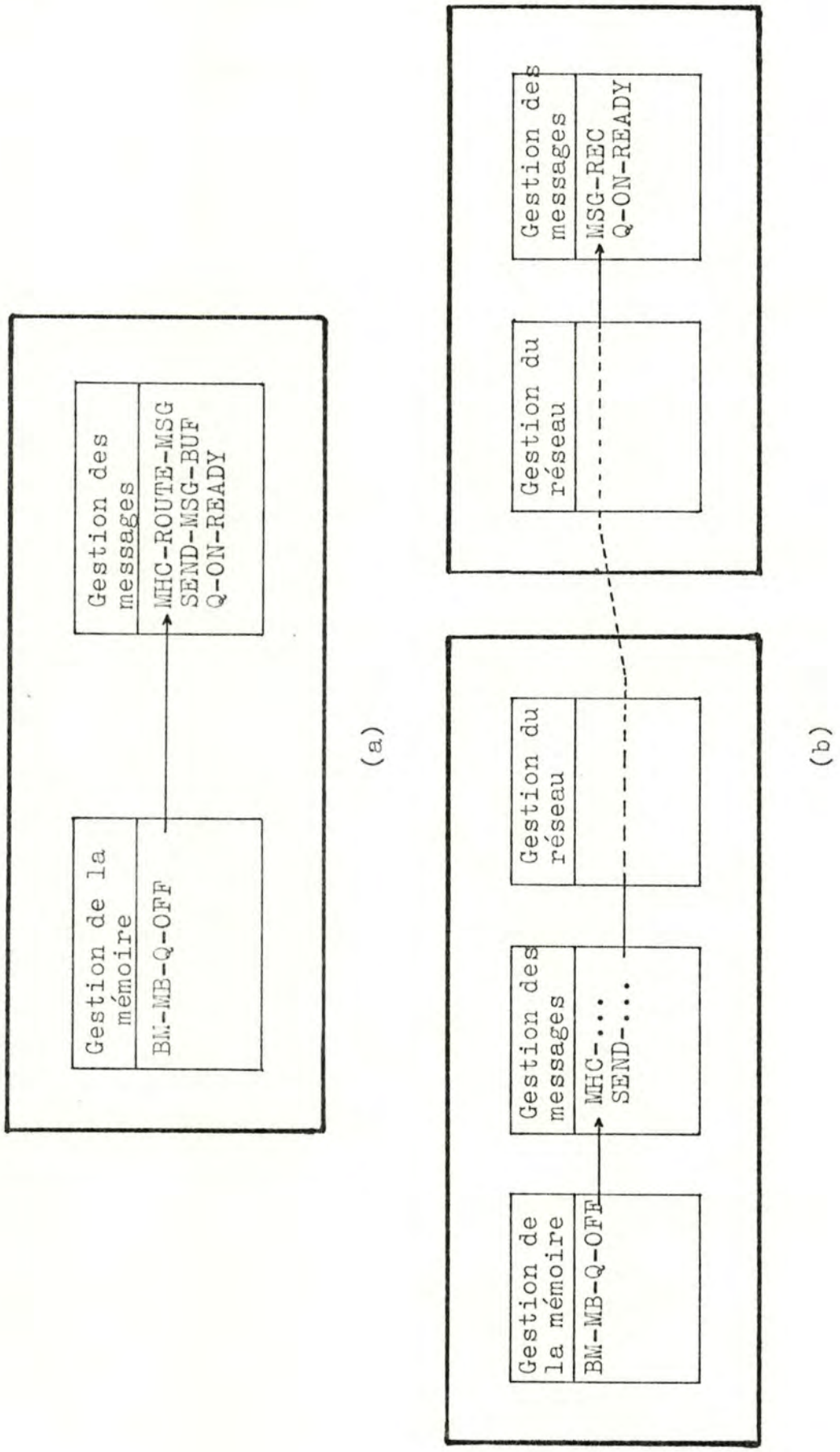


Fig E20(a). envoi local.
(b). envoi entre deux CE différents.

* MSG-WAIT

Cette primitive est du ressort du gérant des processus. Elle a pour effet de mettre un processus dans l'état "attente de message".

Normalement, tôt ou tard un message arrivera pour lui et sera prioritaire lorsque le scheduler (primitive MHC-FIND-WORK) cherchera un nouveau processus à lancer.

Lorsque c'est le cas, la primitive du BM. BM-MB-Q-ON-TAIL est appelée pour restituer le buffer à la mémoire libre.

Ensuite le processus destinataire est activé.

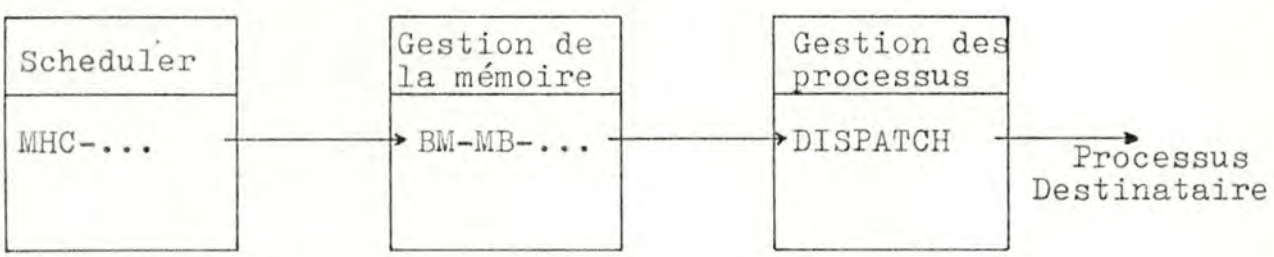


Fig E21. réception d'un message et lancement du processus destinataire.

(ii) AXE

* Structure de la mémoire dans CP.

Il y a quatre mémoires :

- mémoire de programme (PS).
- mémoire de données (DS).
- mémoire de référence (RS).
- mémoire de registres (RM).

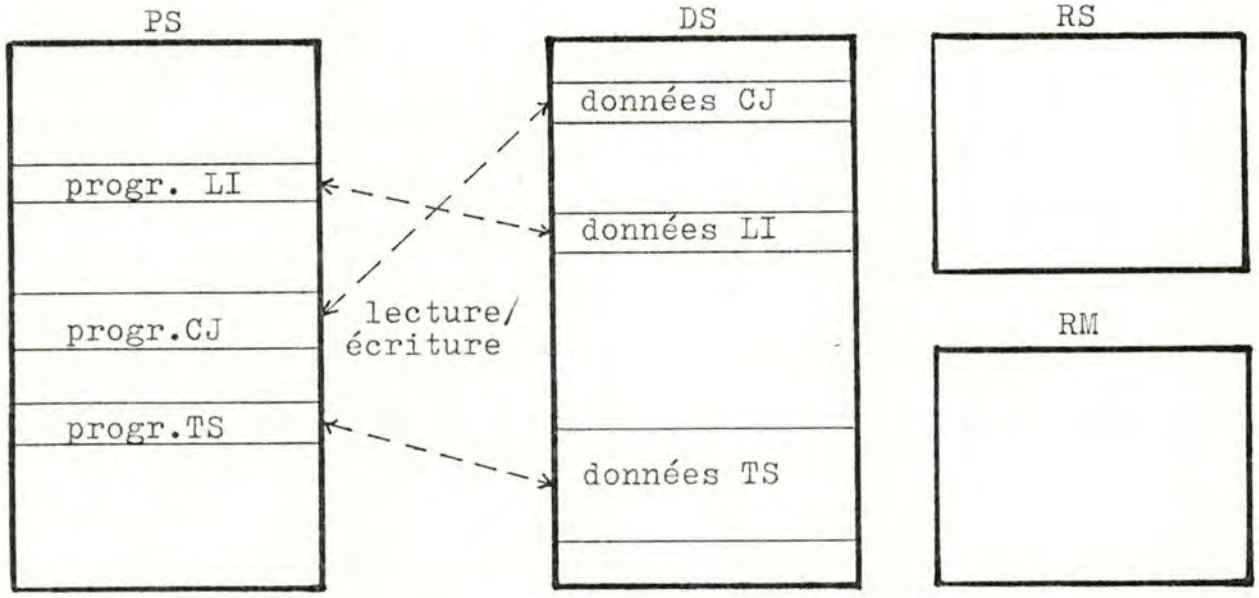


Fig E22. types de mémoires du CP.

Chaque programme se trouve dans un segment de PS et ne peut adresser dans DS que les données qui lui appartiennent. La RM pallie cette restriction en étant commune à tout le système.

* Envoi d'un signal.

Instruction SEND du langage PLEX.

Le déroulement est le suivant :

(1) Le programme d'application origine du signal déplace les données formant le corps du signal, de sa zone en DS vers une zone en RM. Le nom du signal y est également placé.

(2) L'O.S est appelé pour transférer le signal de la RM vers une file d'attente (voir B.5).

* Réception d'un signal.

Lorsque le moniteur veut lancer un tâche, il extrait un signal de la file d'attente. Ce signal a pour but de faire exécuter une séquence particulière d'instructions d'un programme particulier, en lui donnant accès à ses données propres.

Pour cela, le signal porte deux paramètres : BN et SL.

On dispose d'une RS composée de deux parties : la table de référence et la table d'adresses de base.

L'entrée numéro BN de la table de référence donne deux adresses de base.

- L'adresse de départ PSA du programme voulu dans la PS. En ajoutant SL, on obtient l'adresse de départ de la séquence voulue du programme.

- L'adresse de départ BSA, dans la table d'adresses de base concernant le programme voulu. Chacune des adresses de ce bloc référence une variable ou un bloc de données dans DS appartenant au programme voulu.

Le schéma E23 reprend tout le mécanisme.

Le moniteur procède de la manière suivante :

(1) Transfert du signal en RM.
(2) Recherche des zones comme expliqué ci-dessus (au moyen du module RSH).
(3) Lancement du programme.

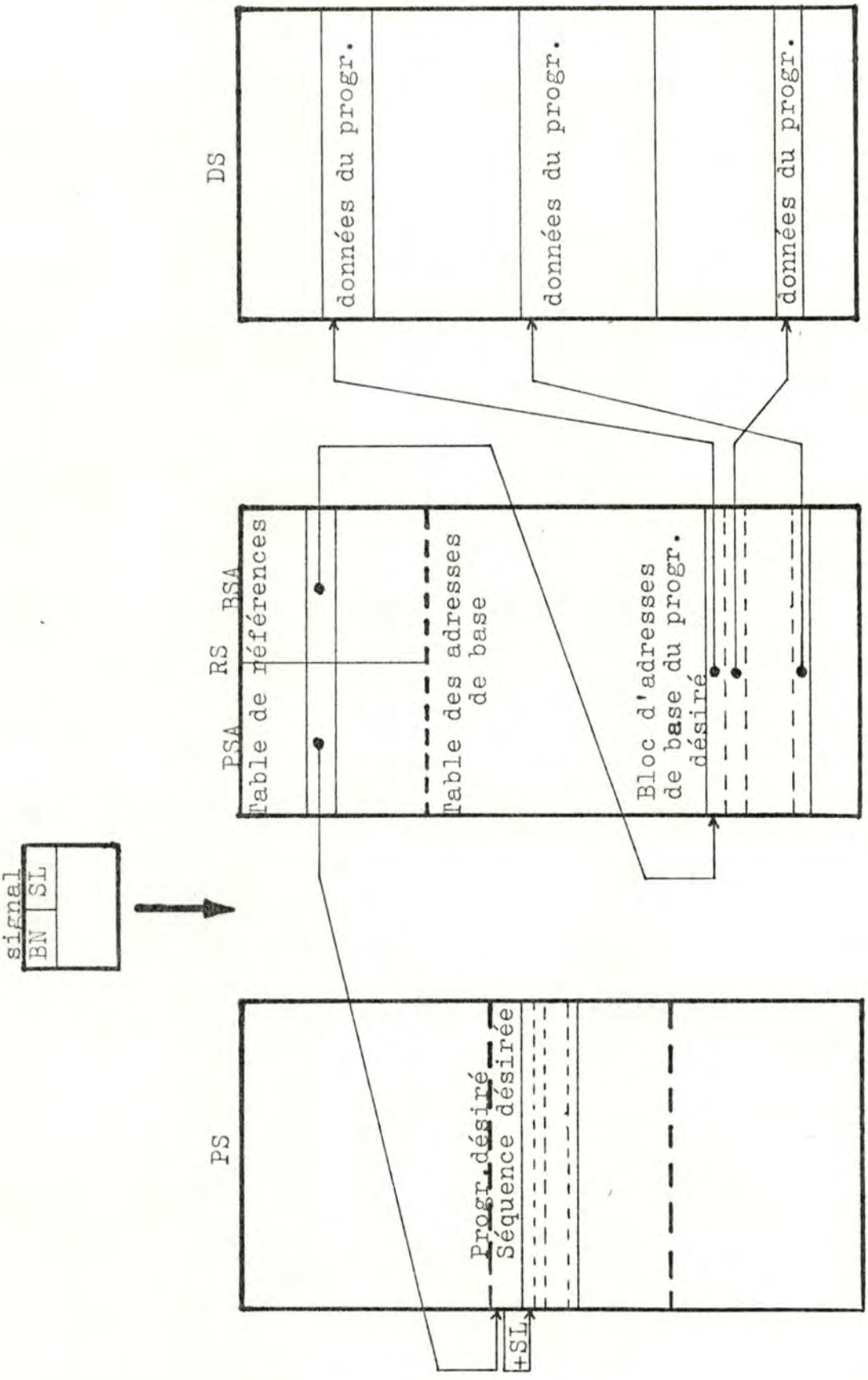


Fig E23. Réception d'un signal.

2.2.2.2.B.5. Files d'attente.

a) Principes.

La notion de file d'attente existe dans tous les OS multiprocessus. Les files contiennent par exemple des processus dans divers états (libre, actif, suspendu, ...) qui attendent d'être traités. En particulier, les processus dans l'état "prêt" attendent d'être exécutés (état "actif"). Dans le contexte d'un nombre limité de processeurs physiques, les processus ne peuvent en effet pas être exécutés simultanément.

Dans le cadre d'un commutateur, les contraintes du temps réel font que l'activation des processus de haut niveau (1240 : FMM ; AXE : software central) dépend de l'apparition d'événements signalés par des messages (1240) ou des signaux (AXE).

Il y a à tout moment de nombreux messages ou signaux envoyés entre les modules du commutateur. La réception de chacun d'eux signifie l'activation du processus destinataire. Etant donné les limites de capacité de traitement CPU, cette activation ne peut se faire simultanément pour tous les destinataires. Par conséquent, les messages ou signaux sont stockés sur des files d'attente.

Dans le cas d'un commutateur en temps réel, les files d'attente de messages prennent donc la place des files d'attente de processus prêts.

En ce qui concerne l'activation de procédures de bas niveau, cela se fait soit par files d'attente (AXE) soit par interruption software sans file d'attente (1240).

De tout cela émerge le problème du choix du prochain processus à lancer. C'est le scheduling, lié aux questions de priorité et de précédence chronologique.

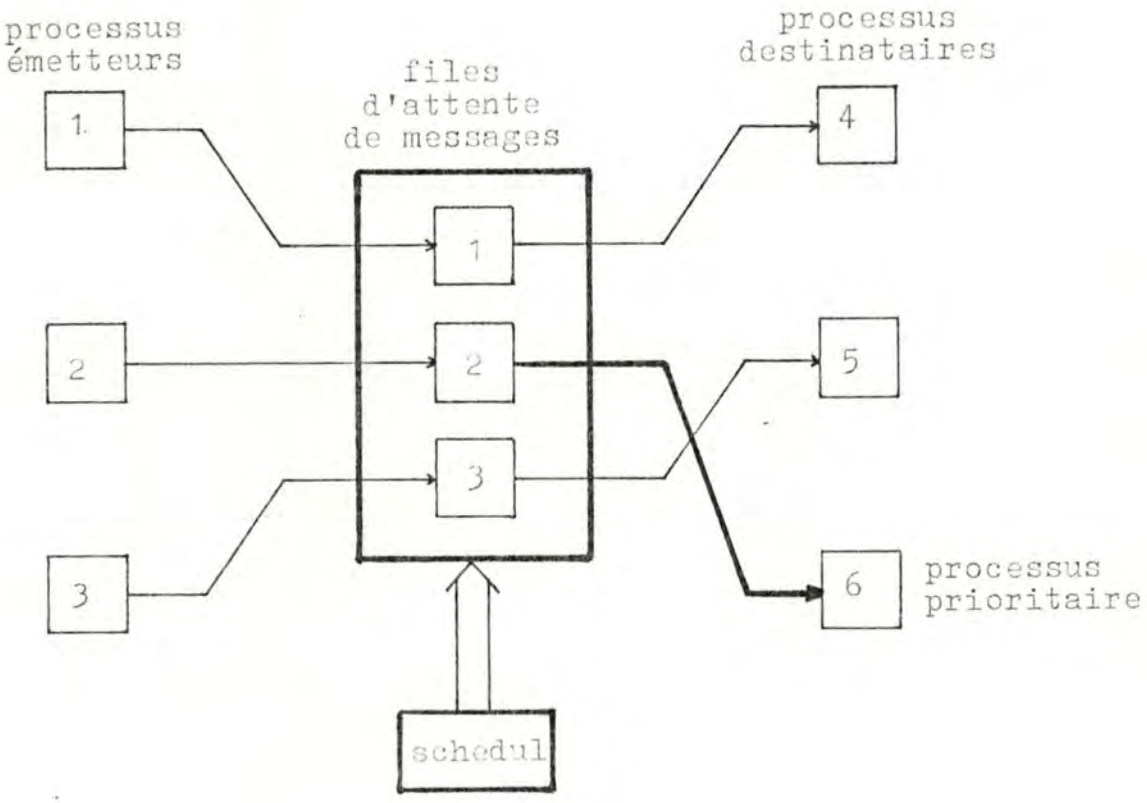


Fig E24. File de messages et scheduling de processus.

b) Files d'attente et scheduling dans AXE 10.

Les files d'attente sont appelées "JOB BUFFERS". Il y en a quatre appelées respectivement A, B, C et D, de priorité décroissante.

Dans chaque buffer, les signaux sont ajoutés en queue et retirés en tête.

Le scheduling procède comme suit :

1. Choisir le buffer de plus haute priorité non vide.

2. Prendre le signal en tête de buffer et lancer le processus destinataire.

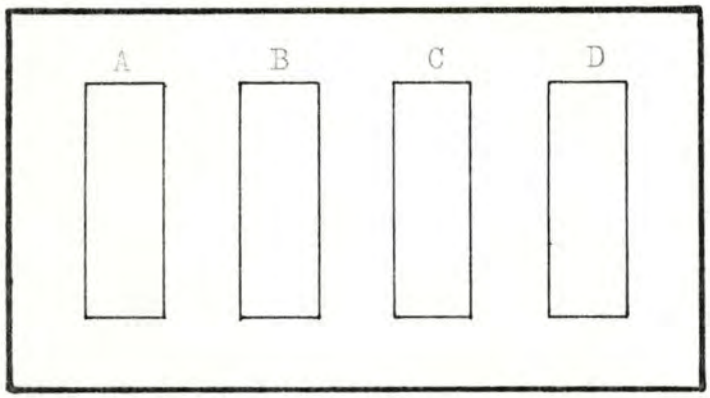


Fig E25. Files d'attente dans AXE.

c) Files d'attente et scheduling dans le 1240.

Les files d'attente sont appelées "READY QUEUES". Il y en a huit numérotées de 0 à 7 et de priorité décroissante.

Le scheduling des processus se fait comme dans AXE.

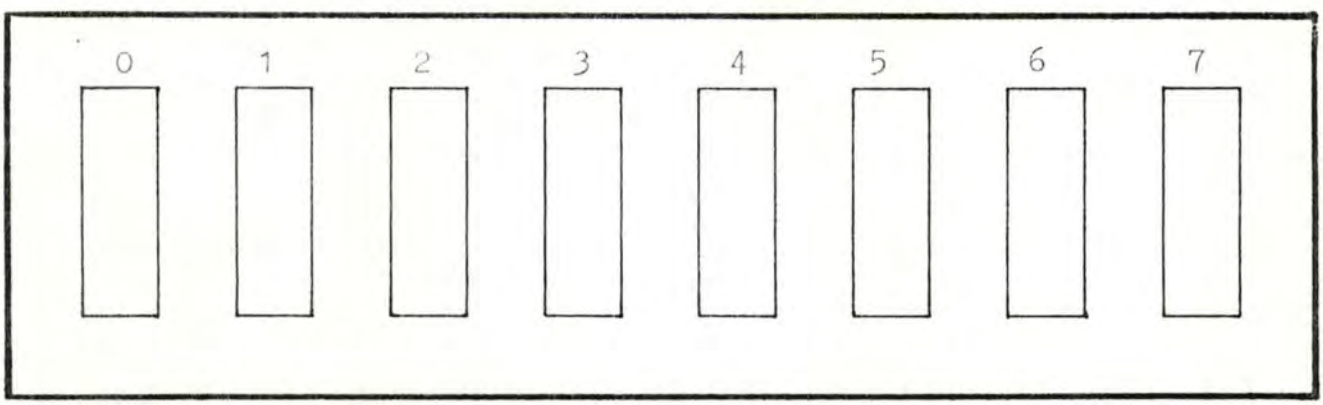


Fig E26. Files d'attente dans le 1240.

C.1 Interruptions.

C.1.1 Principes généraux.

a) Registre des interruptions.

Les interruptions sont classées en quatre catégories:

- asynchrone, tributaire d'une erreur
- asynchrone, non tributaire d'une erreur
- synchrone, tributaire d'une erreur
- synchrone, non tributaire d'une erreur.

De plus, elles peuvent relever du système ou du programme. Elles sont rassemblées et classées par priorité décroissante dans l'INTERRUPT TABLE.

b) Interrupt table.

	PRIORITE		CONDITION D'INTERRUPTION
ASYN. et ERR.	1		baisse de tension
	2		contrôle machine
ASYN. et $\overline{\text{ERR}}$.	3	MASQUE SYSTEME	signal externe n°1
	4		" " n°2
	5		" " n°3
	6		" " n°4
	7		" " n°5
	8		" " n°6
	9		non affecté
	10-15		canal sélecteur 1-5
	16		canal multiplex
SYN. et $\overline{\text{ERR}}$.	17		horloge de décompte de temps
	18		pupitre
	19-20		non affecté
	21		appel superviseur
SYN. et ERR.	22	MASQUE PROGRAMME	opération privilégiée
	23		code opération inconnu
	24		erreur d'adresse
	25		donnée erronée
	26		débordement d'exposant
	27		division erronée
	28		mantisse nulle
	29		exposant trop faible
	30		débordement décimal
	31		débordement en virgule fixe
	32		mode test.

c) Les états machine.

Une machine se trouve toujours dans un des quatre états suivants :

- état P1 : état de traitement de programme.
  Cet état de fonctionnement est celui où le programme d'application est interprété et exécuté.
- état P2 : état de traitement des interruptions
  Cet état est employé par l'exécutif et quelques programmes systèmes.
  Il exécute les opérations requises pour traiter une interruption.
- état P3 : état d'analyse des interruptions.
  La machine passe automatiquement dans cet état lorsque se produit une interruption (excepté pour incident ou baisse de tension).
  Cet état analyse la cause d'interruption et sa priorité. Il lance alors en P2 la routine appropriée pour traiter l'interruption.
- état P4 : état machine (état d'incident HWD).
  Cet état est lancé en cas d'incident machine ou de baisse de tension.

d) Schéma général des interruptions.

Quand un des 32 interrupts possibles est détecté par le hardware, un bit est automatiquement positionné dans l'Interrupt Flag Register.
Ces demandes sont comparées au contenu de l'Interrupt Mask Register de l'état lancé. Ce registre (un par état) autorise ou non cette interruption.

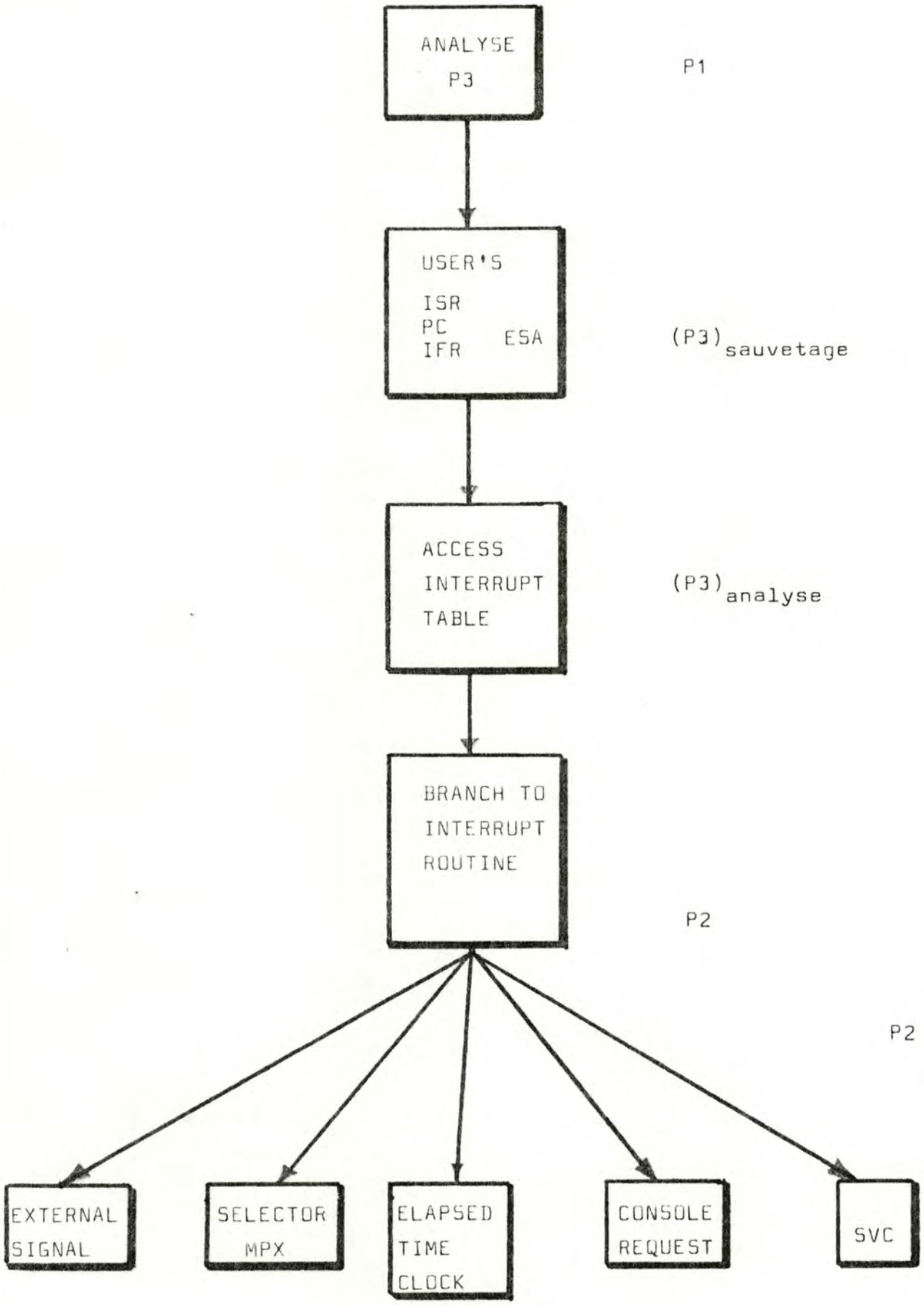
En P1, toutes les interruptions sont autorisées par l'exécutif.

En P2, sont permises, les interruptions pour erreurs de programme ou erreurs de machine.

En P3, sont permises, les interruptions pour erreurs de machine.

En P4, aucune interruption n'est autorisée.

Lorsque la demande est autorisée, le contrôle est donné à P3. L'unité centrale exécute des instructions séquentiellement. Elle trouve l'adresse de l'instruction suivante à exécuter dans les bits 8 à 31 du p.counter. Ce program counter est un registre de 32 bits situé dans la mémoire bloc-note Il y a un p.counter différent pour chacun des états. Lors du passage à l'état P3, nous utiliserons le p.counter de P3.
La routine d'analyse des interruptions stocke les registres généraux RØ, R1 et R2 de l'état P dans une zone du programme utilisateur: l'ESA (Executif Storage Area). Ces registres sont respectivement l'IMR, l'ISR et le p.counter de l'état P1. C'est la phase de sauvetage de l'état P3. Elle permet au programme utilisateur de continuer là où il s'est arrêté en cas de retour ultérieur à l'état P1.
L'Interrupt table est alors consultée pour trouver l'adresse de la routine qui va traiter cette interruption. La consultation survient au terme de la phase d'analyse de l'interruption de l'état P3.
L'adresse est chargée dans le registre 6 de P3.
Ce registre est aussi le p.counter de l'état P2.
L'état P2 est alors lancé par une instruction privilégiée qui ne peut être utilisée que par les programmes-systèmes.


ANALYSE
P3
P1
USER'S
ISR
PC
IFR
ESA
(P3)sauvetage
ACCESS
INTERRUPT
TABLE
(P3)analyse
BRANCH TO
INTERRUPT
ROUTINE
P2
P2
EXTERNAL
SIGNAL
SELECTOR
MPX
ELAPSED
TIME
CLOCK
CONSOLE
REQUEST
SVC
P3

C.1.2 Gestion des interruptions en temps réel.

a) Interrupt label.

PRIORITE	CONDITION D'INTERRUPTION
1	primitive
2	device
3	clock
4	interface
5	memory management
6	events
7-17	erreurs

b) Traitement des appels à des modules de l'OSN.

En l'état P1 : exécution du programme utilisateur. Chargement sur le stack de l'état P1 des paramètres et du numéro de la fonction dont on requiert les services.

En l'état P3 : SAUVETAGE : lors de l'exécution de l'instruction INT PRIMITIVE, qui est une demande d'interruption, les p.counter, code segment register et flags de l'état P1 sont sauvés sur le stack de l'atat P1.
Ensuite les actions suivantes sont prises:
1) le masque d'interruption de l'utilisateur est sauvegardé et le masque courant est mis sur celui de l'OSN.
2) trois variables sont utilisées pour enregistrer le type de job en cours : - type_du_job_actuel
- type_du_job_précédent
- type_du_job_suivant
A l'entrée de cet état la valeur du job actuel est stockée dans type_job précédent et dans type_job_suivant.

Le job actuel sera l'exécution de l'OSN.

3) l'environnement de l'utilisateur est sauvegardé : sp,ss,bp,bx,ds

4) établissement de l'environnement de l'OSN

5) orientation de Bp vers les paramètres dans le User stack.

ANALYSE : l'analyse porte sur les informations sauvées par le programme utilisateur en l'état P1.

Au moyen d'une table qui contient les numéros des fonctions possibles et les adresses de départ des procédures effectuant ces fonctions, l'état P3 peut transmettre à l'état P2 l'adresse de départ de la procédure à exécuter.

Les paramètres sont trouvés dans le stack de l'état P1.

En l'état P2 : exécution de la procédure.

Enfin, une restauration de ca qui a été sauvé en l'état P3 doit être effectuée avant de rendre la main au programme utilisateur.

c) Traitement des interruptions provenant des device, clock et erreur.

En l'état P1 : exécution du programme utilisateur. Survenance de l'interruption.

En l'état P3 : SAUVETAGE : sauvetage des p.counter cs et flags de l'état P1 sur le stack de cet état.

La première partie de la procédure de traitement sert à sauver le registre AX sur le stack; AX contient le numéro du type d'interruption que l'on doit traiter.

La seconde partie effectue ces opérations :

1) Sauver l'environnement du processus interrompu (sp,ss,bp,bx,ds,ax,di,si, dx,cx,es) dans l'Interrupt Control Block associé.

2) Etablir l'environnement de l'OSN.

3) Enregistrer le type de job en cours et mettre le type_de_job_actuel sur OSN.

4) Introduire le contenu de AX dans la variable type_intr qui sera utilisée par l'état P3 (analyse).

ANALYSE :

- contrôle du type du job interrompu.
- SI job = processus
  - ALORS °état du processus passe de running à interruped
    °détecter le pcb du processus en cours
    °mettre le pcb sur interruped
    °détecter l'ICB pour ce processus interrompu
    °positionner un flag sur ACTIVE pour cet ICB
  - SINON °positionner un flag d' EVENT
    °contrôler le type d'interruption.

2.2.2.2.C.2. Processeurs multiples .

a) Introduction .

Dans tous les systèmes informatiques se pose le problème des performances telles que rapidité d'exécution des instructions, promptitude à réagir à des événements extérieurs et capacité de gérer rapidement de multiples contextes .

Dans le cadre du temps réel, ce problème devient particulièrement aigu, car il est indispensable d'avoir d'excellentes performances, sans quoi le système se met en retard sur le monde réel qu'il doit contrôler.

A côté des techniques logicielles complexes visant à améliorer les performances, un moyen matériel apparemment simple et immédiat subsiste : la multiplicité des processeurs physiques. Tous les commutateurs commerciaux passent par cette technique, car les exigences sont telles qu'un seul processeur serait débordé.

Cependant la manière dont les systèmes multiprocesseurs sont conçus diffère largement sur des points comme :

- la correspondance entre unités logicielles et processeurs physiques,

- le type de réseau de connection entre processeurs .

On verra dans la suite comment ITT et Ericsson ont procédé. Un essai de critique sera fait, qui tentera de montrer les avantages et désavantages de l'un et l'autre système.

b) ITT 1240.

b.1) Unités logicielles et processeurs physiques.

* Les **unités logicielles** du 1240 sont les modules automates (FMM) ou de support (SSM). Ces unités sont regroupées en **éléments de contrôle** (CE) ayant une fonction plus globale. Ce sont :

- les TCE : CE terminaux contrôlant des devices, subdivisés en
  - L/TCE pour lignes d'abonnés
  - SC/TCE pour circuits de services (sonneries, etc.
  - T/TCE pour circuits intercommutateurs

- les ACE : CE auxiliaires ne contrôlant aucun device, servant simplement de puissance de traitement supplémentaire, subdivisés en
  - L/ACE pour lignes d'abonnés
  - S/ACE pour ressources système.

* Les processeurs physiques sont affectés à raison de un par CE. Chaque CE est donc à lui seul un organe de traitement, dont il importe de dire qu'il est indépendant des autres CE.

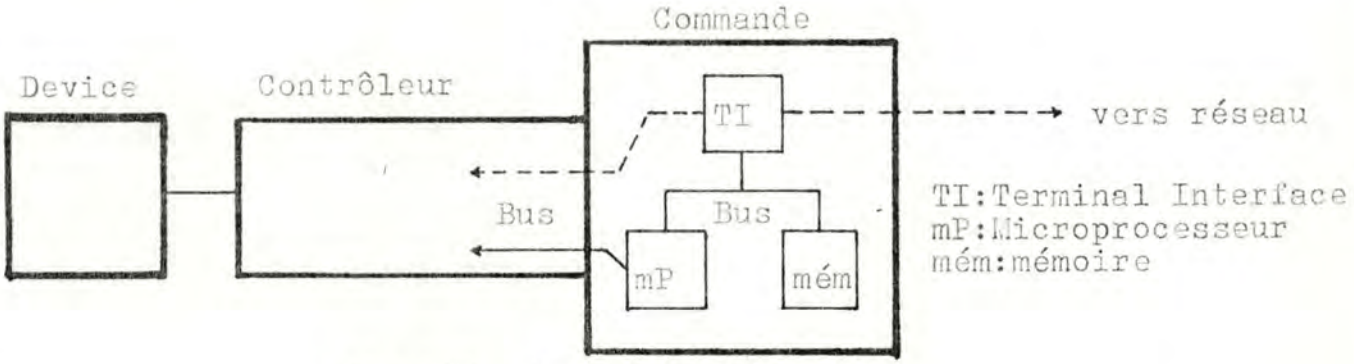


Fig E27.(a). Schéma d'un TCE.

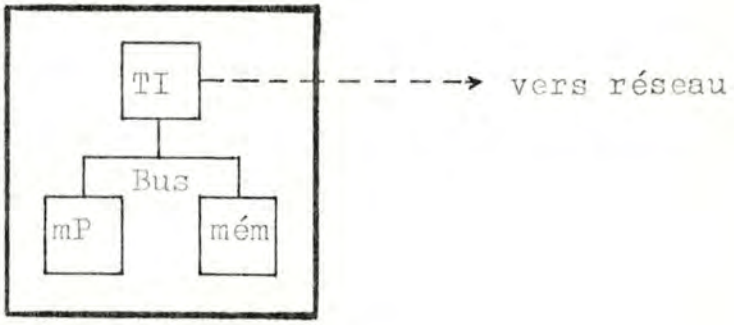


Fig E27.(b). Schéma d'un ACE.

b.2) Réseau de communication entre processeurs.

Tous les CE sont reliés à un même réseau de connection numérique (DSN) via une interface de terminal (TI). On peut se reporter à la section 2.1 concernant la connection .

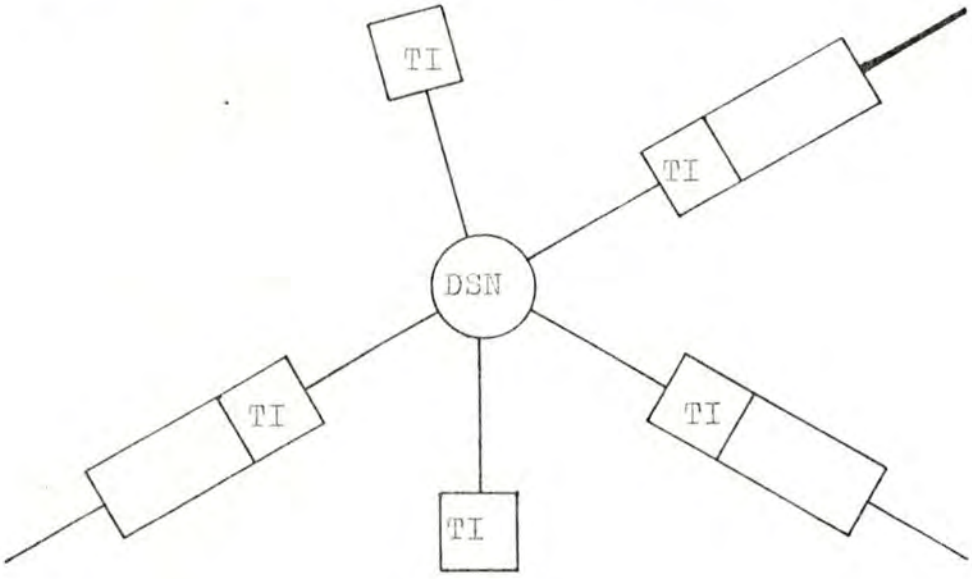


Fig E28. connection entre processeurs 1240.

Ce qui est important à signaler ici est que celui-ci ne comporte pas de microprocesseur, et est donc entièrement contrôlé depuis les extrémités par les microprocesseurs des CE agissant indépendamment l'un de l'autre mais pouvant communiquer entre eux.

b.3) Critique.

* Avantages :

- C'est une disposition en prise directe sur l'architecture fonctionnelle.
- Le réseau de connection est standard et indépendant des mouvements de CE.
- Pas de problème de hiérarchie : tous les microprocesseurs sont sur le même pied.
- On peut théoriquement ajouter ou retirer des CE à volonté, puisque cela ne concerne que le microprocesseur contenu dedans.

* Désavantages :

- Une certaine surcharge en communication via le DSN.
- Absence de maître qui contrôle globalement le commutateur.

c) Ericsson AXE 10.

c.1) Unités logicielles et processeurs physiques.

* Les unités logicielles de AXE sont les unités fonctionnelles software qui, combinées chacune avec une unité fonctionnelle hardware, constituent des blocs fonctionnels.(Fig E29)

* Les processeurs physiques sont hiérarchisés : il y a un processeur central et de multiples processeurs régionaux. La logique de répartition a été la suivante. On constate qu'il y a deux grands types de tâches effectuées dans un commutateur :

- les tâches non répétitives, "intelligentes", demandant l'accès à des bases de données;

- les tâches de routine, simples et répétitives, mais très consommatrices en temps et espace.

Sur base de ces observations, il a été décidé dans AXE de confier les tâches du premier type à un seul processeur central et de reléguer les tâches du second type à un certain nombre de processeurs régionaux.

Il faut bien voir que si tout le logiciel de haut niveau est situé dans le processeur central, il est cependant toujours structuré en unités fonctionnelles logiquement associées aux blocs fonctionnels respectifs du système de commutation.

Quant aux processeurs régionaux (RP), ils peuvent être répartis à volonté, chaque RP contrôlant soit

- un seul device
- plusieurs devices de même type
- plusieurs devices de type différent.

Cette répartition est en principe guidée par les performances des hardwares contrôlés.

De plus, il faut mentionner la différence de langages de programmation utilisés dans le CP et les RP :

RP : assembleur ASA 210 R

CP : assembleur ASA 210 C et surtout langage de haut niveau PLEX-C.

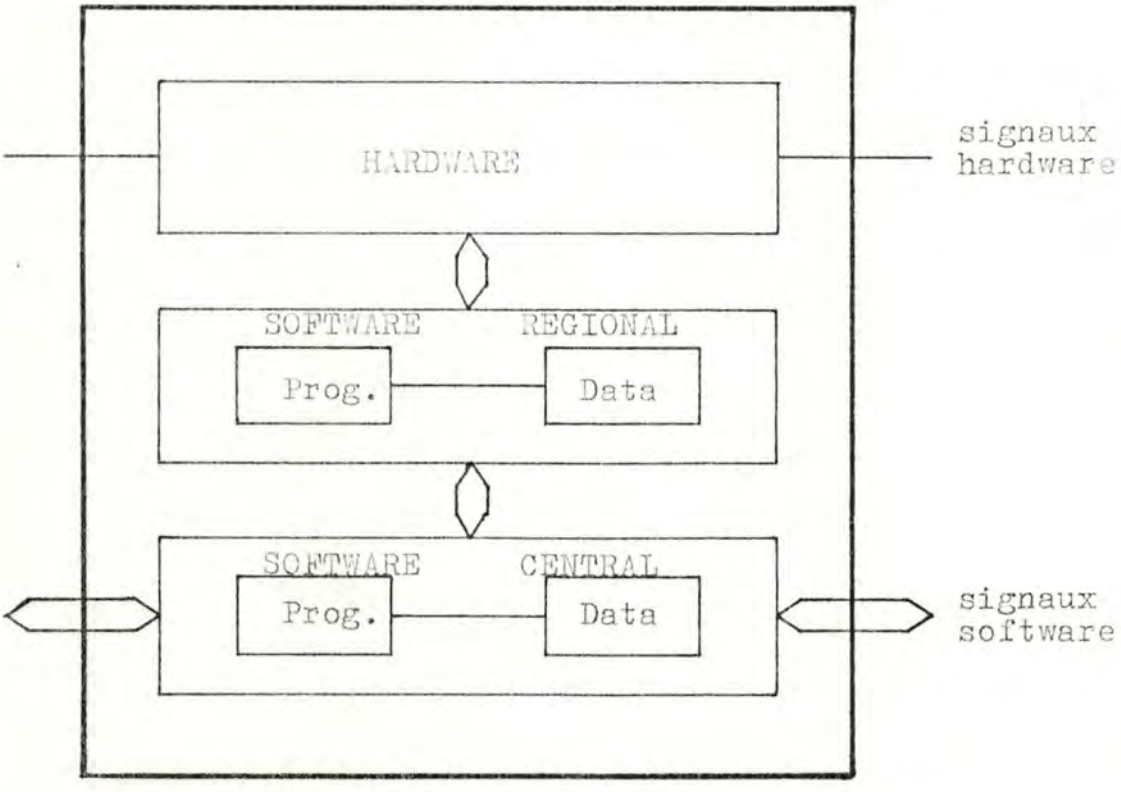


Fig E29. Bloc fonctionnel.

c.2) Réseau de communication entre processeurs.

* Liaison RP-matériel.

Les unités physiques sont divisées en modules d'extension (EM), appelés également magasins. Un RP peut contrôler jusqu'à 64 EM. Chaque EM est une unité possédant son propre adaptateur. La liaison RP-EM se fait via un bus d'EM, EMB.

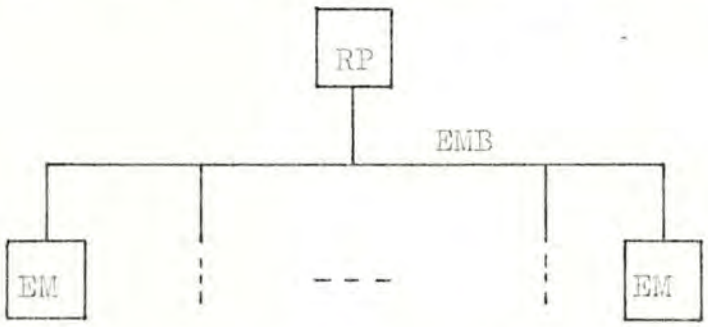


Fig E30. liaison RP-EM

* Liaison CP-RP.

- La liaison RP-CP se fait via un bus d'RP, RPB.

- Le problème de conflit sur RPB est résolu par l'adjonction dans CP d'un module spécial de gestion de RPB, RPH. (voir Fig E31.)

- message de CP vers DP :

. Le message est placé dans le buffer d'output vers RP, BRO.
. RPH l'achemine dans l'unité buffer d'input, IBU.
. RPH interrompt seulement après le RP pour que celui-ci sauve le message dans une zone de sa mémoire de données (DS) pouvant contenir jusqu'à 7 messages.
. Le RP termine son travail courant puis examine le message suivant dans sa DS.

- message de RP vers CP :

. RPH scanne continuellement les unités buffer d'output, OBU, des RP.
. Quand un RP veut envoyer un message à CP, il le place dans son OBU puis continue son travail.
. Quand RPH trouve un OBU plein, il envoie un signal au RP concerné. Celui-ci achemine alors le message dans le buffer d'input de RP, BRI.
. Un signal est envoyé au CP pour qu'il interrompe son travail et qu'il place le contenu de BRI dans sa mémoire, après quoi il retourne à son travail.

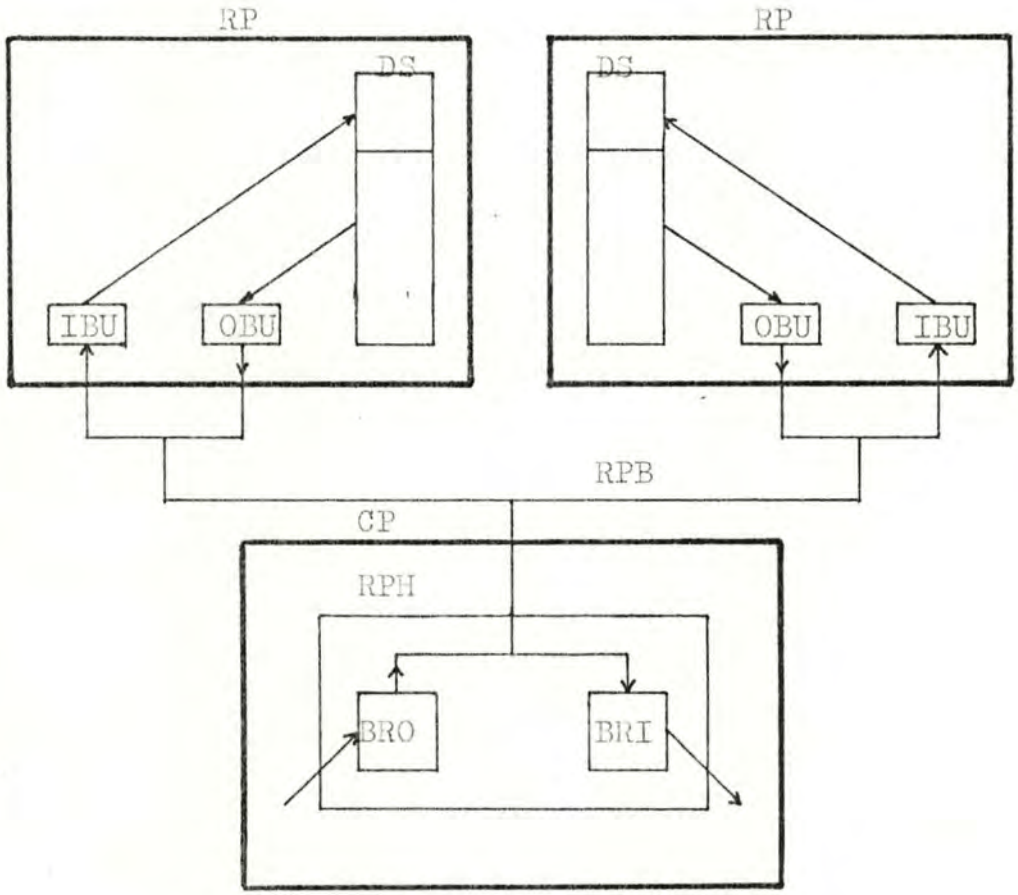


Fig E31. Liaison CP-RP.

c.3) Critique.

* Avantages :

- Lien direct rapide RP-CP.
- Répartition des processeurs et des langages de programmation suivant des conclusions tirées d'observations sur la nature des charges.
- Souplesse de l'affectation et la répartition des RP.
- Rapidité d'un logiciel central entièrement situé dans le même processeur.

* Désavantages :

- Structure moins limpide que le 1240.
- Risques de débordement du CP car il n'est pas multipliable à souhait.

2.2.2.2.D. Exemples d'interfonctionnement entre modules.

Dans le but d'illustrer ce qui a été dit jusqu'à présent, nous allons décrire la dynamique d'un commutateur téléphonique en fonctionnement. Cela permettra de voir comment les modules interagissent entre eux suivant leur type, et comment cet interfonctionnement réalise globalement les différentes phases de la procédure de signalisation téléphonique.

Dans un second point, on examinera comment les modules d'OS interviennent dans l'exécution de l'application.

(1) Interfonctionnement au niveau de l'application.

L'exemple choisi est le segment de procédure compris entre la détection du décrochage sur une ligne d'abonné appelant et la mise en place du système de sorte qu'il puisse recevoir la numérotation.

Ce segment sera décrit pour les deux systèmes 1240 et AXE. Il n'est pas indispensable à ce niveau d'avoir une connaissance approfondie de l'architecture logicielle de ces systèmes. Cependant, pour plus de détails à ce sujet, on peut se reporter à la section 2.3.

a) Description générale du segment de procédure.

1. Un abonné décroche son téléphone.
2. Le commutateur détecte le décrochage.
3. Une tâche de contrôle de la communication est lancée dans dans le commutateur.
4. Des informations sur l'abonné sont extraites des bases de données.
5. Un récepteur de numérotation est affecté à la ligne appelante.
6. Une tonalité d'invitation à numéroter est envoyée à l'appelant.
7. Le système se met en attente de la numérotation.

b) Version 1240.

b.1) Modules concernés.

LCF : matrice de contrôle de ligne. Un bit (SHD) indique le niveau haut ou bas sur la ligne.
DHSSM : procédures de scanning de ligne.
LCDH : FMM de contrôle de ligne.
SLS : FMM de signalisation de ligne (60 lignes).
SIGC : FMM de contrôle de signalisation pour un groupe de lignes (480).
PRECC : FMM de contrôle de communication pour la partie préfixe.
SCDH : FMM de contrôle du récepteur de numérotation.
NH : primitives de gestion du réseau de connexion.
RSIG : récepteur de numérotation.

RCN : réseau de connection numérique.

Note : les FMM ont une partie superviseur et une partie application (voir B.4, D.3).

b.2) Sous-phases.

Note : on suppose les processus d'application LCDH, SCDH et SLS déjà initialisés.

1. DHSSM détecte un décrochage sur une ligne via le bit SHD de la matrice LCF.
2. Message 9832 de DHSSM à LCDH.
3. LCDH lance une procédure DHSSM de retrait de la ligne de l'IDLE-SCAN-LIST (lignes libres).
4. Message 9875 de LCDH à SLS.
5. SLS extrait des données-ligne de la base de données COL (procédure Q-GET-COL).
6. Message 0096 Basic de SLS à SIGC partie superviseur (procédure Q-0096-SZE-ORG).
7. Création d'un processus d'application SIGC par la partie superviseur de SIGC.
8. Message 9197 de SIGC superviseur à SIGC application.
9. Message 0310 Basic de SIGC application à PRECC superviseur.
10. Création d'un processus d'application PRECC par PRECC superviseur.
11. Message 9199 de PRECC superviseur à PRECC application.
12. PRECC extrait des données-abonné de la base de données COS.
13. Message 9527 de PRECC à SIGC.
14. Message 1707 de SIGC à SCDH.
15. SCDH appelle une routine NH de connection avec L/TCE.
16. Message 0107 de NH vers LCDH.
17. LCDH fait une demande d'établissement de chemin au NH.
18. SCDH fait une première demande d'établissement de chemin au NH.
19. SCDH fait une deuxième demande d'établissement de chemin au NH.
20. Message 0115 de SCDH à RSIG.

21. RSIG met le récepteur en condition active.

22. Message 9824 de RSIG à SCDH.

23. SCDH fait une demande de connection à NH.

24. Message 9818 de SCDH à SIGC.

25. Message 8926 de SIGC à SLS.

26. Activation d'une procédure de scanning DHSSM par SLS.

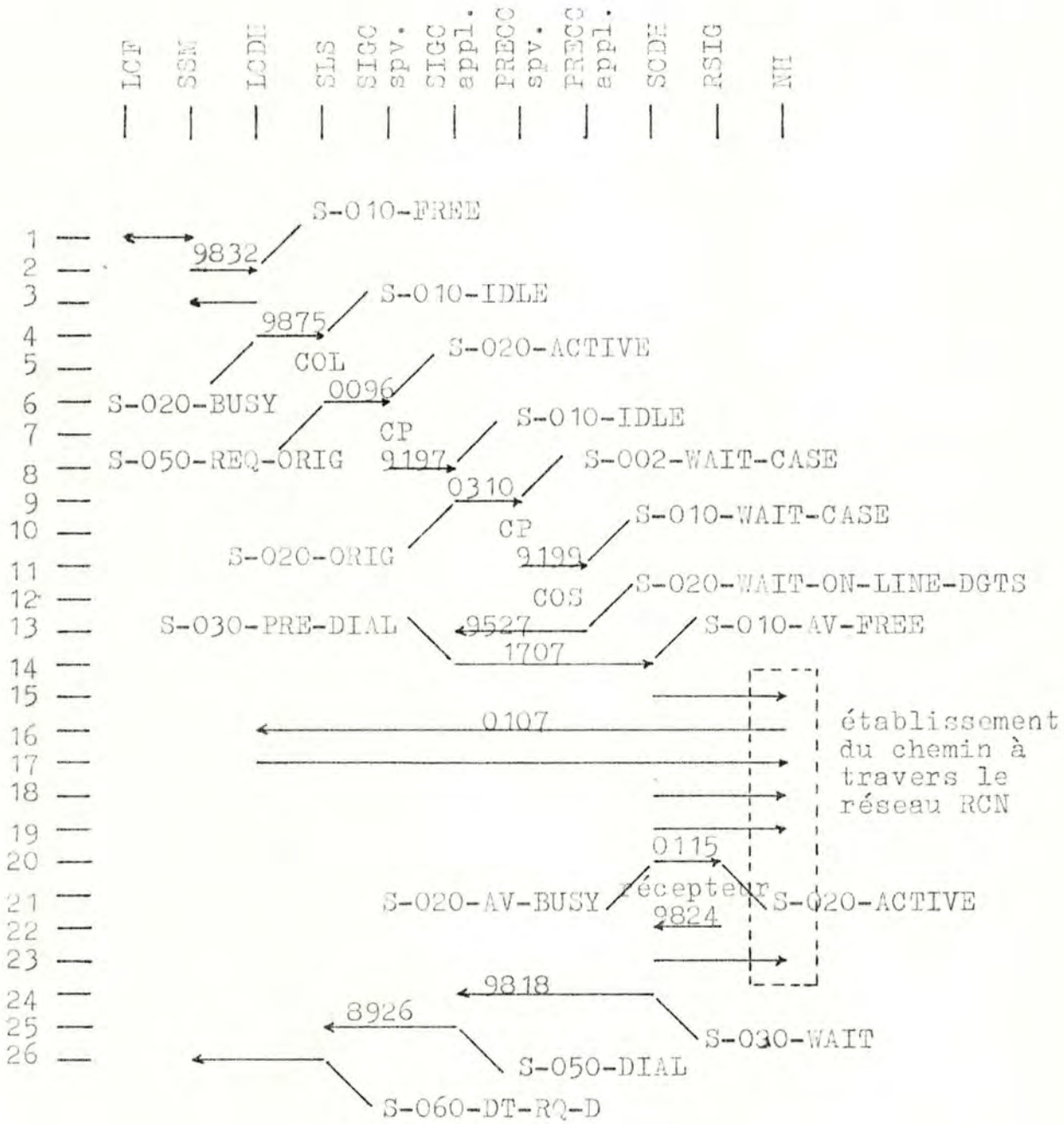


Fig E32. Séquence téléphonique sur 1240.

c) Version AXE 10.

c.1) Modules concernés.

LI : interface de ligne (DHSSM dans le 1240).
TS : multiplexeur temporel.
KR : récepteur de numérotation (SCDH + RSIG dans le 1240).
CJ : joncteur combiné (SIGC dans le 1240).
JT : terminal joncteur (NH dans le 1240).
SC : analyse de la classe d'abonné(SLS + PRECC dans le 1240).
RE : fonctions de registres (SIGC dans le 1240).
DA : analyse de chiffres (PATED + LSIF dans le 1240).
GSS : réseau de connection.

c.2) Sous-phases.

1. LI détecte un décrochage appelant.

2. LI vérifie si cet abonné peut téléphoner.

3. Signal de LI vers CJ : demande de SET-UP et de contrôle de la communication.

4. CJ→JT : demande de réservation d'un canal MIC entre TS et GSS.

5. JT→CJ : accusé de réception.

6. CJ→SC : demande de renseignements sur la classe de l'abonné.

7. SC→CJ : réponse à 6.

8. CJ→LI : demande de connection de l'abonné.

9. LI→TS : connection de l'abonné.

10. LI→CJ : accusé de réception à 8.

11. CJ→KR : demande de sélection d'un récepteur de numérotation.

12. KR→TS : informer TS du device KR choisi. TS connecte alors LI avec ce device.

13. TS→KR : accusé de réception de 12.

14. KR→CJ : accusé de réception de 11. (A ce point, le hardware est prêt)

15. CJ→SC : demande de préparation à l'analyse des chiffres.

16. SC→RE : demande de réservation d'un bloc de registres pour le contrôle de la communication.

17. RE→ DA : envoi des informations de base au module d'analyse de numérotation.

18. DA→ RE : renvoi de l'adresse de départ d'analyse.

19. RE→ CJ : prêt pour l'envoi de la tonalité d'invitation à transmettre.

20. CJ→ LI : mettre la ligne dans l'état réception.

21. CJ→ KR : demande d'envoi de la tonalité d'invitation à transmettre.

22. KR : envoie la tonalité à l'abonné appelant.

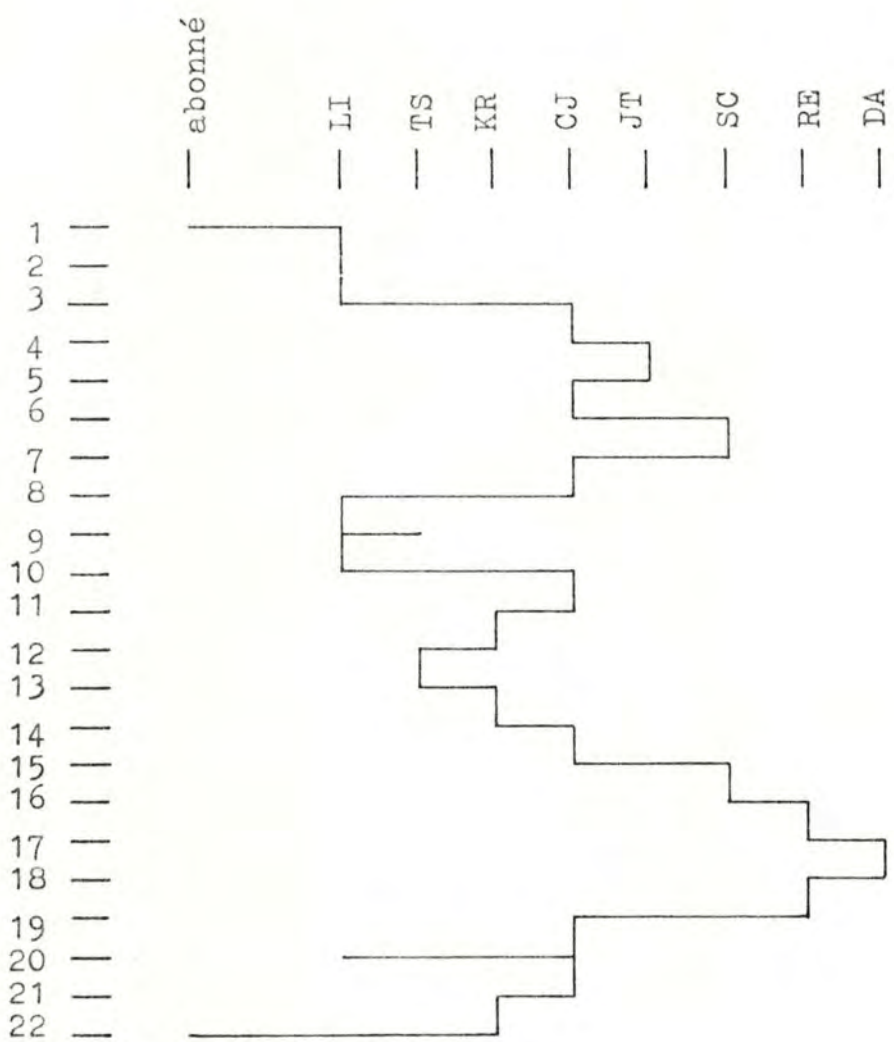


Fig E33. Séquence téléphonique sur AXE.

(2) Interfonctionnement entre application et OS.

a) Situation envisagée pour l'exemple.

Soient 500 lignes :

- la ligne 118 est en état numérotation (scanning 10 msec).
- la ligne 27 est en état conversation (scanning 300 msec).
- les autres lignes sont dans l'état libre (scanning 100 msec).

Les modules suivants (repris du prototype exposé plus loin) interviennent dans le schéma :

Application :

ILS : interface ligne-système.
COLIG : contrôleur de lignes (un seul processus).
COCOM : contrôleur de communications (un processus par communication).
TAXAT : traitement de la taxation.

OS :

GM : gestion de la mémoire.
GX : gestion des échanges .
ST : services de temporisation .

b) Conception du schéma.

- La tâche active est représentée par un trait plein.

- Les tâches concurrentes qui pourraient être actives au même moment sont représentées par des traits discontinus.

- l'axe vertical du temps est gradué en millisecondes. On suppose arbitrairement que chaque morceau de tâche s'exécute en une milliseconde.

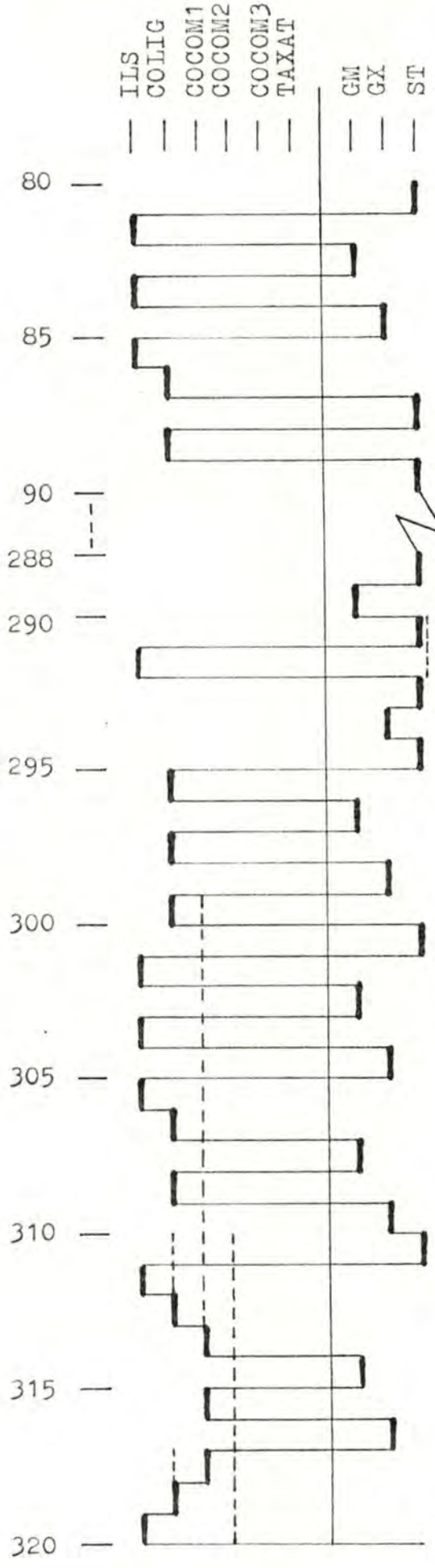
- Les modules moniteur et scheduler intervenant constamment, ils n'ont pas été représentés ici, dans un but de clarté du schéma.

c) Déroulement des événements.

- Le dernier chiffre va être reçu sur la ligne 118.

- La ligne 27 va raccrocher.

- La ligne 384 va être réveillée.



Scanning numérotation
Envoi message 1201: détection événement sur ligne 118 rattachée à COCOM1

Arrêt du timer intercréneau et lancement d'un nouveau timer (200 msec)
Interruption pour scanning 10 msec

(dernier chiffre)
Timer 200 msec arrive à échéance
Interruption pour scanning 10 msec (rien à signaler)

Envoi d'un message pour l'informer de l'échéance du timer

COLIG établit le chiffre et l'envoie àCOCOM1. Ce chiffre est le dernier de la série.

Interruption pour scanning 10, 100 et 300 msec
Détection d'un raccrochage sur la ligne 27 en conversation contrôlée par COCOM2

COLIG le signale à COCOM2 (msg 2307)

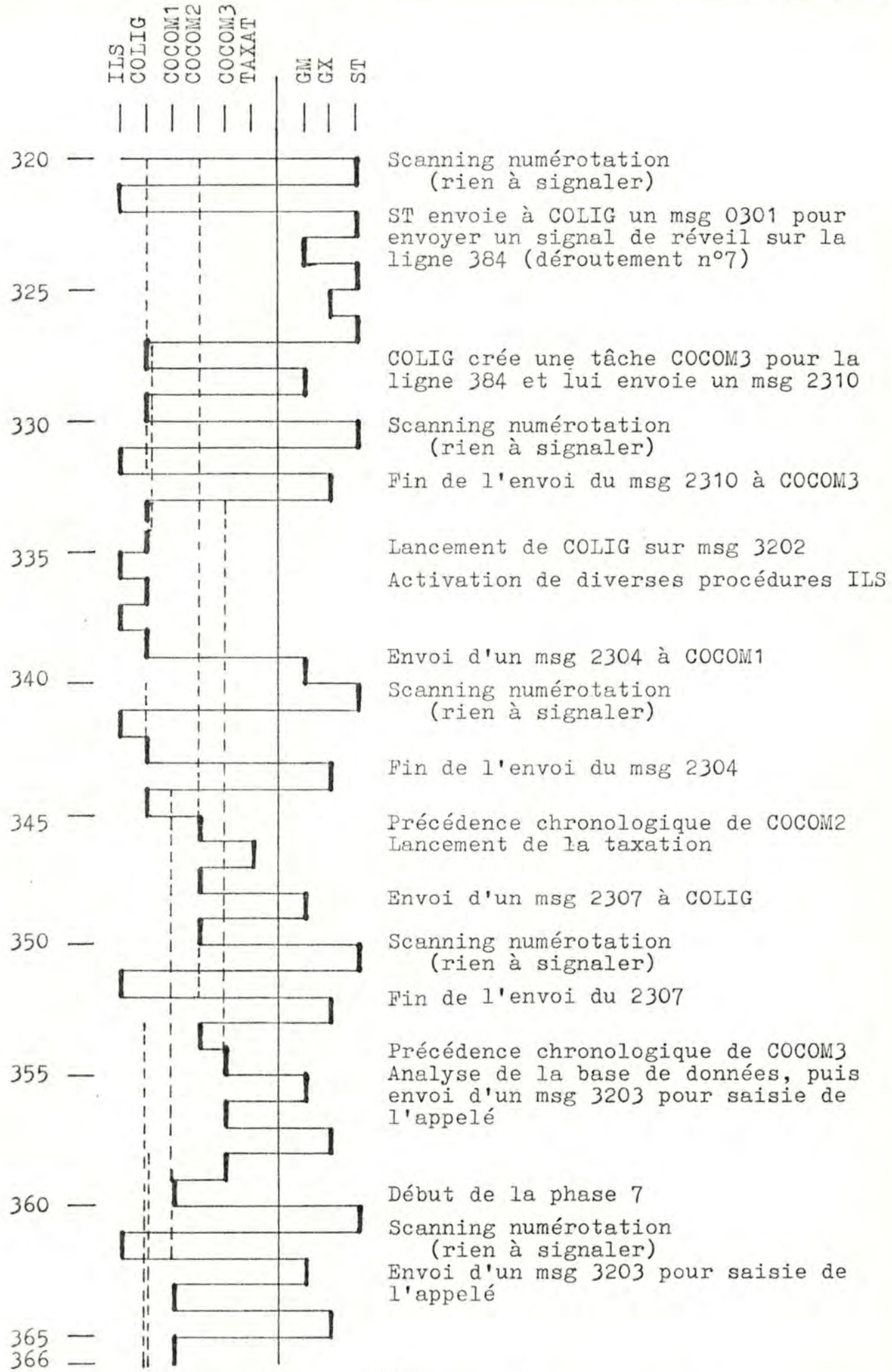
Interruption pour scanning 10 msec (rien à signaler)

COLIG, interrompu, est repris

COCOM1 envoie un msg 3202 à COLIG

COLIG lance ILS

(a)



(b)

Fig E34. Interfonctionnement OS-application.

2.3. Architecture des commutateurs.

2.3.1. Introduction.

Dans un système informatique ordinaire, le software est structuré en vue de réaliser des fonctions de calcul et de traitement de données informatisées. On s'accorde alors généralement sur la démarche nécessaire pour obtenir une architecture logicielle adéquate.

Dans le cas d'un commutateur , le problème est tout différent car il s'agit ici de contrôler un système physique indépendant de toute notion informatique. Ce système est soumis aux contraintes de la réalité, c'est-à-dire du temps réel sous tous ses aspects.

Ces deux éléments, existence d'un système indépendant, et contraintes du temps réel, vont avoir une influence déterminante sur l'architecture du système informatique de contrôle. De plus, ils donnent beaucoup plus de liberté aux principes et à la démarche de conception d'une architecture.

Ceci a pour effet que les architectures logicielles des commutateurs existant actuellement sur le marché diffèrent fortement et sont relativement difficiles à comparer, étant donné la multiplicité des critères de distinction.

Enfin, il ne faut pas négliger l'incidence de la répartition des multiples microprocesseurs sur l'architecture du logiciel. Aussi bien dans AXE que dans le 1240, il y a une tendance à concevoir globalement une architecture matérielle et logicielle intégrée.

2.3.1. Spécification élémentaire d'un commutateur.

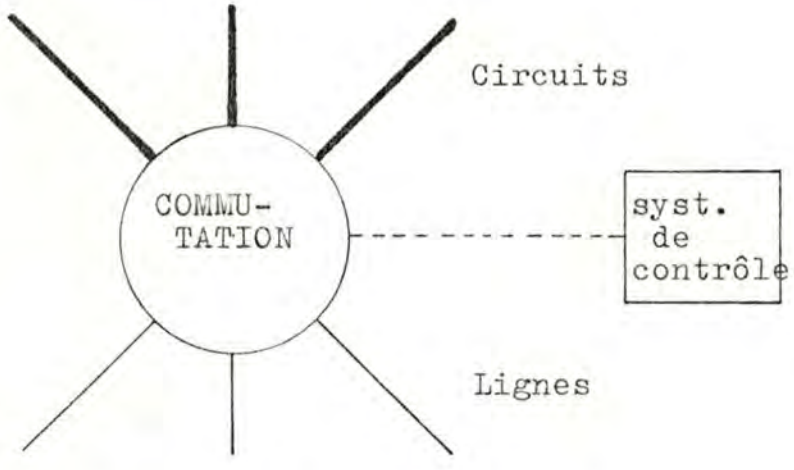


fig E35. Schéma simplifié d'un commutateur.

- Le hardware de commutation doit être en mesure

d'établir une jonction entre n'importe quelle paire de lignes d'abonnés et/ou de circuits intercommutateurs.

- Le système de contrôle gère le fonctionnement du commutateur et en particulier les points suivants :

. signalisation sur les lignes d'abonnés

. signalisation sur les circuits de jonction

. connection numérique des lignes et circuits

Il gère en plus des fonctions auxiliaires d'analyse de numérotation, de taxation, etc...

2.3.2. Emplacement du système de contrôle.

Comme on l'a dit plus haut, les choix sont multiples sur base de critères software et hardware. Les deux schémas qui suivant illustrent les conceptions de AXE et du 1240.

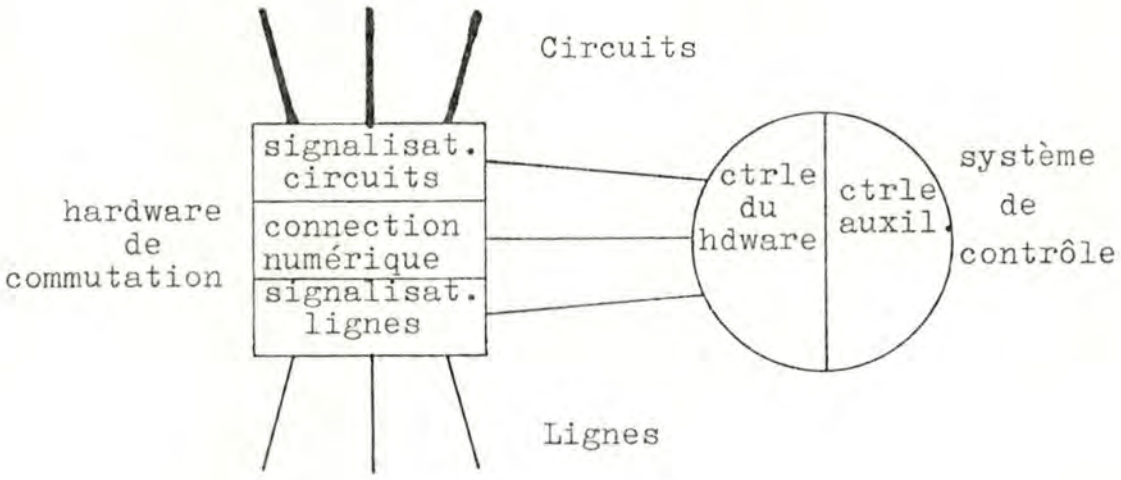


fig E36. Système de contrôle dans AXE.

Dans AXE, le système de contrôle est une unité bien distincte à côté du système de commutation. Une partie contrôle directement le système de commutation et une autre effectue les fonctions auxiliaires. Le système de commutation est directement relié aux lignes et circuits.

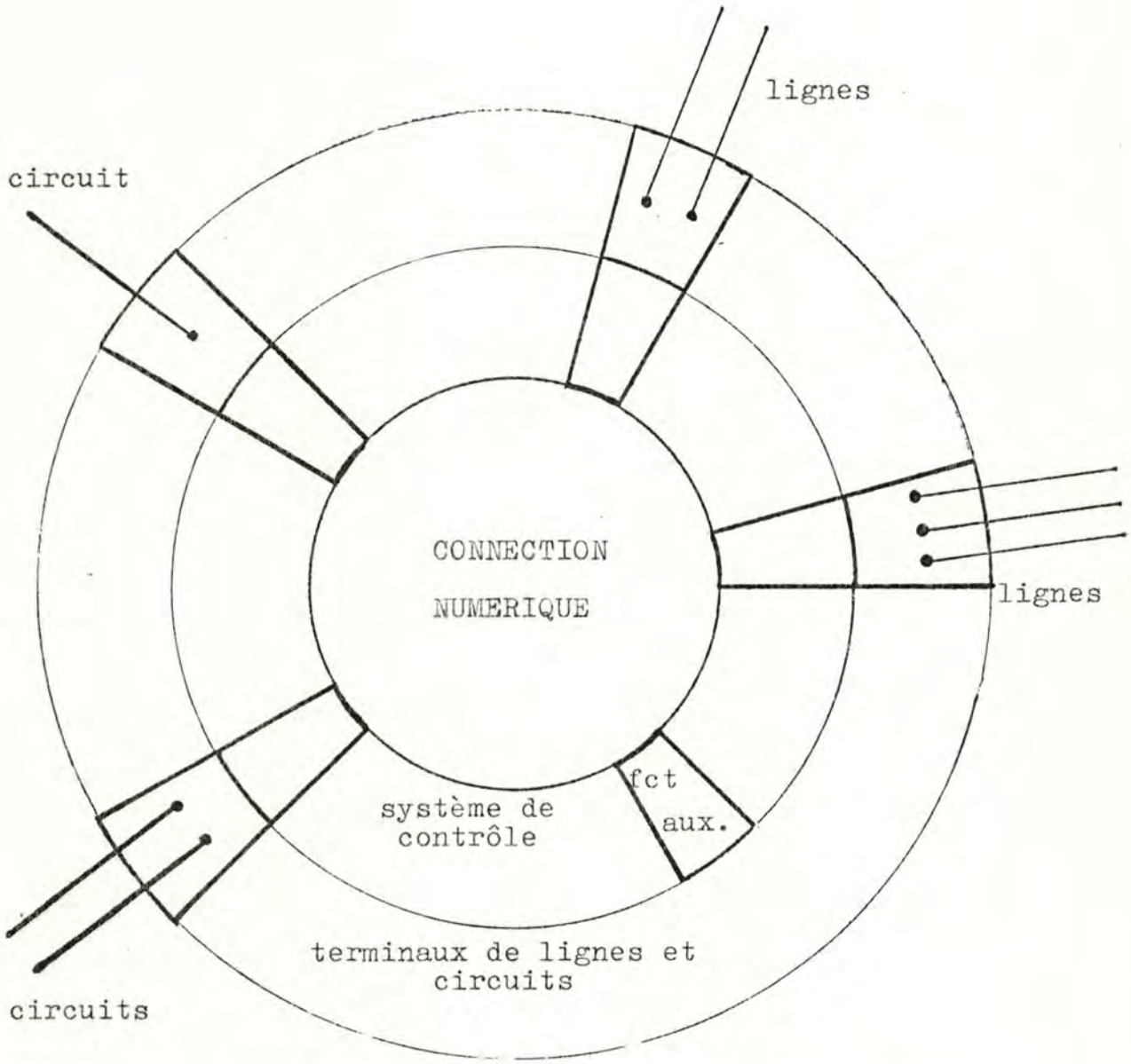


fig E37. Système de contrôle dans le 1240.

On voit que dans le 1240 on a placé le réseau de connection numérique au centre du commutateur. La couche suivante est constituée par le système de contrôle. La dernière couche comprend les terminaux de signalisation reliés aux lignes et circuits.

Le point important dans cette structure est que le système de contrôle est partagé en éléments indépendants affectés chacun soit au contrôle d'un terminal, soit à l'exécution d'une fonction auxiliaire, et qu'ils ne peuvent communiquer entre eux que via le réseau central de connection.

Si l'on se reporte à la section 2.2.2.2.C.2. sur les processeurs multiples, on verra qu'il y a une correspondance immédiate entre ces structures et l'emplacement des processeurs.

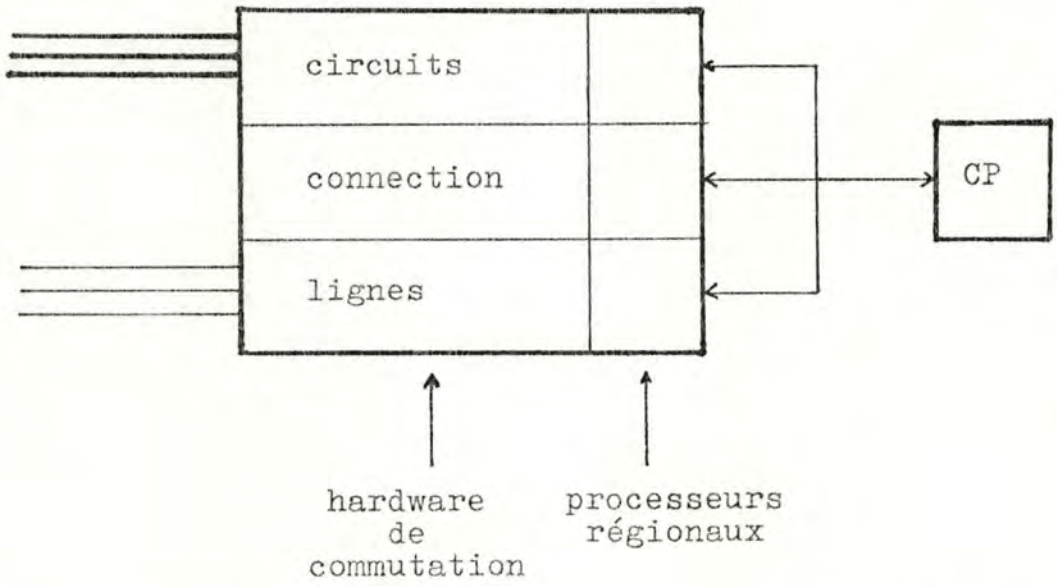


fig E38. Structure des processeurs dans AXE.

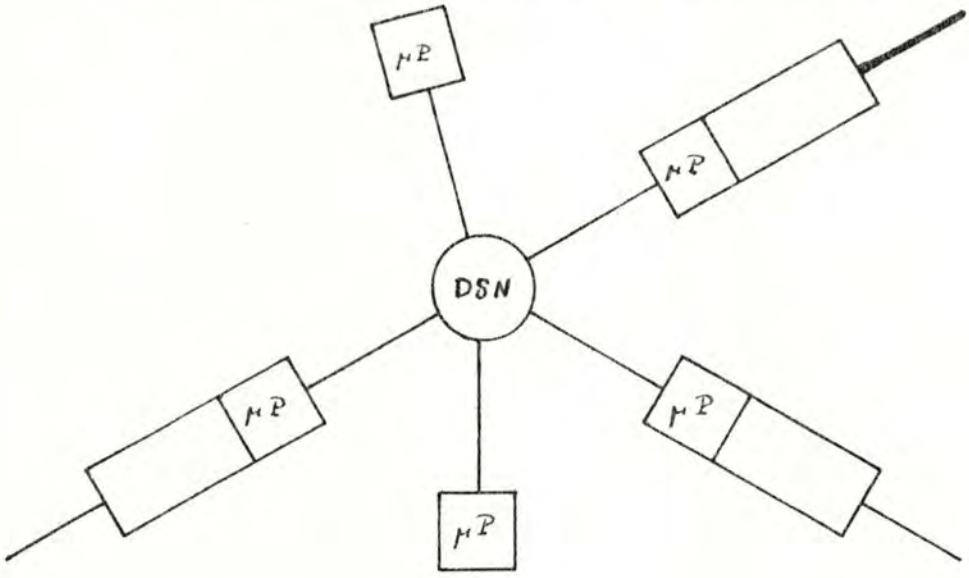


fig E39. Structure des processeurs dans le 1240.

2.3.3. Principes de découpe architecturale.

Lorsqu'on regarde de près les principes de découpe dans AXE et le 1240, on s'aperçoit qu'ils ont de nombreux points communs, au-delà des différences de présentation et de vocabulaire.

Il n'y a tout compte fait que deux diférences marquantes :

. la place et la conception du réseau de connection

. la disposition des processeurs physiques.

Voyons à présent les principes de découpe dans AXE et le 1240.

a) AXE.

C'est une découpe à cinq niveaux :

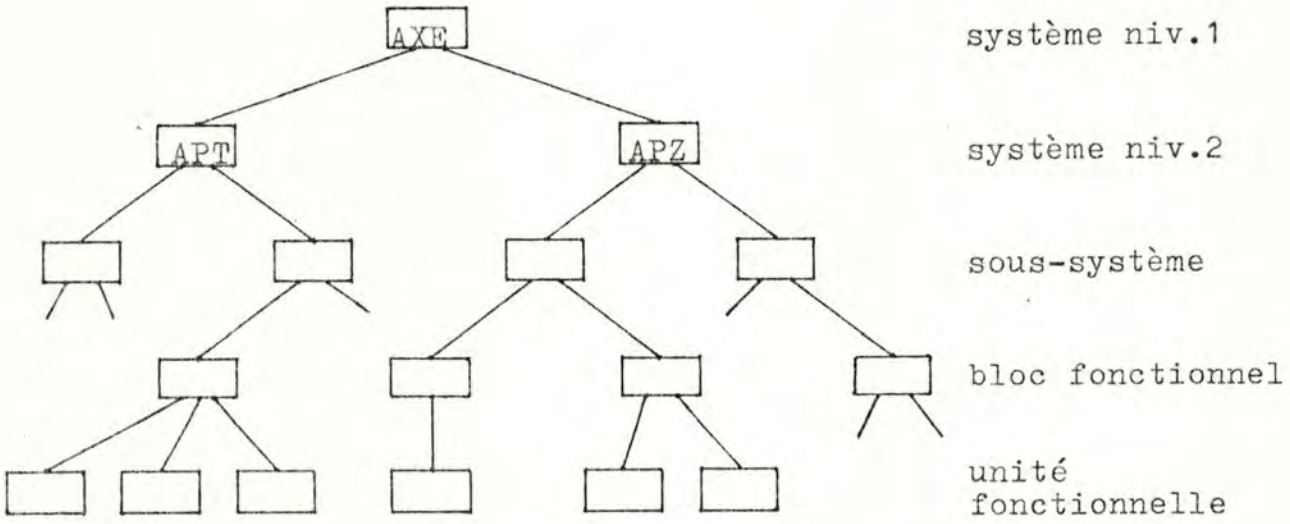


fig E40. Conception architecturale de AXE.

1. Le niveau 1 système est le système AXE lui-même.
2. Le niveau 2 système ne comprend que deux modules :
   . APT : système de commutation.
   . APZ : système de contrôle.
3. Les modules du niveau sous-système reprennent des ensembles de fonctions remplissant un rôle commun. Par exemple :
   . SSS : signalisation sur les lignes d'abonnés
   . TSS : signalisation sur les circuits de jonction

4. Les blocs fonctionnels sont les éléments fonctionnels de base dans la construction du logiciel. Ils ont chacun une fonction bien déterminée à remplir. Par exemple :

   . DA : analyse de la numérotation téléphonique

   . ETC : échanges sur les circuits de jonction

   . LIC : supervision des lignes d'abonnés

5. Les unités fonctionnelles sont de trois types :

   . hardware

   . software régional

   . software central

Chaque bloc fonctionnel est composé d'une combinaison quelconque de une, deux ou trois unités fonctionnelles. Le software régional est le software de bas niveau destiné à contrôler le hardware de très près. Le software central est le software de haut niveau.

b) 1240.

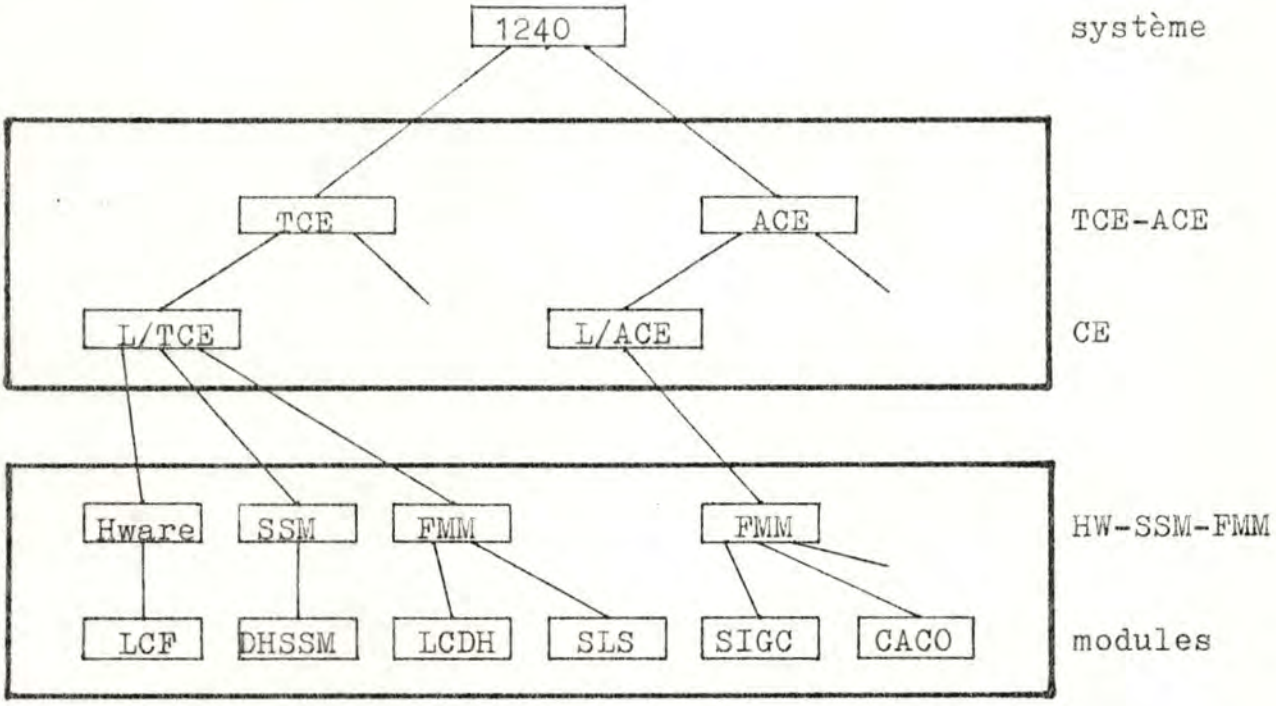


fig E41. Conception architecturale du 1240.

La structure du 1240 est plus simple que celle d'AXE.

Il n'y a en fait que trois niveaux : le système, les éléments de contrôle (CE) et les modules.

Cependant, la figure présente une structure plus détaillée, dans un but de comparaison avec AXE.

C'est ainsi qu'on voit qu'il y a deux grandes classes de CE :

- les TCE ou CE "terminaux" chargés de contrôler du hardware de commutation
- les ACE ou CE "auxiliaires" ne contrôlant aucun hardware, mais apportant simplement un complément de puissance de traitement.

Parmi les modules, on voit qu'il y a deux catégories principales : les SSM et les FMM. Les SSM assument les tâches de contrôle proches du hardware tandis que les FMM ont un rôle de plus haut niveau.

c) Comparaison.

Il y a une correspondance immédiate entre

- sous-système (AXE) et CE (1240)
- unité fonctionnelle de software central (AXE) et module FMM (1240)
- unité fonctionnelle de software régional (AXE) et module SSM (1240)

Quant à cette dernière distinction, AXE est plus systématique. Pour AXE, un bloc fonctionnel complet est composé des trois parties :

fig E42. structure d'un bloc software dans AXE

On retrouve ces trois parties dans le 1240, par exemple dans le cas suivant :

fig E43. exemple de correspondance de modules 1240 avec un bloc fonctionnel AXE

. LCF est la matrice hardware de contrôle d'une ligne d'abonné

. DHSSM est le software de bas niveau de contrôle de LCF

. LCDH est le software de haut niveau de contrôle des lignes

On voit donc la correspondance, bien que la description des modules 1240 ne soit pas aussi systématique.

2.3.4. Les grandes fonctions.

2.3.4.1. Contrôle des lignes d'abonnés.

a) Description.

- Détection des événements sur les lignes au moyen du scanning

- Tenue à jour de tables conservant l'état des lignes et divers renseignements supplémentaires (nature du poste, abonné raccordé ou non,...)

- Contrôle de la signalisation sur les lignes

b) 1240.

Un CE est affecté à cette fonction : le TCE de lignes d'abonnés comprenant :

- DHSSM : SSM de scanning et mise à jour des lignes

- LCDH : FMM de contrôle des lignes

- SLS : FMM de logique de signalisation d'abonnés.

Une base de données (COL) y est adjointe. Elle contient des données concernant les lignes.

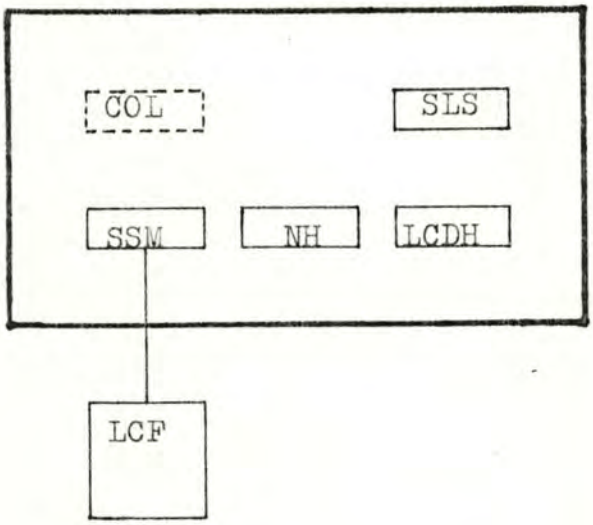


fig E44. ITT 1240 : L/TCE.

c) AXE.

La fonction constitue une partie du sous-système SSS (signalisation d'abonnés). Elle est réalisée dans le module LIC (circuit d'interface de lignes).

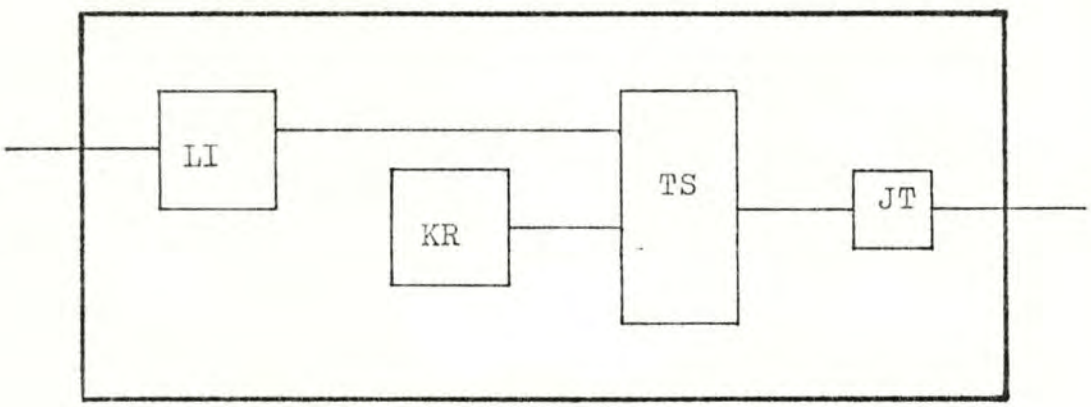


fig E45. AXE 10 : SSS.

2.3.4.2. Contrôle des circuits de jonction.

a) Description.

- Signalisation sur les circuits de jonction.
- Gestion des trames d'information sur les circuits.

b) 1240.

Un CE est affecté à cette fonction : le TCE de circuits comprenant :

- DHSSM : SSM de contrôle du hardware des circuits
- TCDH : FMM de contrôle des circuits
- OLS/ILS : FMM de logique de signalisation sur circuits input/output.

Une base de données (COT) y est adjointe.

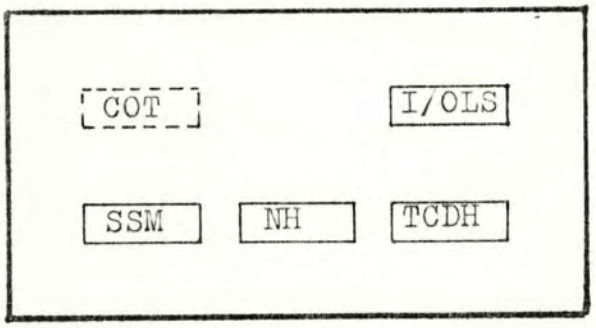


fig E46. ITT 1240 : T/TCE.

c) AXE.

Un sous-système est affecté à cette fonction : TSS (signalisation de circuits). Il comprend des blocs différents suivant que le circuit est analogique ou digital.

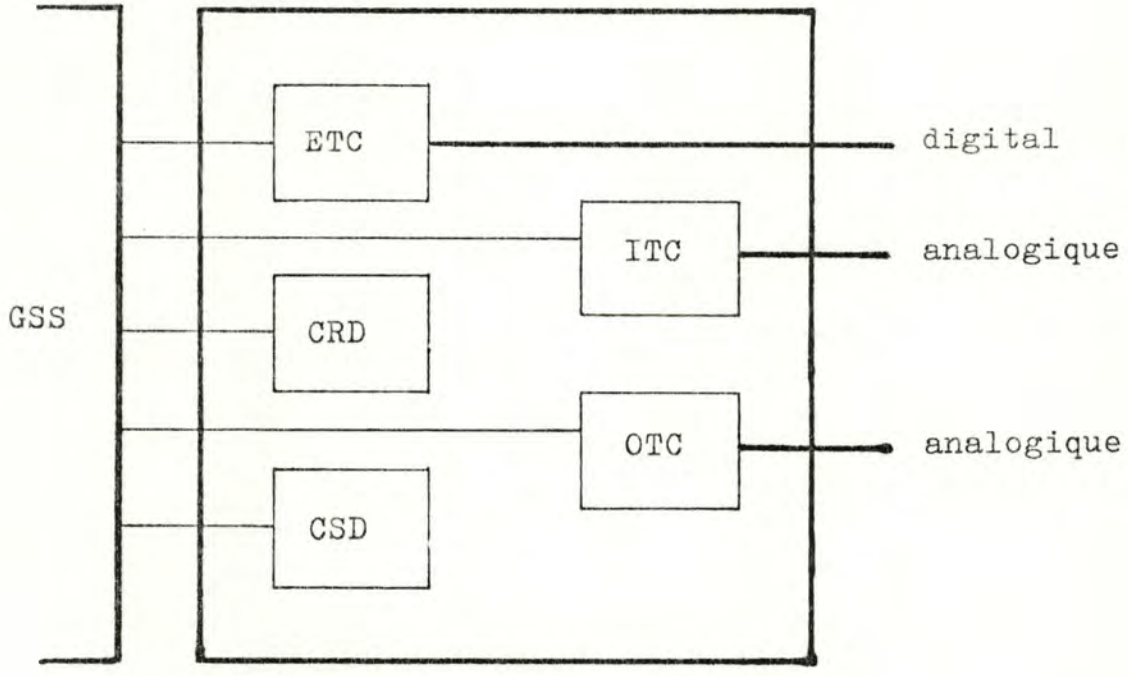


fig E47. AXE 10 : TSS.

2.3.4.3. Connection et commutation.

a)Description.

Cette fonction consiste à effectuer la commutation elle-même, c'est-à-dire la mise en connection de deux canaux.

Pour les figures, voir la section 2.2.1.

b) 1240.

Cette fonction est considérée comme le centre du commutateur et est appelée DSN (réseau de connection numérique). Tous les modules gravitent autour d'elle et doivent passer par elle pour communiquer entre eux. Le DSN n'est donc pas sur le même pied que les autres fonctions. Il n'a pas de commande propre: toute la commande est faite par les extrémités, c'est-à-dire par les autres modules.

Le DSN est constitué uniquement de commutation temporelle.

c) AXE.

Cette fonction est contenue dans le sous-système GSS (commutation de groupe). Elle est réalisée en temps-espace-temps.

2.3.4.4. Services hardware.

a) Description.

Ces services comprennent la réception de numérotation et l'envoi des tonalités et sonneries.

b) 1240.

Les services sont réalisés dans un CE à part : le SC/TCE ou TCE de circuits de services.

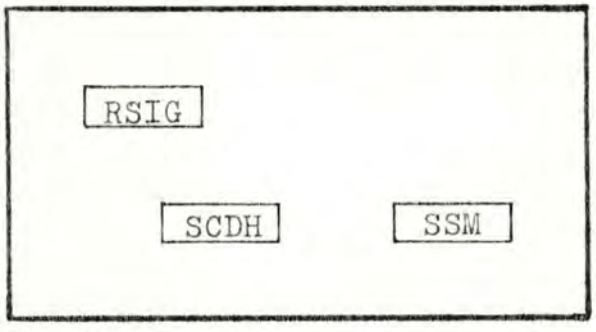


fig E48. ITT 1240 : SC/TCE.

c) AXE.

Les services sont compris dans le sous-système SSS avec la fonction de contrôle des lignes.

2.3.4.5. Contrôle des communications.

a) Description.

Cette fonction consiste en une supervision des communications entre abonnés depuis le début (SET-UP) jusqu'à la fin (RELEASE). Elle comprend l'extraction de données sur les abonnés, l'analyse de la numérotation, la saisie de l'appelé, la taxation, etc...

b) 1240.

Cette fonction est reprise dans les CE suivants :

* L/ACE : ACE de lignes.

* S/ACE : ACE de système.

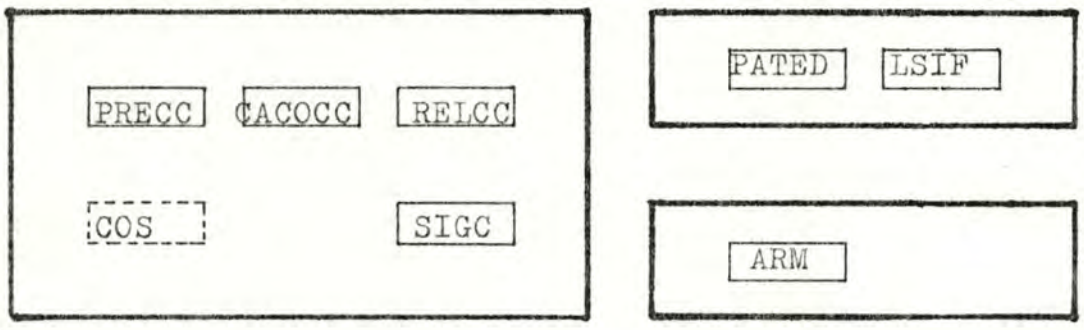


fig E49. ITT 1240 : L/ACE et S/ACE.

c) AXE.

Cette fonction est reprise dans les sous-systèmes suivants :

* TCS : contrôle du trafic.

* CHS : taxation.

* SUS : services divers aux abonnés.

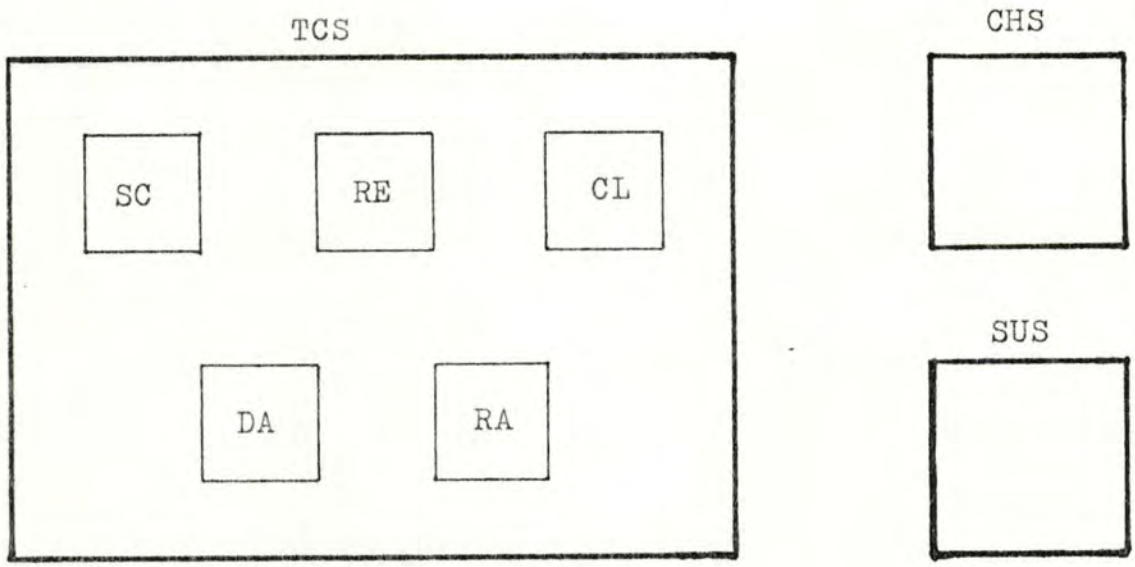


fig E50. AXE 10 : TCS, CHS et SUS.

2.3.5. Configurations.

Nous présentons ici des schémas de configurations moyennes 1240 et AXE.

a) 1240.

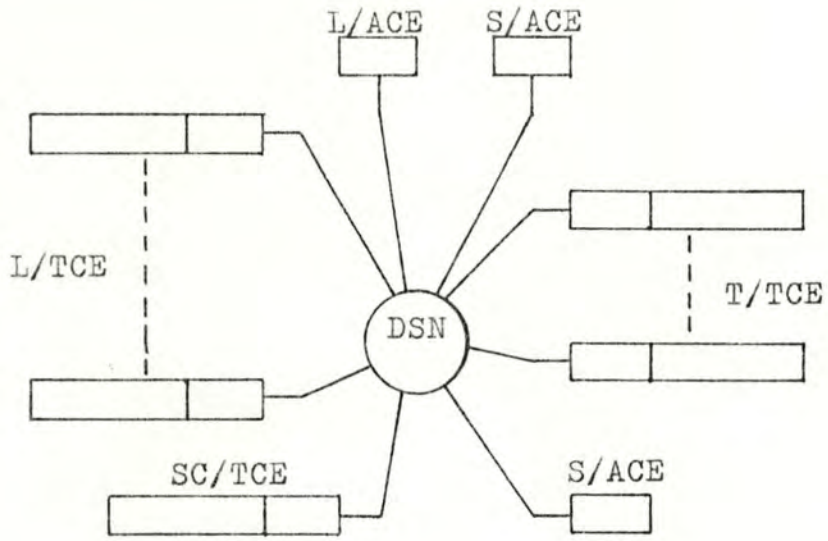


fig E51. 1240 : Schéma hardware.

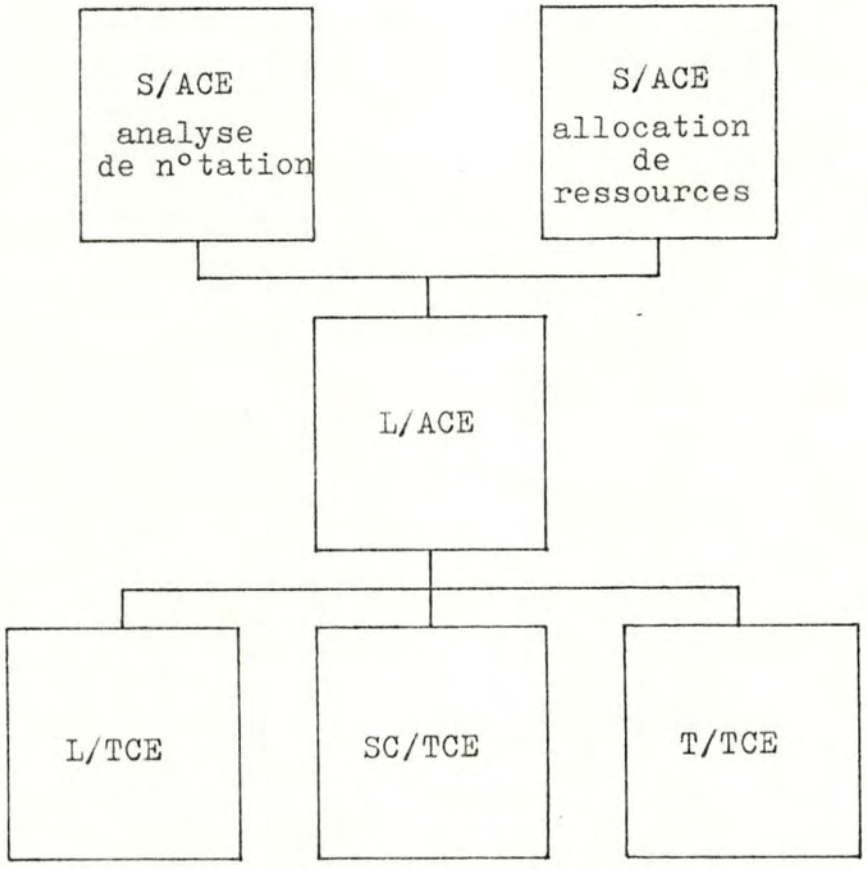


fig E52. 1240 : Structure logicielle.

b) AXE.

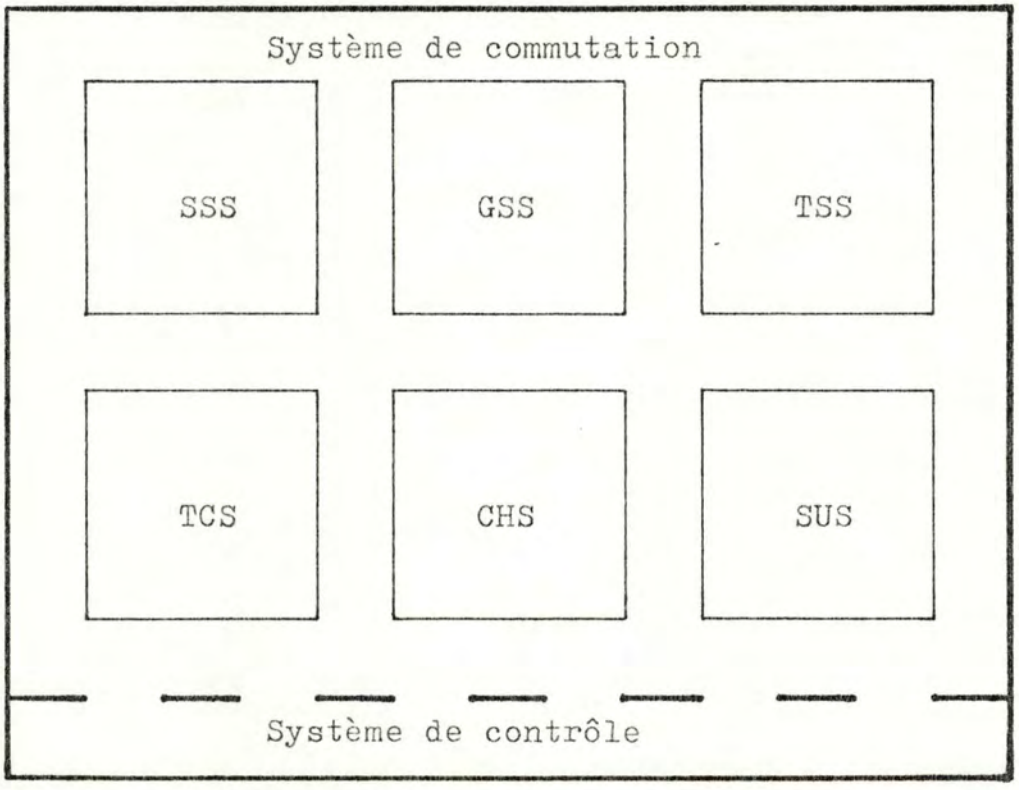


fig E53. AXE : Architecture.

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR (BELGIQUE)

INSTITUT D'INFORMATIQUE

COMMUTATION NUMERIQUE ET TEMPORELLE

Tome Second :

Conception d'un prototype de commutateur local téléphonique

Promoteur : J. Brunin

Mémoire présenté par

Marc EVRARD
Rudy VANGAVER
Benoît TRIGAUX

en vue de l'obtention du grade de

LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1984 - 1985

CHAPITRE 3

LE PROTOTYPE

3.1 Spécifications.

3.1.1. Application téléphonique.

Les chapitres 1 et 2 ont donné les bases générales de la commutation en temps réel. Ces notions peuvent s'appliquer à n'importe quel réseau commuté, qu'il soit dédicacé aux communications téléphoniques, à la transmission de données ou à tout autre service de type RNIS.

Le futur réseau RNIS devrait d'ailleurs permettre à tous les services de passer par le même réseau numérique, moyennant l'adjonction de modules de signalisation propres à chaque service.

Ce que l'on va décrire dans ce chapitre 3 est un prototype de commutateur

- réduit à l'application téléphonique
- réduit à la commutation locale : pas de circuit interurbain
- comprenant tous les détails concernant l'application et l'OS
- ne détaillant pas le hardware de commutation ni la commutation numérique en elle-même; seule la signalisation est traitée
- reprenant en plus de la communication téléphonique ordinaire tous les services particuliers comme l'appel à numérotation prédéterminée, le réveil, etc...

Ce système a reçu le nom de COLO/S, pour "COmmutateur LOcal / partie Signalisation".

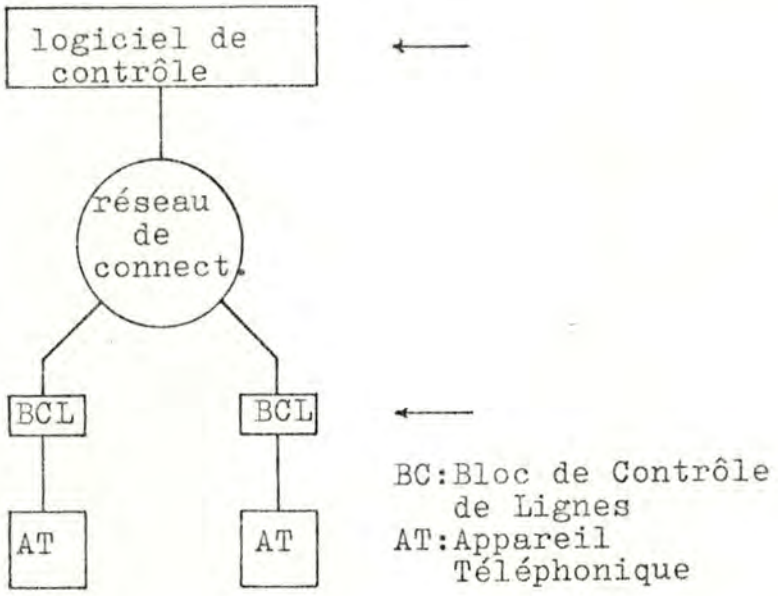


Fig E54. Schéma général de l'application.

Le schéma montre les différents éléments du système. Les flèches indiquent les éléments analysés dans COLO/S. Pour fixer un ordre d'idées, on peut considérer qu'il y a 500 lignes. De plus, on limite les postes d'abonnés aux seuls postes à cadran.

3.1.2. Séquence externe.

On spécifie l'application téléphonique en décrivant les différents événements qui surviennent sur les lignes d'abonnés et les différentes étapes d'une communication téléphonique qui en résultent. On se limite ici à l'aspect externe du commutateur, ce qui signifie que l'on aborde le problème du point de vue de l'utilisateur abonné.

3.1.2.1. Séquence générale.

On envisage ici une communication téléphonique complète, non interrompue avant qu'il y ait conversation entre les abonnés.

Deux lignes sont en communication, une appelante et une appelée.

On décrit pour chacune le niveau de potentiel existant sur la ligne (haut/bas), les événements qui surviennent, les réponses du système, et les états de la ligne.

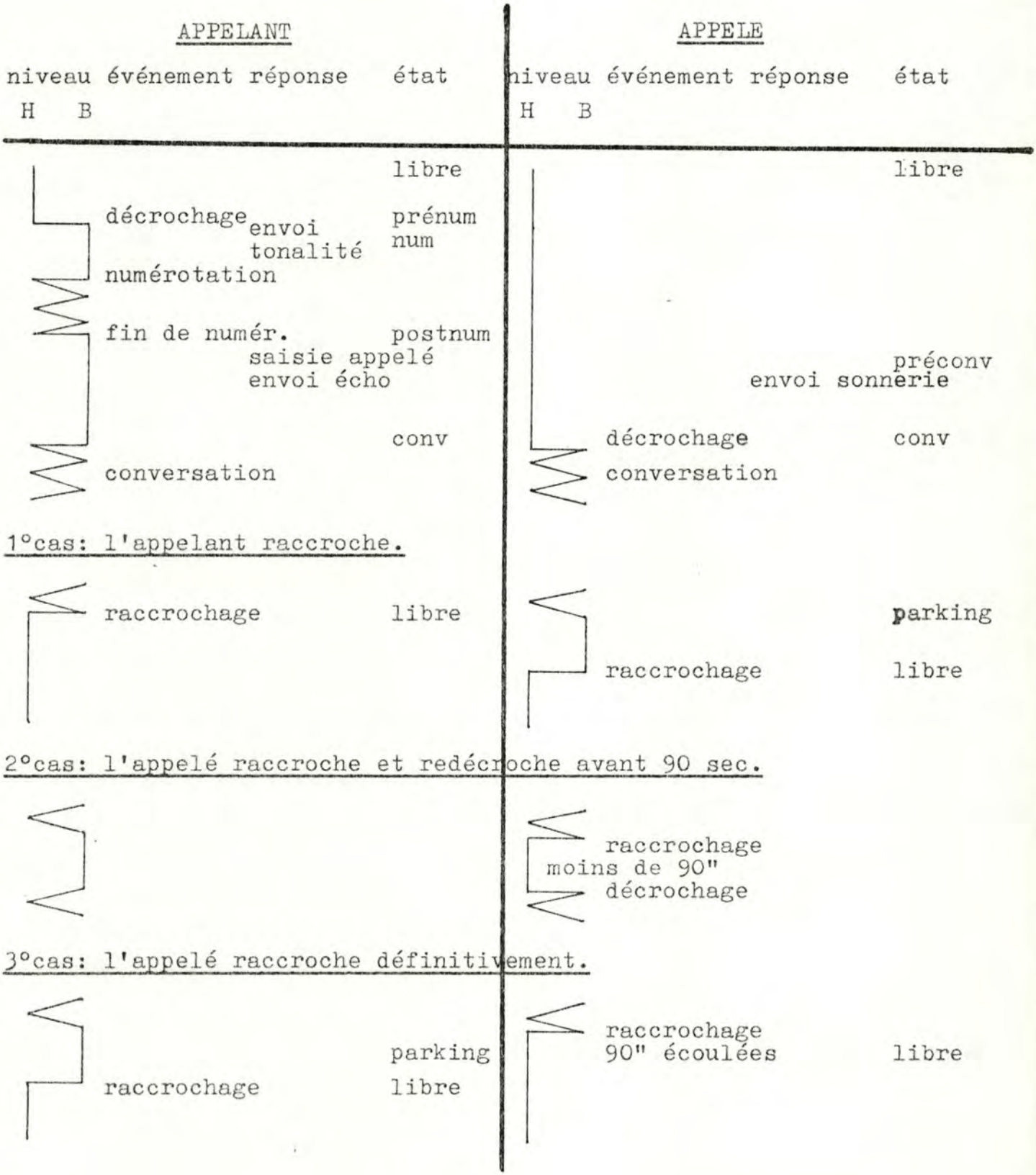


fig E55. Séquence téléphonique externe générale.

Les schémas suivants reprennent les passages d'états :

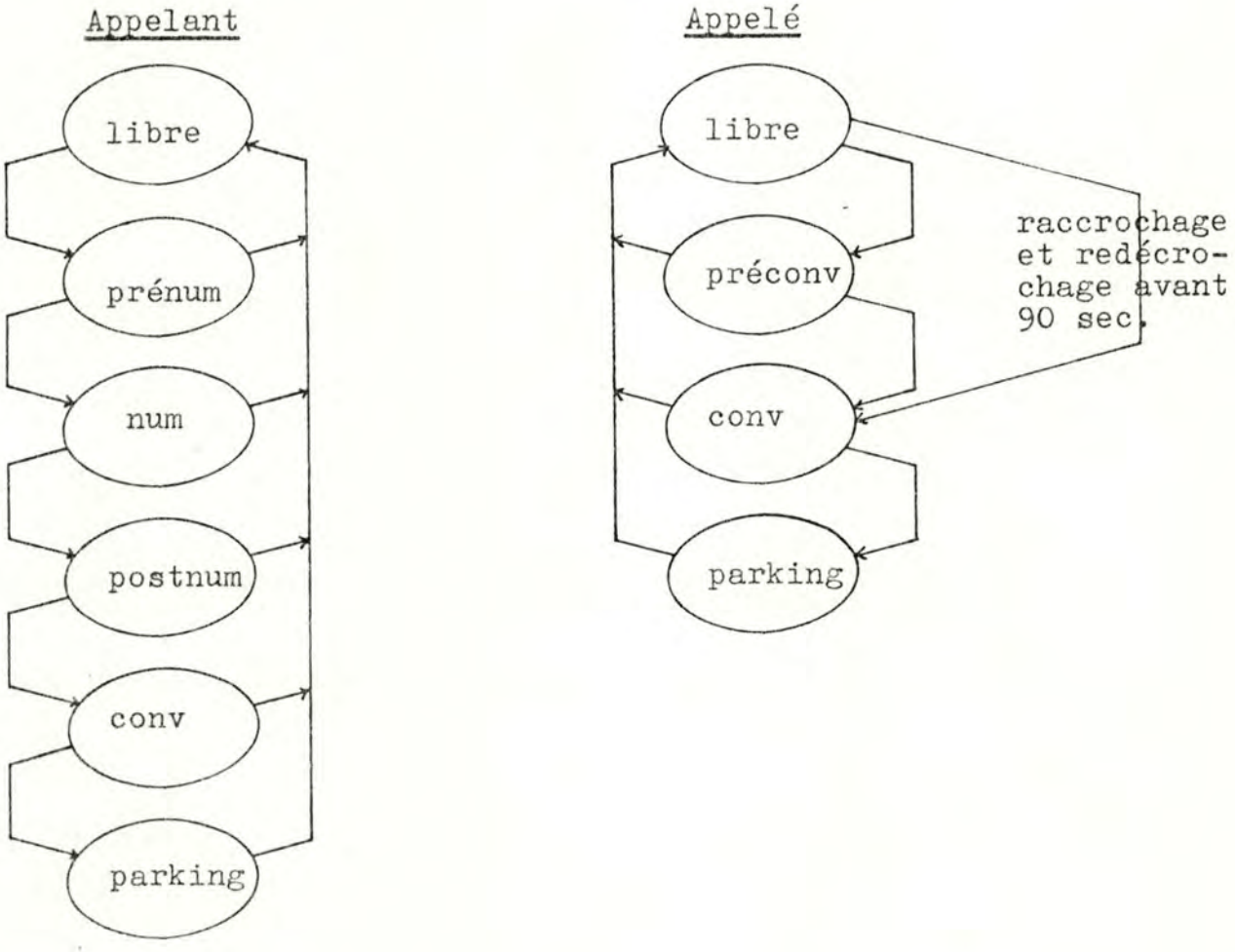


Fig E56. Transitions d'états de communication.

3.1.2.2. Séquence détaillée.

Un certain nombre de contraintes temporelles (TIME-OUT) sont imposées aux différentes phases de la communication. Passés ces délais, les lignes en décrochage sont forcées dans l'état PARKING en attendant d'être à nouveau LIBREs à la suite d'un raccrochage.

1. Un time-out de 24" est établi entre le début de l'invitation à numéroter et l'envoi du premier chiffre.

2. La numérotation se passe comme suit :

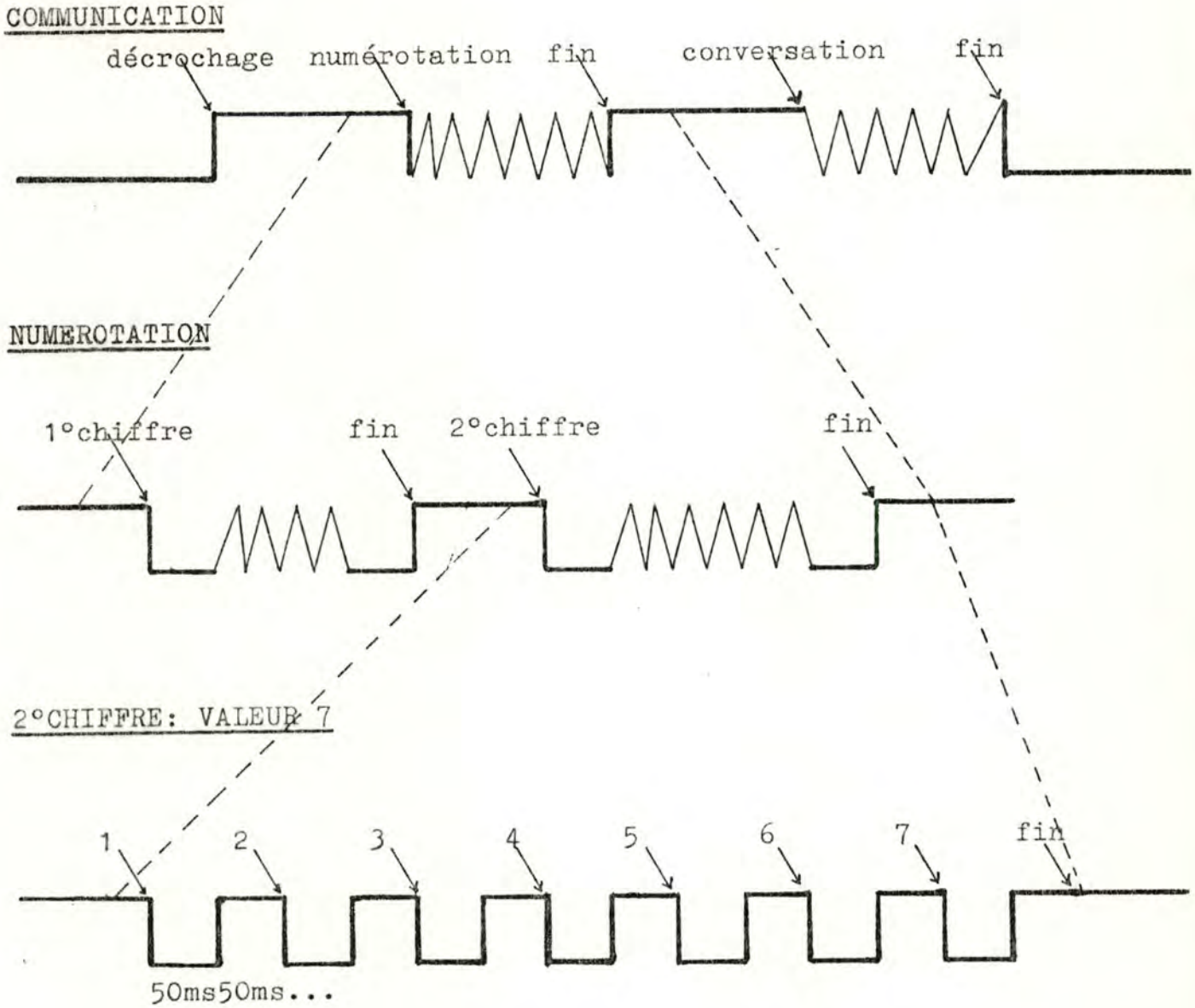


Fig E57. Mécanisme de la numérotation.

Les chiffres sont codés au moyen de créneaux ayant la forme ⌐¬. Avant un chiffre, la ligne est en niveau haut. Le chiffre est une suite de créneaux dont le nombre est égal à la valeur du chiffre lui-même. Dans un créneau, chaque niveau haut ou bas a une durée de 50 msec. Deux chiffres sont séparés par un niveau haut de durée comprise entre 200 msec et 24".

3.1.2.3. Séquences particulières.

Il existe à côté de la communication téléphonique ordinaire un certain nombre de séquences particulières

d'opérations. Ce sont :

- les déroutements : appels à des services particuliers.

- les relâchements qui interrompent une communication plus tôt que prévu.

  .normaux : l'appelant raccroche avant la phase de conversation.

  .forcés : le système met lui-même fin à une communication parce que l'appelant ne réagit pas dans les délais prescrits.

3.1.3. Taxation.

Le calcul de la taxation se fait en deux temps :

-établissement du tarif standard sur base de critères uniquement géographiques.

-application différenciée de ce tarif standard suivant des critères non géographiques.

1) Etablissement du tarif standard.

1.a) Communications zonales.

Une communication zonale est une communication entre deux abonnés appartenant à la même zone géographique. Le tarif est d'une unité de taxation pour 160 secondes.

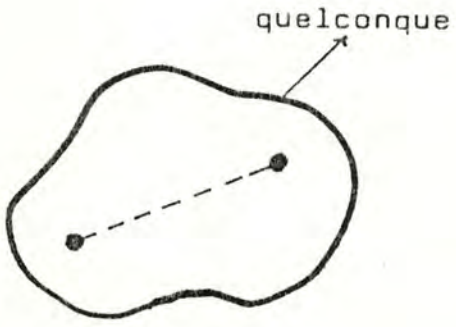


Fig E58. Communication zonale.

1.b)Communications interzonales de type A.

Ce sont des communications établies entre deux abonnés appartenant à deux zones géographiques petites et contiguës. Le tarif est d'une unité de taxation pour 160 secondes.

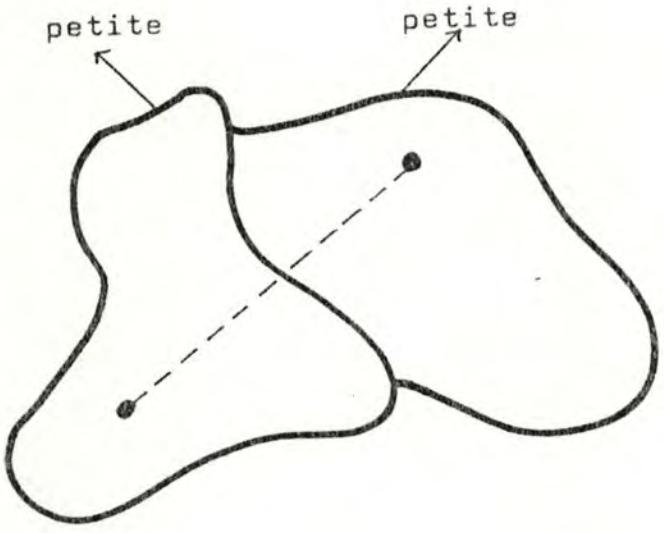
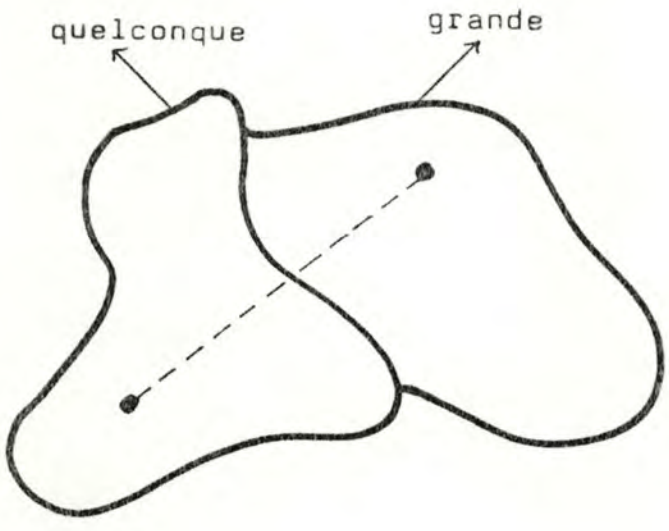


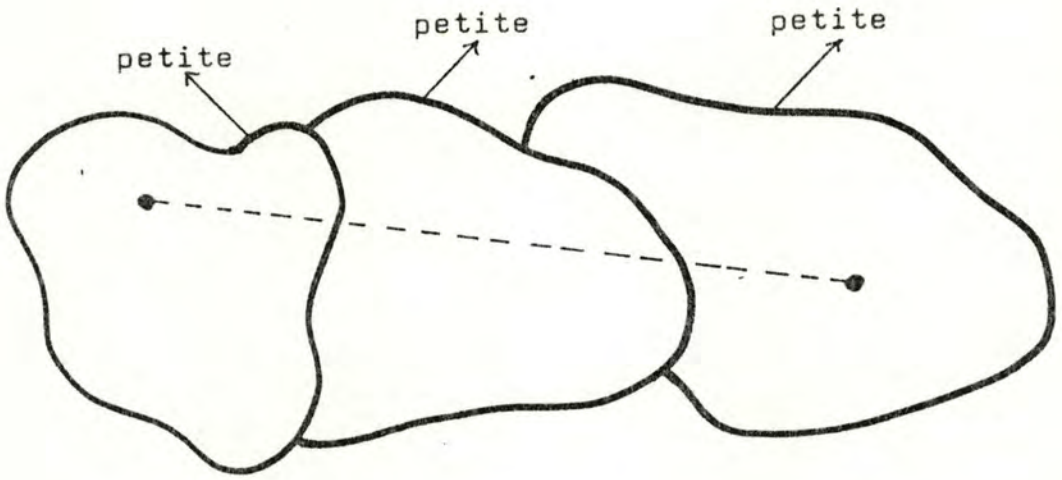
Fig E59. communication interzonale de type A.

1.c) Communications interzonales de type B.

Ce sont des communications établies entre deux abonnés appartenant à deux zones géographiques contiguës dont l'une au moins est une grande zone ou appartenant à deux zones petites séparées par une seule zone géographique petite. Le tarif est d'une unité de taxation pour 80 secondes.



(a)

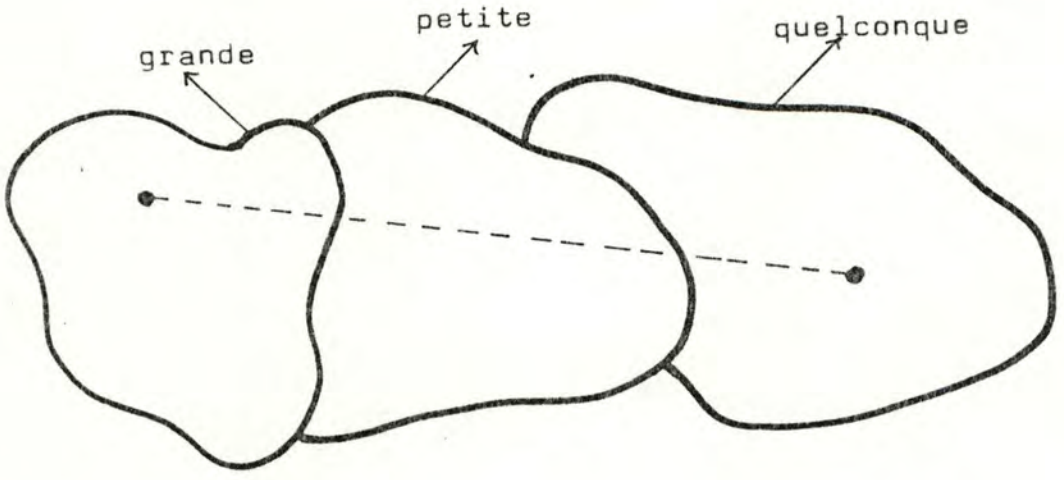


(b)

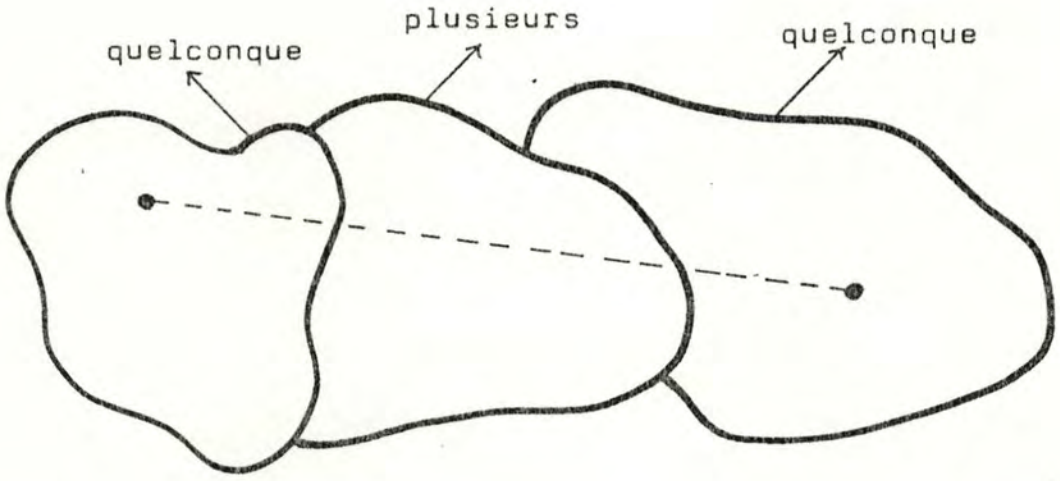
Fig E60. Communications interzonales de type B.

1.d Communications interzonales de type C.

Cette catégorie recouvre tous les autres cas de communications interzonales. Le tarif est d'une unité de taxation pour 40 secondes.



(a)



(b)
Fig E61. Communications interzonales de type C.

2) Application du tarif standard.

Les critères suivants modifient le montant calculé dans la première phase :

-Demi-tarif entre 18h30 et 8h00, et les samedis, dimanches et jours fériés légaux.

-Imputation à l'appelant ou à l'appelé, ou communication gratuite.

3.2 Analyse fonctionnelle.

3.2.1. Découpe fonctionnelle.

3.2.1.1. Niveaux d'analyse.

Etant donné la nature du système, deux niveaux doivent être analysés :

a) L'application, devant satisfaire les spécifications données en 3.1.2.

b) Le système d'exploitation devant satisfaire toutes les contraintes du temps-réel exposées à la section 2.2.

Dans cette section 3.2., on mettra en évidence les grandes fonctions à implémenter. Elles seront décrites avec plus de détails à la section 3.3.

3.2.1.2. L'application.

a) Vue générale.

Le communateur se présente comme un système de deux couches :

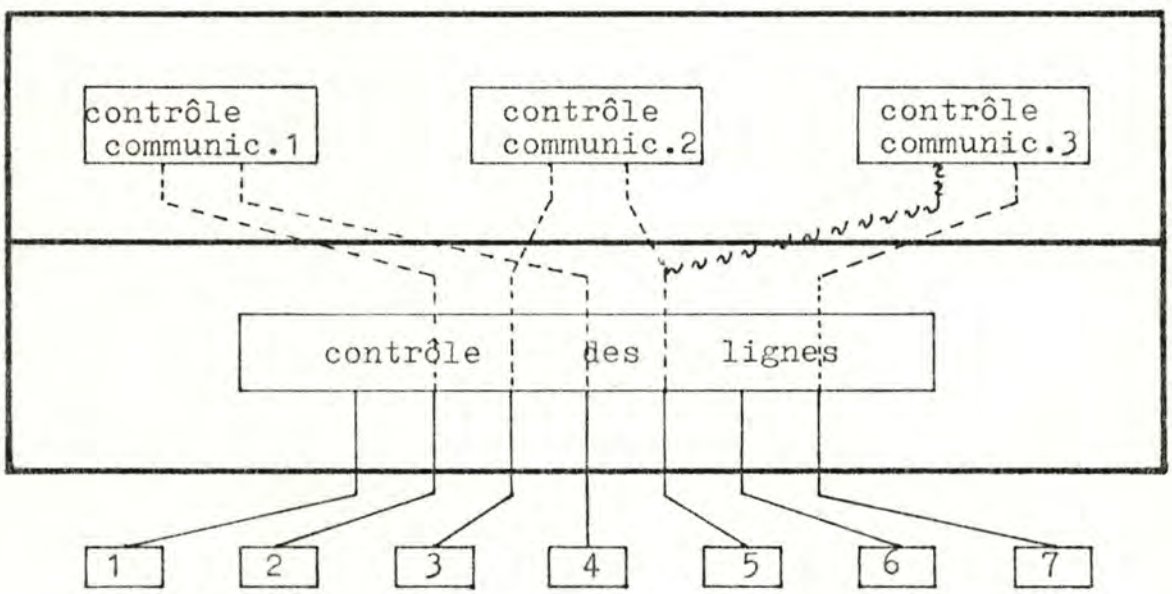


Fig E62. Schéma général du commutateur.
L'abonné 2 téléphone à l'abonné 4.
L'abonné 3 téléphone à l'abonné 5.
L'abonné 7 tente de téléphoner à l'abonné 5,mais celui-ci est occupé.

Dans la première couche, on trouve un contrôleur de lignes qui surveille en permanence les lignes d'abonnés afin de détecter les différents événements tels que décrochage, numérotation, etc... Cette couche se charge également de transmettre les tonalités de sonnerie sur les lignes.

La deuxième couche ou couche de contrôle des communications a pour objet de superviser les communications entre abonnés. Un module particulier de contrôle est créé pour chaque communication différente.

Chaque fois que la première couche détecte un événement sur une ligne, elle le signale à la deuxième couche, laquelle prend des mesures en conséquence : mise à jour des informations maintenues sur la communication, et demande à la première couche d'effectuer certaines opérations sur les lignes.

b)Fonctions.

Il y a quatre fonctions dans l'application : deux dans chaque couche de contrôle.

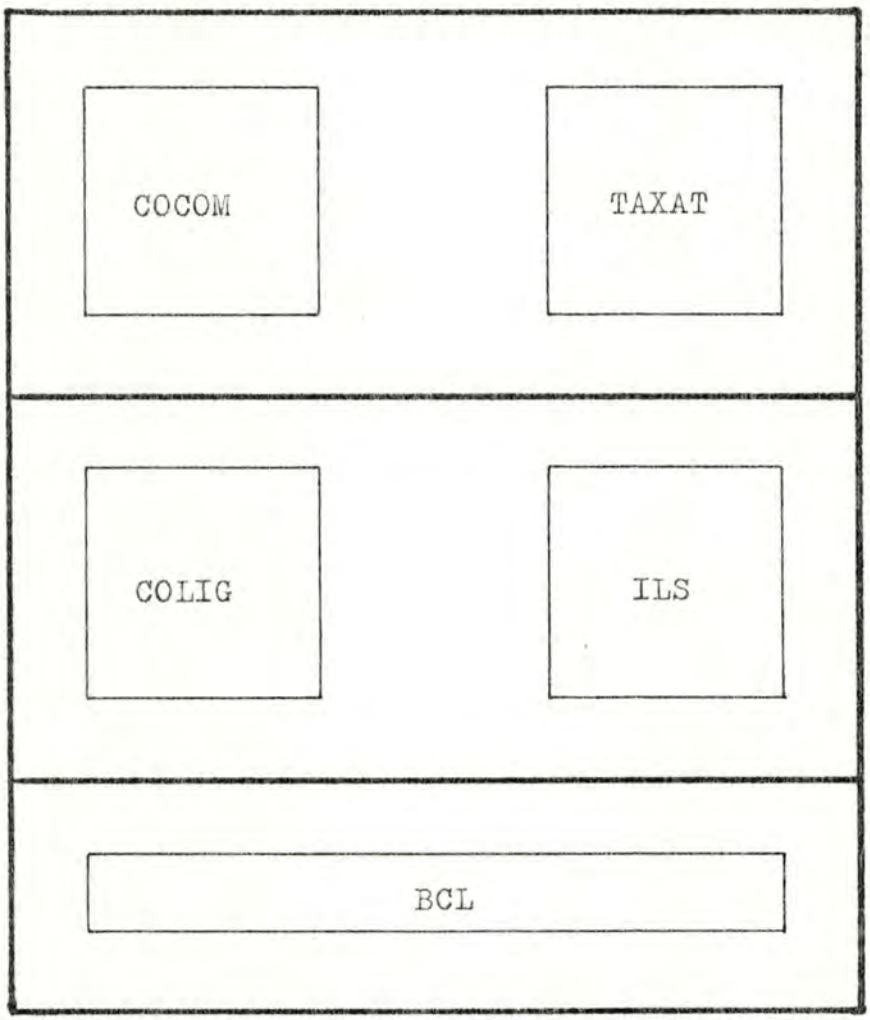


fig E63. Fonctions de l'application.

* Couche de lignes :

- ILS : Interface Ligne-système.

  Fonction de bas niveau proche du hardware. Elle comprend le scanning des lignes d'abonnés avec des fréquences variables, la gestion des différentes sonneries et tonalités, et la gestion de tables d'information sur les lignes.

  ILS est un ensemble de procédures activées par interruption software.

- COLIG : COntrôleur des LIGnes.

  Automate contrôlant les séquences d'actions et de réponses consécutives aux événements survenant sur les lignes et signalés par ILS.

* Couche des communications :

- COCOM : COntrôleur des COMmunications.

  Automate contrôlant le déroulement des communications téléphoniques.

- TAXAT : gestion de la TAXATion.

  Automate effectuant la taxation des communications.

3.2.1.3. Le système d'exploitation.

Quatre fonctions sont nécessaires dans l'OS :

- GM : Gestion de la mémoire.

- GX : Gestion des échanges.

- GT : Gestion des tâches.

- ST : Services de temporisation.

3.2.2. Base de données.

Trois fichiers sont tenus à jour :

- FAB : Fichier des abonnés, constitué d'enregistrements ABONNE, comprenant les renseignements suivants:

  . numéro de ligne
  . numéro d'abonné
  . indicatif des zones géographiques

. signalétique de l'abonné
. numéros de déroutement
. identificateurs de time-outs de réveil
. type de taxation
. montant à facturer

- FGEO : Fichier des zones géographiques, composé d'enregistrements ZONE, comprenant les renseignements suivants:

. indicatif de zone
. taille de zone
. indicatifs des zones contiguës

- FHIST : Fichier historique, composé d'enregistrements HISTO, comportant les renseignements suivants:

. numéro d'abonné appelant et appelé
. heure/jour de début/fin de communication
. indicatif de taxation pour appelé ou appelant ou gratuite
. montant facturé

3.2.3. Dynamique du système.

On étudiera successivement la séquence ordinaire, les déroutements et les relâchements forcés et normaux.

La structure des numéros de messages est la suivante: wxyz est le yz-ième message envoyé par le module w au module x. Les modules sont numérotés comme suit:

ILS : 1
COLIG : 2
COCOM : 3
TAXAT : 4

3.2.3.1. Séquence ordinaire.

Phase 1 : Décrochage appelant.

1) ILS détecte un événement haut sur une ligne libre.

2) ILS -> 1201 -> COLIG : signale l'événement.

3) COLIG crée une tâche COCOM pour superviser la nouvelle communication.

4) COLIG -> 2301 -> COCOM : envoi d'informations sur la ligne et l'abonné appelant.

5) COLIG active une procédure ILS qui retire la ligne de l'état libre et la met dans l'état prénumérotation.

6) COLIG active une procédure ILS de scanning prénumérotation.

7) COCOM consulte la base de données pour extraire des informations sur la ligne et l'abonné.

8) COCOM → 3201 → COLIG : demande de se préparer à recevoir les chiffres.

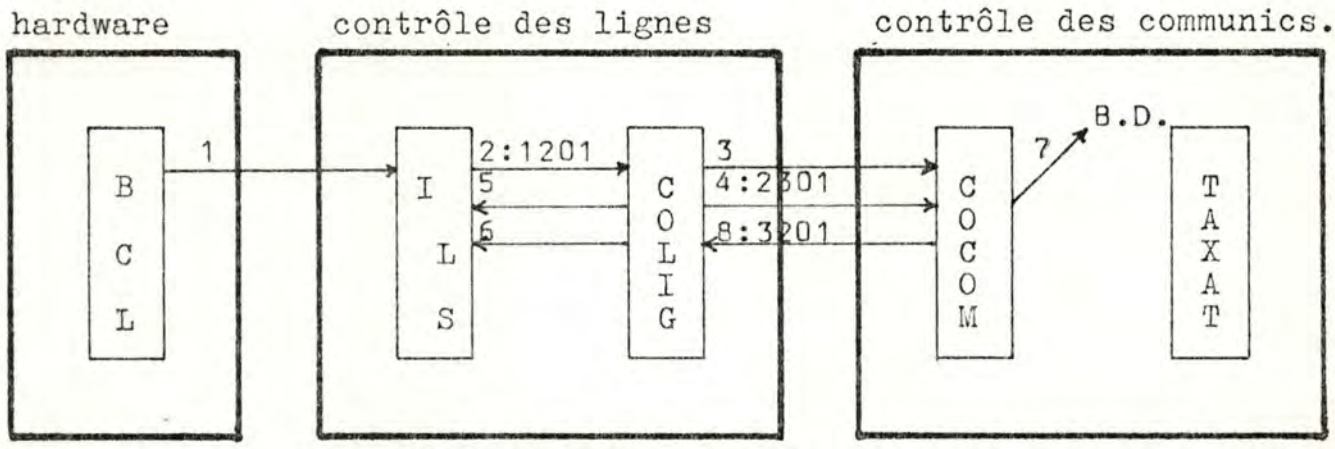


fig E64. Décrochage côté appelant.

Phase 2 : Invitation à numéroter.

1) COLIG active la procédure ILS de changement d'état qui retire la ligne de l'état prénumérotation et la met dans l'état numérotation.

2) COLIG active une procédure ILS de scanning numérotation.

3) COLIG désactive la procédure de scanning prénumérotation s'il n'y a plus de ligne en état prénumérotation.

4) COLIG active la procédure ILS d'envoi de la tonalité d'invitation à numéroter.

5) COLIG → 2302 → COCOM : informe de l'envoi de la tonalité d'invitation à numéroter.

6) COCOM demande au module ST de l'OS l'activation d'un time-out de 24 secondes.

7) ST → 0301 → COCOM : renvoi de l'ID d'un time-out.

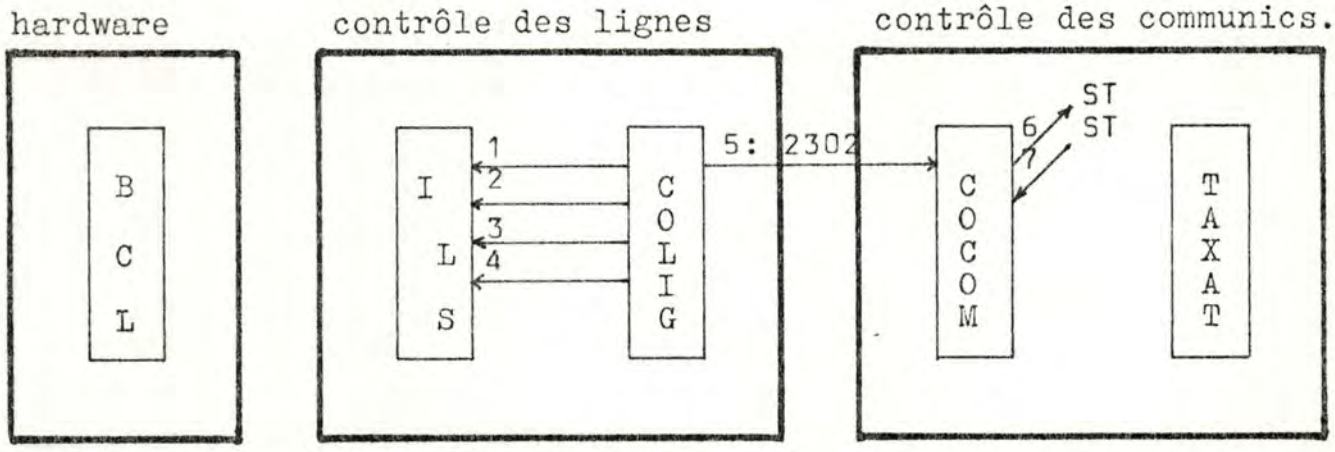


fig E65. Invitation à numéroter.

Phase 3 : Détection des trois premiers chiffres.

1) ILS détecte un changement de niveau sur la ligne appelante.

2) ILS → 1201/1202 → COLIG : signale l'événement.

2') COLIG active la procédure ILS qui interrompt la tonalité (uniquement pour le premier passage).

3) COLIG traite le changement de niveau en vue d'établir le chiffre.

4) COLIG → 2303 → COCOM : envoi du chiffre détecté.

5) COCOM met à jour les time-outs (préchiffre, inter-chiffre).

---→ répéter 3 fois les points 1 à 5.

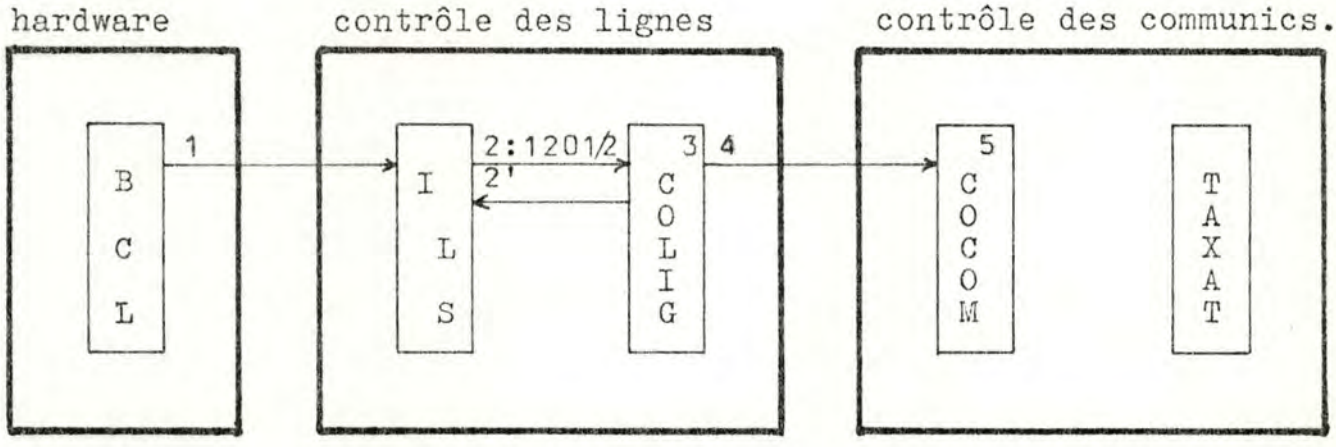


fig E66. Détection des trois premiers chiffres.

Phase 4 : Analyse du préfixe.

1) COCOM analyse le préfixe.

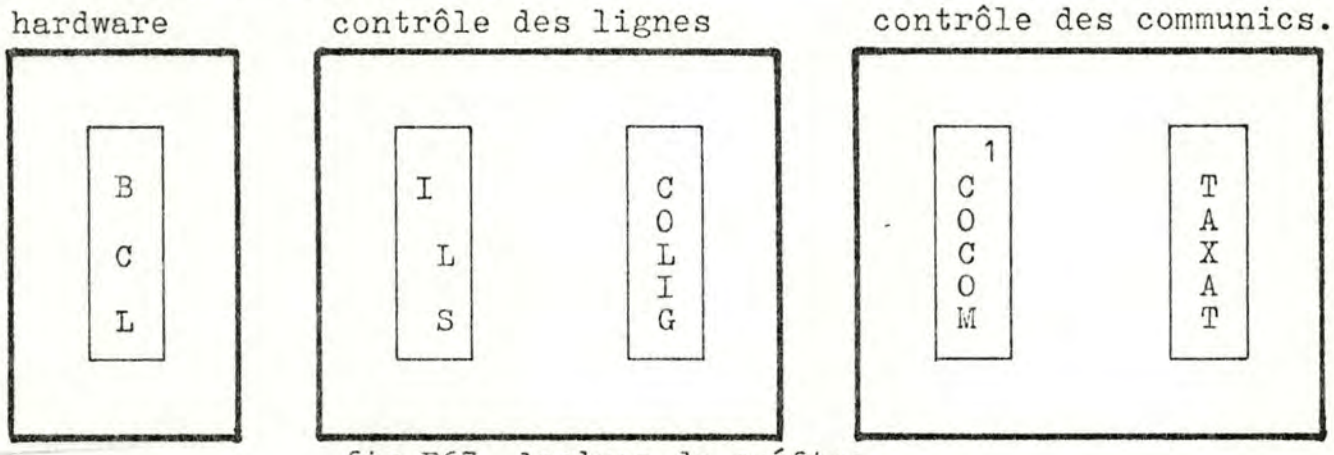


fig E67. Analyse du préfixe.

Phase 5 : Fin de numérotation.

1- 5) reprendre les sous-phases 1 à 5 de la phase 3. Répéter n fois ces sous-phases, n dépendant du résultat de l'analyse du préfixe.

6) COCOM extrait de la base de données le numéro de ligne de l'appelé et sa catégorie.

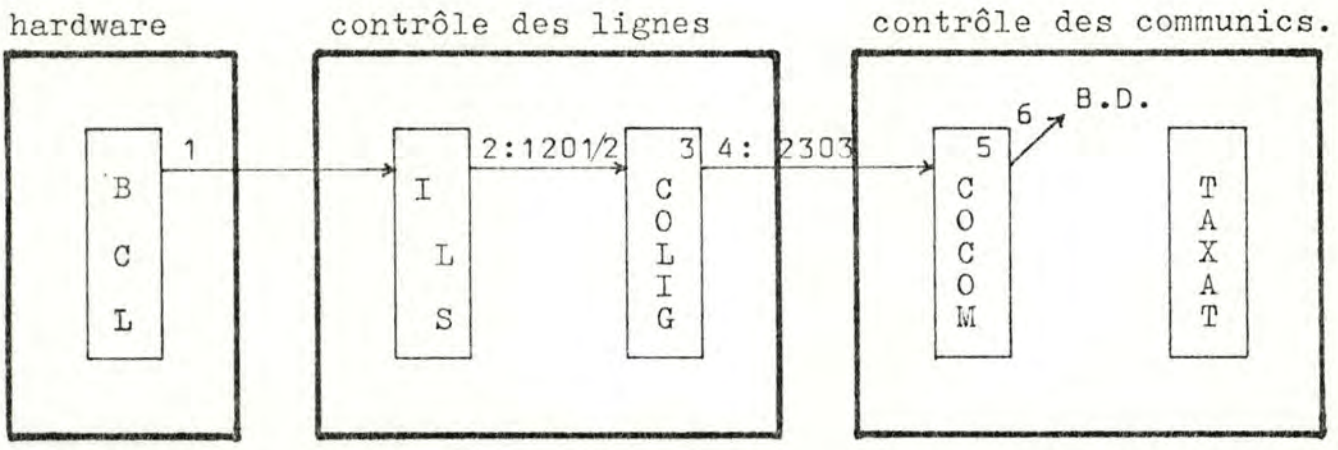


fig E68. Fin de numérotation.

Phase 6 : Fin de la réception de numérotation.

1) COCOM → 3202 → COLIG : demande de fin de réception.

2) COLIG active la procédure ILS de changement de la ligne appelante de l'état numérotation à l'état postnumérotation.

3) COLIG active la procédure ILS de scanning postnumérotation.

4) COLIG désactive la procédure ILS de scanning numérotation si aucune ligne n'est dans l'état numérotation.

5) COLIG → 2304 → COCOM : indique que la réception est libérée.

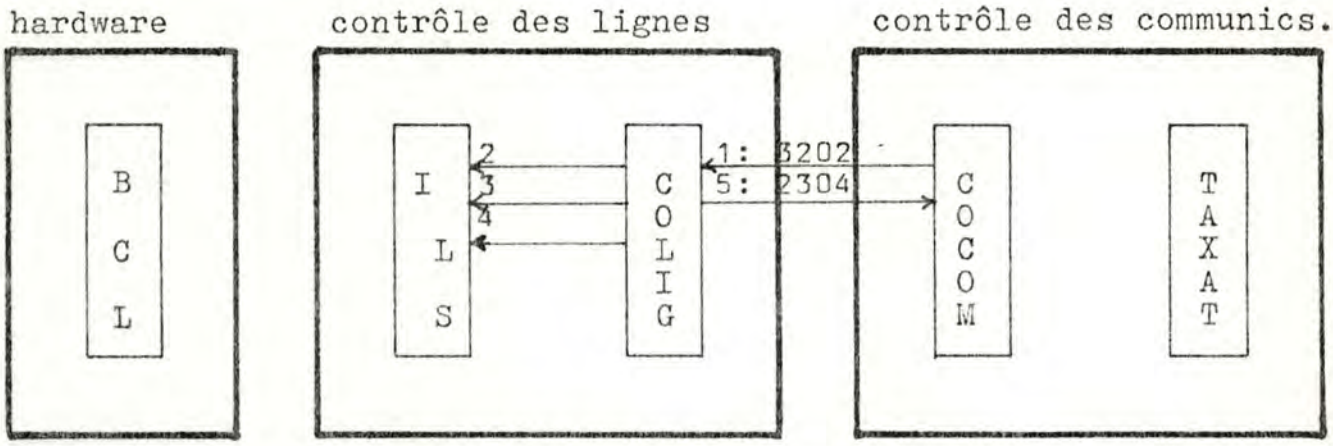


fig E69. Fin de la réception de numérotation.

Phase 7 : Envoi du courant de sonnerie.

1) COCOM → 3203 → COLIG : demande de saisie de l'appelé.

2) COLIG active la procédure ILS de mise de la ligne appelée dans l'état préconversation, si celle-ci est dans l'état libre.

3) COLIG active la procédure de scanning préconversation.

4) COLIG active la procédure ILS qui envoie le courant de sonnerie sur la ligne appelée.

5) COLIG active la procédure ILS qui envoie l'écho de sonnerie sur la ligne appelante.

6) COLIG → 2305 → COCOM : informe que la saisie de l'appelé est terminée.

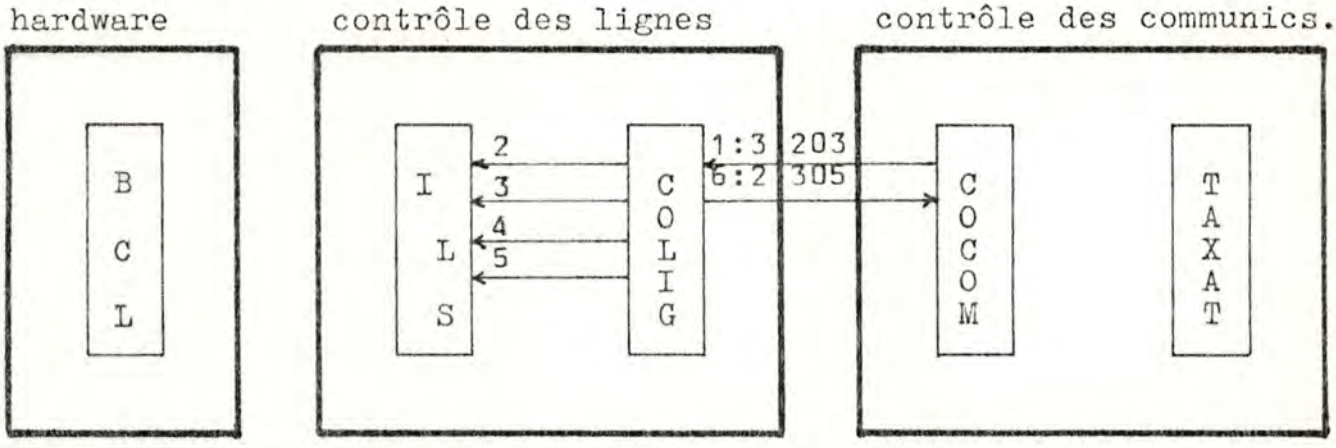


fig E70. Envoi du courant de sonnerie.

Phase 8 : Passage à l'état "stable".

1) COCOM → 3204 → COLIG : envoi de données pour les côtés appelant et appelé.

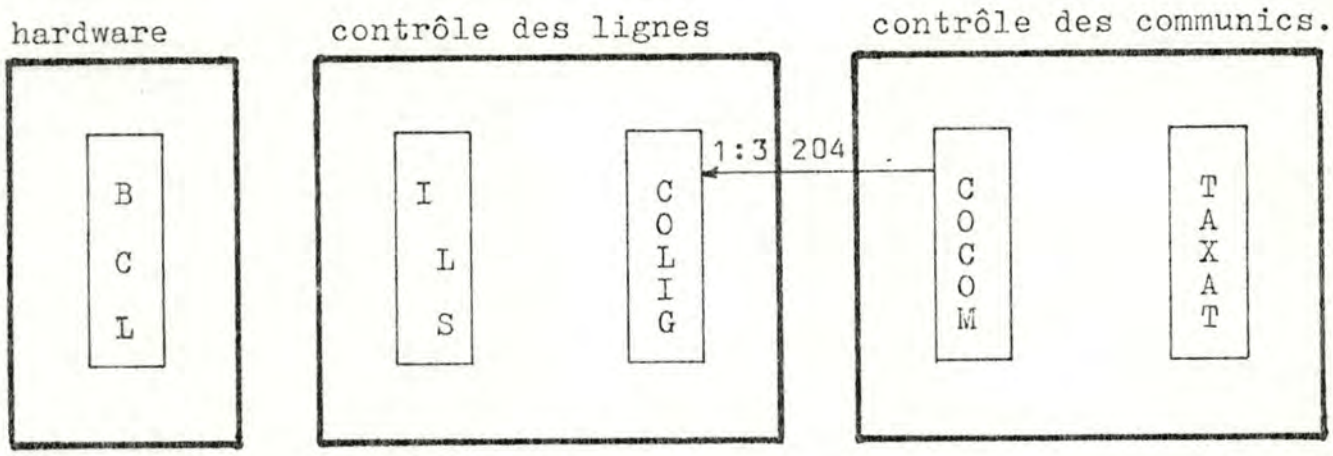


fig E71. Passage à l'état stable.

Phase 9 : Réponse de l'abonné appelé.

1) ILS détecte un événement haut sur la ligne appelée.

2) ILS → 1201 → COLIG : signale le changement d'état de la ligne.

3) COLIG active la procédure ILS qui déconnecte la sonnerie sur la ligne appelée.

4) COLIG active la procédure ILS de changement de la ligne appelée de l'état préconversation à l'état conversation.

5) COLIG → 2306 → COCOM : avertit du décrochage sur la ligne appelée.

6) COCOM → 3205 → COLIG : demande de saisie de l'appelant.

7) COLIG active la procédure ILS de changement de la ligne appelante de l'état postnumérotation à l'état conversation.

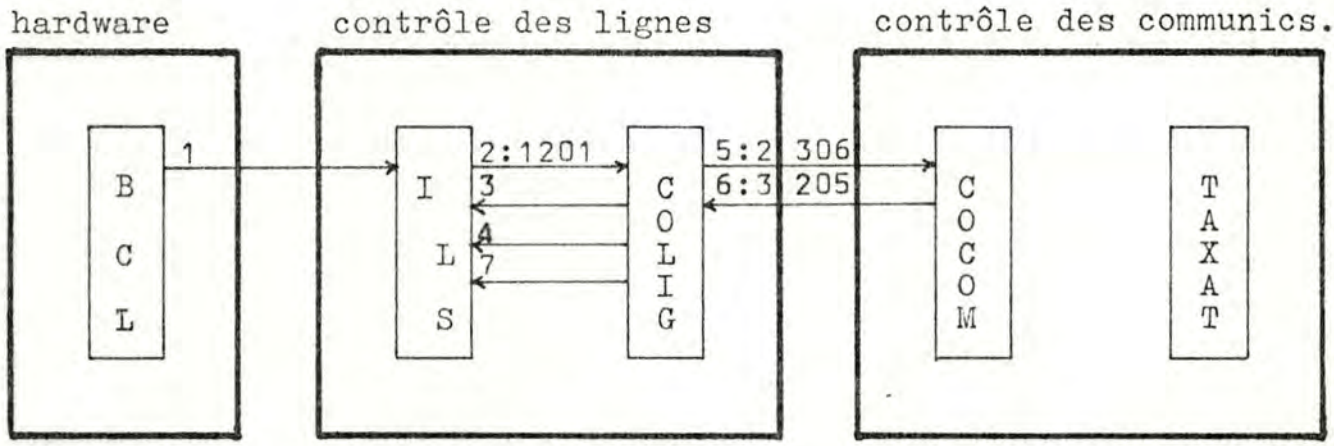


fig E72. Réponse de l'abonné appelé.

Phase 10 : Conversation.

Phase 11 : Libération vers l'avant.

1) ILS détecte un événement bas sur la ligne appelante.

2) ILS -> 1202 -> COLIG : signale le changement d'état de la ligne.

3) COLIG -> 2307 -> COCOM : informe du raccrochage.

4) COCOM -> 3401 -> TAXAT : demande la taxation.

5) COCOM -> 3206 -> COLIG : demande de libérer les deux côtés.

6) Destruction de la tâche COCOM.

7) COLIG active une procédure ILS de changement de la ligne appelante de l'état conversation à l'état libre.

8) COLIG active une procédure ILS de changement de la ligne appelée de l'état conversation à l'état parking.

9) ILS détecte un événement bas sur la ligne appelée.

10) ILS -> 1202 -> COLIG : signale l'événement.

11) COLIG active une procédure ILS de changement de la ligne appelée de l'état parking à l'état libre.

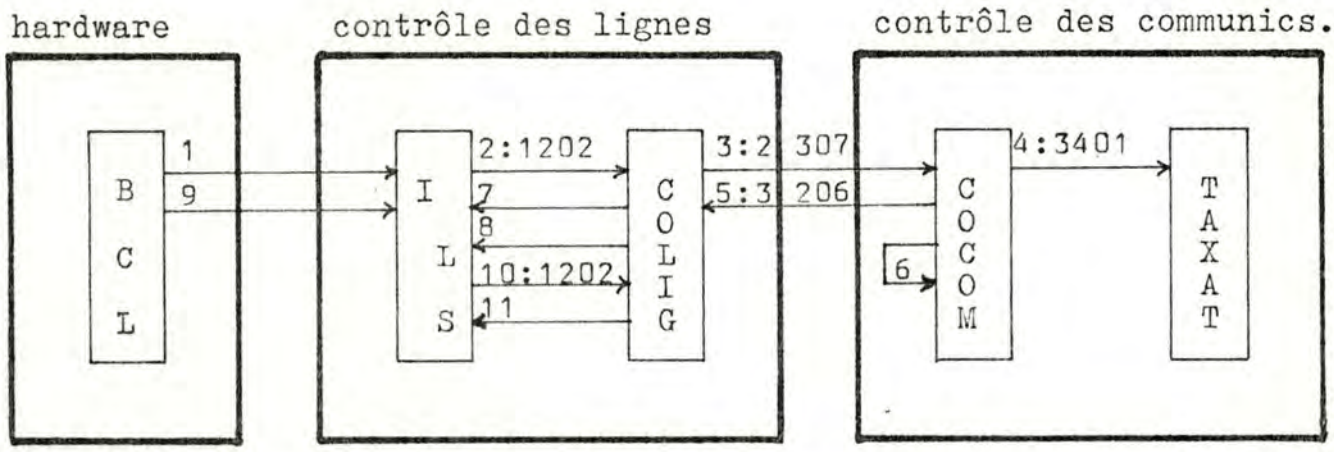


fig E73. Libération vers l'avant.

Phase 12 : Libération vers l'arrière.

1) ILS détecte un événement bas sur la ligne appelée.

2) ILS → 1202 → COLIG : signale l'événement.

3) COLIG → 2307 → COCOM : informe du raccrochage.

4) COCOM active un time-out de 90 secondes pour permettre à l'appelé de redécrocher.

5) ST → 0301 → COCOM : envoi de l'identifiant du time-out de 90".

6) ST → 0302 → COCOM : signale que le time-out est expiré.

7) COCOM → 3401 → TAXAT : demande de taxation.

8) COCOM → 3206 → COLIG : demande de libérer les deux côtés.

9) Destruction de la tâche COCOM.

10) COLIG active une procédure ILS pour retirer la ligne appelée de l'état conversation et la mettre dans l'état libre.

11) COLIG active une procédure ILS pour retirer la ligne appelante de l'état conversation et la mettre dans l'état parking.

12) ILS détecte un événement bas sur la ligne appelante.

13) ILS → 1202 → COLIG : signale l'événement.

14) COLIG active une procédure ILS pour retirer la ligne appelante de l'état parking et la mettre dans l'état libre.

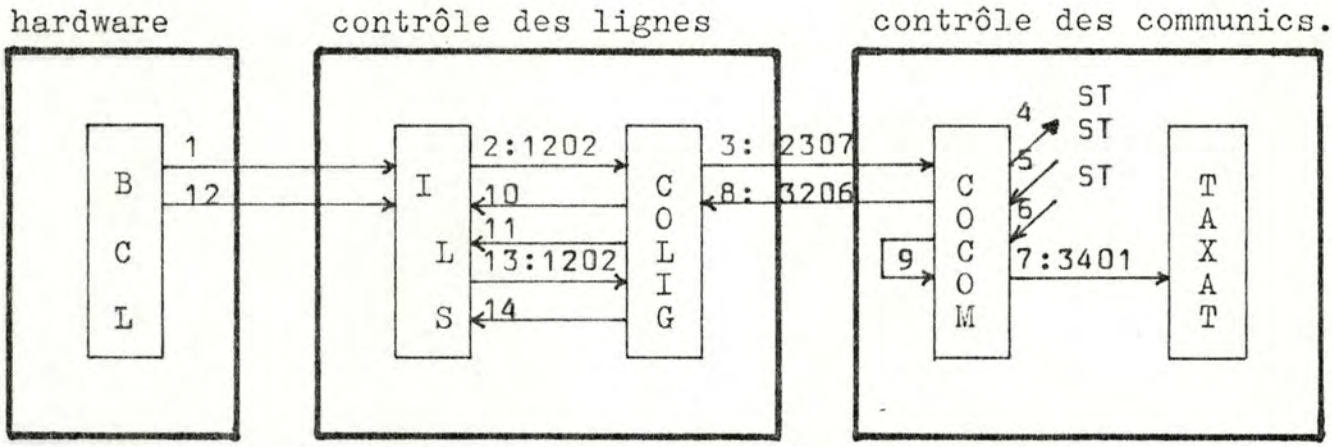


fig E74. Libération vers l'arrière.

3.2.3.2. Déroutements.

Déroutement 1 : l'appelé décroche avant l'expiration du time-out de 90 secondes durant la libération vers l'arrière.

--> phase 12.4 de la séquence ordinaire.

1) ILS détecte un événement haut sur la ligne appelée.

2) ILS -> 1201 -> COLIG : signale le changement d'état.

3) COLIG -> 2308 -> COCOM : informe du redécrochage.

4) COCOM demande à TS de désactiver le time-out de 90 secondes.

--> phase 10

Déroutement 2 : ligne appelée PBX simple occupée.

--> phase 7.2.

1) COLIG -> 2309 -> COCOM : signale que la ligne est occupée.

2) COCOM accède à la base de données et recherche une autre ligne du PBX.(S'il n'y a plus de ligne, voir le relâchement forcé numéro 7). COCOM retire de la base de données le numéro de ligne et la catégorie de l'abonné.

3) COCOM -> 3203 -> COLIG : demande de saisie de l'apelé.

--> phase 7.2.

Déroutement 3 : ligne appelée PBX spéciale.

--> phase 4.1.

Suite à l'analyse du préfixe, COCOM a encore besoin d'un chiffre de COLIG. Il procède à l'analyse de ce quatrième chiffre.

--> phase 4.1.

Déroutement 4 : formation d'un numéro prédéterminé (12 suivi de 6 ou 9 chiffres suivis de 21).

--> phase 4.1.

1) COCOM analyse le préfixe et remarque la séquence 12. Il sait alors qu'il aura encore 11 chiffres à recevoir au maximum avec les deux derniers qui doivent être 21.

2) Enregistrement dans la base de données du numéro prédéterminé et du numéro de l'appelant.

--> relâchement de l'appelant.

Déroutement 5 : ligne appelée prédéterminée.

--> phase 1.6.

COCOM extrait de la base de données les informations concernant la ligne et l'abonné. On y trouvera le numéro prédéterminé de la ligne appelée.

--> phase 7

Déroutement 6 : Formation d'une heure de réveil (13/hh/mm).

--> phase 4.1.

1) COCOM reçoit le préfixe et s'aperçoit de la séquence 13. Il sait alors qu'il doit recevoir au maximum 6 chiffres dont les deux derniers doivent être 31.

2) Enregistrement dans la base de données de l'ID du time-out absolu et du numéro de l'appelant.

3) COCOM demande à ST d'activer le time-out.

--> relâchement de l'appelant.

Déroutement 7 : Réveil à une heure déterminée.

1) ST → 0301 → COLIG : arrivée à échéance du time-out de réveil.

2) COLIG crée une tâche COCOM qui va établir une communication entre une ligne virtuelle et la ligne de l'appelant qui désire le réveil.

3) COLIG → 2310 → COCOM : transmet l'ID du time-out.

4) COCOM accède à la base de données de laquelle il extrait le numéro de l'appelant qui correspond à l'ID de ce time-out.

--> phase 7.

Déroutement 8 : ligne en observation d'appels malveillants.

--> phase 5.6.

Enregistrement du numéro appelant et de l'heure à laquelle s'est effectué l'appel.

Déroutement 9 : déroutement du numéro appelé vers un numéro prédéterminé (formation : 14/suite de 6 ou 9 chiffres/41).

--> phase 4.1.

1) COCOM analyse le préfixe, s'aperçoit de la séquence 14. Il sait qu'il ne devra plus recevoir que 11 chiffres au maximum dont les deux derniers doivent être 41.

2) Enregistrement dans la base de données du numéro de déroutement et de celui de l'appelant.

--> relâchement de l'appelant.

Déroutement 10 : déroutement du numéro appelé vers un numéro prédéterminé.

--> phase 5.6.

COCOM extrait de la base de données le numéro prédéterminé.

--> phase 6.

3.2.3.3. Relâchements forcés.

Relâchement 1 : expiration du time-out de 24 secondes avant l'envoi du premier chiffre.

--> phase 2.6.

1) COCOM -> 3208 -> COLIG : demande de libération de l'appelant.

2) COLIG détruit la tâche COCOM.

3) COLIG active une procédure ILS qui supprime la tonalité d'invitation à numéroter sur la ligne appelante.

4) COLIG active une procédure ILS qui retire la ligne appelant de l'état numérotation et la met dans l'état libre.

Relâchement 2 : expiration du time-out de 24 secondes inter-chiffre dans le préfixe.

--> phase 3.5.

1) COCOM -> 3208 -> COLIG : demande de libération de l'appelant.

2,3,4) voir 2,3,4 du relâchement 1.

Relâchement 3 : expiration du time-out de 24 secondes interchiffre en numérotation.

--> phase 5.5.

Même relâchement que le 2.

3.2.3.4. Relâchements normaux.

Relâchement 1 : Raccrochage entre décrochage et tonalité d'invitation à transmettre.

--→ phase 3.3.

La détection n'est pas immédiate. En 3.3, un time-out est positionné. Après 200 msec, COLIG constate que le niveau bas n'a pas changé. Il détecte par conséquent un raccrochage de l'appelant.

1) ST → 0303 → COLIG : expiration du time-out.

2) COLIG active une procédure ILS de changement de la ligne appelante de l'état numérotation à l'état libre.

3) COLIG détruit la tâche COCOM.

Relâchement 2 : Raccrochage pendant la tonalité d'invitation à transmettre.

cf. R.N.1.

Relâchement 3 : Raccrochage pendant un chiffre.

cf. R.N.1.

Relâchement 4 : Raccrochage à la fin d'un chiffre avant 200 msec.

cf. R.N.1.

Relâchement 5 : Raccrochage à la fin d'un chiffre entre 200 msec et 24 sec.

cf. R.N.1.

Relâchement 6 : Raccrochage entre le dernier chiffre et l'écho de sonnerie.

--> n'importe quelle sous-phase entre 6.4 et 9.6.

1) ILS détecte un événement bas sur la ligne appelante.

2) ILS → 1202 → COLIG : signale l'événement.

3) COLIG → 2310 → COCOM : signale l'événement.

4) COCOM → 3220 → COLIG : demande à COLIG de libérer l'appelant et l'appelé.

5) COLIG détruit la tâche COCOM.

6) COLIG demande à ILS de mettre la ligne appelante en état libre.

7) COLIG demande à ILS de mettre la ligne appelée en état libre.

3.3. Analyse organique.

3.3.1. Architecture.

a) Application.

Il y a quatre modules implémentant le quatre fonctions décrites dans l'analyse fonctionnelle. Trois de ces modules (COLIG, COCOM et TAXAT) sont des automates et le dernier (ILS) est un ensemble de procédures appelées par interruption software.

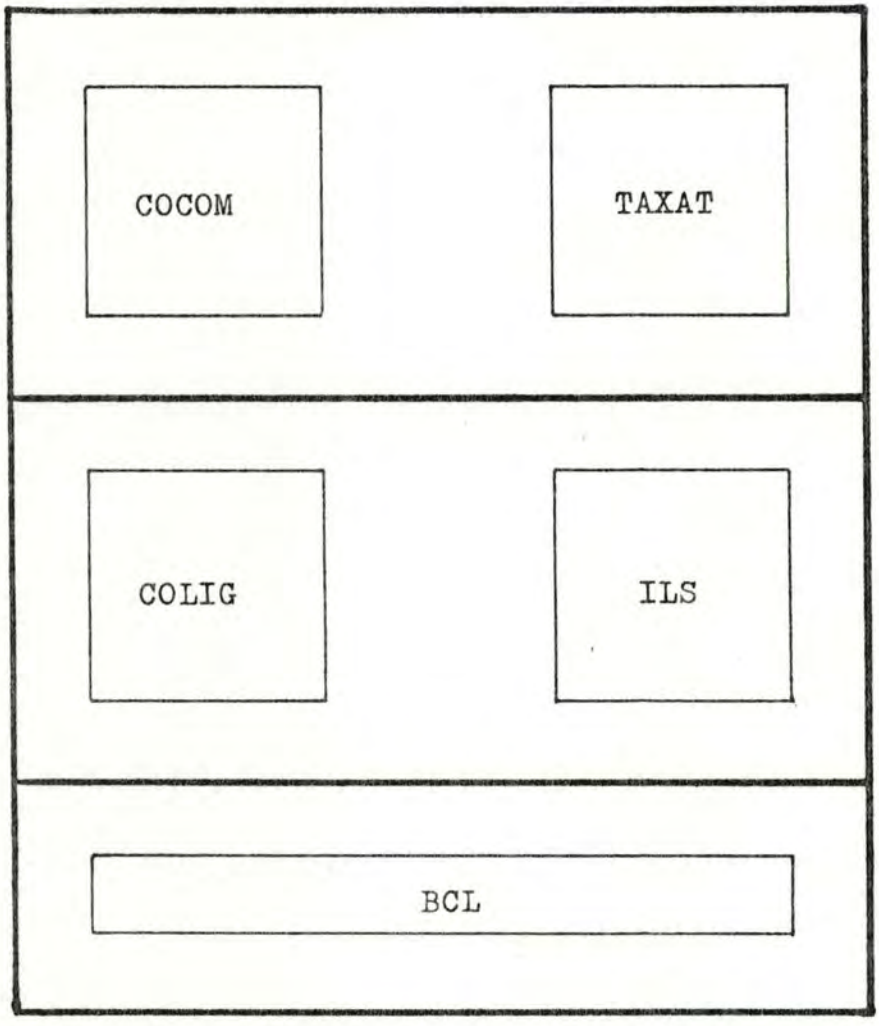


fig E75. Architecture de l'application.

La figure indique les seules relations possibles entre modules.

b) Système d'exploitation.

Il y a quatre modules implémentant les quatre fonctions décrites dans l'analyse fonctionnelle.

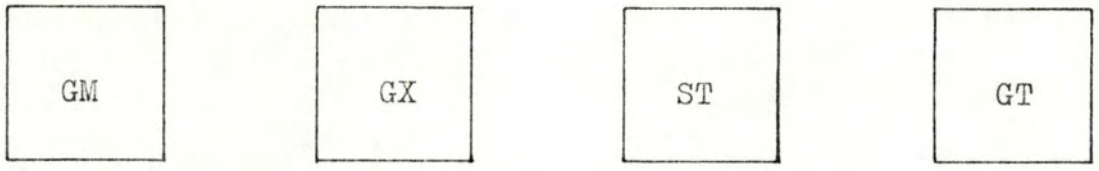


fig E76. Architecture du système d'exploitation.

Les relations de module OS à module OS, et de module d'application à module OS sont implémentées sous forme d'interruptions software.

3.3.2.1 Gestion des processus.

A. Introduction.

D'un point de vue statique, le système peut être vu comme un ensemble de modules fonctionnels.
D'un point de vue dynamique, le système apparaît comme un ensemble de morceaux de programme (correspondant plus ou moins aux modules) en fonctionnement parallèle ou pseudo-parallèle. Les processus sont créés et détruits suivant les besoins du moment et se déroulent indépendamment les uns des autres. Toutefois, ils peuvent communiquer entre eux.

B. Notion de processus.

Un processus est une unité de code (par exemple un module) s'exécutant sur une période bien déterminée, qui l'identifie parmi les autres processus.
Le processus est créé pour s'appliquer à un objet du système lorsque celui-ci apparaît; il est détruit lorsque l'objet disparaît.

Remarques :

1) Si plusieurs processus sont basés sur le même code et contrôlent des objets de même type apparaissant et disparaissant en même temps, ils peuvent être regroupés en un seul processus contrôlant plusieurs objets.
2) Si plusieurs objets de même type ont des situations dans le temps et des durées de vie différentes, il est préférable de leur allouer des processus différents.

C. Découpe du système en processus.

Dans le cas présent, l'unité de code choisie comme base pour les processus est le module. La découpe suivante en processus est donc également une découpe modulaire du système.(Tabl. suivant).

Processus / Module	Rôle et objet contrôlé	Période de vie
Moniteur		
GMT	Gérer le multiprocessus Objet : processus	De la mise en marche jusqu'à l'arrêt du système
GI	Gérer les interrupts Objet : table des interrupts	idem
Scheduler	Gérer la programmation des processus Objet : liste des processus	idem
Application		
ILS	Scanner les lignes	idem
COLIG	Contrôler les lignes	idem
COCOM	Contrôler UNE SEULE communication	Durée de vie de la communication
TRATAX	Gérer la taxation	De la mise en marche jusqu'à l'arrêt du système
O.S.		
GT	Complément à le gestion des processus	idem
GM	Gestion de la mémoire	idem
ST	Services de temporisation	idem
GX	Gestion de la file de messages entre processus	idem

D. Fonctionnement du système.

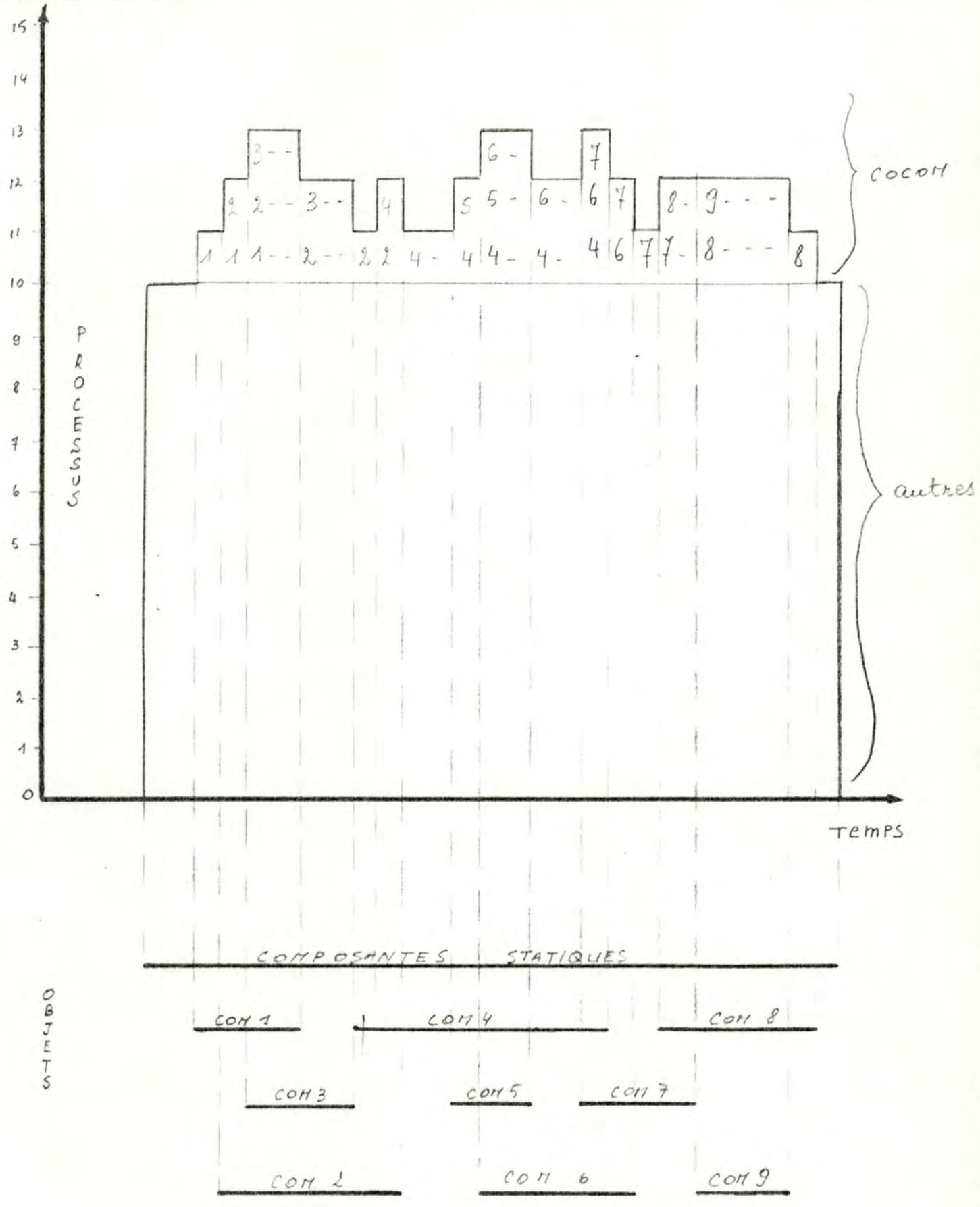
D'après le tableau précédent, il apparaît que les objets du système sont :

- d'une part les composantes "statiques" dont la durée de vie équivaut à celle du système global. Les processus y affectés sont donc créés à la mise en marche du système et détruits lors de l'arrêt du système.
- d'autre part les composantes "dynamiques", c'est-à-dire les communications dont la durée de vie et la position dans le temps sont aléatoires.

Remarques :

1) En vertu de B.Rem1, un seul processus COLIG et un seul processus ILS seront affectés au contrôle de toutes les lignes.
2) En vertu de B.Rem2, un processus COCOM particulier sera affecté à chaque communication et ne vivra que le temps de la communication.

La figure suivante décrit le fonctionnement multiprocessus du système.


hbre de Processus
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PROCESSUS
COCOM
autres
Temps
COMPOSANTES STATIQUES
OBJETS
COM 1
COM 4
COM 8
COM 3
COM 5
COM 7
COM 2
COM 6
COM 9

E. Moniteur temps réel.

E.1 Gestion des interruptions.

Nous nous basons sur le microprocesseur Intel 8086. Celui-ci peut gérer jusqu'à 256 interruptions classées par priorité. toutes les sortes d'interruptions s'y retrouvent : hardware/software, asynchrone/ synchrone, avec/sans erreur.
Lorsqu'on prévoit d'utiliser une interruption n°N, on écrit une routine destinée à la gérer. L'adresse de cette routine se retrouve dans la TABLE D'INTERRUPTION dans l'élément d'indice N. Cet élément est composé de 4 bytes capables d'adresser toute la mémoire.
Lors d'une interruption, le GI sauve le contexte du processus interrompu et se branche sur la routine d'interruption via la table.(Voir figure V.7)

E.2 Gestion des multiprocessus.

Le moniteur gère plusieurs processus en parallèle
A chaque passage d'un processus à un autre, le GMT intervient pour lancer le nouveau processus et cloturer le précédent. Les événements provoquant ce changement de processus sont : fin d'un processus, arrêt volontaire d'un processus (pour attente de message par exemple), interruption d'un processus par un autre processus prioritaire,...
Dans tous les cas, le GMT fait appel au Scheduler (voir point F) pour sélectionner le prochain processus à lancer. Une exception cependant : lors d'un appel de primitive (voir figure V.7), la primitive est immédiatement lancée.

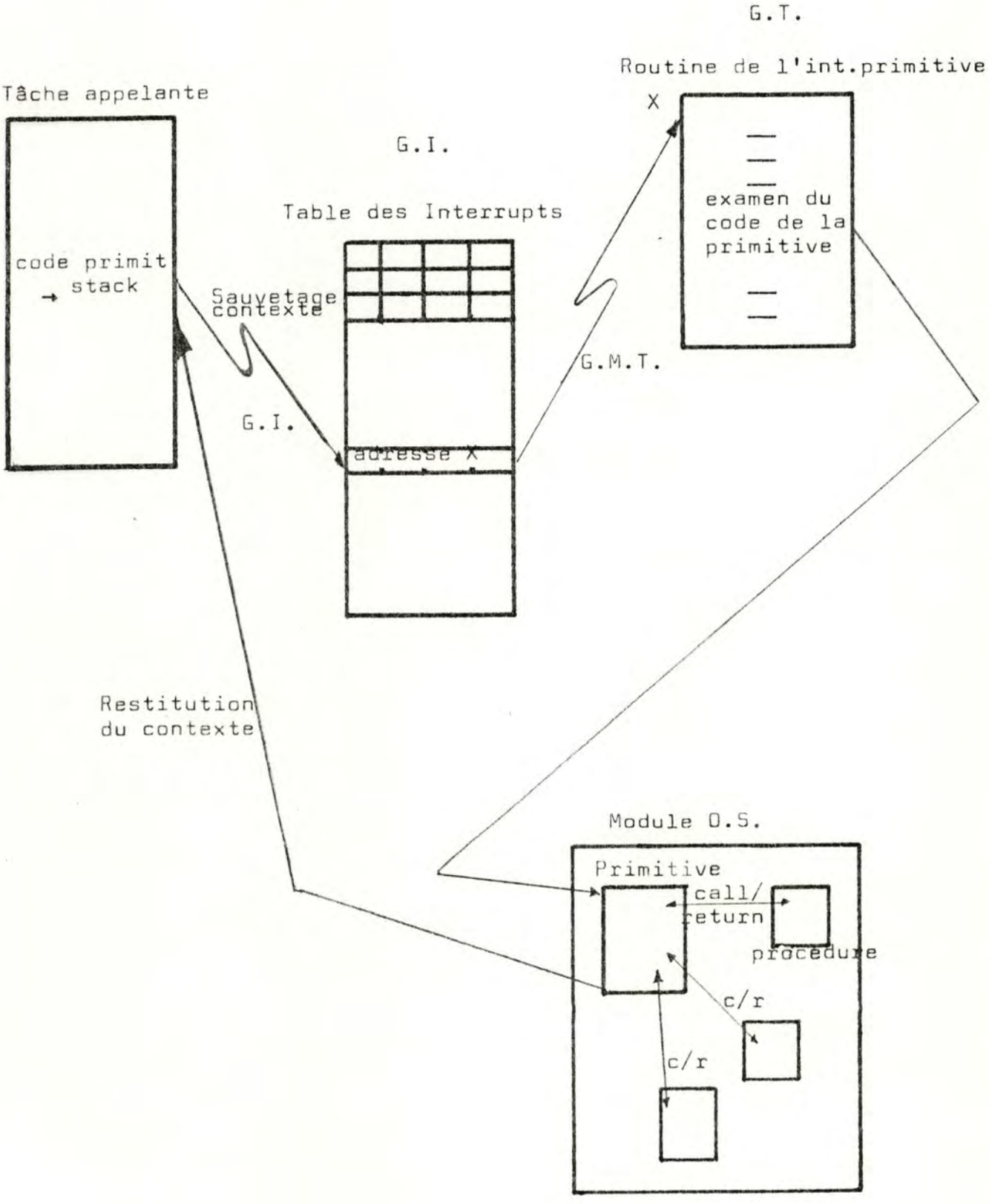


FIGURE V.7 : Mécanisme d'appel de primitives.

F. Scheduler.

F.1 Rôle du scheduler.

Le scheduler est le module chargé de sélectionner le processus à lancer lors du passage d'un processus à un autre. Il est appelé par le GMT du moniteur dans tous les cas de changement de processus, sauf dans le cas d'un appel de primitive.

F.2 Critères de sélection d'un processus.

F.2.1 Etat des processus.

Tout processus se trouve à tout moment dans un des trois états suivants :

1) interrompu : le processus s'est interrompu volontairement ou a été interrompu par un processus prioritaire.
2) en attente : le processus attend un message (cas des processus automates)
3) prêt : le processus est prêt à être lancé.

F.2.2 Priorité des états.

Lorsque le scheduler doit sélectionner le processus à lancer, il parcourt d'abord les interrompues; s'il n'y en a pas, il parcourt les processus en attente; s'il n'y en a pas, il parcourt les processus prêts.

F.2.3 Sélection d'un processus interrompu.

1) Choix des processus les plus prioritaires (4 priorités)
2) Parmi ces processus, choisir celui qui a été interrompu le premier chronologiquement.

F.2.4 Sélection d'un processus en attente.

Les messages envoyés portent une priorité (4 en tout). Le processus récepteur prendra cette priorité après réception du message.

1) Choix des processus devant recevoir les messages les plus prioritaires.
2) Parmi ces processus, choisir celui qui doit recevoir le message envoyé le premier chronologiquement.

F.2.5 Sélection d'un processus prêt.

Sous réserve de modifications ultérieures, le scheduler n'intervient pas dans ce cas-ci.
En effet, si l'on reprend les trois types de modules présentés au point C, on voit que :

1) le moniteur est lancé automatiquement;
2) les automates sont toujours en attente de messages, ou en exécution ou interrompus, mais ils ne sont jamais prêts;
3) Les modules de support sont constitués de primitives, or le scheduler n'intervient pas dans les appels de primitives.

3.3.2.2 Gérant de la mémoire

A) Introduction

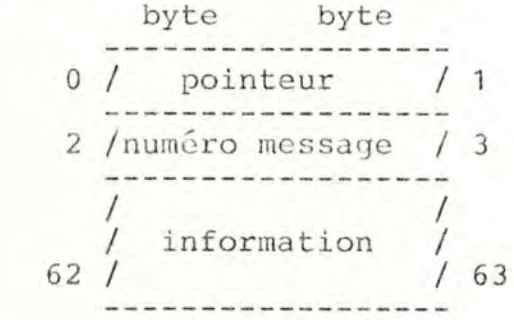
Le Gérant de la Mémoire (GM) est une tâche du système d'exploitation qui contrôle l'affectation et la libération de mémoire pour le moniteur, les différentes tâches du système et les tâches d'application.

Après avoir défini les différents types d'affectations de mémoire que l'on trouve dans le système, ainsi que l'organisation des blocs de mémoire libres, nous étudierons les primitives constituant le corps du Gérant de la Mémoire et qui permettent d'obtenir et de libérer les différents types de blocs mémoire préalablement définis.

B) Typologie des Ressources Mémoire

a.- le Bloc de Message : BLC-MSG

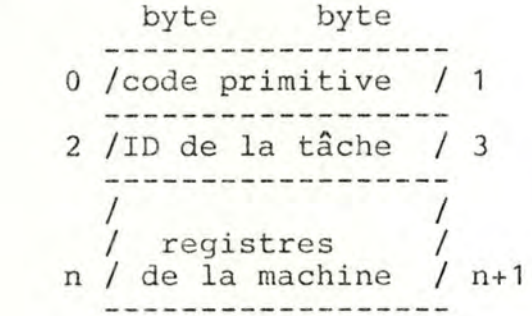
Il s'agit d'un bloc de mémoire dans lequel une tâche introduira un message à destination d'une tâche d'application automate.



b.- le Bloc de Contexte : BLC-CTX

Le bloc de contexte est un bloc de mémoire dans lequel une tâche qui va s'interrompre via un appel de primitive introduit des informations spécifiques à cette primitive.

Dans ce bloc, le Gérant des Interruptions introduira également au moment de l'appel de la primitive les informations qui seront nécessaires à la restauration de l'environnement de la tâche interrompue.



c.- le Bloc Controleur de Tâche : BLC-CTL-TCH

Ce bloc de mémoire contient l'identité de la tâche et les informations nécessaires à son exécution.

d.- le Bloc Controleur de Temporisation : BLC-CTL-TMP

Ce bloc contient les informations nécessaires au Gérant du Temps pour contrôler la temporisation.

```
   byte          byte
  --------------------
0 / ID temporisation / 1
  --------------------
2 /  pointeur        / 3
  --------------------
4 / ID de la tâche   / 5
  --------------------
6 / délai de temps   / 7
  --------------------
```

C) Organisation de la Mémoire libre

Le Gérant de la Mémoire contrôle une file de ressources libres, c'est-à-dire non affectées à une tâche, pour chaque type de ressource. Chaque bloc de mémoire d'un type particulier sera relié par un pointeur vers le bloc libre le suivant dans la file. A chaque demande d'affectation d'un bloc, le Gérant de la Mémoire accède au premier bloc de la file correspondant au type de mémoire demandée via un pointeur de tête. Il reste au Gérant de la Mémoire à fournir à la tâche ayant fait la demande d'affectation le pointeur vers le bloc de mémoire.

De manière systématique, toute libération par une tâche d'un bloc de mémoire verra celui-ci réinséré en tête de la file correspondant à son type (figure T37).

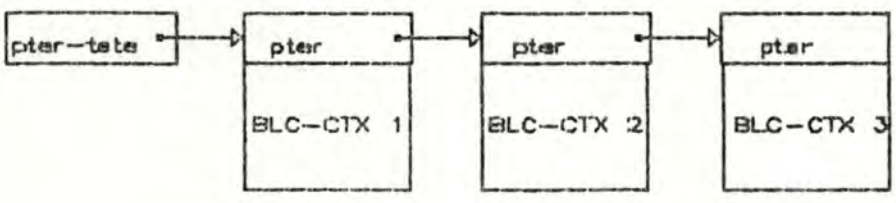


figure T37 : file des BLC-CTX libres

Les files des BLC-MSG, des BLC-CTL-TCH et des BLC-CTL-TMP sont organisées de la même manière.

D) Primitives de Traitement des BLC-MSG

OBT-BLC-MSG (pter-tâche)

fonction : lorsqu'une tâche demande un BLC-MSG, le Gérant de la Mémoire lui fournit un pointeur (se trouvant dans le BLC-CTL-TCH de la tâche) vers un BLC-MSG libre.

méthode : procédure OTE-BLC-QUE (file-blc-msg; pter-tâche)

RET-BLC-MSG (pter-tâche)

fonction : lorsqu'une tâche libère un BLC-MSG, cette tâche fournit au Gérant de la Mémoire un pointeur (se trouvant dans le BLC-CTL-TCH de la tâche) vers le BLC-MSG libéré.

méthode : procédure MET-BLC-DEB-QUE (file-blc-msg; pter-tâche)

RET-MSG-QUE-PRT (pter-début-queue-prête; pter-tâche)

fonction : lorsque le Scheduler a choisi une tâche en attente d'un message, le Gérant Multi-Tâches appelle la primitive RET-MSG-QUE-PRT. Le Gérant de la Mémoire retire le message indiqué par le pointeur fourni par le Gérant Multi-Tâches dans une des quatre queues prêtes et renvoie à la tâche programmée un pointeur (se trouvatn dans le BLC-CTL-TCH de la tâche) vers le message.

méthode : procédure OTE-BLC-QUE (queue-prête-n; pter-tâche)

MET-MSG-QUE-PRT (pter-fin-queue-prête; pter-tâche)

fonction : lorsqu'une tâche désire envoyer un message vers une tâche d'application automate, elle doit d'abord remplir un BLC-MSG demandé préalablement au Gérant de la Mémoire. Ce message ne sera pas transmis tout de suite au destinataire mais prendra place dans une des quatre files d'attente selon sa priorité via la primitive MET-MSG-QUE-PRT.

méthode : procédure MET-BLC-FIN-QUE (queue-prête-n; pter-tâche)

E) Primitives de Traitement des BLC-CTX

OBT-BLC-CTX (pter-tâche)

fonction : idem OBT-BLC-MSG

méthode : procédure OTE-BLC-QUE (file-blc-ctx; pter-tâche)

RET-BLC-CTX (pter-tâche)

fonction : idem RET-BLC-MSG

méthode : procédure MET-BLC-DEB-QUE (file-blc-ctx; pter-tâche)

F) Primitives de Traitement des BLC-CTL-TCH

OBT-BLC-CTL-TCH (pter)tâche)

fonction : lorsque le Gérant multi-Tâches crée une nouvelle tâche, le Gérant de la Mémoire lui fournit un pointeur vers un BLC-CTL-TCH libre.

méthode : procédure OTE-BLC-QUE (file-bloc-ctl-tch; pter-tâche)

RET-BLC-CTL-TCH (pter-tâche)

fonction : lorsque le Gérant multi-Tâches détruit une tâche, il fournit au Gérant de la Mémoire un pointeur vers le BLC-CTL-TCH libéré.

méthode : procédure MET-BLC-DEB-QUE (file-bloc-ctl-tch; pter-tâche)

G) Primitives de Traitement des BLC-CTL-TMP

OBT-BLC-CTL-TMP (pter-ST)

fonction : lorsque le Gérant du Temps demande un BLC-CTL-TMP, le Gérant de la Mémoire lui fournit un pointeur vers un bloc libre.

méthode : procédure OTE-BLC-QUE (file-bloc-ctl-tmp; pter-ST)

RET-BLC-CTL-TMP (pter-ST)

fonction : lorsque le Gérant du Temps détruit une temporisation, il renvoie au Gérant de la Mémoire un pointeur vers le BLC-CTL-TMP libéré.

méthode : procédure MET-BLC-DEB-QUE (file-blc-ctl-tmp; pter-ST)

H) Procédures de Traitement des files de blocs de mémoire libre

OTE-BLC-QUE (type-file; pter)

fonction : cette procédure accède au premier bloc via le pointeur de tête de la file type-file et le retire.

méthode (figure T38) :

```
pter := pter-début-type-file
pter-début-type-file := pter^ (pter-ar)
```

"type-file" peut représenter une des 4 files de blocs de mémoire libre ainsi qu'une des 4 files de messages prêts .

MET-BLC-DEB-QUE (type-file; pter)

fonction : cette procédure insère en tête de la file type-file le bloc de mémoire indiqué par le pointeur pter.

méthode (figure T39) :

```
pter^ (pter-ar) := pter-début-type-file
pter-début-type-file := pter
```

"type-file" peut représenter une des 4 files de blocs de mémoire libre .

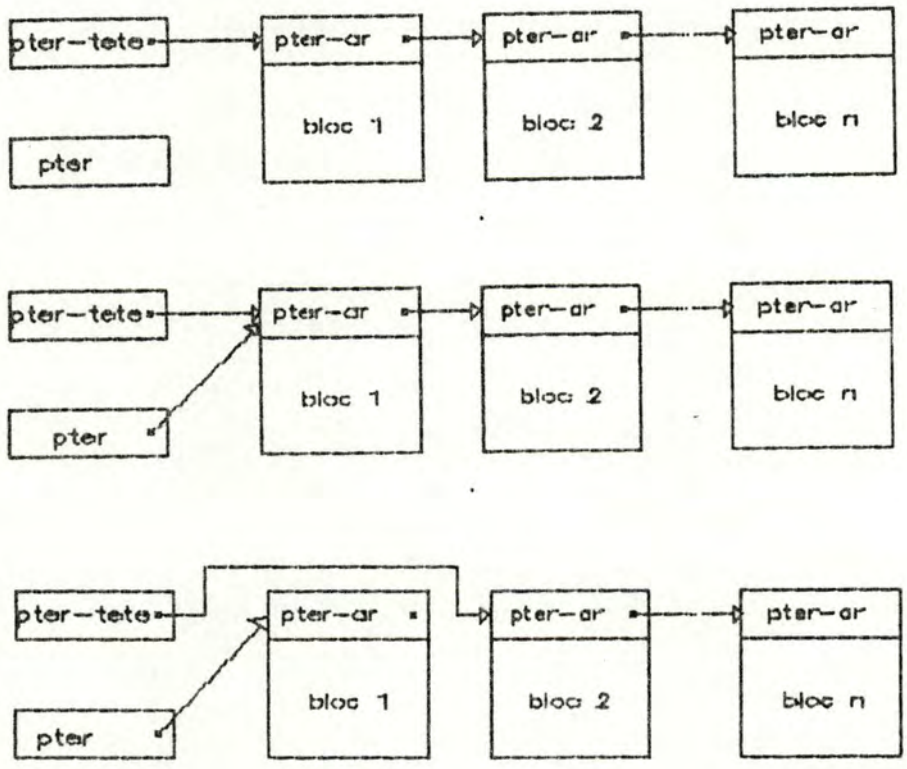


figure T38 : procedure OTE-BLC-QUE

MET-BLC-FIN-QUE (type-file; pter)

fonction : cette procédure insère en queue de la file type-file le bloc de mémoire indiqué par le pointeur pter .

méthode (figure T40):

```
pter^ (pter-ar) := pter-fin-type-file
pter-fin-type-file := pter
```

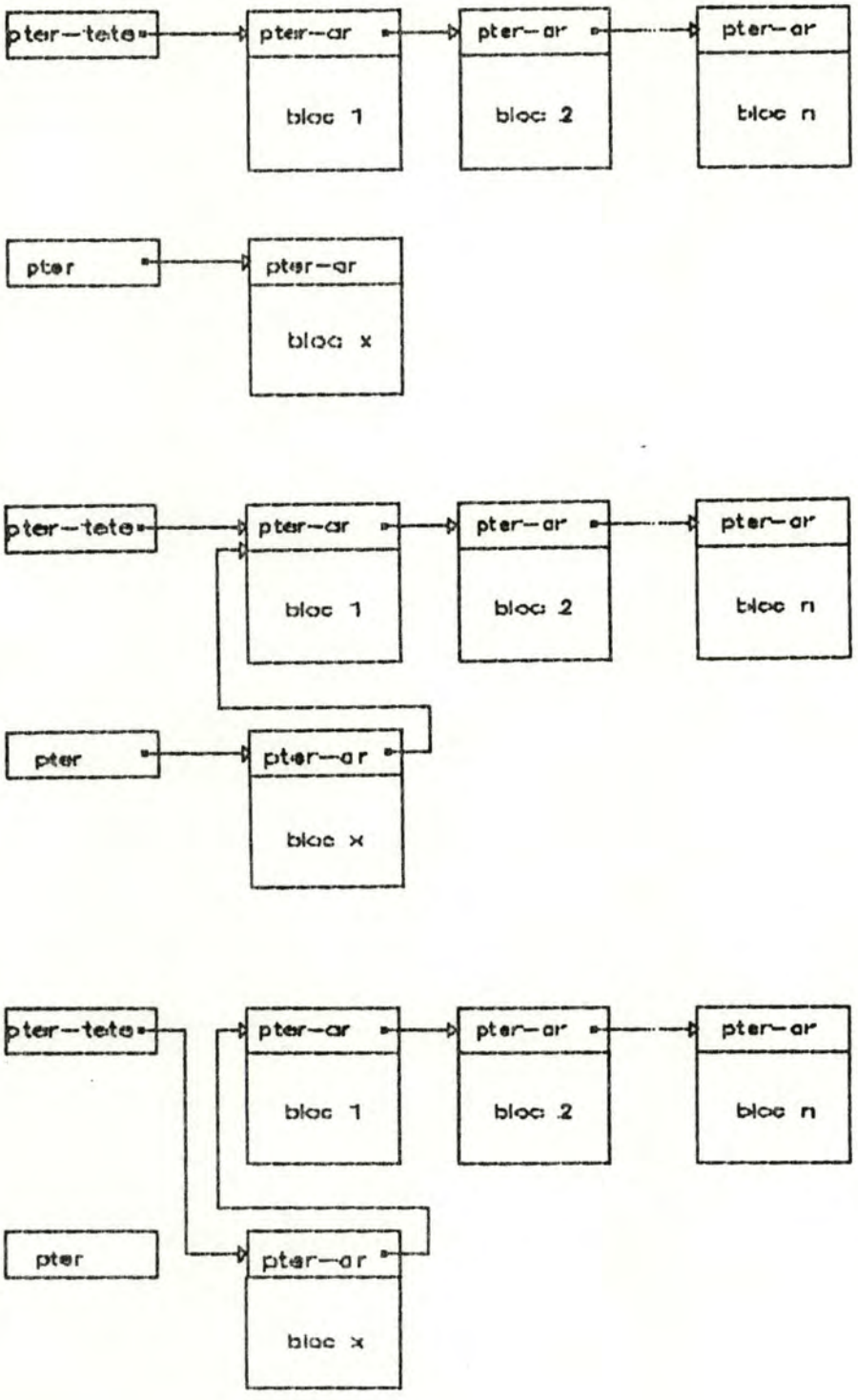


figure T39 : procedure MET-BLC-DEB-QUE

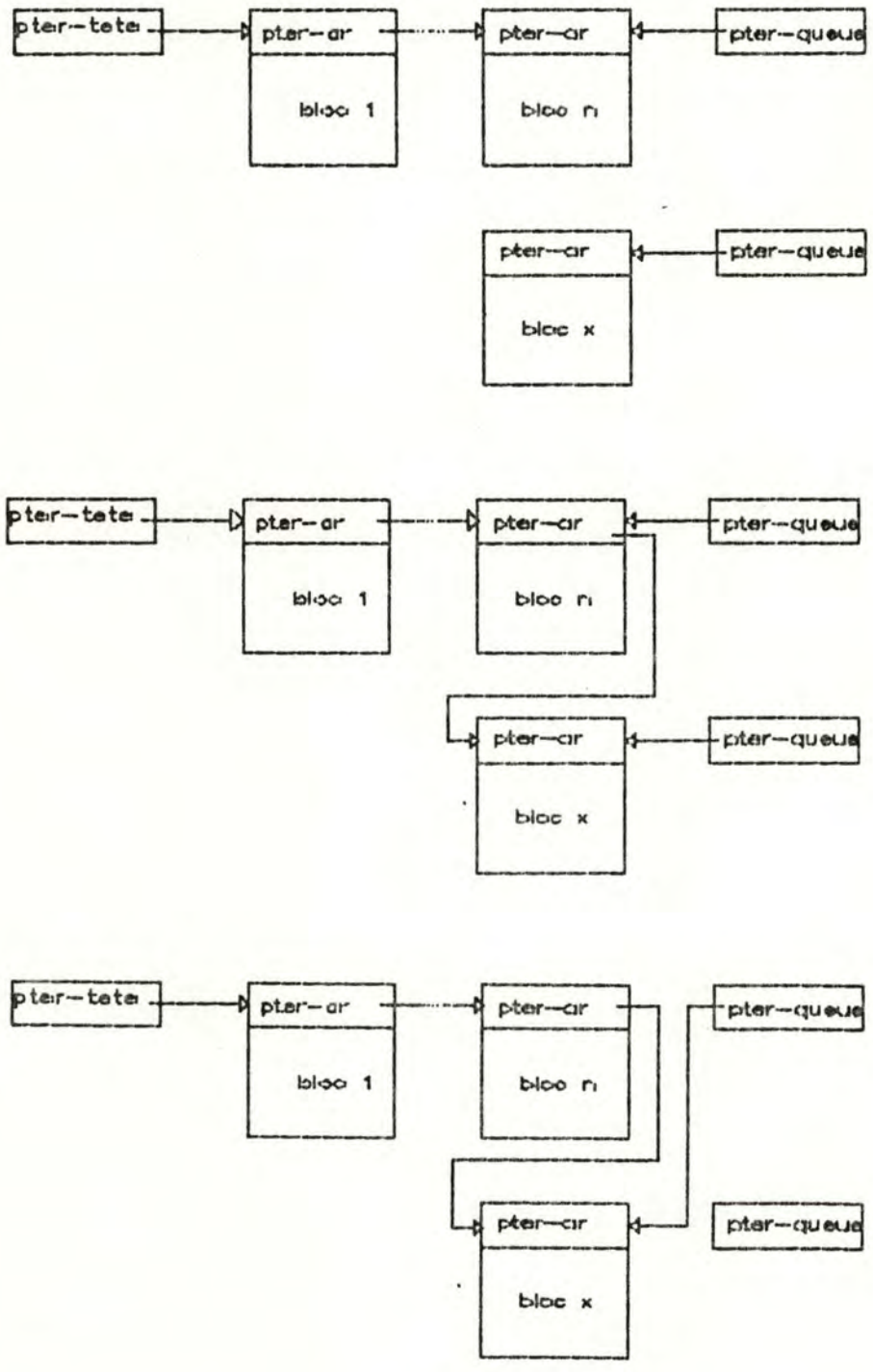


figure T40 : procedure MET-BLC-FIN-QUE

3.3.2.3. Gestion des messages.

a) Description.

Ce module (GX : gestion des échanges) est chargé du transfert de messages de n'importe quel module d'application ou d'OS vers un module automate de l'application.

Il y a quatre files d'attente de messages de priorités décroissantes. La structure d'un message est décrite dans le gestionnaire de la mémoire.

b) Primitive.

Nom : ENVOI-MSG

Paramètres : contenus dans le message lui-même

Description :

Transfert d'un message de la tâche envoyeur vers une des quatre files d'attente. Le choix de la file d'attente est fait d'après le module envoyeur:

ST : 1
ILS : 2
COLIG : 3
COCOM : 4

Le message est inséré en queue de file.

c) Séquence d'envoi d'un message.

Supposons que le module A désire envoyer un message au module B.

c.1) Obtention d'un BLC-MSG.

La tâche A doit d'abord obtenir de GM un bloc de message pour y placer le contenu de contrôle et d'information du message. Cette première étape se fait via la primitive OBT-BLC-MSG de GM qui renvoie un pointeur vers un BLC-MSG. Ce pointeur se trouve dans le BLC-CTL-TCH de A.

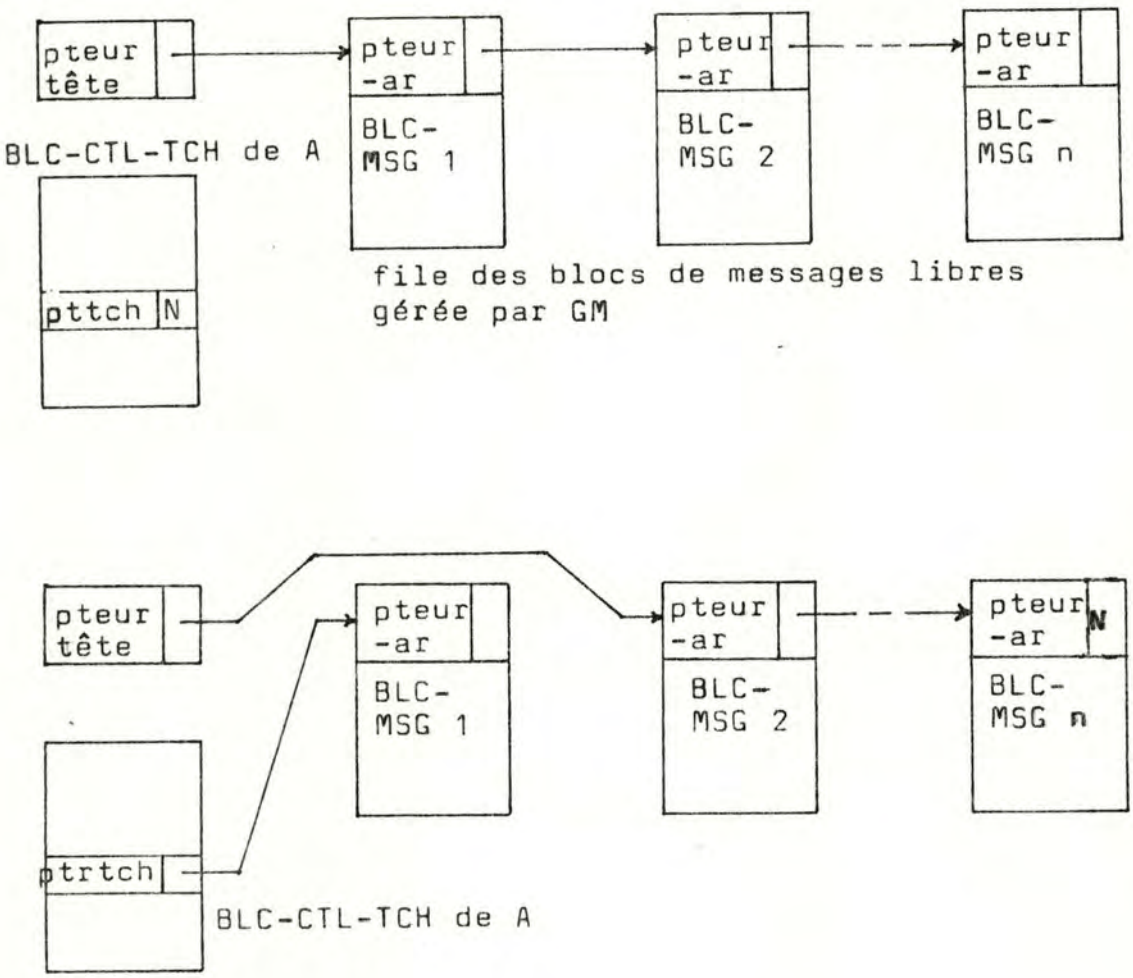


fig E77. Obtention d'un BLC-MSG.

c.2) Placement du message dans le BLC-MSG.

Le module A inscrit dans BLC-MSG le contenu du message:

- numéro de message
- heure d'envoi
- information à transmettre au destinataire

c.3) Mise du message prêt en attente.

Le module A fait appel à la primitive ENVOI-MSG DE GX. La primitive va chercher dans le BLC-CTL-TCH de A le pointeur vers le BLC-MSG.

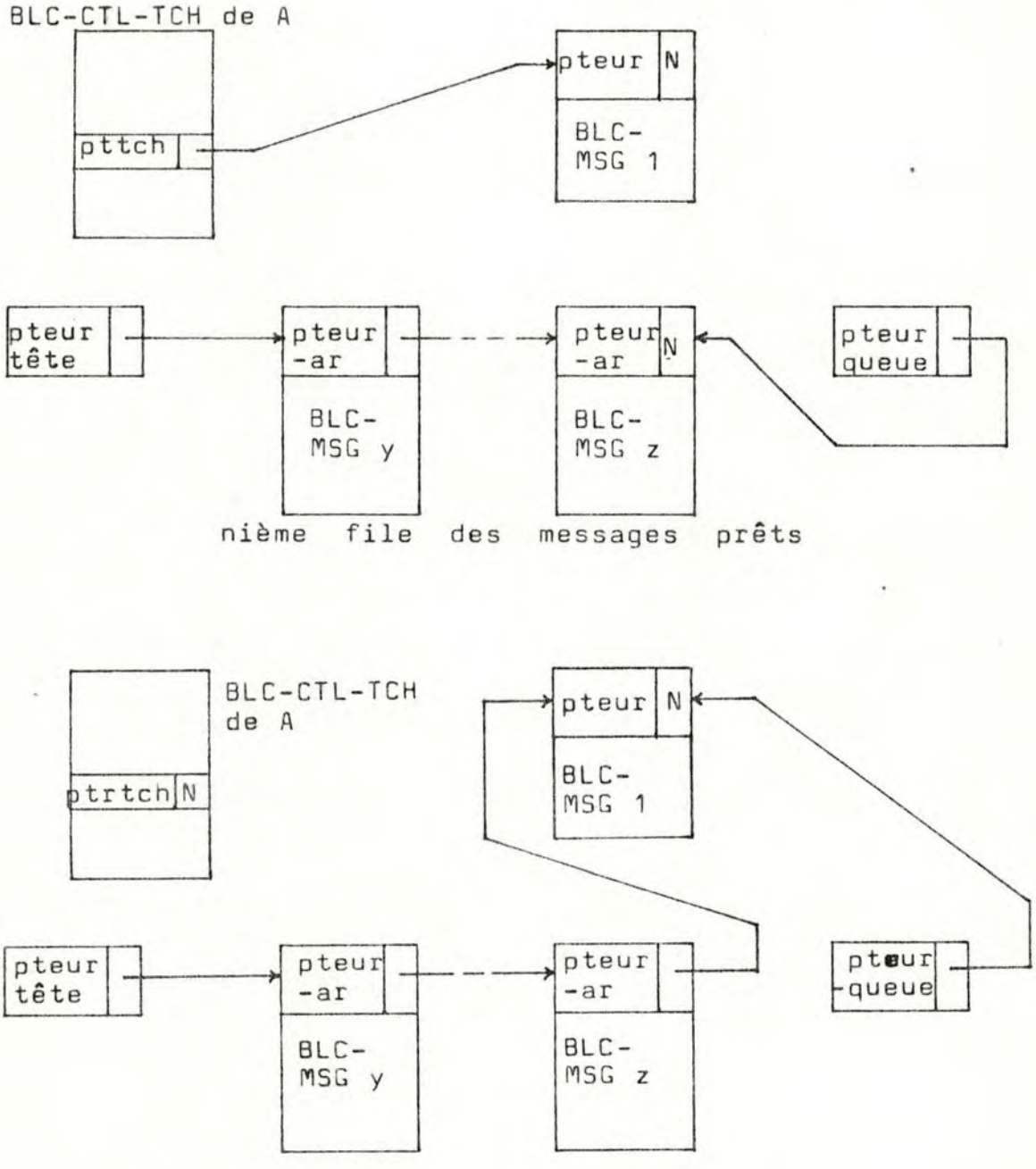


fig E78. Mise du message prêt en attente.

c.4) Transmission du message au module B.

Ceci est du ressort de GT qui achemine un message lorsqu'il doit activer une nouvelle tâche. Le BLC-MSG est ensuite restitué à GM par appel à la primitive RET-MSG-QUE-PRT de GM.

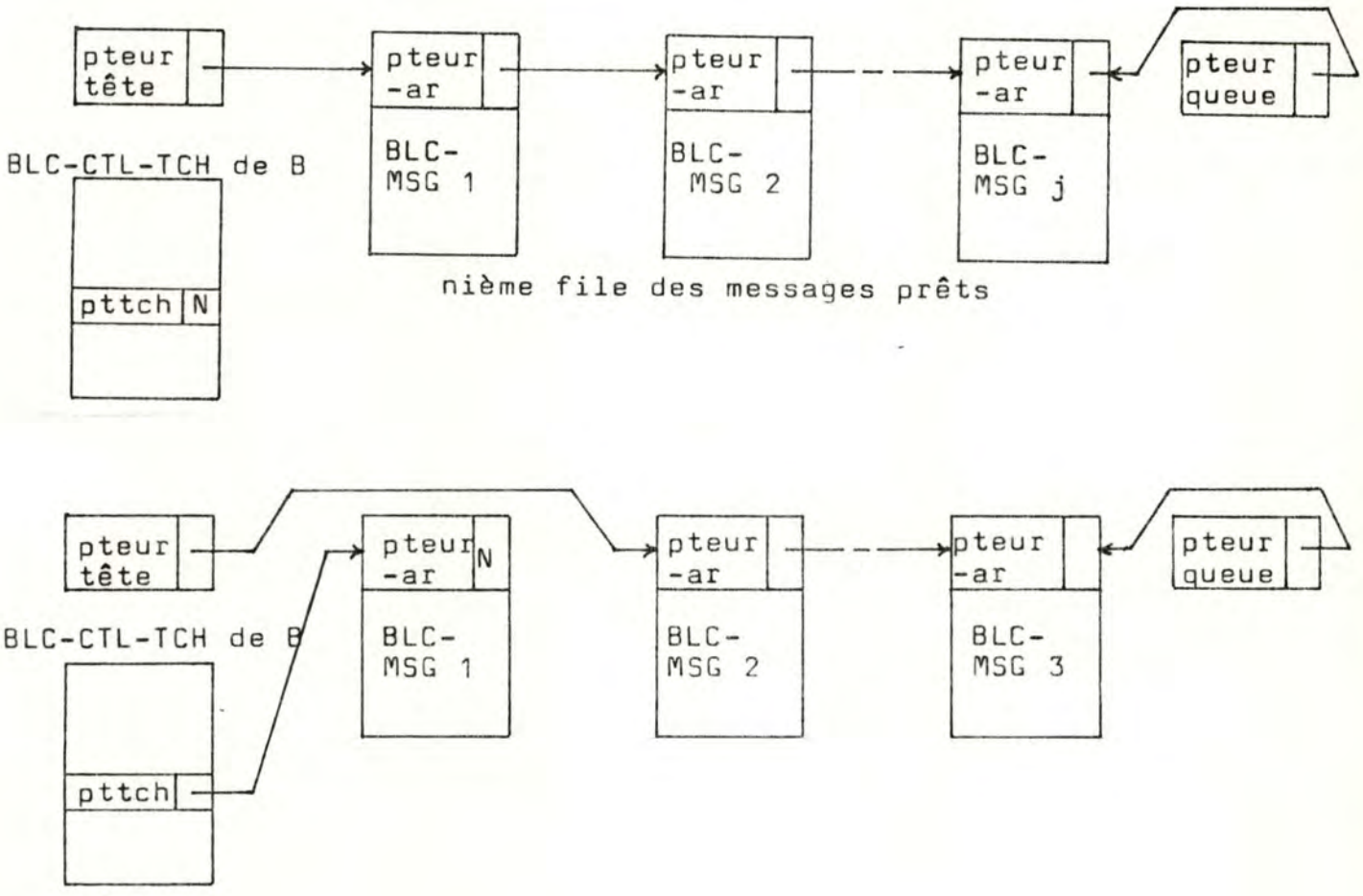


fig E79. Transmission du message à la tâche B.

3.3.2.4 Gérant du Temps

A) Introduction

Le Gérant du Temps est une tâche du système d'exploitation qui gère les fonctions reliées au temps sur base d'une horloge hardware provoquant une interruption toutes les 10 millisecondes.

Ces fonctions sont :
a) la programmation de procédures périodiques,
b) le contrôle de séquence et d'erreurs par temporisation.

B) Programmation de procédures périodiques

Les procédures périodiques étant principalement des primitives du module ILS, la description détaillée du fonctionnement est reprise au point 3.3.3.1. Néanmoins, le Gérant du Temps joue dans ce mécanisme un rôle précis et important.

Lors de chaque interruption d'horloge, le Gérant des Interruptions exécute la tâche du Gérant du Temps. Celle-ci incrémente un compteur périodique indiquant une entrée dans la table de programmation. A cette entrée, se trouvent les procédures d'horloge qui doivent être exécutées par le Gérant du Temps à cette période précise. Néanmoins, pour être exécutée, une procédure d'horloge doit être préalablement active. Le Gérant du Temps effectue cette vérification en allant tester, pour chaque procédure à exécuter, l'état actif ou inactif dans la table des statuts (figure T41).

C) Contrôle de Séquence et d'Erreurs par Temporisation

a.- Typologie des demandes de temporisation

Il existe deux types de temporisation.

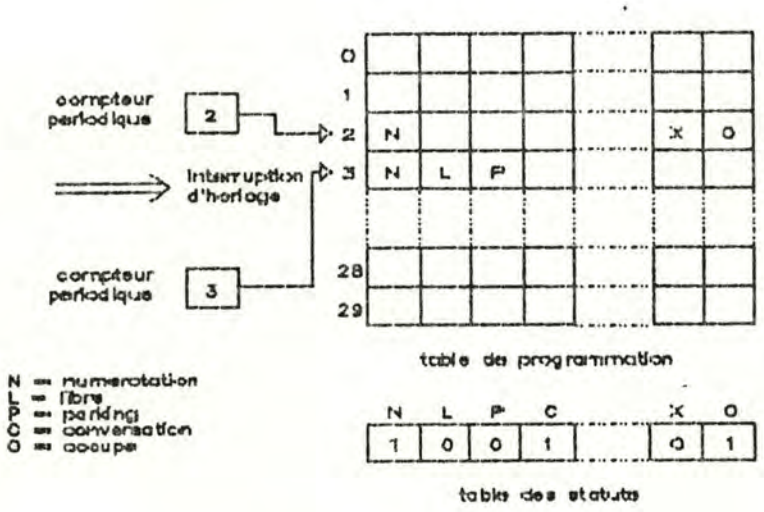


figure T-41 : programmation des procedures periodiques

La temporisation relative consiste à déclencher un processus quelconque après que ce soit écoulé un délai de temps, ce délai étant indépendant de l'heure à laquelle il fût demandé ou de l'heure à laquelle il arrive à expiration.

La temporisation absolue consiste à déclencher un processus quelconque à une heure précise, étant bien entendu que cette heure doit être supérieure à l'heure à laquelle la temporisation fût demandée.

Toute demande de temporisation absolue est directement ramenée à une temporisation relative en faisant la différence entre l'heure d'expiration et l'heure de la demande, cette différence devenant donc un délai.

b.- Création de Temporisation

Une temporisation peut être demandée à l'aide de deux primitives du Gérant du Temps.

TMP-REL (délai; id-tâche; id-temp)

fonction :

Le Gérant du Temps reconnait une demande de temporisation relative. Pour gérer toute temporisation, un bloc de contrôle de temporisation (BLC-CTL-TMP) est nécessaire.

Le Gérant du Temps appelle la primitive du Gérant de la Mémoire OBT-BLC-CTL-TMP (pter-ST).

Le Gérant du Temps gère une file de BLC-CTL-TMP organisée de manière croissante sur le délai de TMP-REL, le Gérant du Temps calcule l'endroit où le nouveau BLC-CTL-TMP doit venir s'insérer dans la file pour respecter l'organisation croissante.

Le Gérant du Temps peut alors fournir le pter-ST au Gérant de la Mémoire et un identificateur du BLC-CTL-TMP à la tâche demandeur.

TMP-ABS (heure; id-tâche)

fonction :

Le Gérant du Temps reconnait une demande de temporisation absolue. Le Gérant du Temps calcule la différence entre l'heure de TMP-ABS et l'heure courante, obtenant ainsi un délai.

Le Gérant du Temps appelle la procédure TMP-REL (délai; id-tâche; id-temp).

c.- Gestion des Temporisations

Lors de chaque interruption hardware d'horloge, le Gérant du Temps appelle la procédure suivante :

MAJ-TMP (pter-début-file)

fonction :

Cette procédure soustrait 10 msec du délai de chaque BLC-CTL-TMP de la file. Si le délai du premier BLC-CTL-TMP reste différent de 0, alors il en sera de même pour les suivants et la procédure se termine.

Si le délai du premier BLC-CTL-TMP s'annule, alors le Gérant du Temps passe en phase d'expiration de temporisation, puis vérifie le deuxième et ainsi de suite.

c.- Expiration de Temporisation

Si le délai d'un BLC-CTL-TMP a touché à sa fin, le Gérant du Temps envoie un message d'expiration de temporisation à la tâche demandeur via la procédure MAJ-TMP. Cet envoi de message suit la procédure classique décrite dans le Gérant des Echanges au point 3.3.2.3.

L'envoi du message terminé, le Gérant du Temps libère le BLC-CTL-TMP et le renvoie au Gérant de la Mémoire via la primitive RET-BLC-CTL-TMP (pter-ST).

d.- Annulation de Temporisation

Pour une raison ou pour une autre, la tâche qui fût à l'origine de la demande de temporisation garde le droit d'annuler à tout instant cette temporisation via la primitive suivante du Gérant du Temps.

DET-TMP (id-temp)

fonction :

Cette procédure retrouve dans la file le BLC-CTL-TMP dont l'identificateur est id-temp. Une fois retrouvé, le Gérant du Temps libère le BLC-CTL-TMP et le renvoie au Gérant de la Mémoire via la primitive RET-BLC-CTL-TMP (pter-ST).

3.3.3.1 I.L.S. (Interface Lignes/Système)

La fonction du module ILS est d'effectuer le scanning des lignes. Pour ce faire deux fonctions auxiliaires lui sont confiées :

- la gestion des tables de scanning,
- le traitement des événements dûs au scanning.

A) Scanner les lignes.

Une ligne peut se trouver dans un des sept états suivants :

- libre,
- pré-numérotation,
- numérotation,
- conversation,
- pré-conversation,
- post-conversation,
- parking.

Les différents états sont des "états de scanning". Une ligne ne devant pas être scannée n'aura donc aucun de ces sept états particuliers. En fait, une ligne est toujours scannée, car elle est libre lorsque elle n'est pas dans un des autres états. A chacun de ces sept états correspond une table d'un nombre de bits égal au nombre de lignes dans le système. Le bit correspondant à une ligne particulière sera positionné à "1" si cette ligne se trouve dans l'état correspondant à la table, à "0" sinon.

A chacun des états de scanning correspond une procédure de scanning. Une procédure de scanning est caractérisée par :

- son statut actif ou inactif,
- sa fréquence d'exécution,
- sa table de scanning.

La fréquence d'exécution peut être déterminée de la manière statique suivante :

- scanning libre : 100 msec
- scanning de numérotation : 10 msec
- scanning de pré-numérotation : 10 msec
- scanning de post-numérotation : 10 msec
- scanning de conversation : 300 msec
- scanning de pré-conversation : 300 msec

- scanning de parking : 100 msec

La programmation des différentes procédures selon leur fréquence d'exécution sera déterminée à l'aide d'une horloge Hardware générant une interruption toutes les dix millisecondes, d'une table de programmation et d'un compteur périodique.

La table de programmation est statique et comporte trente entrées. Chaque entrée, qui correspond à un multiple de dix millisecondes, contiendra l'identificateur des procédures de scanning devant être activées lorsque le compteur périodique indiquera cette entrée particulière. Ce compteur est incrémenté de un à chaque interruption d'horloge et remis à zéro toutes les trois cents millisecondes (figure T41).

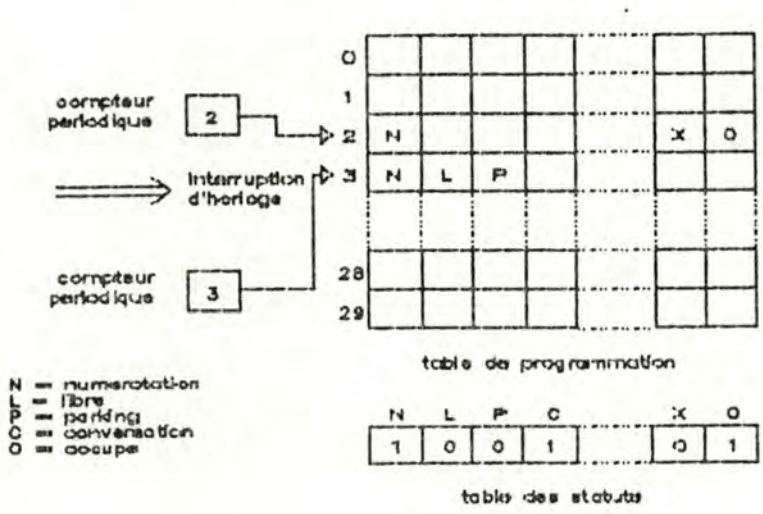


figure T41 : programmation des procedures periodiques

Le statut actif ou inactif d'une procédure de scanning est déterminé de la manière dynamique suivante : si aucune ligne ne se trouve dans l'état de scanning correspondant à la procédure, le statut de celle-ci devient inactif.

Cette gestion dynamique est réalisée grâce à la table de statuts. Chacune des cinq entrées de un bit correspond à une procédure. Ce bit est mis à "1" si le statut est actif, à "0" sinon.

Résumé.

Tous les multiples de dix millisecondes, le compteur périodique est incrémenté et identifie une entrée dans la table de programmation. Chaque procédure de scanning de cette entrée sera exécutée si l'entrée correspondante de la table des statuts l'indique comme étant active.

Procédures de Scanning

Lorsqu'une des six procédures (LIBRE, CONVERSATION, PARKING, POST-NUMEROTATION, PRE-NUMEROTATION, PRE-CONVERSATION) est exécutée, elle commence par parcourir la table de l'état correspondant pour s'informer des lignes à scanner. Pour chacune des lignes à scanner, la procédure accède au VECLIG correspondant à la ligne. Si Np ≠ Na alors, un événement a lieu. Il reste à déterminer s'il s'agit d'un événement bas, (Np = 0 & Na = 1), ou d'un événement haut, (Np = 1 & Na = 0).

Exemple.

	11	12	13	14
Np	0	0	1	1
Na	0	1	0	1
Np xor Na = Wx	0	1	1	0
Np and Wx = Eh	0	0	1	0
Na and Nx = Eb	0	1	0	0

Si Eh = "1", alors, un événement haut a eu lieu et la procédure positionne le flag d'événement haut dans le banc des événements.

Si Eb = "1", alors, un événement bas a eu lieu et la procédure positionne le flag d'événement bas dans le banc des événements.

Procédure de scanning NUMEROTATION.

Les chiffres sont codés en nombre de créneaux (figure T42).

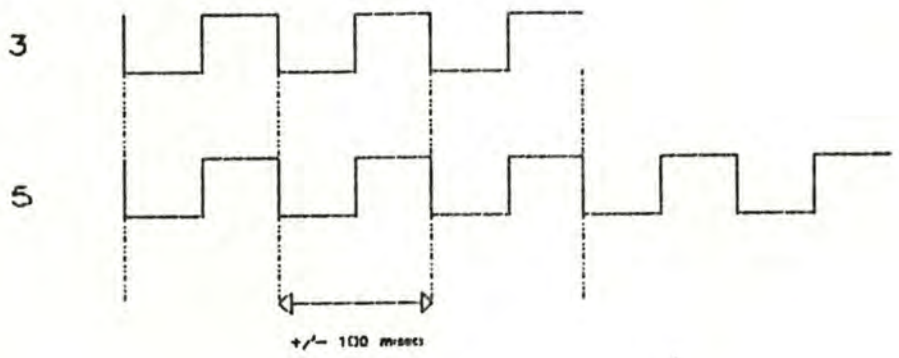


figure T42 : codage des chiffres

Détection de la fin d'un chiffre :
étant donné que le nombre de créneaux pour un chiffre est inconnu, un time-out de deux cents millisecondes est activé. Si celui-ci expire avant d'avoir reçu un nouveau créneau, c'est que l'émission du chiffre est terminée.

Reconnaissance du chiffre :
à chaque passage de la ligne d'un niveau haut à un niveau bas, ou d'un niveau bas à un niveau haut, Np sera différent de Na et un événement sera positionné. Dans ce cas, la reconnaissnce du chiffre n'est pas effectuée par ILS.

Positionnement de l'événement :
cfr. la reconnaissance du chiffre.

B) Gérer les tables.

a.- Tables d'états.

- procédure qui enlève une ligne d'une des tables d'états.
- procédure qui positionne une ligne dans une des tables d'état.

b.- Tables des statuts.

- procédure qui active une des procédures de scanning.
- procédure qui désactive une des procédures de scanning.

C)Traiter les événements.

Il existe une procédure de traitement d'événements par types d'événements possibles.

Types d'événements possibles :
- événement haut, (passage du niveau haut au niveau bas).
- événement bas, (passage du niveau bas au niveau haut).

Le rôle des procédures de traitement d'événements consistera principalement dans l'envoi d'un message vers le module COLIG et auxiliairement dans des fonctions comme positionner la valeur de l'état antécédent à la valeur de l'état présent.

D) Primitives ILS.

CHGT-ETAT (nr-ligne; nr-état-ancien;
nr-état-nouveau)

fonction :
mettre la ligne dans la table d'état nr-état-nouveau,
appeler la procédure ACT-SCANNING(nr-état-nouveau),
appeler la fonction booléenne SIGN-ETAT(nr-ligne; nr-état-ancien),
si SIGN-ETAT = vrai
alors : retirer la ligne nr-ligne de la table d'état nr-état-ancien,
appeler la procédure DESACT-SCANNING (nr-état-ancien);
sinon : appeler une procédure d'erreur.

ACT-HDW (nr-ligne; nr-appareillage)

fonction :
activer dans le VECLIG de la ligne nr-ligne l'appareillage hardware nr-appareillage.

DESACT (nr-ligne; nr-appareillage)

fonction :
désactiver dans le VECLIG de la ligne nr-ligne l'appareillage hardware nr-appareillage.

SCANNING (nr-état)

fonction :
pour chaque ligne active dans la table d'état nr-état
accéder à VECLIG,
si Np ^{23} Na
alors si Np = 1
alors envoyer à COLIG le message 1201
sinon envoyer à COLIG le message 1202.

E) Fonctions ILS

SIGN-ETAT (nr-ligne; nr-état)

fonction :
variable booléenne qui prend la valeur VRAI si la ligne nr-ligne se trouve dans la table d'état nr-état et prend la valeur FAUX sinon.

ETAT-OCCUPE (nr-état)

fonction :
variable booléenne qui prend la valeur VRAI si il existe au moins une ligne active dans la table d'état nr-état et prend la valeur FAUX si il n'y en a aucune.

F) Procédures ILS

ACT-SCANNING (nr-état)

fonction :
activer dans la table des statuts la procédure de scanning correspondant à l'état nr-état.

DESACT-SCANNING (nr-état)

fonction :
appeler la fonction booléenne ETAT-OCCUPE (nr-état)
si ETAT-OCCUPE = FAUX
alors désactiver dans la table des statuts la procédure de scanning correspondant à l'état nr-état.

3.3.3.2. Colig.

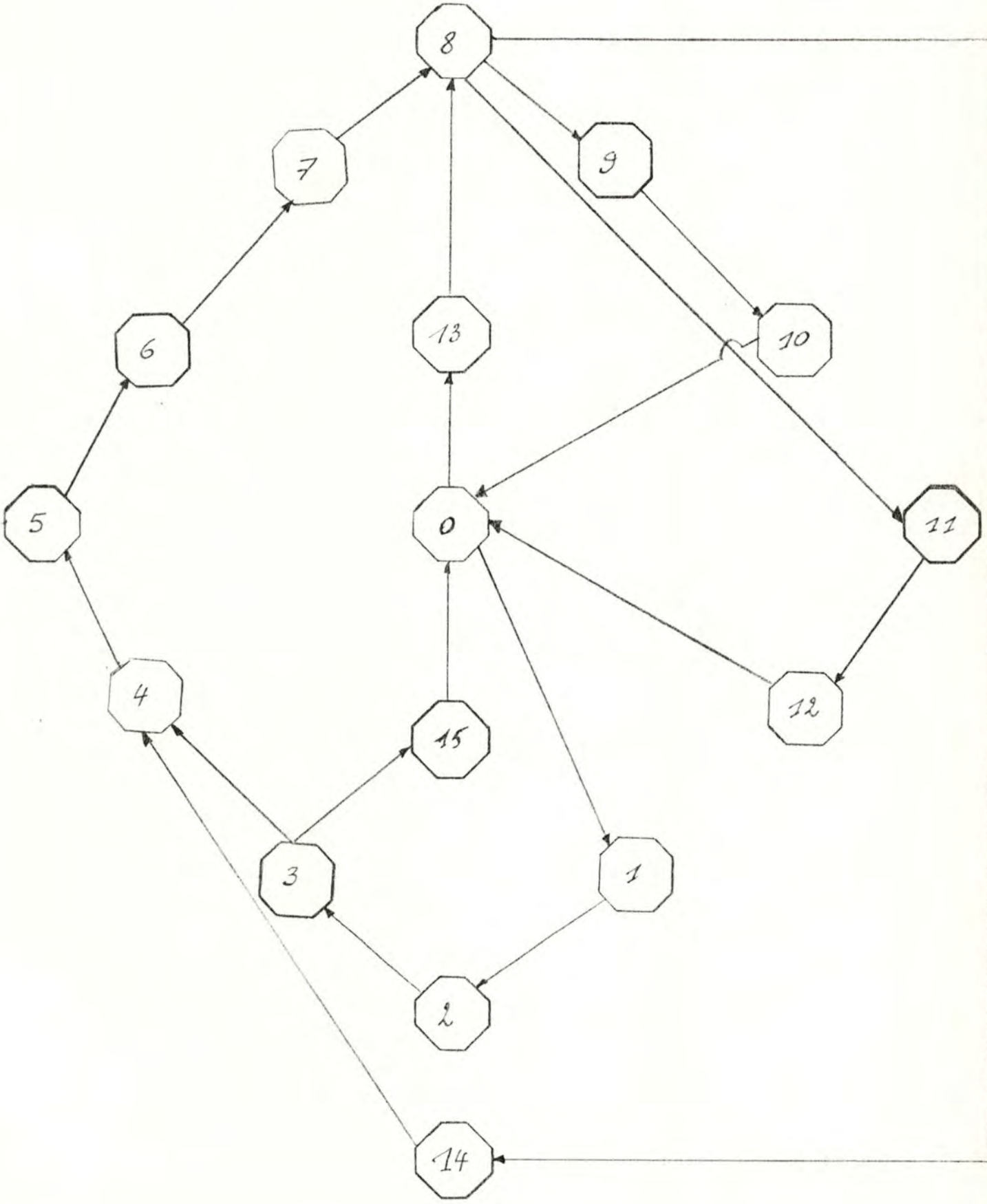
A. Table des transitions.

Etat présent	Action	Evénement	Etat suivant
Ø	repos	msg 1201 msg 0202	1 13
1	Création d'un processus Cocom Envoi du msg 2301 Activation de la procédure ILS CHGT-ETAT Activation d'une procédure de scanning pré-numérotation	msg 3201	2
2	Activation de la procédure ILS CHGT-ETAT Activation d'une procédure scanning numérotation Désactivation de la procédure scanning pré-numérotation Activation de la procédure ILS qui va positionner à 1 le bit 2 Envoi du msg 2302	msg 1202	3
3	Activation de la procédure ILS qui va positionner à Ø le bit 2 de l'appelant Traitement du changement de niveau en vue d'établir le chiffre Envoi du chiffre détecté msg 2303	msg 0201 msg 3202	15 4
4	Activation de la procédure ILS CHGT-ETAT Activation de la procédure ILS de scanning post-numérotation		

Etat présent	Action	Evénement	Etat suivant
	Désactivation de la procédure ILS de scanning numérotation		
	Envoi du msg 2304	msg 3203	5
5	Activation de la procédure ILS CHGT-ETAT		
	Activation de la procédure de scanning pré-conversation		
	Activation de la procédure ILS qui va positionner le bit 6 de VECLIG de l'appelé à 1		
	Activation de la procédure ILS qui va positionner le bit 3 de VECLIG de l'appelant à 1		
	Envoi du msg 2305	msg 1201	6
6	Activation de la procédure ILS qui va positionner à Ø le bit 6 du VECLIG de l'appelé et le bit 3 de l'appelant		
	Activation de la procédure ILS CHGT-ETAT		
	Envoi du msg 2306	msg 3205	7
7	Activation de la procédure ILS qui va retirer la ligne appelante de l'état post-numération et le mettre dans l'état conversation	msg 1202	8
8	Envoi du msg 2307	msg 3206 msg 1201 msg 3207	9 14 11
9	Destruction du processus Cocom		
	Activation de la procédure ILS CHGT-ETAT (appelant)		
	Idem (appelé)	msg 1202	10

Etat présent	Action	Evénement	Etat suivant
10	Activation de la procédure ILS CHGT-ETAT (appelé)		Ø
11	Destruction du processus Cocom		
	Activation de la procédure ILS CHGT-ETAT (appelé)		
	Idem(appelant)	msg 1202	12
12	Activation de la procédure ILS CHGT-ETAT (appelant)		Ø
13	Envoi du msg 2309	msg 1202	8
14	Création d'un processus Cocom		
	Envoi du msg 2310		4
15	Destruction du processus Cocom		
	Activation de la procédure CHGT-ETAT qui va remettre l'appelant dans l'état libre		Ø

B. Diagramme des transitions.



3.3.3.3. Cocom.

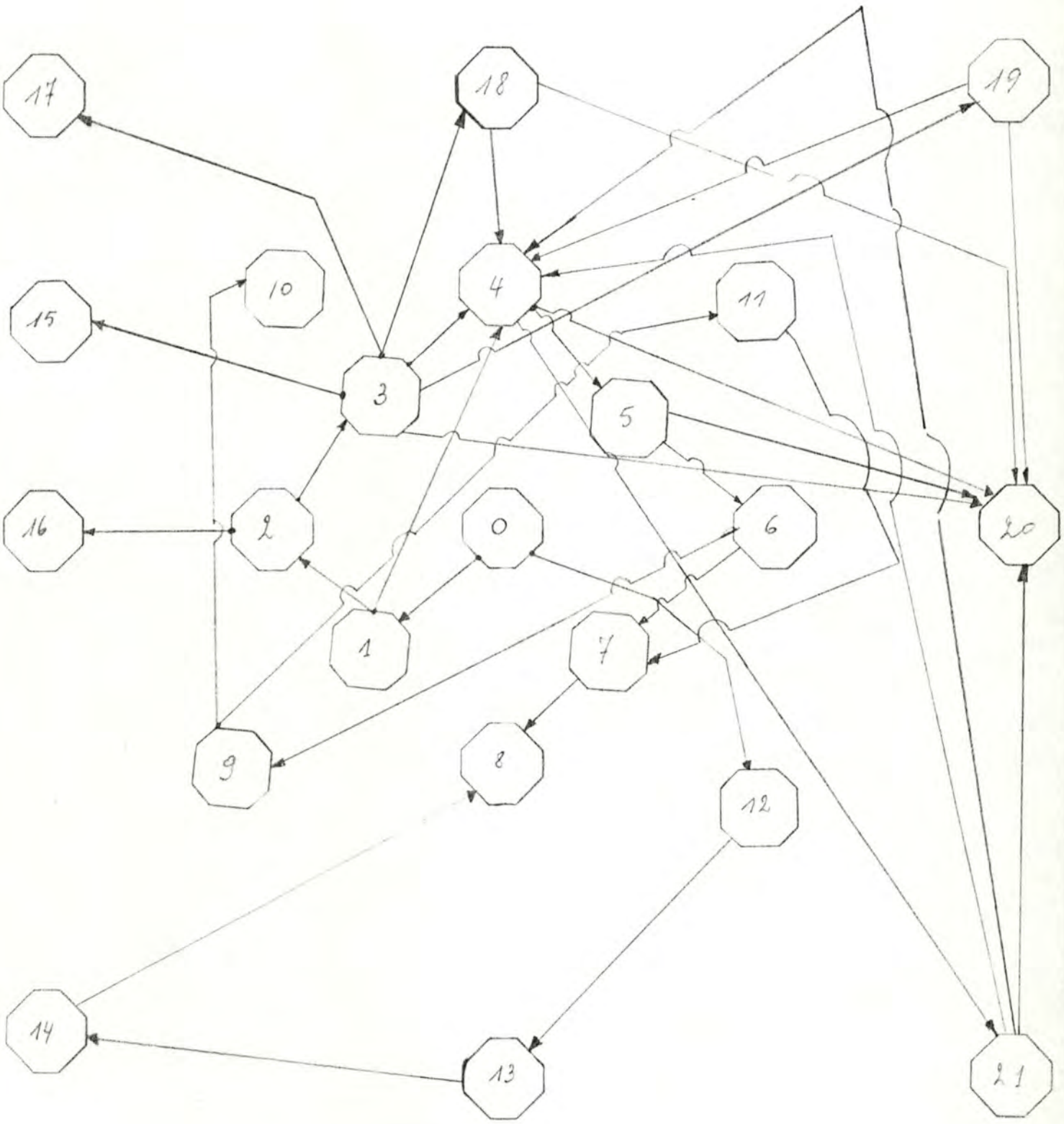
A. Table des transitions.

Etat présent	Action	Evénement	Etat suivant
naissance	repos	msg 2301 msg 2310	1 12
1	consultation de la base de données pour extraire des informations sur la ligne et l'abonné envoi du msg 3201	si appelé pré-déterminé msg 2302	4 2
2	demande au service de temps de l'operating system d'activer un time out de 24 secondes	msg 2303 msg 0302	3 16
3	Cocom met à jour les T.O. Cocom procède à l'analyse du préfixe Il continue par l'enregistrement de n chiffres; n dépend du résultat de l'analyse. Accès à la B.D. pour voir si l'appelé ne doit pas être dérouté ou si l'appelé est classé malveillant Envoi du msg 3202	→suite à l'analyse →si c'est le cas msg 2304 msg 2307	15,17 18,19 4 20
4	Envoi du msg 3203	msg 2305 msg 2307 msg 2309	5 20 21
5	Envoi du msg 3204	msg 2306 msg 2307	6 20
6	Envoi du msg 3205	msg 2307	7 9

Etat présent	Action	Evénement	Etat suivant
7	Envoi du msg 3401	msg 4301	8
8	Envoi du msg 3206 ou 3207		mort
9	Demande au time service de positionner un T.O. de 90 secondes	msg 2308	11
	Envoi du msg 3402	msg 4301	10
10	Envoi du msg 3206		mort
11	Désactivation du T.O. de 90 secondes	msg 2307	7
12	Accès à la B.D. de laquelle on extrait le numéro de l'appelant		
	Envoi du msg 3203	msg 2305	13
13	Envoi du msg 3204	msg 2306	14
14	Cocom sait que la ligne appelante est la ligne virtuelle,il demande dès lors une libération vers l'avant.		
	Création du tratax		
	Envoi du msg 3401	msg 4301	8
15	Déclenchement Enregistrement du numéro prédéterminé dans la base de données		
	Envoi du msg 3207		mort
16	Envoi du msg 3207		mort

Etat présent	Action	Evénement	Etat suivant
17	(cas de la formation d'une heure de réveil)		
	Cocom procède à l'enregistrement dans la table de l'identification du time out absolu et du numéro de l'appelant.		
	Activation du processus		
	Envoi du msg 3207		mort
18	Extraction de la B.D. du numéro vers lequel il faut dérouter l'appelé	msg 2304	4
	Envoi du msg 3202	msg 2307	20
19	Enregistrement de l'heure à laquelle est effectué l'appel et du numéro de l'appelant.	msg 2304	4
	Envoi du msg 3202	msg 2307	20
20	Envoi du msg 3206		mort
21	Recherche, dans la B.D., s'il existe d'autres numéros d'appelés pouvant se substituer au premier introduit		
	Si oui		4
	Si non		20

B. Diagramme des transitions.



3.3.3.4. Taxation.

3.3.3.4.1. Module principal TAXAT.

a) Spécification.

Elles ont été données en 3.1.3.

b) Appel à TAXAT.

L'appel se fait par l'envoi du message 3401 contenant les paramètres.

c) Paramètres.

NUMLIGA : numéro de ligne appelante, entier(1..500)

NUMLIGB : numéro de ligne appelée, entier(1..500)

HDEB : heure de début de conversation, entier HHMMSS

HFIN : heure de fin de conversation, entier HHMMSS

JDEB : jour de début de conversation, entier AAMMJJ

JFIN : jour de fin de conversation, entier AAMMJJ

INDCHRG : indicatif de charge, entier = 1 : appelant
= 2 : appelé
= 3 : gratuit

d) Organigramme de TAXAT.

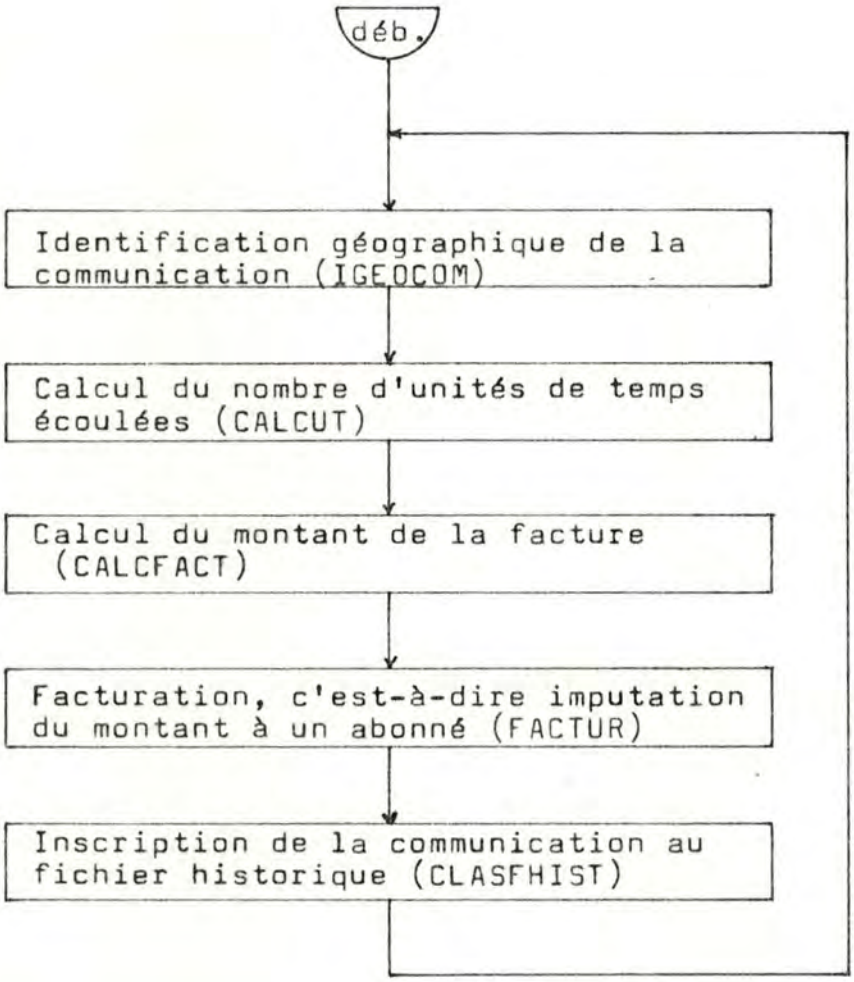


fig E80. Organigramme de TAXAT.

e) Tables.

Une table des tarifs est utilisée. C'est un vecteur appelé TARIF, composé de 4 records pour les 4 types géographiques de communications, comprenant chacun une partie JOUR et une partie NUIT pour le tarif normal et le tarif réduit.

3.3.3.4.2. Procédures.

1) IGEOCOM.

1.a) Spécification.

Identifier le type géographique de communication. Ceci correspond à la première phase des spécifications.

1.b) Paramètres.

* input:

```
        NUMLIGA : comme TAXAT

        NUMLIGB : comme TAXAT

    * output:

        TGEOCOM : type géographique de communication,
                  entier = 1 : zonale
                         = 2 : interzonale type A
                         = 3 : interzonale type B
                         = 4 : interzonale type C
```

1.c) Algorithme.

```
GET RECORD (fichier:FAB, CLE:NUMLIGA)
IGEOZONEA := ABONNE.IGEOZONE

GET RECORD (fichier:FAB, CLE:NUMLIGB)
IGEOZONEB := ABONNE.IGEOZONE

GET RECORD (fichier:FGEO, CLE:IGEOZONEA)
ZONEA := ZONE

GET RECORD (fichier:FGEO, CLE:IGEOZONEB)
ZONEB := ZONE

IF IGEOZONEA = IGEOZONEB
     THEN TGEOCOM := 1
          EXIT

FOR I := 1 TO 5
     IF ZONEA.IZONECONT(I) = IGEOZONEB
          THEN IF  ZONEA.TAILLEZONE = 1
                 & ZONEB.TAILLEZONE = 1
                    THEN TGEOCOM := 2
                         EXIT
                    ELSE TGEOCOM := 3
                         EXIT

FOR I := 1 TO 5
     FOR J := 1 TO 5
          IF ZONEA.IZONECONT(I) = ZONEB.IZONECONT(J)
               THEN GET RECORD(fichier:FGEO,
                               clé:ZONEA.IZONECONT(I))
                    IF ZONE.TAILLEZONE = 1
                         THEN TGEOCOM := 3
                              EXIT
TGEOCOM := 4
```

2) CALCUT.

2.a) Spécification.

Calculer la durée d'une conversation en unités de

temps. L'unité par défaut est la seconde. La durée est ventilée en unités à tarif normal et unités à tarif réduit.

2.b) Paramètres.

* input:

HDEB, HFIN, JDEB, JFIN : comme TAXAT

* output:

NBUTTN : nombre d'unités en tarif normal

NBUTTR : nombre d'unités en tarif réduit

2.c) Algorithme.

(Utilise procédure ISOLTRANCHE)

```
NBUTTN := NBUTTR := 0
H1 := HDEB
J1 := JDEB
WHILE (H1 ≠ HFIN) OR (J1 ≠ JFIN)
     ISOLTRANCHE (H1,HFIN,J1,JFIN,NBUT,TT)
     IF TT = 1 THEN NBUTTN := NBUTTN + NBUT
               ELSE NBUTTR := NBUTTR + NBUT
```

3) CALCFACT.

3.a) Spécification.

Calculer le montant de la facture.

3.b) Paramètres.

* input:

NBUTTN, NBUTTR : comme CALCUT

TGEOCOM : comme IGEOCOM

* output:

MONTFACT : montant à facturer

3.c) Algorithme.

```
MONTFACT := NBUTTN * TARIF.JOUR (TGEOCOM)
          + NBUTTR * TARIF.NUIT (TGEOCOM)
```

4) FACTUR.

4.a) Spécification.

Facturer la communication à l'appelant, l'appelé ou éventuellement à personne si la communication est gratuite.

4.b) Paramètres.

* input:

NUMLIGA, NUMLIGB, HDEB, HFIN, JDEB, JFIN, INDCHRG : comme TAXAT

MONTFACT : comme CALCFACT

* output:

néant

4.c) Algorithme.

```
IF INDCHRG = 1
     THEN GET RECORD (fichier:FAB, clé:NUMLIGA)
          ABONNE.MONTFACT := ABONNE.MONTFACT + MONTFACT
          WRITE RECORD (fichier:FAB, clé:NUMLIGA)

IF INDCHRG = 2
     THEN GET RECORD (fichier:FAB, clé:NUMLIGB)
          ABONNE.MONTFACT := ABONNE.MONTFACT + MONTFACT
          WRITE RECORD (fichier:FAB, clé:NUMLIGB)
```

5) CLASFHIST.

5.a) Spécification.

Classer la communication dans le fichier historique des communications.

5.b) Paramètres.

* input:

NUMLIGA, NUMLIGB, HDEB, HFIN, JDEB, JFIN, INDCHRG : comme TAXAT

MONTFACT : comme CALCFACT

* output:

néant

6) ISOLTRANCHE.

6.a) Spécification.

Isoler un tranche de temps entre deux dates et deux heures données suivant le principe suivant : la tranche est la période de temps comprise entre la date et l'heure de début, et le moment le plus proche entre soit la date et l'heure de fin, et soit le plus proche passage entre deux types de tarifs différents, soit l'heure de minuit.

6.b) Paramètres.

* input:

H1, H2 : heure de début et de fin (HHMMSS)

J1, J2 : jour de début et de fin (AAMMJJ)

* output:

H1, J1 : heure et jour de fin de la période isolée

NBUT : nombre d'unités de temps contenues dans la période

TT : type de tarif = 1 : normal
= 2 : réduit

6.c) Algorithme.

```
( Utilise procédures JOURSUIV, FERIE et CALCINT)

IF FERIE (j1)
     THEN IF (H2<MINUIT) & (J1=J2)
               THEN CALCINT (H1,H2,NBUT)
                    H1 := H2
               ELSE CALCINT (H1,MINUIT,NBUT)
                    H1 := 0
          TT := 2
          EXIT
IF H1<HDJOUR
     THEN IF (H2<HDJOUR) & (J1=J2)
               THEN CALCINT (H1,H2,NBUT)
                    H1 := H2
               ELSE CALCINT (H1,HDJOUR,NBUT)
                    H1 := HDJOUR
          TT := 2
          EXIT
IF H1<HFJOUR
     THEN IF (H2<HFJOUR) & (J1=J2)
               THEN CALCINT (H1,H2,NBUT)
                    H1 := H2
               ELSE CALCINT (H1,HFJOUR,NBUT)
                    H1 := HFJOUR
```

```
                TT := 1
                EXIT
     IF (H2<MINUIT) & (J1=J2)
          THEN CALCINT (H1,H2,NBUT)
                H1 := H2
          ELSE CALCINT (H1,MINUIT,NBUT)
                H1 := 0
                J1 := JOURSUIV(J1)
     TT := 2
```

7) JOURSUIV.

7.a) Spécification.

fonction donnant le jour suivant celui fourni en paramètre.

7.b) Paramètres.

JPREC : jour précédent (AAMMJJ)

7.c) Algorithme.

```
     ( Utilise procédures MOISSUIV et BISSEXT)

     JOUR := (JPREC/100 - INT(JPREC/100))*100
     MOIS := ((JPREC-JOUR)/10000-INT(JPREC-JOUR)/10000)*10000
     ANNEE := INT(JPREC/10000)
     IF (JOUR=28)&(MOIS=2)&NOT BISSEXT(ANNEE)
          THEN MOISSUIV(JPREC)
                EXIT
     IF (JOUR=29)&(MOIS=2)& BISSEXT(ANNEE)
          THEN MOISSUIV(JPREC)
                EXIT
     IF (JOUR=30)&(MOIS=4 OR 6 OR 9 OR 11)
          THEN MOISSUIV(JPREC)
                EXIT
     IF (JOUR=31)&(MOIS=1 OR 3 OR 5 OR 7 OR 8 OR 10 OR 12)
          THEN MOISSUIV(JPREC)
                EXIT
     JOURSUIV := JPREC + 1
```

8) MOISSUIV.

8.a) Spécification.

Faire passer une date au début du mois suivant.

8.b) Paramètres.

* input:

ANNEE, MOIS, JOUR : entiers à deux chiffres

* output:

ANNEE, MOIS, JOUR : modifiés

8.c) Algorithme.

```
IF MOIS = 12
     THEN MOIS := 1
          ANNEE := ANNEE + 1
     ELSE MOIS := MOIS + 1
JOUR := 1
```

9) BISSEXT.

9.a) Spécification.

Fonction déterminant si une année est bissextile.

9.b) Paramètres.

* input:

ANNEE : entier à deux chiffres

* output:

Bissext = 0 si l'année est bissextile.

9.c) Algorithme.

```
IF (((ANNEE/4=0)&(ANNEE/100 ≠0))OR(ANNEE/400=0))
     THEN BISSEXT := 0
     ELSE BISSEXT := 1
```

10) CALCINT.

10.a) Spécification.

Calculer le nombre d'unités de temps (secondes) entre deux heures d'un même jour.

10.b) Paramètres.

* input:

H1, H2 : heures, entiers HHMMSS

* output:

NBUT : nombre de secondes entre H1 et H2

10.c) Algorithme.

```
NBUT := H2 - H1 - 4040
```

11) FERIE.

11.a) Spécification.

fonction déterminant si une date est un jour férié.

11.b) Paramètres.

* input:

JOUR : AAMMJJ

* output:

FERIE = 0 si JOUR est férié

11.c) Algorithme.

On se sert d'un tableau JFER contenant au maximum MAXJFR jours fériés et en contenant en réalité NEFJFR.

```
FOR I := 1 TO NEFJFR
     IF JOUR = JFER(I)
          THEN FERIE := 0
               EXIT
```

3.3.3.5. Base de données.

1) Fichier des abonnés.

- fichier : FAB
- organisation : séquentiel indexé
- accès : indexé sur le numéro de ligne NUMLIG
- record : ABONNE
- items :
  - NUMLIG : entier(1..500)
  - NUMABO : entier
  - IGEOZONE : entier
  - SIGABO : tableau de 100 caractères
  - NUMDER : tableau de 5 entiers
  - ITOREV : tableau de 5 entiers
  - TYPETAX : entier
  - MONTFACT : entier

2) Fichier des zones géographiques.

- fichier : FGEO
- organisation : séquentiel indexé
- accès : indexé sur l'indicatif de zone IGEOZONE
- record : ZONE
- items :
  - IGEOZONE : entier
  - TAILLEZONE : entier
  - IZONECONT : tableau de 5 entiers

3) Fichier historique.

- fichier : FHIST
- organisation : séquentiel
- accès : séquentiel
- record : HISTO
- items :
  - NUMLIGA : entier
  - NUMLIGB : entier
  - HDEB : entier HHMMSS
  - HFIN : entier HHMMSS
  - JDEB : entier AAMMJJ
  - JFIN : entier AAMMJJ
  - INDCHRG : entier
  - MONTFACT : entier

3.3.3.6. Contenu des messages.

a) ST → COLIG.

0201

Rôle : Renvoyer à COLIG l'ID du time-out dont il vient de demander l'initialisation à une certaine valeur.

Contenu : - ID de la ligne
- ID du time-out

0202

Rôle : signaler l'arrivée à échéance d'un time-out.

Contenu : - ID du time-out
- ID de la ligne

b) ST → COCOM.

0301

Rôle : Renvoyer à COCOM l'ID du time-out dont il vient de demander l'initialisation à une certaine valeur.

Contenu : - ID de la ligne
- ID du time-out

0302

Rôle : Signaler l'arrivée à échéance d'un time-out

Contenu : - ID de la ligne
- ID du time-out

c) ILS → COLIG.

1201

Rôle : Signaler le passage d'une ligne du niveau haut au niveau bas

Contenu : - numéro de ligne

1202

Rôle : Signaler le passage d'une ligne du niveau bas au niveau haut

Contenu : - numéro de ligne

d) COLIG → COCOM.

Contenu pour tous les messages : numéro de ligne

2301

Transmettre des informations sur la ligne et l'abonné appelant

2302

Informer que l'envoi de la tonalité d'invitation à transmettre pour une ligne a été positionné.

2303

Transmettre un chiffre (Contenu : numéro de ligne et chiffre)

2304

Indiquer que la réception pour une ligne est libérée

2305

Informer que la saisie de l'appelé est terminée

2306

Avertir que l'appelé vient de décrocher

2307

Informer du raccrochage de l'appelant ou de l'appelé

2308

Informer du redécrochage de l'appelé

2309

Informer que la ligne appelée est occupée

2310

Transmettre l'ID d'un time-out (Contenu : numéro de ligne et ID du time-out).

e) COCOM → COLIG.

Contenu pour tous les messages : ID de la ligne.

3201

Demander de se préparer à recevoir des chiffres pour une ligne donnée.

3202

Demander d'arrêter la réception de chiffres pour une ligne donnée.

3203

Demander de saisir l'appelé.

3204

Envoi de données sur l'abonné

3205

Demander la saisie de l'appelant

3206

Demander la libération des interlocuteurs

3207

Demander la libération de l'appelant

f) COCOM → TAXAT.

3401

Rôle : demander la taxation d'une communication

Contenu : - numéro de l'appelant
- numéro de l'appelé
- heure/jour de début/fin de communication
- indicatif de taxation

3.4.1 Introduction.

Lors de l'implémentation du prototype nous avons été confronté au problème du choix d'un système d'exploitation en temps réel.

La maquette établie dans les analyses fonctionnelle et organique nous permettait de le construire nous-même . Et d'autre part, elle nous servait de base de référence pour la sélection d'un système d'exploitation existant. Il va de soi qu'il était plus simple pour nous d'utiliser de tels systèmes, avec l'aide du professeur Brunin, nous avons entamé notre prospection.

Premièrement, le RTM 80, ce système conçu et mis en place aux A.C.E.C. de Charleroi ne gérant que 16 processus en parallèle et son accès étant, par ailleurs, impossible fut exclus.

Deuxièmement, le iRMX 86-88 convenait idéalement à notre application mais malheureusement il ne fonctionne que sur une architecture de type SBC coûtant très cher.

In fine, monsieur Pierson nous a proposé le système XINU. Ce système convenait plus ou moins bien comme support de notre application Il fut implémenté pour un micro-processeur LSI 11 et notre application devant tourner sur un Olivetti M 24 avec micro-processeur INTEL 8086, il était nécessaire de porter XINU sur cette machine. Ce système arrivant un peu tard dans l'année académique, il ne nous fut pas possible de réaliser cette opération.

3.4.2.1 Gestion des processus.

A. Scheduling et context switching.

Dans cette partie, le lecteur trouvera une présentation des mécanismes de Xinu mis en oeuvre pour la réalisation du scheduling et du context switching.
Il trouvera en annexe le détail des procédures utilisées à cette fin.

A.1 La table des processus.

Le système garde toutes les informations concernant les processus dans la table des processus. Il y a une entrée par processus.
La table des processus de Xinu est un tableau de NPROC entrées. Chaque processus est identifié par un entier. La règle suivante donne la relation entre ces nombres et la table des processus:

> Les processus sont référencés par leur identificateur, lequel est l'index de l'état d'information sauvé dans la table des processus.

Les informations sauvées sont:

- état du processus,
- priorité du processus,
- registres de la machine,
- sémaphore si processus est en attente,
- message envoyé au processus,
- informations relatives à la gestion du stack,
- nom du processus,
- nombre initial d'arguments,
- adresse initiale du code.

A.2 L'état des processus.

Le système utilise le champ **pstate** de la table des processus pour garder trace de se que fait un processus. Cela lui permet, conséquemment, d'opérer un contrôle sémantique des opérations réalisées sur ce processus. Dans Xinu, on dénombre 6 états: current, ready, receiving, sleeping, suspended, et waiting.

A.3 Sélection d'un processus prêt.

Dans Xinu, la politique de sélection est la suivante:

> A chaque instant, le processus de priorité la plus haute éligible au service du CPU est exécuté. Parmi les processus d'égales priorité, le scheduling opère en round-robin.

Par round-robin nous entendons que les processus sont sélectés l'un aprés l'autre pendant un quota de temps déterminé ; aprés leur exécution ils sont replacés à la fin de la file d'attente.
Les priorités sont des entiers que l'utilisateur de Xinu doit attribué à ses processus.
Pour rendre la sélection d'un nouveau processus plus rapide, tous les processus prêts sont rangés dans une liste ordonnée selon la priorité de telle sorte que le processus de plus haute priorité soit pris directement.

B. La suspension et la réactivation des processus.

Il peut être trés utile de pouvoir suspendre un processus dans un système temps réel. Cela nous amène à considérer un nouvel état pour les processus: l'état suspendu. Deux procédures sont utilisées: resume et suspend.

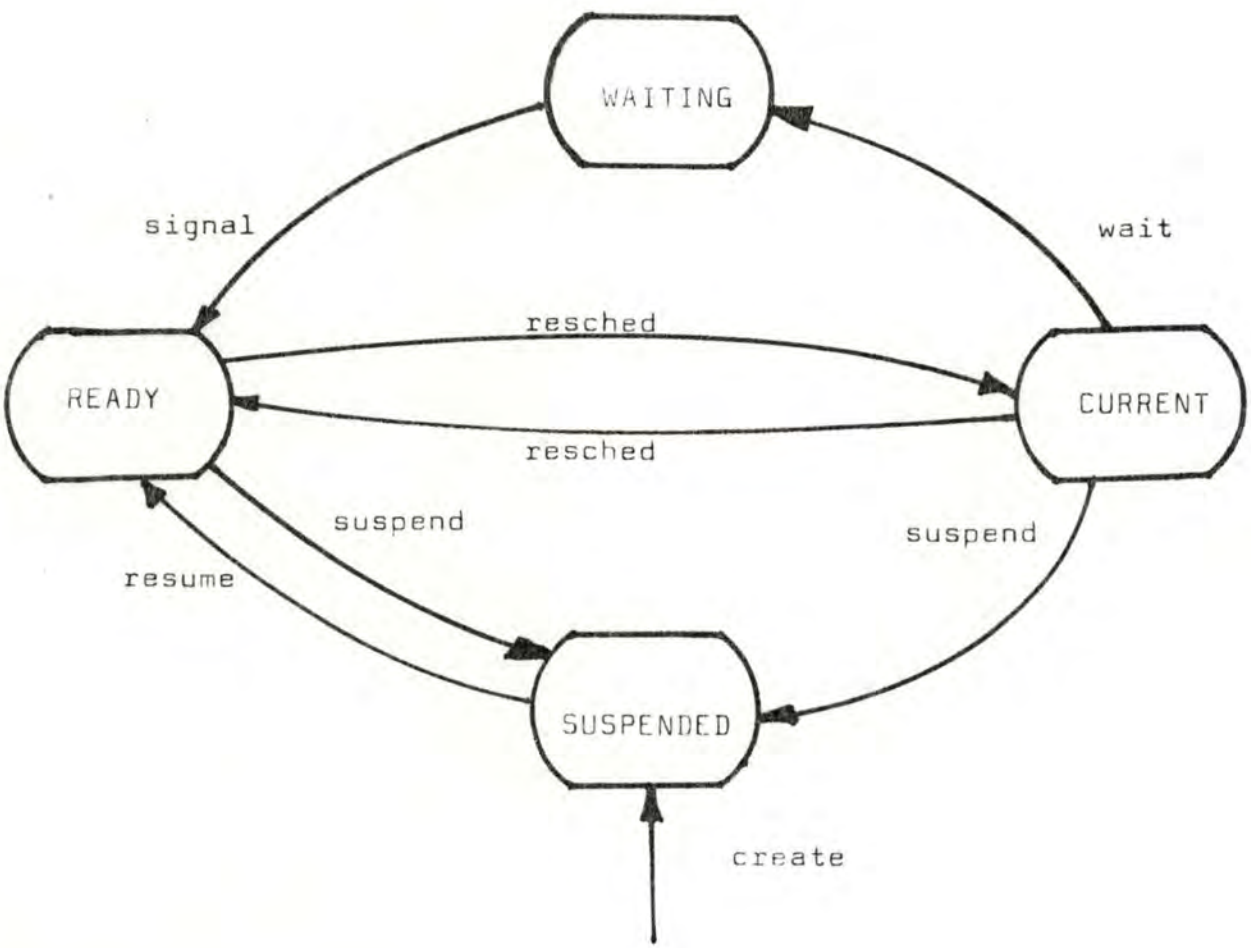
C. La création et la terminaison des processus.

La création est réalisée au moyen de la procédure **create** qui va cherché une entrée libre dans la table des processus, alloué un espace pour le stack du nouveau processus.
La terminaison se fait par l'appel à la procédure **kill** .

D. La coordination entre les processus.

Cette coordination est réalisée par l'utilisation de sémaphores. Quand un processus veut se mettre en attente, le système le place dans un liste de processus associée au sémaphore qui permet la mise en attente.
Les procédures associées à la manipulation des sémaphores sont **wait** et **signal**.
Dans Xinu, les sémaphores sont organisés dans une table dont chaque entrée contient le chiffre de comptage. Chaque sémaphore est identifié par son index dans la table appelée **semaph**.

E. Diagramme des états des processus dans Xinu.



3.4.2.2 Gérant de la Mémoire

Le système d'exploitation XINU gère la mémoire d'une manière très simple. Deux types de blocs mémoire sont considérés : les blocs simples et les blocs de stack.

Les **blocs de stack** sont d'une utilité interne au système d'exploitation XINU et sortent de notre propos.

Les **blocs de mémoire simple** sont, par contre, à la base de la gestion mémoire du système d'exploitation en temps réel. Néanmoins, XINU gère ces blocs indépendemment de la définition des quatre types de blocs étudiés au point 3.3.2.2 : le bloc de message, le bloc de contexte, le bloc contrôleur de tâche et le bloc contrôleur de temporisation. On ne trouve donc dans XINU que deux primitives, GETMEM et FREEMEM, allouant et désallouant un nombre de bytes donné.

Il faut remarquer que le bloc de contexte manipulé par le système temps réel, comme défini au point 3.3.2.2.B)b. perd de son utilité. En effet, XINU gère lui-même le sauvetage des registres. Seul donc le premier byte contenant le code de la primitive sera maintenu.

L'interface entre ces deux primitives de XINU et les primitives du système temps réel consiste à transformer les demandes d'allocation et de désallocation d'un type particulier de bloc mémoire en une allocation et/ou désallocation d'un nombre de bytes correspondant au type de bloc référencé.

Alors que le système temps réel prévoyait un partitionnement de la mémoire en quatre parties correspondant chacune à un type de bloc, ces quatre parties étant gérées sous la forme de files chaînées, nous adoptons pour l'implémentation une vue uniforme de la mémoire, vue que l'on retrouve dans XINU.

Les procédures de traitement des files de blocs de mémoire libre tombent en désuétude : OTE-BLC-QUE, MET-BLC-DEB-QUE et MET-BLC-FIN-QUE.

De plus, GETMEM est une fonction qui retourne l'adresse de début du bloc qui sera le pointeur prévu dans le système temps réel.

. OBT-BLC-MSG (pter-tâche)

méthode : pter-tâche := GETMEM (64)

. RET-BLC-MSG (pter-tâche)

méthode : FREEMEM (pter-tâche,64)

. OBT-BLC-CTX (pter-tâche)

méthode : pter-tâche := GETMEM (2)

. RET-BLC-CTX (pter-tâche)

méthode : FREEMEM (pter-tâche,2)

. OBT-BLC-CTL-TCH (pter-tâche)

méthode : pter-tâche := GETMEM (12)

. RET-BLC-CTL-TCH (pter-tâche)

méthode : FREEMEM (pter-tâche,12)

. OBT-BLC-CTL-TMP (pter-tâche)

méthode : pter-tâche := GETMEM (8)

. RET-BLC-CTL-TMP (pter-tâche)

méthode : FREEMEM (pter-tâche,8)

3.4.2.3. Gestion des messages.

1) Principe de la communication dans Xinu.

Xinu offre deux systèmes de communication entre processus : un de bas niveau et un de haut niveau.

Le système de bas niveau permet d'envoyer directement un message, mais avec l'inconvénient qu'il écrase tout message déjà présent.

Le système de haut niveau utilise un pool tel que décrit plus haut. Ce système convient mieux au prototype COLO/S pour plusieurs raisons : il permet une gestion de priorités de messages et il n'écrase pas un message déjà présent.

Le pool est en fait un ensemble de "ports", chacun étant assigné à un processus particulier. Les messages s'enchaînent à leur port destinataire.

2) Primitives d'envoi et de réception.

2.a) PSEND.

Paramètres :

PORTID : identifiant du port

MSG : pointeur vers le bloc de message

2.b) PRECEIVE.

Paramètre :

PORTID : identifiant du port

3.4.2.4 Gérant du Temps

L'interruption hardware de base a lieu 60 fois par seconde c'est-à-dire plus ou moins toutes les 16,5 msec. Deux problèmes essentiels doivent être réglés

(1) lors de l'implémentation, adapter les cycles de scanning pour l'application en fonction d'un cycle de base de 16,5 msec contre 10 msec auparavant;

(2) modifier la primitive SLEEP10 offerte par XINU qui ne permet comme unité de temporisation que le dixième de seconde.

A) Cycles de Scanning

On trouve les cycles de scanning prévus lors de la conception au point 3.3.3.1.A. Ces temps, allant de 10 à 300 msec, sont peu représentatifs en valeur absolue mais dénotent, en valeur relative, les priorités des différents scannings.

Il s'agit donc, avec une fréquence relative de base de 16,5 msec, d'élaborer de nouvelles périodes, en maintenant les relations de priorité. La liste qui suit donne pour chaque procédure de scanning, sa fréquence d'exécution lors de l'implémentation.

- scanning LIBRE : 165 msec
- scanning de NUMEROTATION : 16,5 msec
- scanning de PRE-NUMEROTATION : 16,5 msec
- scanning de POST-NUMEROTATION : 16,5 msec
- scanning de CONVERSATION : 495 msec
- scanning de PRE-CONVERSATION : 495 msec
- scanning de PARKING : 165 msec

B) Modification de la Primitive SLEEP10

Le corps de la procédure SLEEP10 gère le contrôle d'erreurs quant au paramètre, à savoir le nombre de secondes, et endors le processus demandeur.

Lors de l'interruption hardware de l'horloge, ce n'est pas SLEEP10 qui est exécuté mais bien une routine assembleur CLKINT qui, appelle SLEEP10 toutes les 6 interruptions. Il suffit dès lors de supprimer ce décompte de sorte que CLKINT, qui est en fait une routine de service d'interruption, puisse, après avoir gérer sa file de timers et réveiller les processus en fonction des besoins, appeler SLEEP10, non plus tous les dixièmes de seconde, mais bien tous les soixantièmes de seconde.

3.4.3.1 I.L.S (Interface Lignes/Système)

A) Introduction

La conception du module ILS a été dirigée par les données (Data Driven). Nous avons défini deux structures de données logiques et indépendantes, BCL et SCAN (figure T43), pour lesquelles nous avons défini des opérations. Les seuls accès autorisés aux structures de données sont ceux définis à l'intérieur des opérations. Nous avons ainsi construit des structures connues sous le vocable "types abstraits".

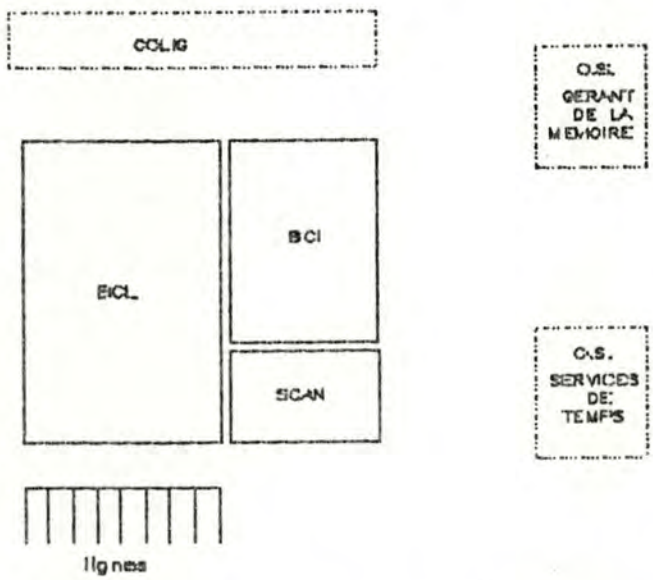


figure T43 : structures BCL, SCAN et BCI

BCL = Bloc de Contrôle de Lignes
BCI = Bloc de Contrôle Interne
SCAN = Bloc de Contrôle de Scanning

Le premier type de données pour lequel nous avons spécifié un type abstrait concerne les fréquences des différentes procédures de scanning, une procédure de scanning correspondant, comme on le sait déjà, à une étape de la conversation. Sont concentrées dans ce module SCAN les relations avec la partie du système d'exploitation offrant les services de temps.

Le second type de données, beaucoup plus complexe, regroupe toutes les données contrôlant les lignes elles-mêmes et reprend toutes les relations directes avec le module Contrôleur de Lignes COLIG. Les relations avec COLIG nécessitent les services du Gérant des Echanges. Pour le bon fonctionnement de ce module BCL, une structure de données BCI complètement interne à ILS est définie. Cette structure n'intervient d'ailleurs que dans l'opération de scanning.

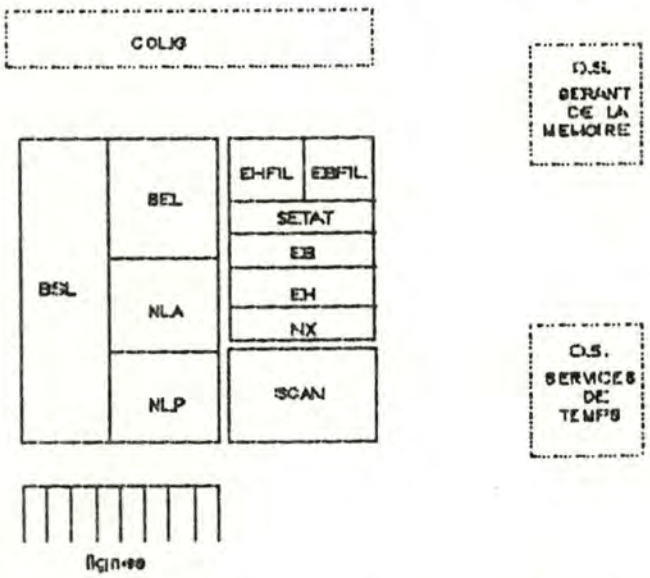


figure T44 : sous-structures

SCAN : bloc des états scannés
SETAT : bloc des lignes référencées
EHFIL, EBFIL : lignes référencées ayant subi des événements hauts ou bas
BSL : bloc des sonneries
BEL : bloc des états
NLA : niveaux de ligne antécédents
NLP : niveaux de lignes présents
NX : bloc des événements
EH : bloc des événements hauts
EB : bloc des événements bas

La structure de données BCL regroupe en fait plusieurs sous-structures tendant chacune au contrôle des lignes mais pour des caractéristiques différentes de ces dernières (figure T44). La structure BSL renferme les informations concernant les différentes sonneries et tonalités pouvant agir sur une ligne. BSL est modifié par deux procédures METSON et RETSON directement accessibles par COLIG via interruptions, ayant ainsi le statut de primitive ILS (figure T45). L'interface physique des lignes

accède directement à cette structure pour positionner les tonalités selon le déroulement de l'appel téléphonique.

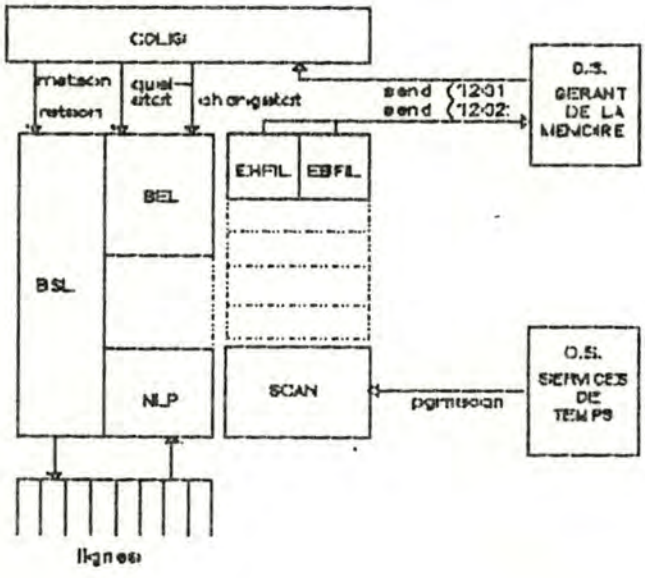


figure T45 : interfaces

L'information véhiculée sur les lignes est purement binaire et correspond à deux niveaux en logique positive, haut et bas, que l'interface physique vient positionner dans un vecteur NLP (Niveaux de Lignes Présents) (figure T44). Il est nécessaire, pour pouvoir déceler un changement de niveau sur une ligne, de conserver les niveaux de lignes enregistrés au préalable pour pouvoir les comparer à ceux du vecteur NLP. Les Niveaux de Lignes Antécédents sont mémorisés dans un vecteur NLA.

Comme on le sait, l'avancement d'une conversation téléphonique correspond à des activités différentes sur une ligne. Ces activités diffèrent de par leur fréquence, leur signification, leur traitement et permettent de définir ainsi des états logiques pour les lignes. Pour chacune d'entre elles, son état logique est maintenu dans une matrice BEL (figure T44) à laqquelle n'a accès que COLIG par le biais de deux primitives CHANGETAT et QUELETAT (figure T45).

La structure BCI très particulière (figure T44) est définie fonctionnellement par ses relations avec la structure BCL et autres primitives. La seule opération de scanning peut y avoir accès.

B) module SCAN

La structure SCAN consiste en une table à une dimension d'une longueur égale au nombre d'états possibles dans le système c'est-à-dire 7. Les fréquences de ces états sont rangées par ordre croissant dans la table pour une facilité de programmation.

$$0 \leq f_1 \leq f_2 \leq \ldots \leq f_n$$

Une fréquence relative de base MIN est définie de la manière suivante :

$$MIN = (f_i - f_{i-1} \; ; \; f_0 \leq 0 \text{ et } 1 \leq i \leq n)$$

La seule opération ayant accès à la structure SCAN est l'opération de programmation de scanning PGMSCAN. Cette opération est en fait un processus du système qui se réveille et est exécuté chaque fois qu'une période de temps correspondant à la fréquence relative de base est écoulée. Ceci est réalisé grâce à la primitive SLEEP (MIN) du Gestionnaire de Temps. Les procédures de scanning dont le cycle se termine à ce moment sont déterminées et activées et cela, indépendemment de leur nombre, de leur fréquence et de la fréquence d'horloge. L'activation prendra forme une fois que l'information sera passée par vecteur à une procédure SCANNING du module BCL (figure T46).

C) module BCL

La structure BCL consiste en une matrice de vecteurs d'une longueur égale au nombre de lignes dans le système. Ces vecteurs sont regroupés en quatre sous-matrices distinctes :

- NLP : un seul vecteur qui informe au moyen d'un bit par ligne du niveau de la ligne lors de l'exploration en cours;

- NLA : un seul vecteur qui informe au moyen d'un bit par ligne du niveau de la ligne lors de l'exploration précédente;

- BSL : un certain nombre de vecteurs égal au nombre de sonneries/tonalités possibles dans le système. Chaque ligne ou vecteur de cette sous-matrice représentant une sonnerie/tonalité a le nième bit à 1 si son appareillage hardware est

connecté à la nième ligne. Une colonne de cette sous-matrice représentant une ligne peut soit ne contenir que des bits à 0, soit en contenir un seul à 1 et tous les autres à 0;

- BEL : un certain nombre de vecteurs égal au nombre d'états possibles pour une ligne dans le système. Chaque ligne ou vecteur de cette sous-matrice représentant un état de ligne a le nième bit à 1 si la nième ligne se trouve dans son état. Une colonne de cette sous-matrice représentant une ligne devra toujours contenir un et un seul bit à 1 et tous les autres à 0.

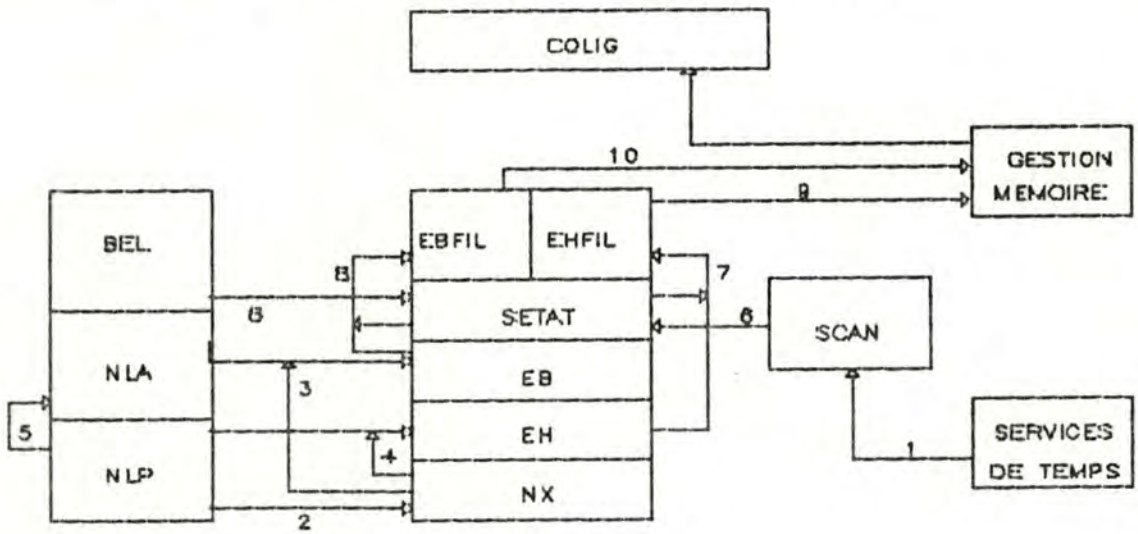


figure T46 : scanning

1. pgmscan
2. exclor (nla,nlp,nx)
3. conjonc (nlp,nx,eh)
4. conjonc (nlp,nx,eb)
5. egal (nlp,nla)
6. grouplg (scan,setat)
7. conjonc (setat,eh,ehfil)
8. conjonc (setat,eb,ebfil)
9. send (1201)
10. send (1202)

Le module BCL comprend trois types d'opérations :

- celles relatives au scanning;
- celles relatives aux accès BSL;
- celles relatives aux accès BEL.

a.- Scanning

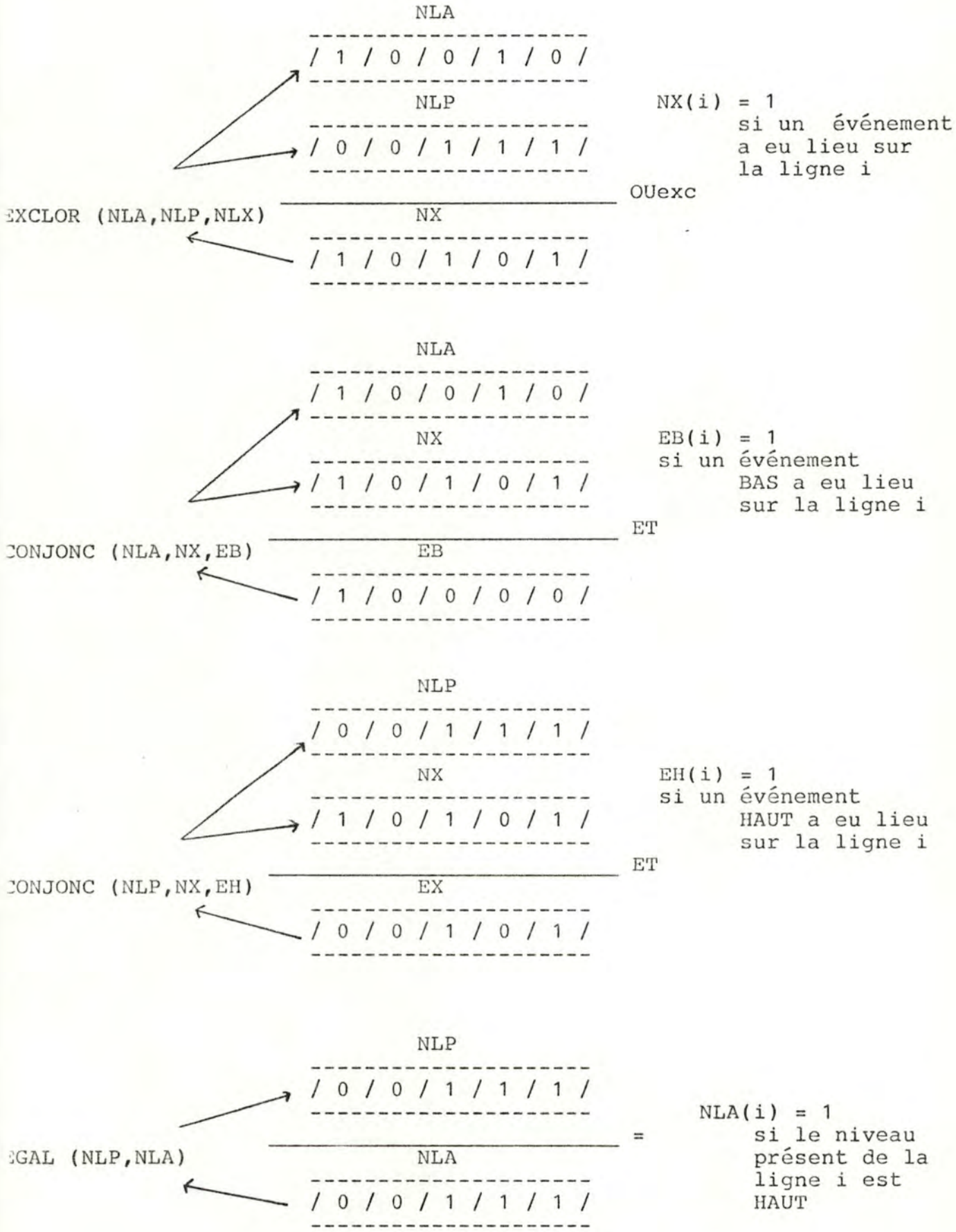
Ces opérations explorent toutes les lignes dans les états (0, 1 ou plusieurs) référencés par un vecteur en provenance du module SCAN. La procédure envoie au module COLIG un message 1201 pour chaque événement haut détecté sur une ligne et un message 1202 pour chaque événement bas détecté sur une ligne. Cet envoi de mesages est rendu possible par les appels suivants de primitives du système d'exploitation : GETMEM et PSEND.

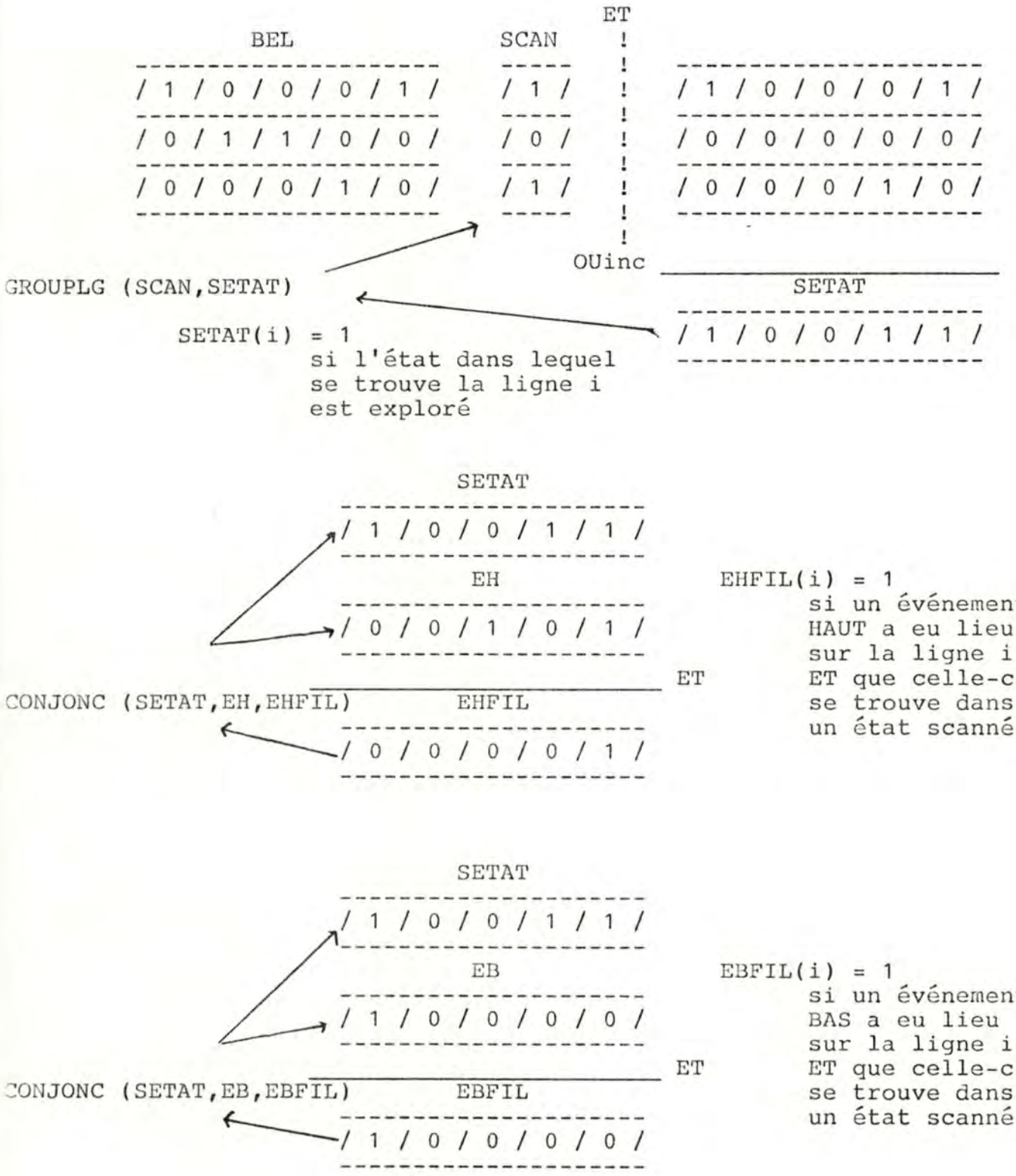
L'opération complète décrite via un exemple au paragraphe suivant, utilise de nombreuses procédures (EXCLOR, CONJONC, EGAL) permettant des manipulations combinatoires sur des vecteurs ayant un nombre de bits égal au nombre de lignes dans le système. Il existe également une procédure qui regroupe les lignes se trouvant dans un des états scannés (GROUPLG). Finalement, l'envoi de messages mentionné plus haut, sera fonction de la valeur de certains bits (VALBIT).

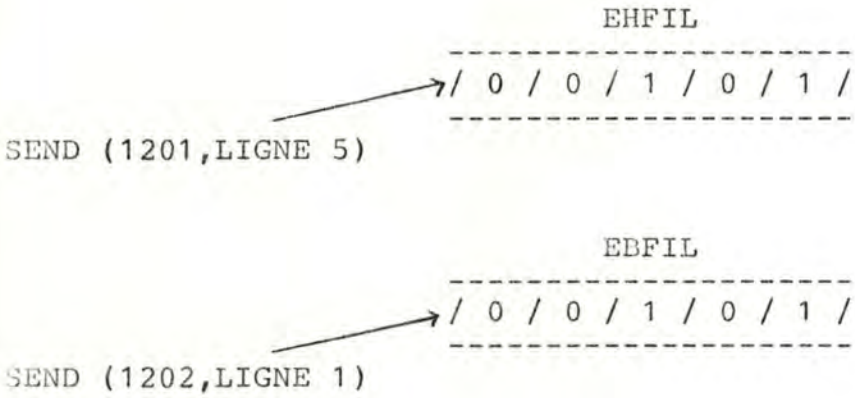
Voici à présent un exemple basé sur un système comprenant 5 lignes pouvant se trouver dans 3 états possibles. La situation initiale est la suivante :

la ligne 1 dans l'état 1 ;
la ligne 2 dans l'état 2 ;
la ligne 3 dans l'état 2 ;
la ligne 4 dans l'état 3 ;
la ligne 5 dans l'état 1 ;

Un nouveau cycle d'une fréquence relative de base arrivant à terme, une exploration des lignes se trouvant dans les états 1 et 3 est sollicitée. La séquence des opérations est la suivante (figure T47):


NLA
/ 1 / 0 / 0 / 1 / 0 /
NLP
/ 0 / 0 / 1 / 1 / 1 /
NX(i) = 1
si un événement
a eu lieu sur
la ligne i
OUexc
EXCLOR (NLA,NLP,NLX)
NX
/ 1 / 0 / 1 / 0 / 1 /
NLA
/ 1 / 0 / 0 / 1 / 0 /
NX
/ 1 / 0 / 1 / 0 / 1 /
EB(i) = 1
si un événement
BAS a eu lieu
sur la ligne i
ET
CONJONC (NLA,NX,EB)
EB
/ 1 / 0 / 0 / 0 / 0 /
NLP
/ 0 / 0 / 1 / 1 / 1 /
NX
/ 1 / 0 / 1 / 0 / 1 /
EH(i) = 1
si un événement
HAUT a eu lieu
sur la ligne i
ET
CONJONC (NLP,NX,EH)
EX
/ 0 / 0 / 1 / 0 / 1 /
NLP
/ 0 / 0 / 1 / 1 / 1 /
=
NLA(i) = 1
si le niveau
présent de la
ligne i est
HAUT
EGAL (NLP,NLA)
NLA
/ 0 / 0 / 1 / 1 / 1 /


BEL
SCAN
ET
/ 1 / 0 / 0 / 0 / 1 /
/ 0 / 1 / 1 / 0 / 0 /
/ 0 / 0 / 0 / 1 / 0 /
/ 1 /
/ 0 /
/ 1 /
/ 1 / 0 / 0 / 0 / 1 /
/ 0 / 0 / 0 / 0 / 0 /
/ 0 / 0 / 0 / 1 / 0 /
OUinc
SETAT
/ 1 / 0 / 0 / 1 / 1 /
GROUPLG (SCAN,SETAT)
SETAT(i) = 1
si l'état dans lequel
se trouve la ligne i
est exploré
SETAT
/ 1 / 0 / 0 / 1 / 1 /
EH
/ 0 / 0 / 1 / 0 / 1 /
EHFIL(i) = 1
si un événement
HAUT a eu lieu
sur la ligne i
ET que celle-ci
se trouve dans
un état scanné
ET
CONJONC (SETAT,EH,EHFIL)
EHFIL
/ 0 / 0 / 0 / 0 / 1 /
SETAT
/ 1 / 0 / 0 / 1 / 1 /
EB
/ 1 / 0 / 0 / 0 / 0 /
EBFIL(i) = 1
si un événement
BAS a eu lieu
sur la ligne i
ET que celle-ci
se trouve dans
un état scanné
ET
CONJONC (SETAT,EB,EBFIL)
EBFIL
/ 1 / 0 / 0 / 0 / 0 /



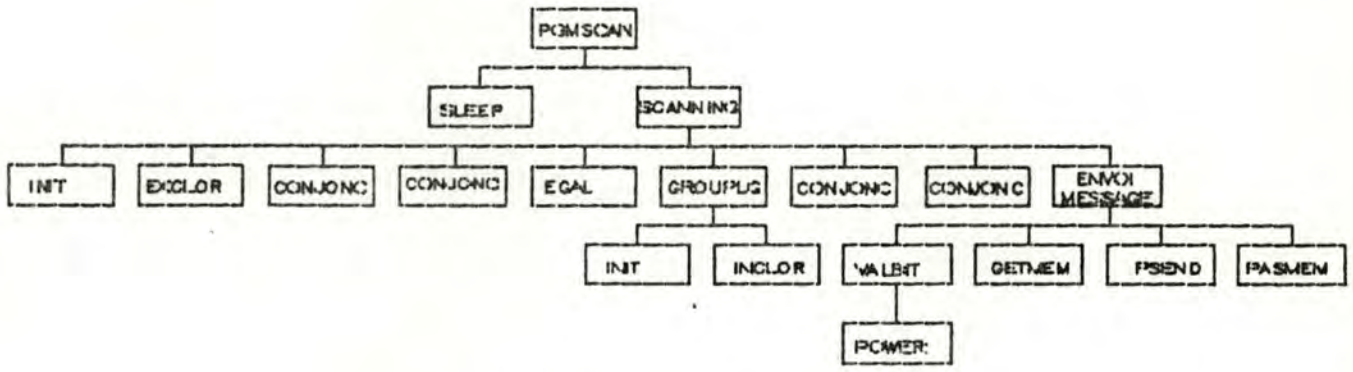


Figure T47 : La fonction SCANNING activée par la fonction PGMSCAN

b.- Accès BEL

L'opération CHANGETAT permet à une ligne de passer d'un état dans un autre. L'opération QUELETAT permet de connaître l'état d'une ligne donnée.

Ces deux procédures accèdent directement aux bits de la matrice BEL pour les modifier (UNBIT, NULBIT) ou simplement en connaître la valeur (VALBIT). CHANGETAT contrôle aussi ces accès (RETETAT) (figure T48).

c.- Accès BSL

La procédure METSON active une sonnerie sur une ligne, tandis que la procédure RETSON réalise l'opération inverse en désactivant une sonnerie sur une ligne.

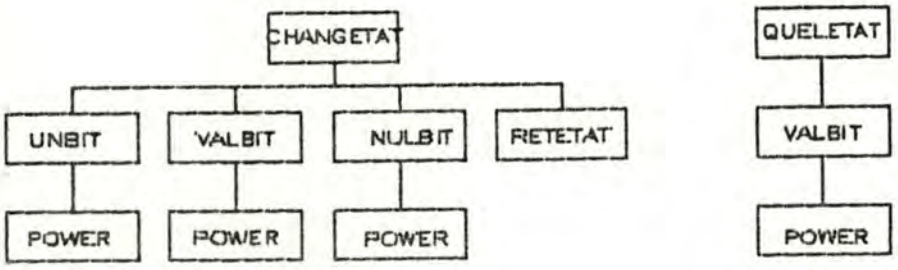


figure T48 : primitives d'acces a BEL

Ces deux procédures positionnent (1 ou 0) les bits de la matrice BSL (UNBIT, NULBIT) (figure T49).

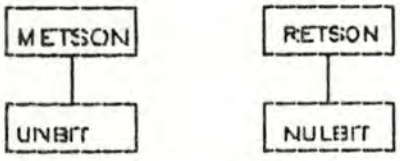


figure T49 : primitives d'acces a BSL

d.- Procédures utilitaires

CONJONC : effectue en ET logique entre deux vecteurs ayant un nombre de bits égal au nombre de lignes dans le système.

INCLOR : effectue en OU inclusif logique entre deux vecteurs ayant un nombre de bits égal au nombre de lignes dans le système.

EXCLOR : effectue en OU exclusif logique entre deux vecteurs ayant un nombre de bits égal au nombre de lignes dans le système.

EGAL : affecte la valeur d'un vecteur ayant un nombre de bits égal au nombre de lignes dans le système à un autre vecteur répondant aux mêmes caractéristiques.

VALBIT : retourne la valeur (0 ou 1) d'un bit dans un entier.

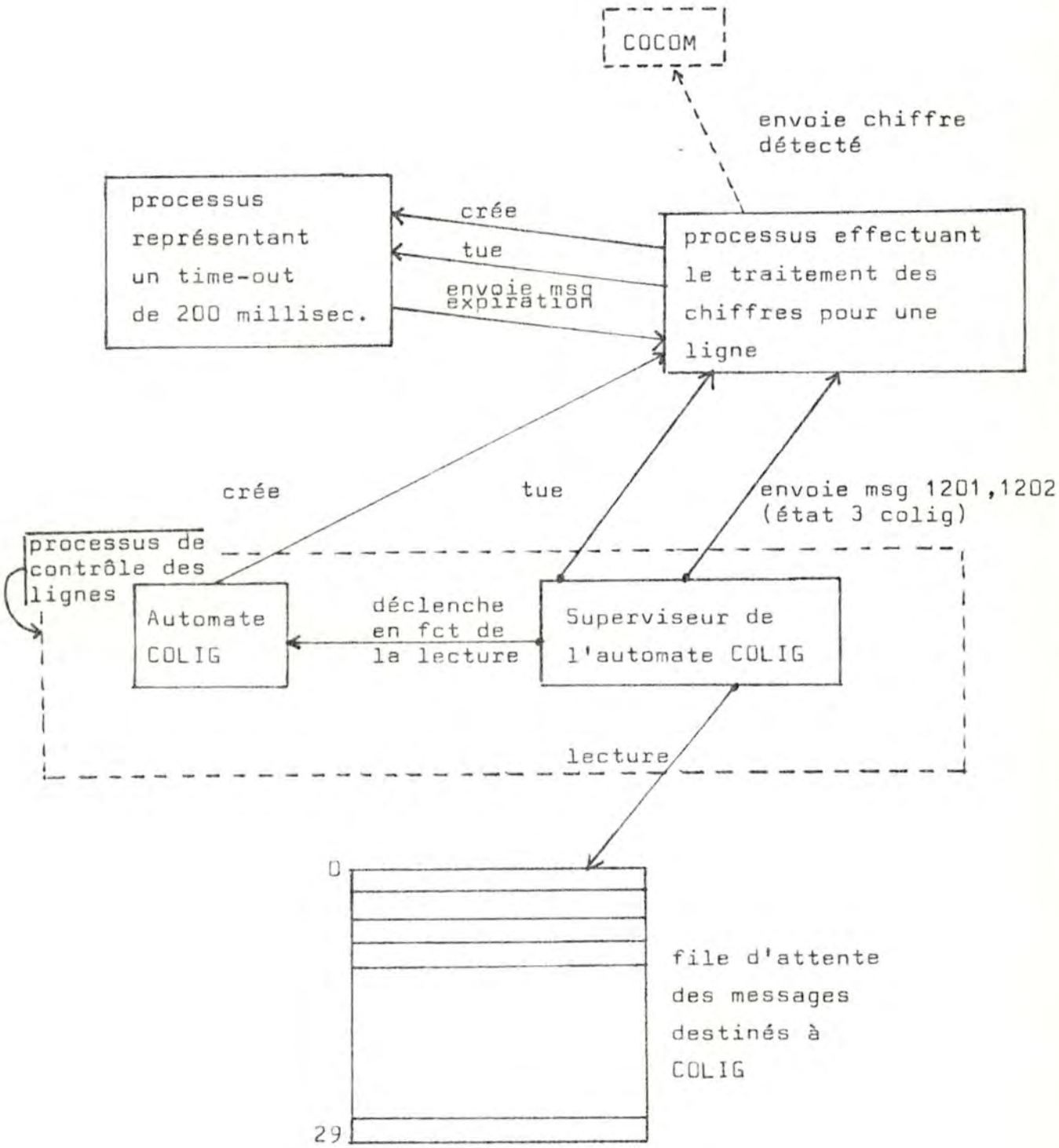
UNBIT : affecte la valeur 1 à un bit dans un entier.

NULBIT : affecte la valeur 0 à un bit dans un entier.

RETETAT : traitement d'une erreur lors d'un changement d'état.

3.4.3.2 COLIG.

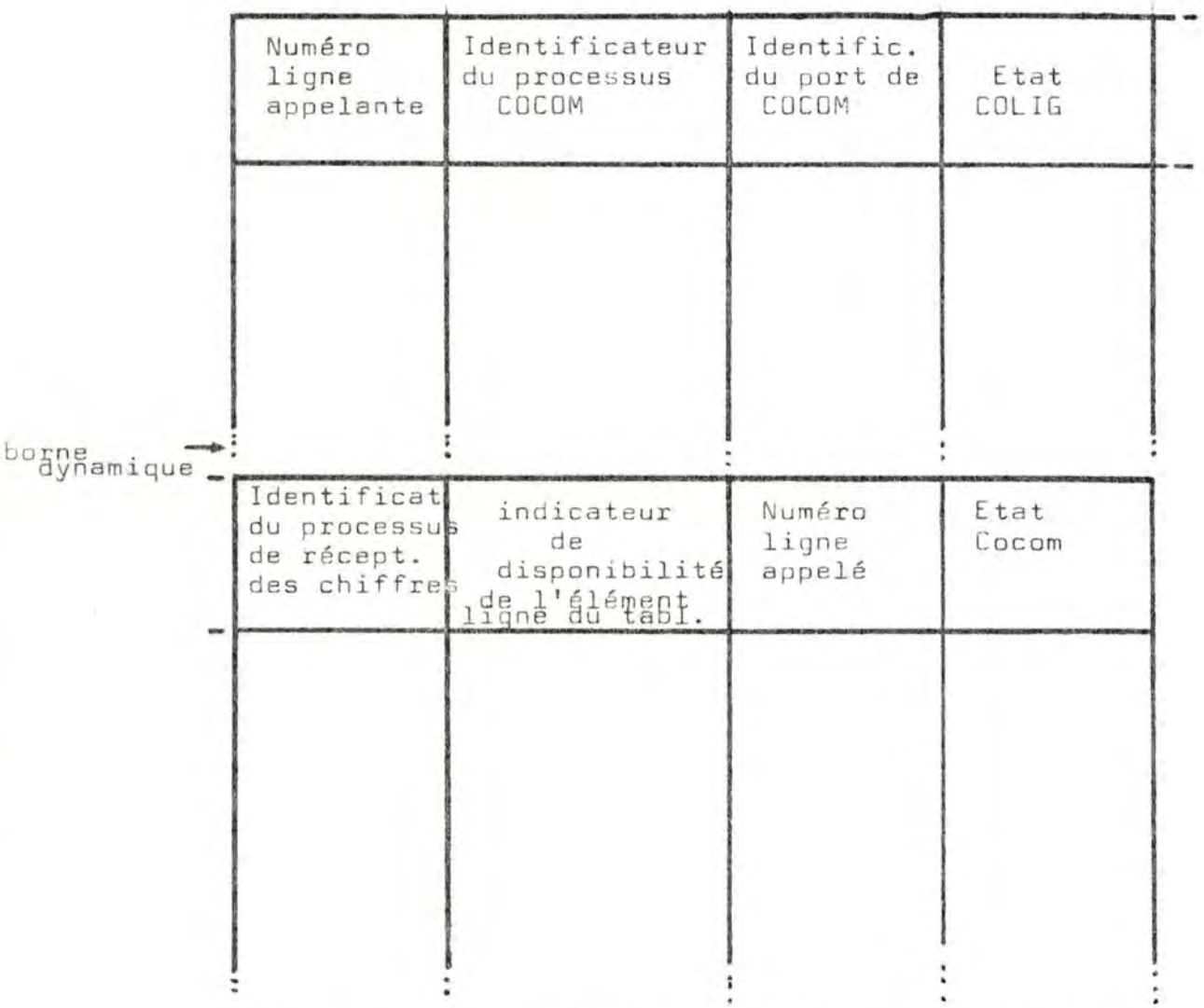
A. Architecture fonctionnelle de l'automate COLIG.



Cette architecture est soutenue par une table qui permet à tout moment de connaître pour une ligne appelante ou appelée:

- l'identificateur du processus COCOM qui gère la communication.
- l'identificateur de la file d'attente des messages destinés à ce processus COCOM.
- l'état de COLIG.
- l'identificateur du processus qui est chargé de la réception des chiffres.
- le numéro de l'appelé.

Cette table est construite avec une borne dynamique ce qui permet de traiter un nombre de lignes non prédéfini .

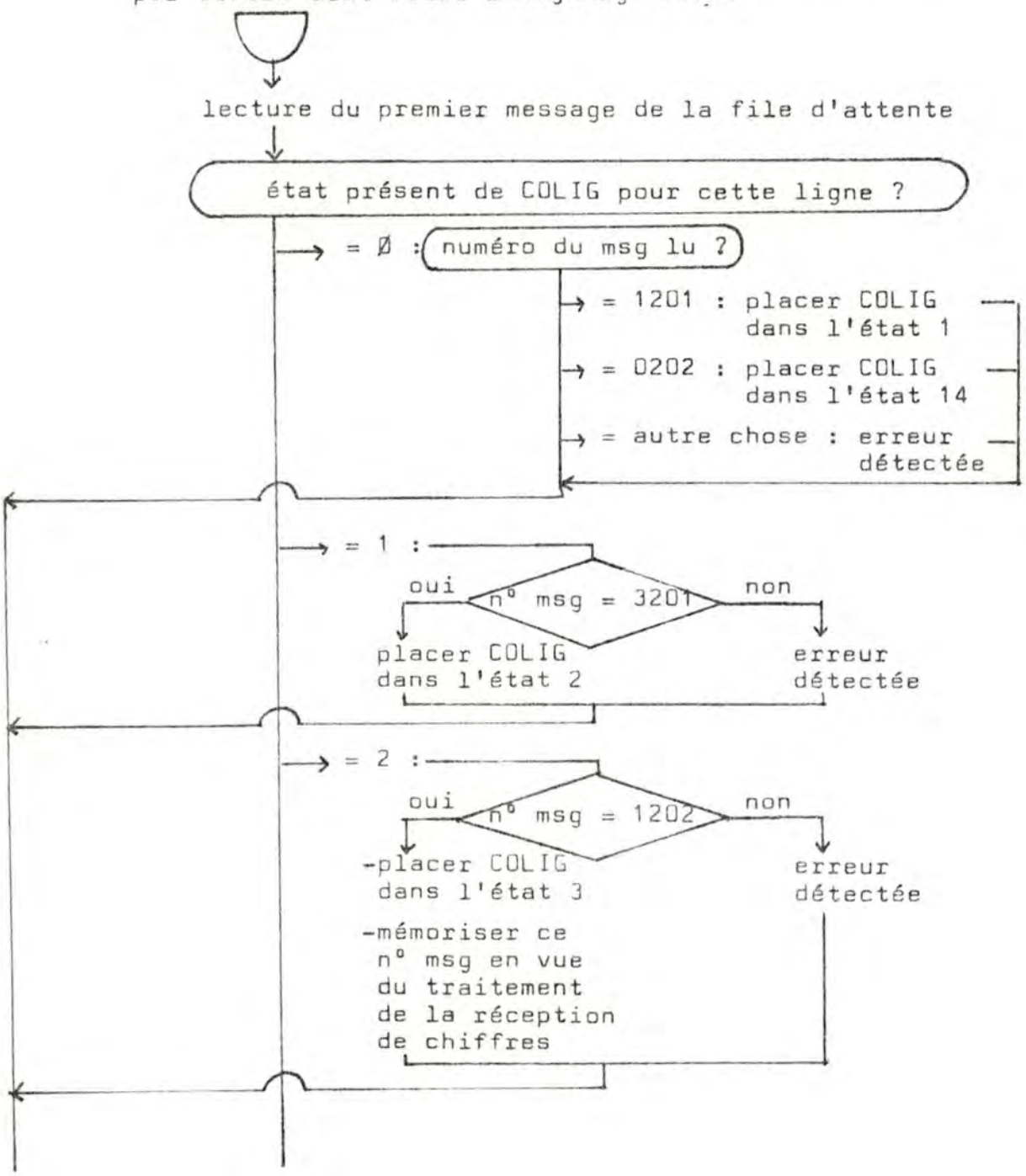


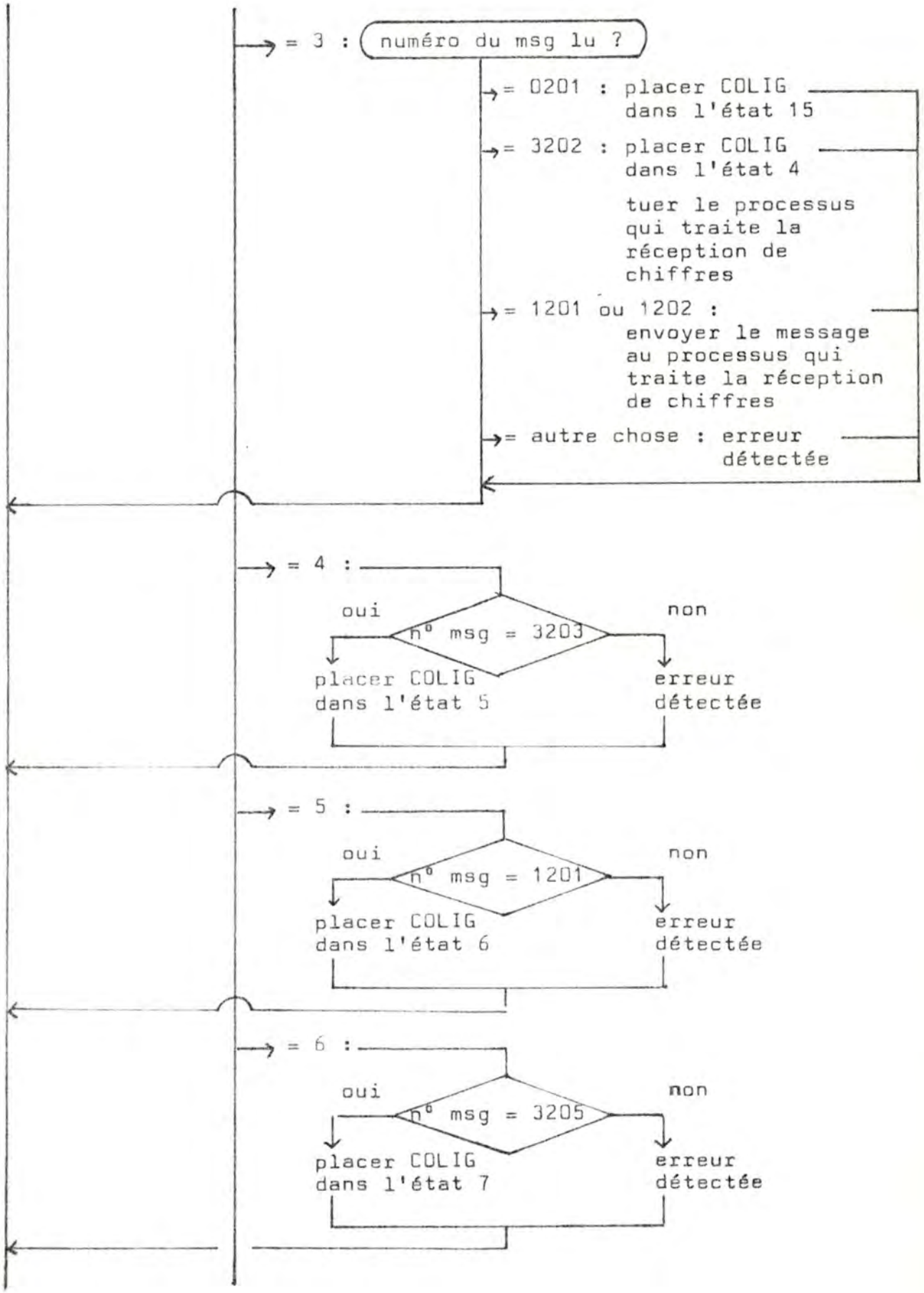
Lorsqu'une communication se termine, les éléments la concernant enregistrés dans la table peuvent être remplacés par ceux qui se rapportent à une communication future. L'indicateur de disponibilité permet de repérer une ligne du tableau contenant de tels éléments.

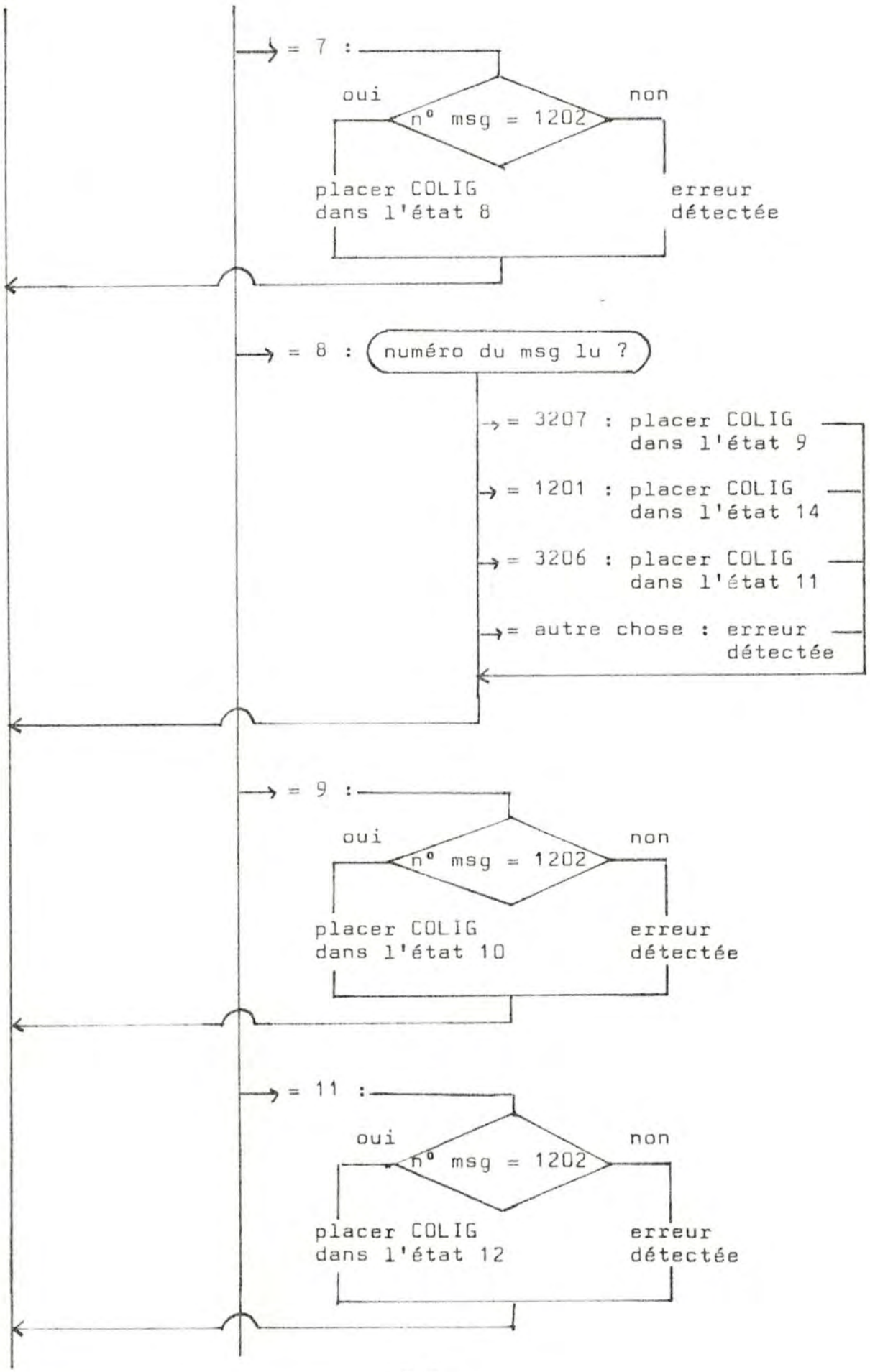
B. Automate COLIG.

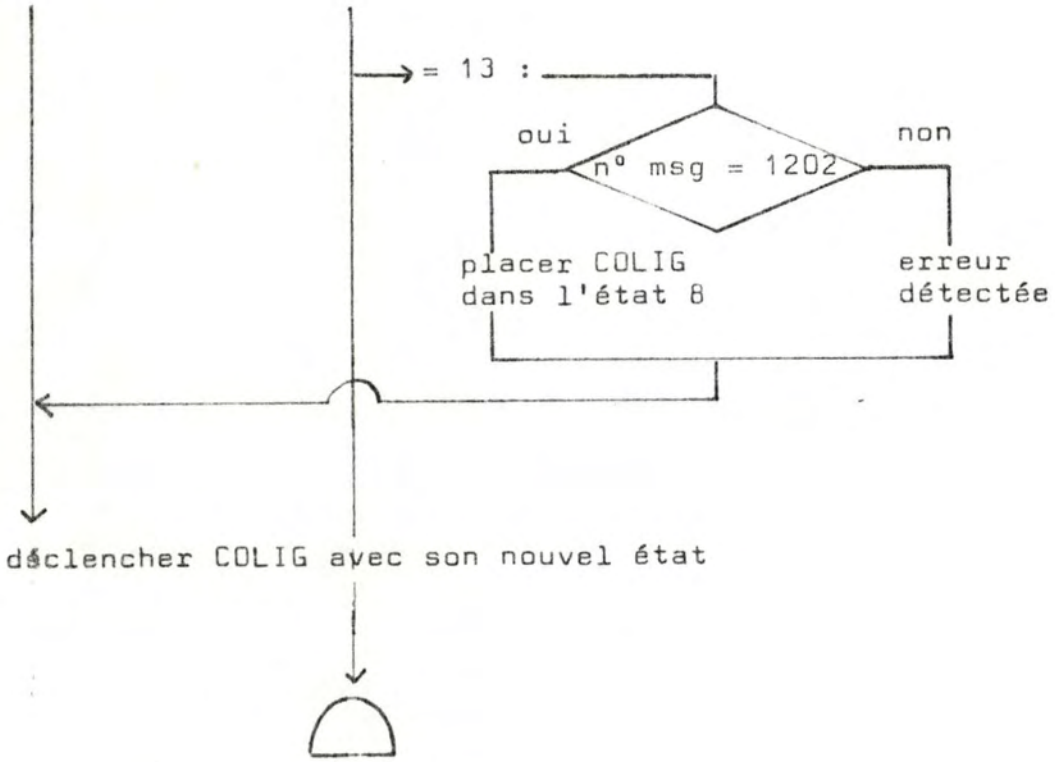
B.1 Supervision de l'automate.

Contrairement à l'automate COCOM, COLIG a besoin d'être supervisé. En effet, la file d'attente de ses messages contient des informations qui se rapportent à différentes communications pour lesquelles COLIG aura des états différents. Ils seront gérés par SUPCLG dont voici l'organigramme :

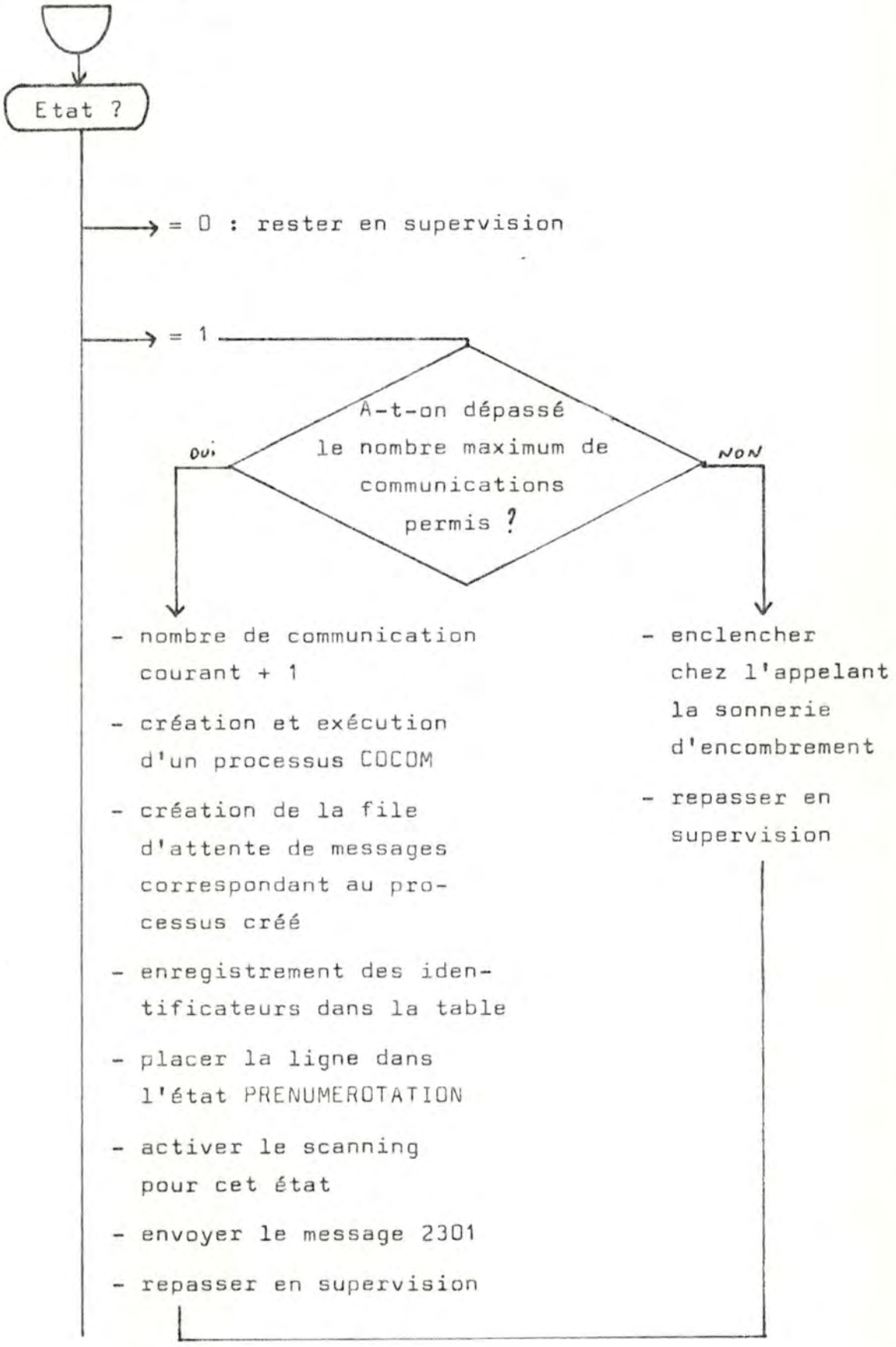


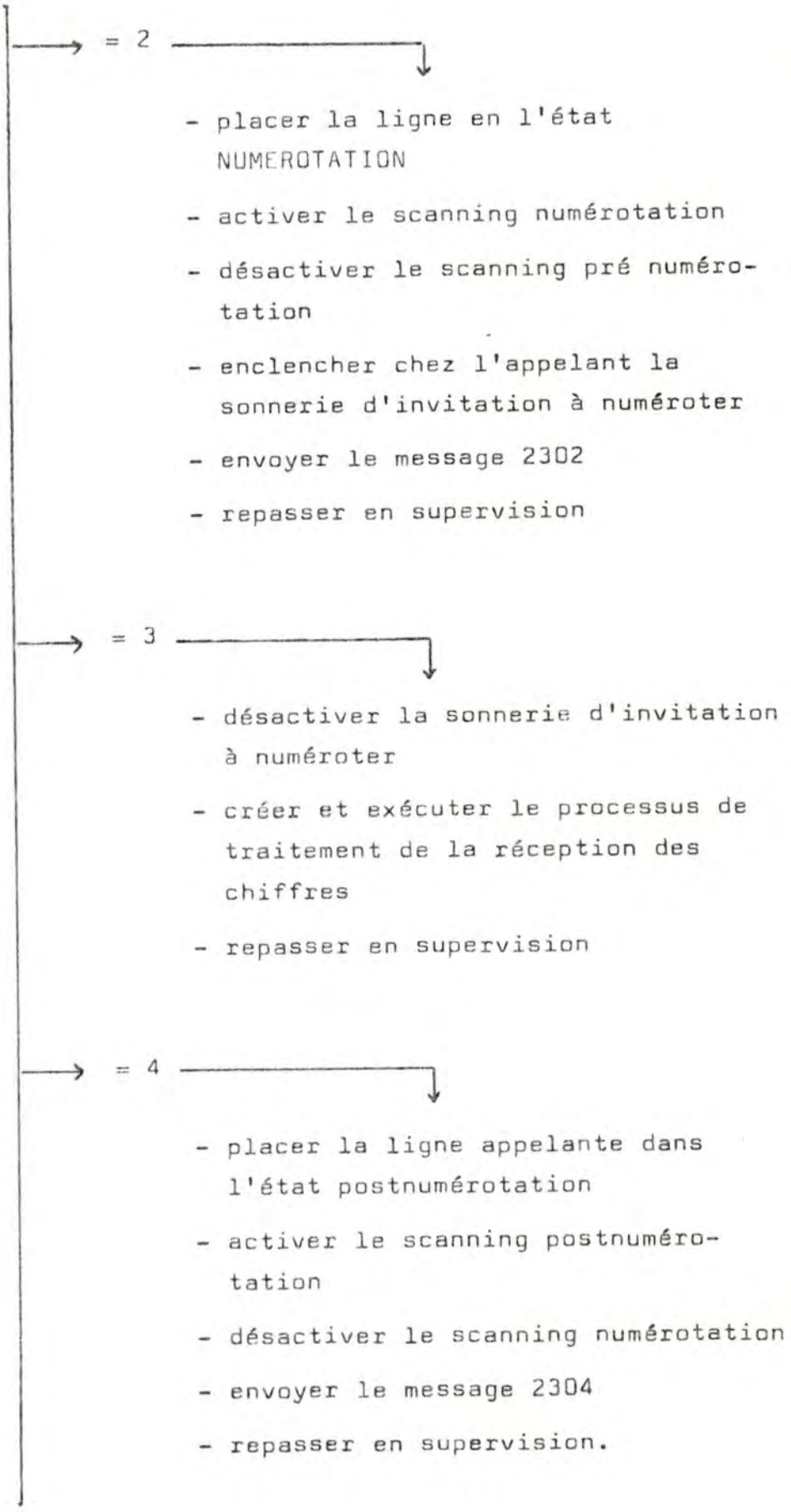

= 3 : numéro du msg lu ?
= 0201 : placer COLIG dans l'état 15
= 3202 : placer COLIG dans l'état 4
tuer le processus qui traite la réception de chiffres
= 1201 ou 1202 : envoyer le message au processus qui traite la réception de chiffres
= autre chose : erreur détectée
= 4 :
oui
n° msg = 3203
non
placer COLIG dans l'état 5
erreur détectée
= 5 :
oui
n° msg = 1201
non
placer COLIG dans l'état 6
erreur détectée
= 6 :
oui
n° msg = 3205
non
placer COLIG dans l'état 7
erreur détectée

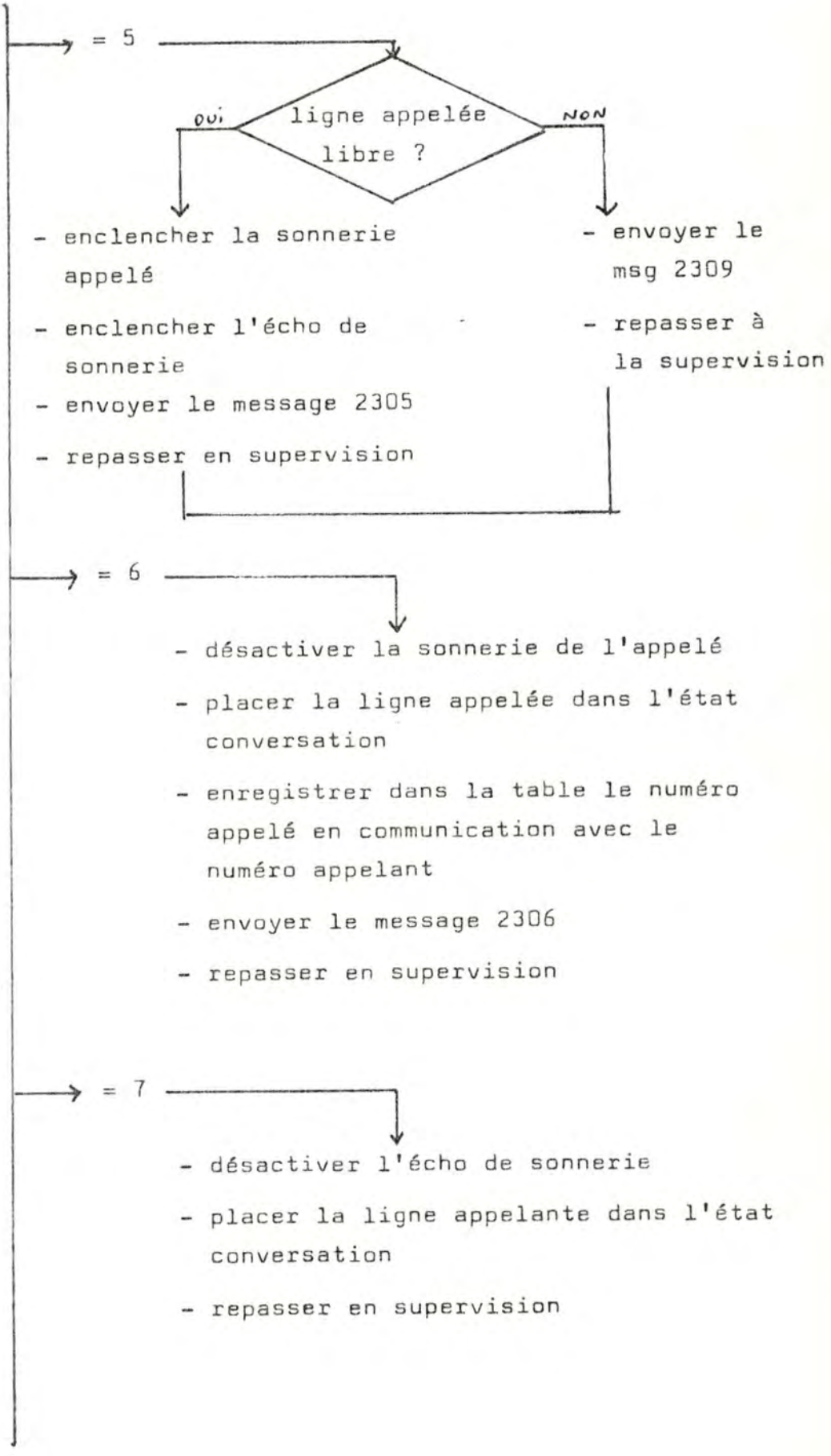

= 7 :
oui
n° msg = 1202
non
placer COLIG
dans l'état 8
erreur
détectée
= 8 :
numéro du msg lu ?
= 3207 : placer COLIG
dans l'état 9
= 1201 : placer COLIG
dans l'état 14
= 3206 : placer COLIG
dans l'état 11
= autre chose : erreur
détectée
= 9 :
oui
n° msg = 1202
non
placer COLIG
dans l'état 10
erreur
détectée
= 11 :
oui
n° msg = 1202
non
placer COLIG
dans l'état 12
erreur
détectée

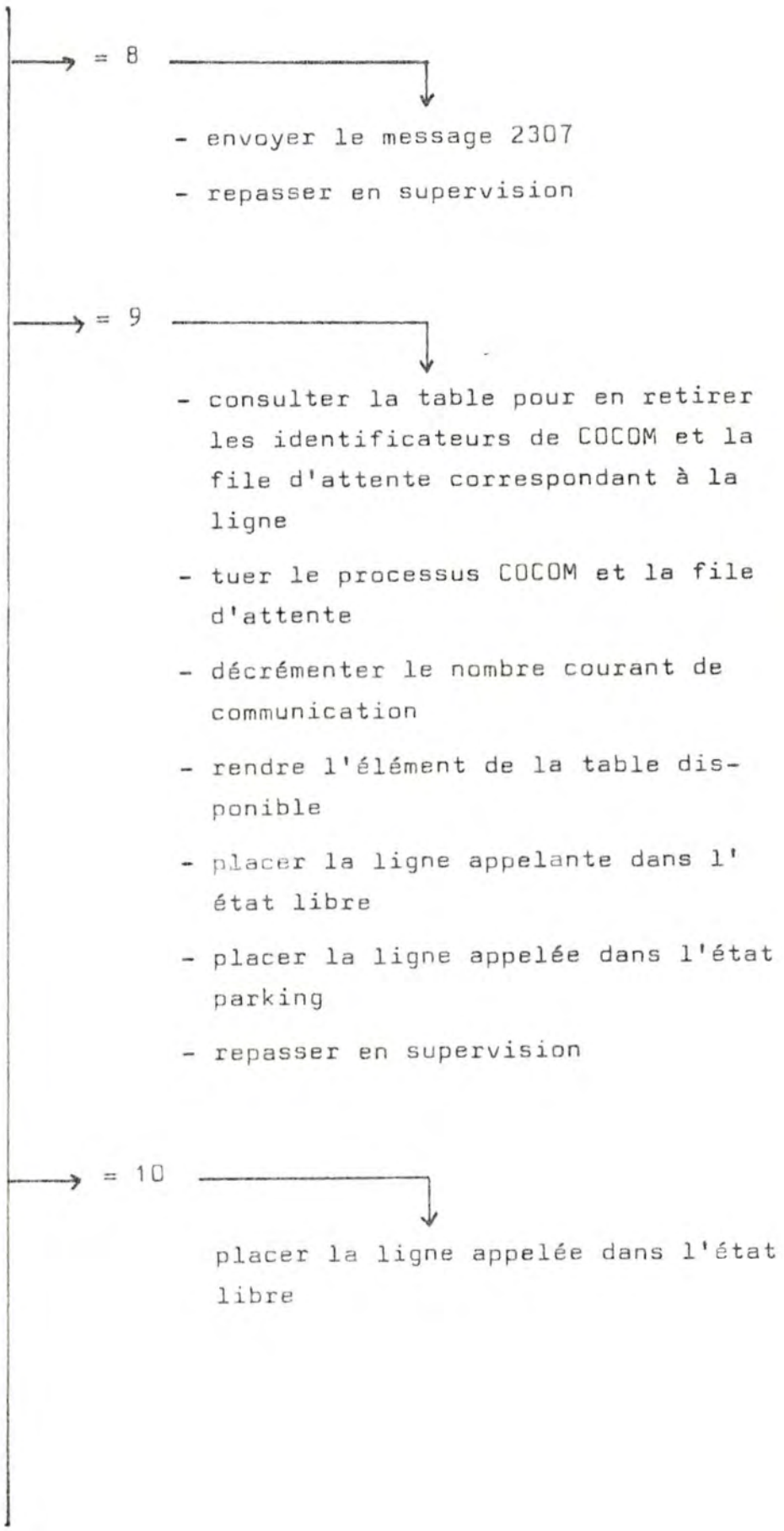

= 13 :
oui
non
n° msg = 1202
placer COLIG
dans l'état 8
erreur
détectée
déclencher COLIG avec son nouvel état

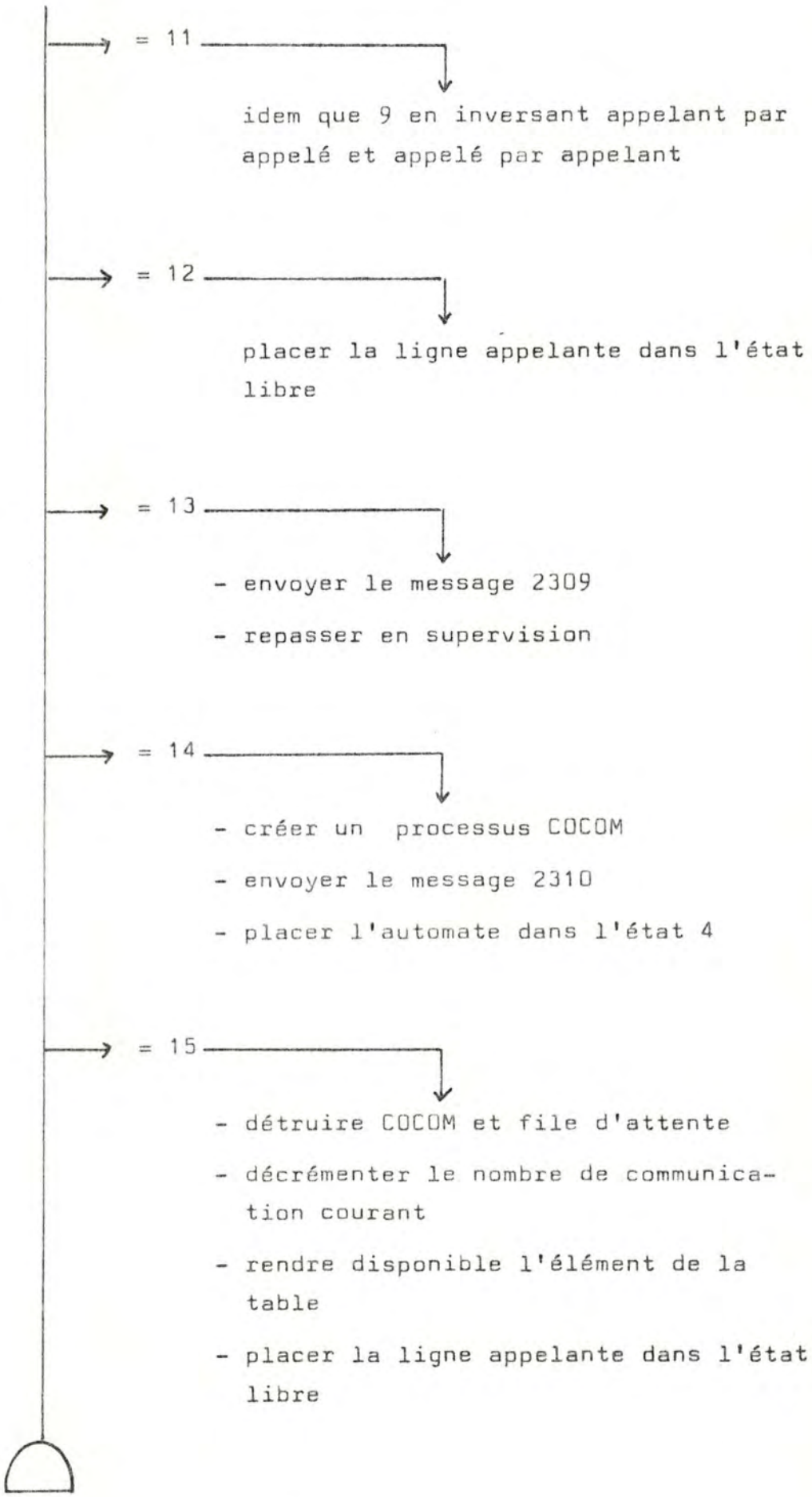
B.2 Automate COLIG.



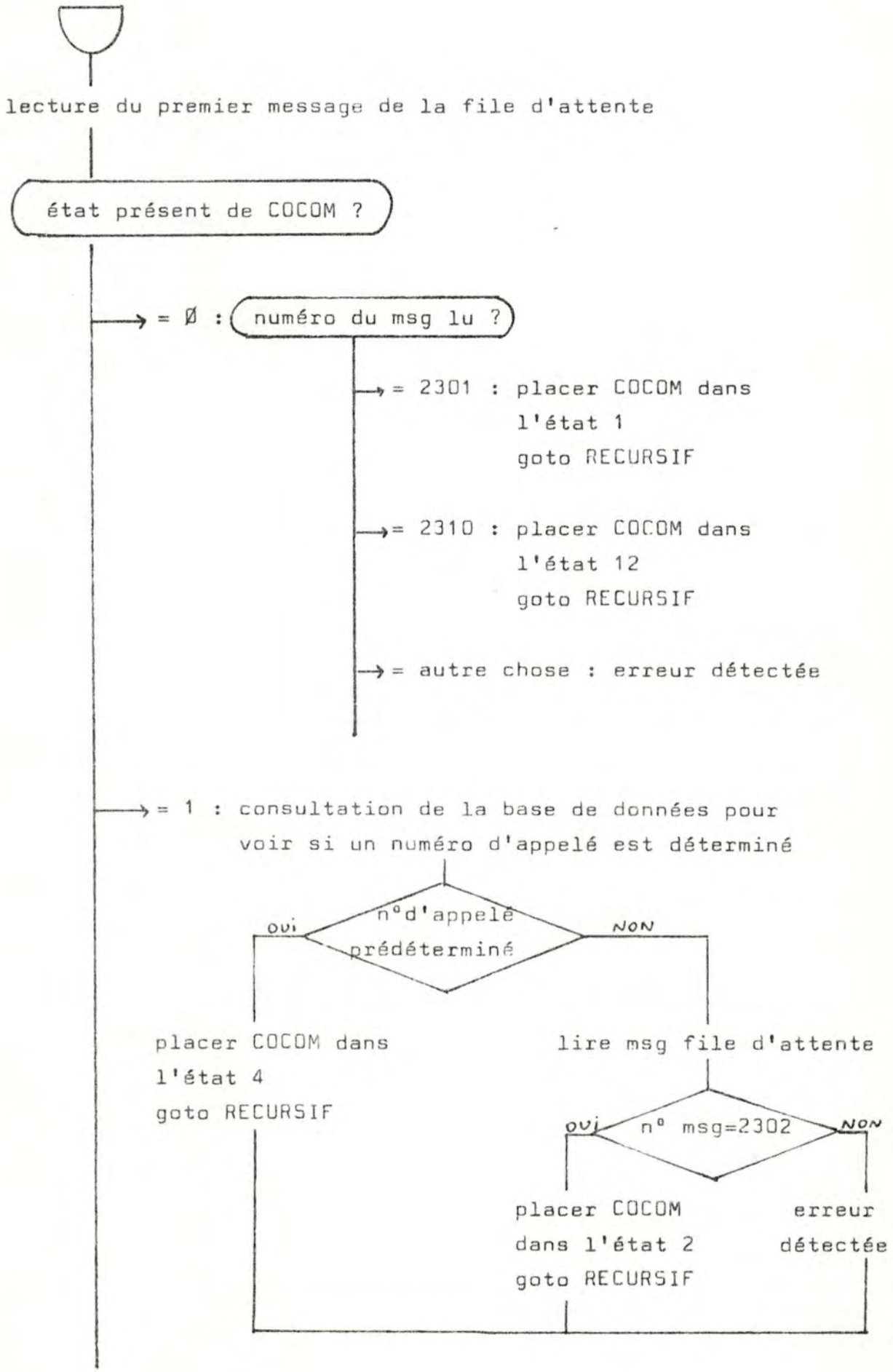

= 2
- placer la ligne en l'état NUMEROTATION
- activer le scanning numérotation
- désactiver le scanning pré numérotation
- enclencher chez l'appelant la sonnerie d'invitation à numéroter
- envoyer le message 2302
- repasser en supervision
= 3
- désactiver la sonnerie d'invitation à numéroter
- créer et exécuter le processus de traitement de la réception des chiffres
- repasser en supervision
= 4
- placer la ligne appelante dans l'état postnumérotation
- activer le scanning postnumérotation
- désactiver le scanning numérotation
- envoyer le message 2304
- repasser en supervision.


= 5
ligne appelée libre ?
OUI
NON
- enclencher la sonnerie appelé
- enclencher l'écho de sonnerie
- envoyer le message 2305
- repasser en supervision
- envoyer le msg 2309
- repasser à la supervision
= 6
- désactiver la sonnerie de l'appelé
- placer la ligne appelée dans l'état conversation
- enregistrer dans la table le numéro appelé en communication avec le numéro appelant
- envoyer le message 2306
- repasser en supervision
= 7
- désactiver l'écho de sonnerie
- placer la ligne appelante dans l'état conversation
- repasser en supervision


= 8
- envoyer le message 2307
- repasser en supervision
= 9
- consulter la table pour en retirer les identificateurs de COCOM et la file d'attente correspondant à la ligne
- tuer le processus COCOM et la file d'attente
- décrémenter le nombre courant de communication
- rendre l'élément de la table disponible
- placer la ligne appelante dans l'état libre
- placer la ligne appelée dans l'état parking
- repasser en supervision
= 10
placer la ligne appelée dans l'état libre


= 11
idem que 9 en inversant appelant par appelé et appelé par appelant
= 12
placer la ligne appelante dans l'état libre
= 13
- envoyer le message 2309
- repasser en supervision
= 14
- créer un processus COCOM
- envoyer le message 2310
- placer l'automate dans l'état 4
= 15
- détruire COCOM et file d'attente
- décrémenter le nombre de communication courant
- rendre disponible l'élément de la table
- placer la ligne appelante dans l'état libre

3.4.3.3 COCOM.


lecture du premier message de la file d'attente
état présent de COCOM ?
= Ø : numéro du msg lu ?
= 2301 : placer COCOM dans l'état 1 goto RECURSIF
= 2310 : placer COCOM dans l'état 12 goto RECURSIF
= autre chose : erreur détectée
= 1 : consultation de la base de données pour voir si un numéro d'appelé est déterminé
n°d'appelé prédéterminé
oui
NON
placer COCOM dans l'état 4 goto RECURSIF
lire msg file d'attente
n° msg=2302
oui
NON
placer COCOM dans l'état 2 goto RECURSIF
erreur détectée

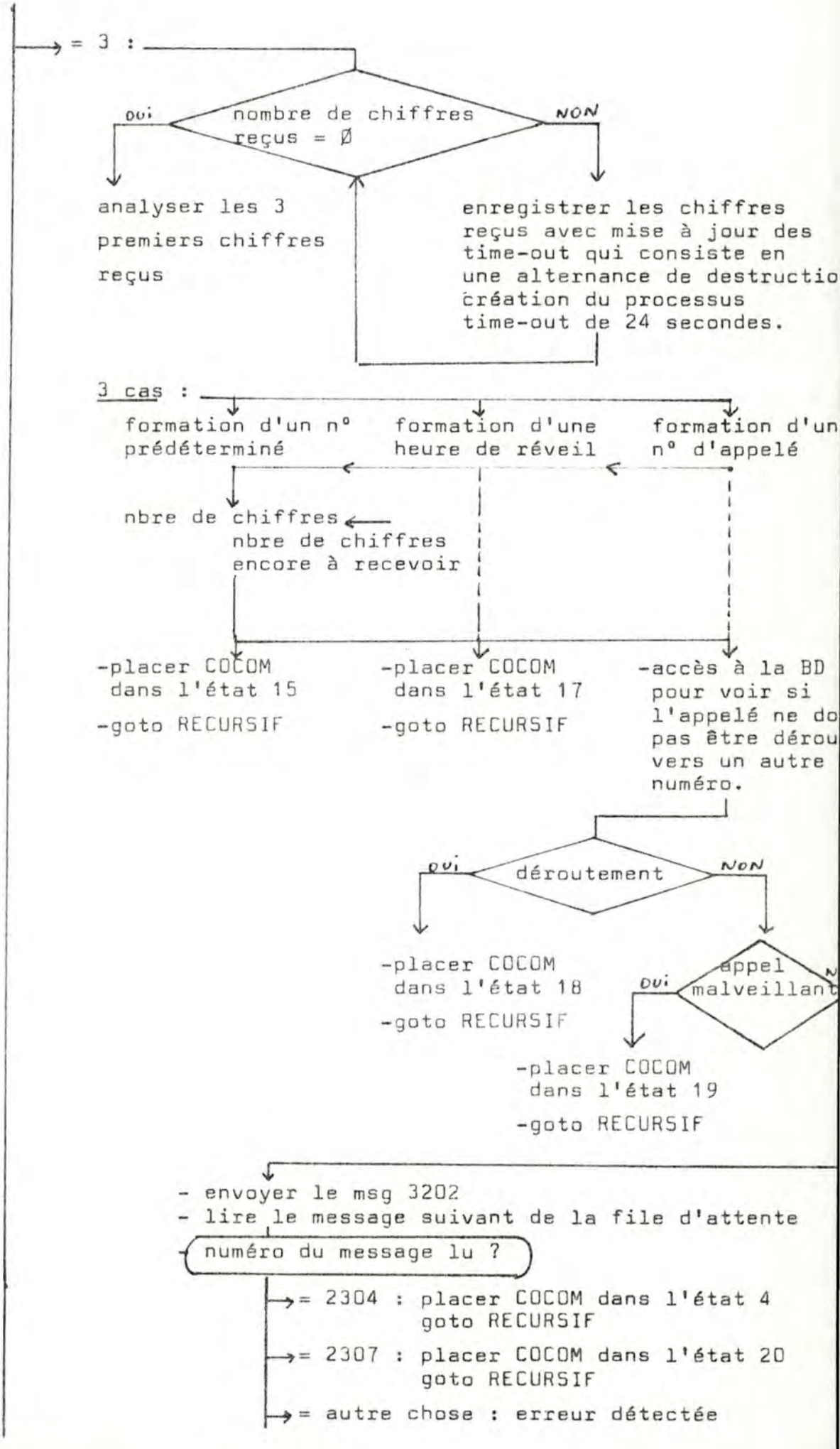
→ = 2 : - création d'un processus qui s'endort
24 secondes, au réveil il envoie au
COCOM qui l'a déclenché un msg (0302)
Cette création est une simulation d'un
déclenchement d'une réception de msg
(ici le chiffre) qui n'attend celui-ci
que 24 secondes (primitive non fournie
par XINU)

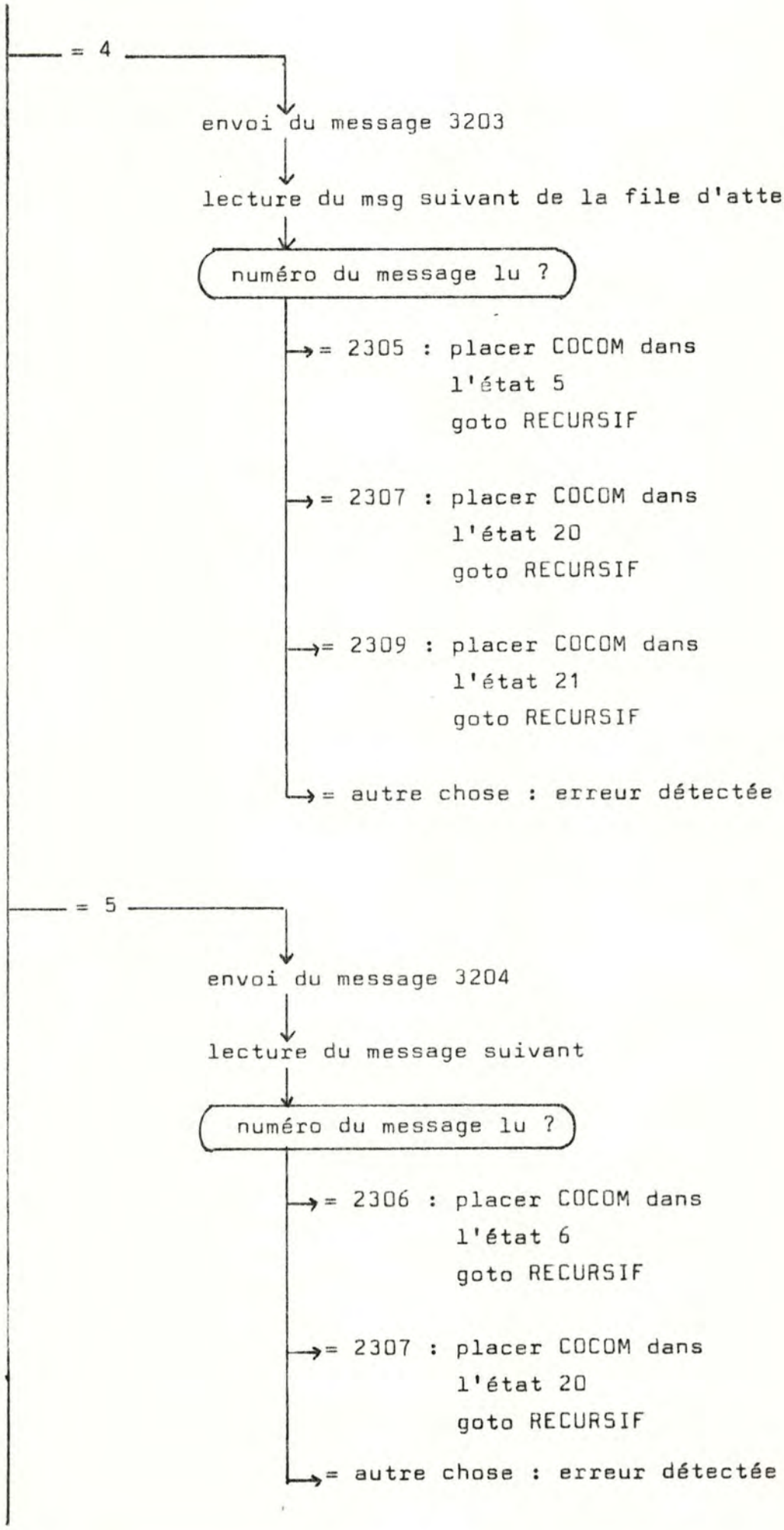
- lecture du message de la file d'attente

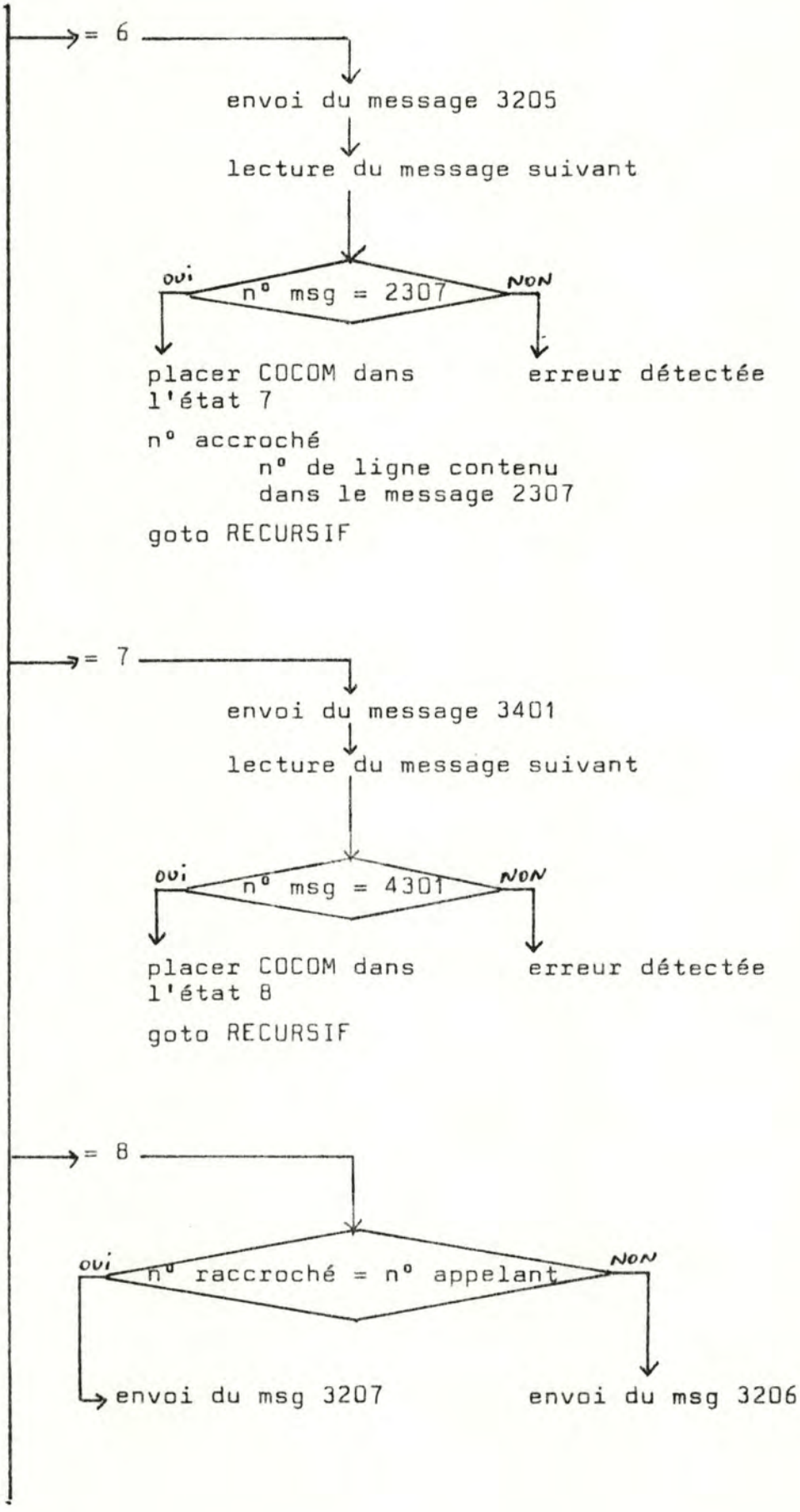
- (numéro du message lu ?)

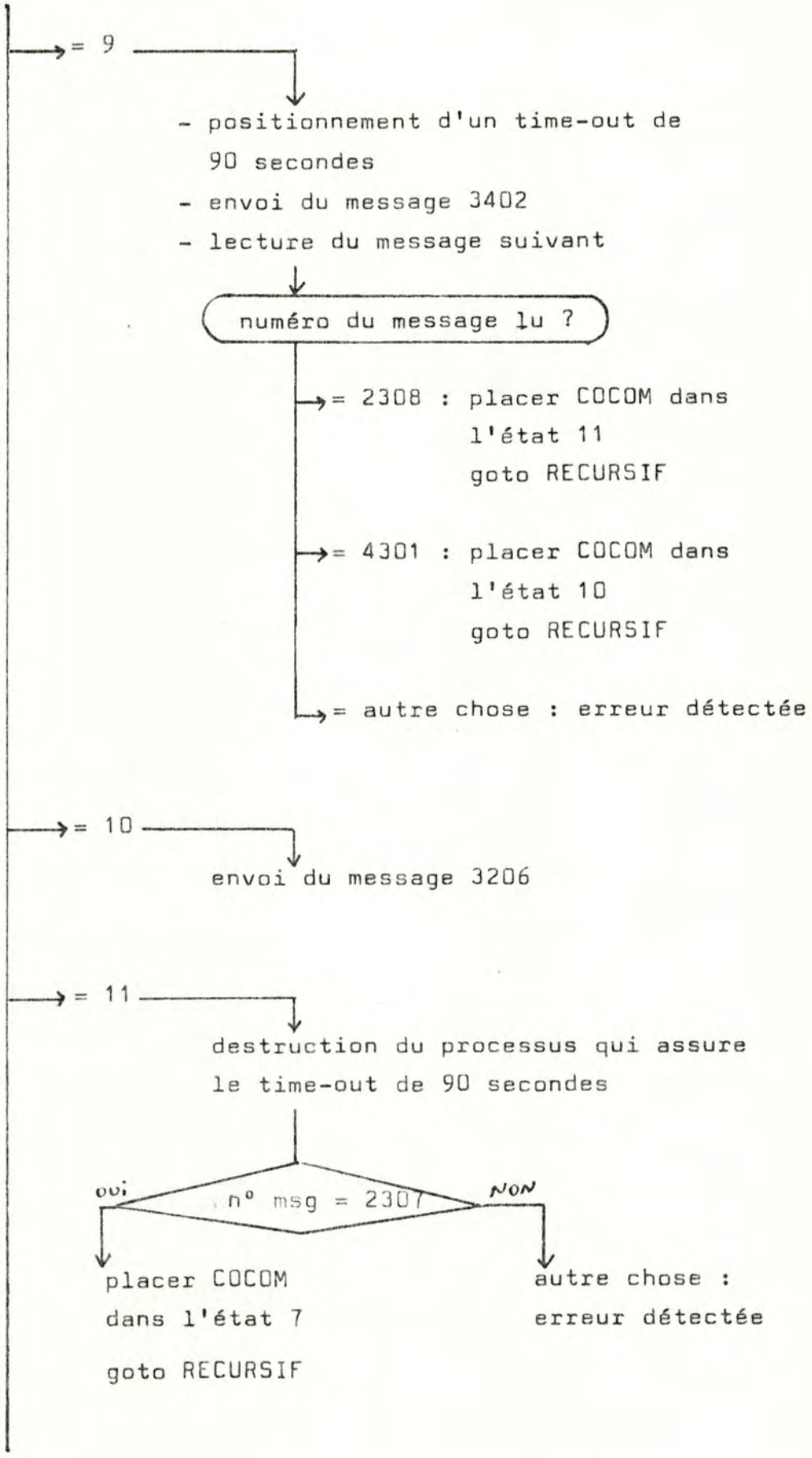
→ = 2303 : placer COCOM dans
l'état 3
goto RECURSIF

→ = 0302 : placer COCOM dans
l'état 16
goto RECURSIF


= 3 :
nombre de chiffres reçus = Ø
OUI
NON
analyser les 3 premiers chiffres reçus
enregistrer les chiffres reçus avec mise à jour des time-out qui consiste en une alternance de destructio création du processus time-out de 24 secondes.
3 cas :
formation d'un n° prédéterminé
formation d'une heure de réveil
formation d'un n° d'appelé
nbre de chiffres
nbre de chiffres encore à recevoir
-placer COCOM dans l'état 15
-goto RECURSIF
-placer COCOM dans l'état 17
-goto RECURSIF
-accès à la BD pour voir si l'appelé ne do pas être dérou vers un autre numéro.
déroutement
OUI
NON
-placer COCOM dans l'état 18
-goto RECURSIF
appel malveillant
OUI
-placer COCOM dans l'état 19
-goto RECURSIF
- envoyer le msg 3202
- lire le message suivant de la file d'attente
numéro du message lu ?
= 2304 : placer COCOM dans l'état 4
goto RECURSIF
= 2307 : placer COCOM dans l'état 20
goto RECURSIF
= autre chose : erreur détectée


= 4
envoi du message 3203
lecture du msg suivant de la file d'attente
numéro du message lu ?
= 2305 : placer COCOM dans l'état 5 goto RECURSIF
= 2307 : placer COCOM dans l'état 20 goto RECURSIF
= 2309 : placer COCOM dans l'état 21 goto RECURSIF
= autre chose : erreur détectée
= 5
envoi du message 3204
lecture du message suivant
numéro du message lu ?
= 2306 : placer COCOM dans l'état 6 goto RECURSIF
= 2307 : placer COCOM dans l'état 20 goto RECURSIF
= autre chose : erreur détectée


= 6
envoi du message 3205
lecture du message suivant
oui
n° msg = 2307
NON
placer COCOM dans l'état 7
n° accroché
n° de ligne contenu dans le message 2307
goto RECURSIF
erreur détectée
= 7
envoi du message 3401
lecture du message suivant
oui
n° msg = 4301
NON
placer COCOM dans l'état 8
goto RECURSIF
erreur détectée
= 8
oui
n° raccroché = n° appelant
NON
envoi du msg 3207
envoi du msg 3206


= 9
- positionnement d'un time-out de 90 secondes
- envoi du message 3402
- lecture du message suivant
numéro du message lu ?
= 2308 : placer COCOM dans l'état 11
goto RECURSIF
= 4301 : placer COCOM dans l'état 10
goto RECURSIF
= autre chose : erreur détectée
= 10
envoi du message 3206
= 11
destruction du processus qui assure le time-out de 90 secondes
n° msg = 2307
oui
NON
placer COCOM dans l'état 7
goto RECURSIF
autre chose : erreur détectée

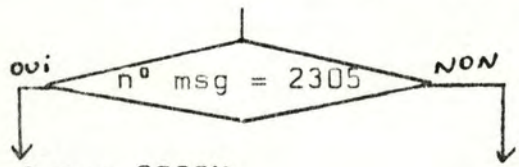
→ = 12 : accès à une table à deux dimensions

n° de ligne	n° du processus time-out de réveil
≡	

on extrait le n° de ligne qui correspond au n° du processus qui a été déclenché lors de la demande d'un réveil

↓

envoi du message 3204



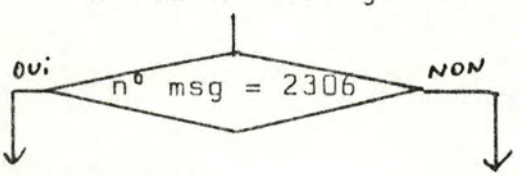
placer COCOM dans l'état 13

goto RECURSIF

erreur détectée

→ = 13

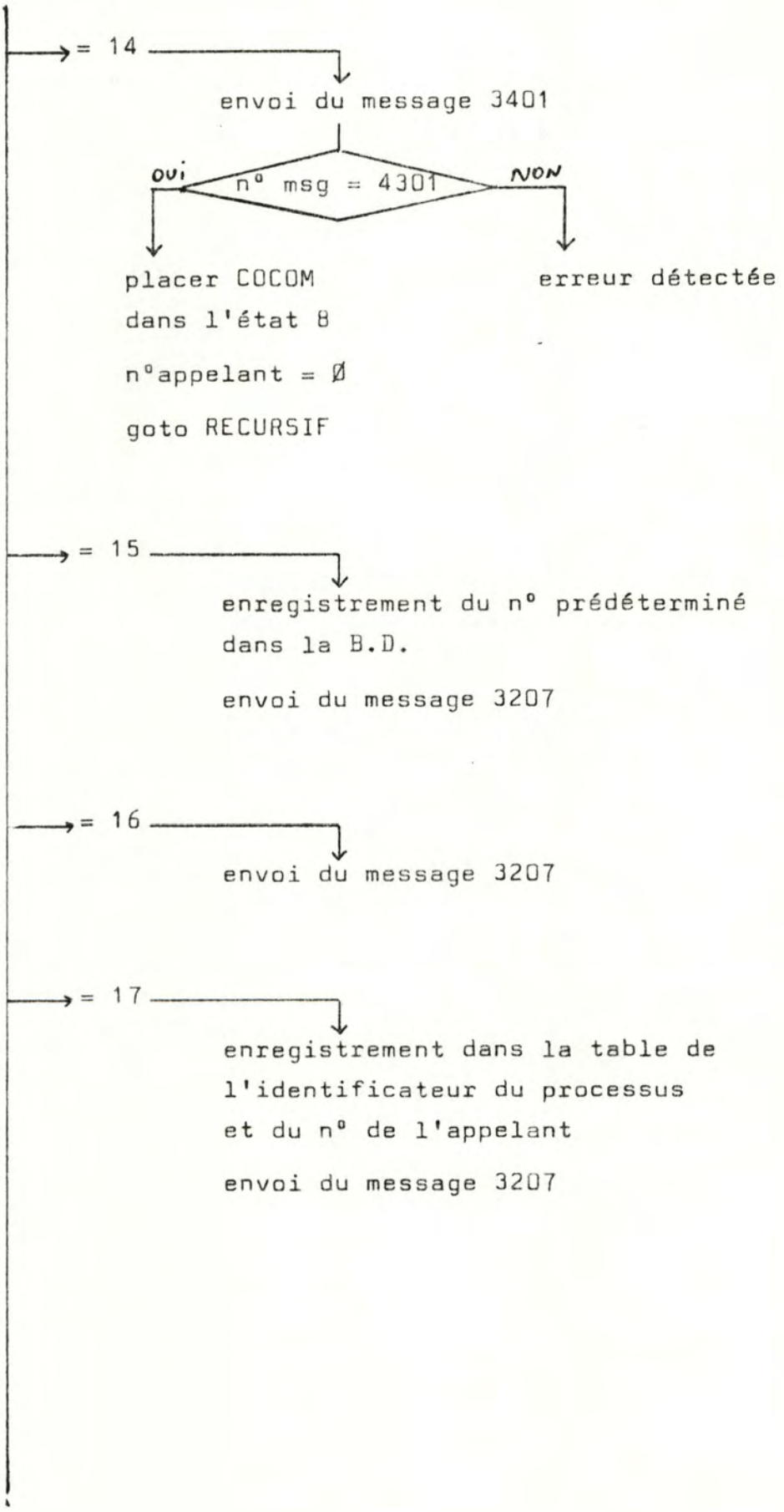
envoi du message 3204

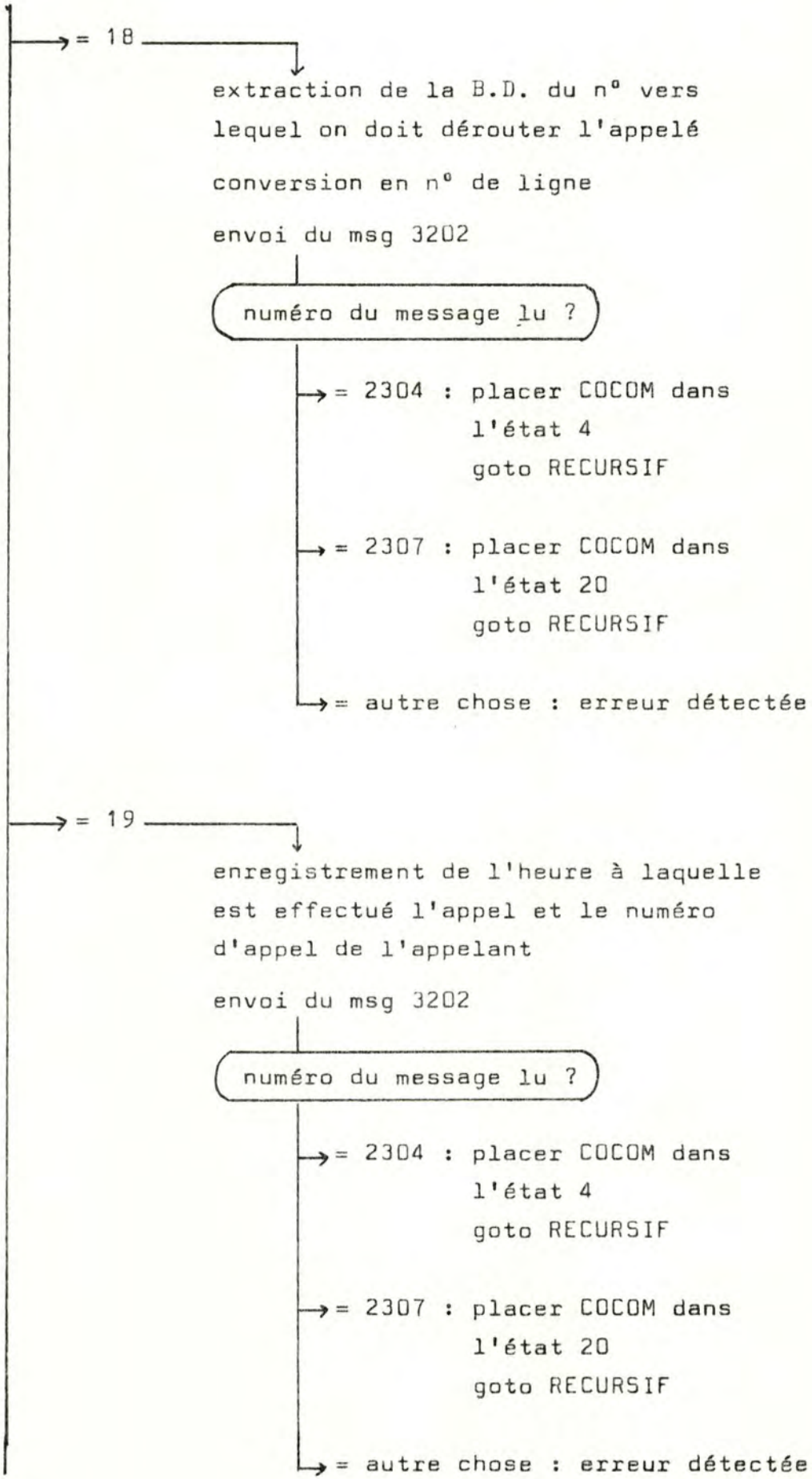


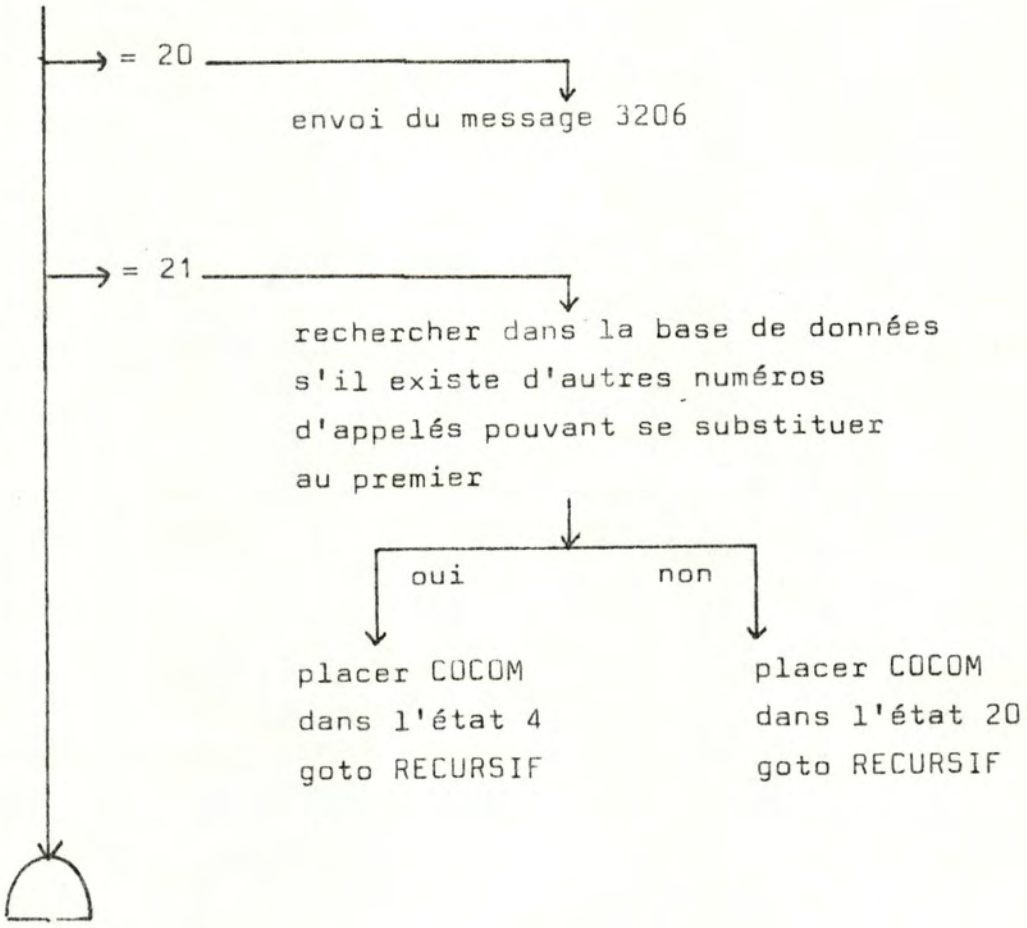
placer COCOM dans l'état 14

goto RECURSIF

erreur détectée


= 14
envoi du message 3401
OUI
n° msg = 4301
NON
placer COCOM
dans l'état 8
n°appelant = Ø
goto RECURSIF
erreur détectée
= 15
enregistrement du n° prédéterminé
dans la B.D.
envoi du message 3207
= 16
envoi du message 3207
= 17
enregistrement dans la table de
l'identificateur du processus
et du n° de l'appelant
envoi du message 3207


= 18
extraction de la B.D. du n° vers
lequel on doit dérouter l'appelé
conversion en n° de ligne
envoi du msg 3202
numéro du message lu ?
= 2304 : placer COCOM dans
l'état 4
goto RECURSIF
= 2307 : placer COCOM dans
l'état 20
goto RECURSIF
= autre chose : erreur détectée
= 19
enregistrement de l'heure à laquelle
est effectué l'appel et le numéro
d'appel de l'appelant
envoi du msg 3202
numéro du message lu ?
= 2304 : placer COCOM dans
l'état 4
goto RECURSIF
= 2307 : placer COCOM dans
l'état 20
goto RECURSIF
= autre chose : erreur détectée



REMARQUE sur l'état de cocom = 8

Cocom possède deux variables qui sont le numéro de ligne APPELE et le numéro de ligne APPELANT afin de pouvoir les différencier lors du raccrochage de l'un des deux abonnés.

3.4.3.4. Taxation.

Ce module ne subit pas d'adaptation importante, si ce n'est qu'il utilise une base de données sous forme de tableaux au lieu de fichiers sur disque si le contrôleur de device n'est pas disponible dans Xinu.

3.4.3.5. Base de données.

Xinu n'offre qu'une gestion de fichiers très simplifiée. Il n'y a que des fichiers séquentiels. Cependant les procédures de bas niveau de gestion des tables de référence des fichiers sont très bien documentées et pour cette raison il a été possible d'écrire une gestion de fichiers séquentiels indexés.

Le texte de ce programme est listé dans les annexes et chaque procédure est bien documentée par elle-même.

CONCLUSION

Dans ce mémoire nous avons voulu aborder un ensemble de questions concernant les réseaux commutés que nous avions peu ou pas approfondies au cours de nos études.

Ainsi nous avons dû nous plonger dans des domaines tels que la théorie des automates, les techniques du temps réel et la gestion avancée de processus parallèles; tous ces problèmes devaient être éclaircis pour une bonne compréhension de la commutation elle-même. L'étude de cette dernière nous a ensuite amenés à l'analyse des méthodes de signalisation et de connexion numérique.

Enfin, nous voulions également réaliser un prototype de commutateur certes réduit dans ses dimensions, mais reprenant tous les outils théoriques abordés précédemment.

Lorsque l'heure de rédiger est venue, nous avons délibérément opté pour une structure reflétant la succession chronologique des différentes étapes de nos investigations.

L'étude des fondements théoriques est reprise dans les deux premiers chapitres. Le chapitre un brosse un tableau des différents aspects des réseaux, avec une attention toute particulière aux réseaux commutés. Le deuxième chapitre se concentre sur les commutateurs qui sont les noeuds de ces réseaux.

Le chapitre trois est une analyse complète d'un commutateur téléphonique local que nous avons conçu sur les bases de commutateurs existant actuellement sur le marché.

Une expérience pratique de l'un de ces derniers nous a été fournies par la firme Ericsson à Rijen et Zwijndrecht, à l'occasion d'une visite agrémentée d'exposés techniques.

Pour terminer, la réalisation de ce mémoire nous a apporté le bénéfice de l'étude de domaines relativement nouveaux pour nous, ainsi qu'une approche concrète d'une branche de l'informatique riche de promesses pour l'avenir.

BIBLIOGRAPHIE

AMIRCHAH'Y, M., NEEL, D., La Conception des Systèmes répartis, I.N.R.I.A., 1979.

COLLECTIF LORRAINS, Réseaux téléinformatiques, Hachette, 1979.

COMER, Douglas, Operating System Design : The XINU approach, Prentice-Hall, 1967.

FREEDMAN, A.L., LEES, R.A., Real-Time Computer Systems, Edward Arnold, 1977.

GLOWINSKI, Albert, groupe de prospective, Télécommunications : Objectif 2000, Dunod, 1980.

GRIES, David, Compiler construction for digital Computers, J.Wiley and sons, 1971.

GRINSEC, La Commutation électronique, vol.1 : Structure de systèmes spatiaux et temporels, Eyrolles, 1984.

GRINSEC, La Commutation électronique, vol.2 : Logiciel, exploitation et maintenance, Eyrolles, 1984.

LISTER, Andrew M., Principes fondamentaux des Systèmes d'Exploitation, Eyrolles, 1984.

MACCHI, César, GUILBERT, Jean-François, téléinformatique, Dunod, 1983.

MARTIN, James, Design of Real-time Computer Systems, Prentice-Hall, 1967.

MEINADIER, Jean-Pierre, Structure et Fonctionnement des Ordinateurs, Larousse, 1971.

KERNIGHAN, Brian W., RITCHIE, Dennis M., le Langage C, Masson, 1984.

KNUTH, Donald E., The Art of Computer programming, vol.1 : Fundamental Algorithms, Addison-Wesley, 1975.

KNUTH, Donald E., The Art of Computer programming, vol.2 : Sorting and Searching, Addison-Wesley, 1975.

Numéro spécial sur le Central numérique Système 12, Revue des Télécommunications, vol.56, nros 1, 1981.

Numéro spécial sur le Réseau numérique à Intégration de Services, Revue des Télécommunications, vol.56, nros 1, 1981.

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR (BELGIQUE)

INSTITUT D'INFORMATIQUE

COMMUTATION NUMERIQUE ET TEMPORELLE

Tome Trois :

Annexes

Promoteur : J. Brunin

Mémoire présenté par

Marc EVRARD
Rudy VANGAVER
Benoît TRIGAUX

en vue de l'obtention
du grade de

LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1984 - 1985

TABLE DES ANNEXES.

Auteurs des annexes :

B.Trigaux : 2.3.,1.1.

R.Vangaver: 1.2,1.3,2.1,2.2.

M.Evrard : 1.4,1.5,2.4,3,4,5.

Annexe 1 :

Listings des programmes en C du prototype COLO/S.

Annexe 1.1 : I L S.

```
/*****************************************************************************
******************************************************************************

                              CONSTANTES DU SYSTEME

                              [ fichier SWITCH.H ]

  ****************************************************************************
*****************************************************************************/

#define  NBRLIG   500   /* nombre de lignes dans le systeme */
#define  BITINT   16   /* nombre de bits par entier - dependant de la machine */
#define  NBETAT   7   /* nombre d'etats possibles pour une ligne */
#define  NBRSON   7   /* nombre de sonneries dans le systeme */

#define  nbrint(a,b)  (a%b == 0) ? (a/b) : (a/b+1)
#define  nbrbit(a,b)  (a%b == 0) ? (b) : (a%b)

/*****************************************************************************
******************************************************************************

                          MODULE TABLE DE PROGRAMMATION

                              [ fichier TABPRG.C ]

  ****************************************************************************
*****************************************************************************/

#include <switch.h>

/*
                                  CONSTANTES
*/

/* frequences de programmation exprimees en nombre de pulsations relatives ou
   en nieme de secondes pour les scannings NUMEROTATION
                                          PRE_NUMEROTATION
                                          POST_NUMEROTATION
                                          LIBRE
                                          PARKING
                                          PRE_CONVERSATION
                                          CONVERSATION
*/

#define  FRNUM     1
#define  FRPREN    1
#define  FRPOSN    1
#define  FRLIBRE   3
#define  FRPARK    3
#define  FRPREC    6
#define  FRCONV    6

#define  MIN       1   /* frequence relative de base */

/*
                      TYPE ABSTRAIT TABLE DE PROGRAMMATION
*/

static int frequence [NBETAT] = { FRNUM, FRPREN, FRPOSN, FRLIBRE, FRPARK,
                                  FRPREC, FRCONV } ;
```

```
/*****************************************************************************/
/*****************************************************************************

                                    PGMSCAN

     Se reveille periodiquement pour programmer le scanning des lignes
     se trouvant dans le ou les etats references periodiquement.

  *****************************************************************************/

pgmscan ()

{
     int i ;
     static int frcompt = 0 ;          /* compteur du nombre de frequences */
     static int scan [NBETAT] ;        /* interface avec le module BCL */
     extern scanning (), sleep () ;

     for (;;) {

          sleep (MIN) ;

          frcompt += MIN ;

          for (i=0;i<NBETAT;i++)
               scan[i] = ((frcompt % frequence[i]) == 0) ? (1) : (0) ;

          scanning (scan) ;

     }
}

/*****************************************************************************/
```

```
*****************************************************************************

                                MODULE BCL
                              [ fichier BCL.C ]

  ***************************************************************************
*****************************************************************************/

#include <switch.h>
#include <stdio.h>

#define  SYSERR  -1
/*
                              TYPE ABSTRAIT BCL
*/

static unsigned int   bcl [nbrint(NBRLIG,BITINT) * (2+NBRSON+NBETAT)] ;

static unsigned int  *nlp ; /* niveau present des lignes */
static unsigned int  *nla ; /* niveau antecedent des lignes */
static unsigned int  *bsl [NBRSON] ; /* bloc de controle des sonneries */
static unsigned int  *bel [NBETAT] ; /* bloc de controle des etats */

/*
                              INITIALISATION DE BCL
*/

initbcl ()
{
     int i ;
     unsigned int *init ;

/* initialisation des pointeurs */

     nlp = & bcl [0];
     nla = & bcl [nbrint(NBRLIG,BITINT)];
     for(i=0;i<NBRSON;i++)
         bsl [i] = & bcl [(2+i)*nbrint(NBRLIG,BITINT)] ;
     for(i=0;i<NBETAT;i++)
         bel [i] = & bcl [(2+NBRSON+i)*nbrint(NBRLIG,BITINT)] ;

/* initialisation des lignes dans l'etat libre */

     for(i=0,init=bel[0];i<nbrint(NBRLIG,BITINT);i++)
         *(init++) = ~0 ;
}

/*****************************************************************************/
/*****************************************************************************

                                  SCANNING

     Effectue le scanning sur toutes les lignes dans le ou les etats
     references et envoie a COLIG un message 1201 pour chaque evenement
     haut detecte sur une ligne et un message 1202 pour chaque evenement
     bas detecte sur une ligne.

*****************************************************************************/

scanning (scan)

int scan [NBETAT] ;

{
     unsigned int tab [6 * nbrint(NBRLIG,BITINT)] ;
```

```
      char *cpter ;
      extern int pcolig ;
      extern char *getmem() ;
      extern psend() ;
      extern exclor(), conjonc(), egal(), groupIg(), pasmem() ;
      extern unsigned int valbit() ;

/* initialisation des pointeurs */

      nx      = & tab [0 * nbrint (NBRLIG,BITINT)] ;
      eh      = & tab [1 * nbrint (NBRLIG,BITINT)] ;
      eb      = & tab [2 * nbrint (NBRLIG,BITINT)] ;
      setat   = & tab [3 * nbrint (NBRLIG,BITINT)] ;
      ehfil   = & tab [4 * nbrint (NBRLIG,BITINT)] ;
      ebfil   = & tab [5 * nbrint (NBRLIG,BITINT)] ;

/* corps */

      exclor(nla,nlp,nx) ;

      conjonc(nla,nx,eh) ;
      conjonc(nlp,nx,eb) ;

      egal(nlp,nla) ;

      groupIg(netat,setat) ;

      conjonc(setat,eh,ehfil) ;
      conjonc(setat,eb,ebfil) ;

      for (i=0;i<nbrint(NBRLIG,BITINT);i++,ebfil++,ehfil++)
           for (j=0;j<BITINT;j++) {

                if (valbit(ehfil,j+1))
                     if ( *(cpter = getmem(4)) != SYSERR ) {
                          * cpter = 1201 ;
                          * (cpter+2) = (i+1)*BITINT + (j+1) ;
                          psend (pcolig,cpter) ;
                     } else
                          pasmem() ;

                if (valbit(ebfil,j+1))
                     if ( *(cpter = getmem(4)) != SYSERR ) {
                          * cpter = 1202 ;
                          * (cpter+2) = (i+1)*BITINT + (j+1) ;
                          psend (pcolig,cpter) ;
                     } else
                          pasmem() ;
           }


*****************************************************************************/
*****************************************************************************

                                   CHANGETAT

     Effectue le passage d'une ligne d'un etat dans un autre.

*****************************************************************************/

changetat (nligne,etancien,etnouveau)
```

```
int nligne, etancien, etnouveau ;

{
     unsigned int * pint ;
     int j ;
     extern actstat(), dactstat() ;
     extern unbit(), nulbit(), retetat() ;
     extern unsigned int valbit() ;



     j = nbrint (nligne,BITINT) - 1 ;
     unbit (bel [etnouveau] + j , nbrbit(nligne,BITINT));
     if ( valbit (pint = (bel [etancien] + j) , nbrbit (nligne,BITINT)))
         nulbit (pint,nbrbit(nligne,BITINT)) ;
     else
         retetat () ;

}

/*****************************************************************************/
/*****************************************************************************

                                    METSON

     Active une sonnerie sur une ligne.

*****************************************************************************/

etson (nligne,nson)

nt nligne, nson ;


     extern unbit() ;

     unbit (bsl [nson] + nbrint(nligne,BITINT)-1 , nbrbit(nligne,BITINT)) ;


/*****************************************************************************/
/*****************************************************************************

                                    RETSON

     Desactive une sonnerie sur une ligne.

*****************************************************************************/

etson (nligne,nson)

nt nligne, nson ;


     extern nulbit() ;

     nulbit (bsl [nson] + nbrint(nligne,BITINT)-1 , nbrbit(nligne,BITINT)) ;

}

/*****************************************************************************/
/*****************************************************************************

                                    GROUPLG

     Concentre dans un seul vecteur l'ensemble des lignes se trouvant
```

```
   dans les états référencés.

**********************************************************************/

anetat (compt,solpter)

t compt [NBETAT] ;
signed int * solpter ;


   extern inclor () ;
   int i,j ;

 initialisation du vecteur OUTPUT */

   for (i=0 ; i<nbrint (NBRLIG,BITINT) ; i++)
        * (solpter+i) = 0 ;

 corps */

   for (i=0 ; i< nbrint(NBRLIG,BITINT) ; i++)
        for (j=0 ; j<NBETAT ; j++)
             if (compt [j] == 1)
                 inclor (bel [j]+i, solpter+i, solpter+i) ;



**********************************************************************/
**********************************************************************

                              CONJONC

   Effectue un ET logique entre deux vecteurs de NBRLIG bits.

**********************************************************************/

njonc (apter,bpter,solpter)

nsigned int *apter, *bpter, *solpter ;


   unsigned int * jpter ;
   int j ;

   jpter = solpter ;
   for (j=0;j< nbrint(NBRLIG,BITINT);j++)
        * jpter++ = * apter++ & * bpter++ ;



**********************************************************************/
**********************************************************************

                              INCLOR

   Effectue un OU inclusif logique entre deux vecteurs de NBRLIG bits.

**********************************************************************/

nclor (apter,bpter,solpter)

nsigned int *apter, *bpter, *solpter ;


   unsigned int * jpter ;
   int j ;
```

```
     jpter = solpter ;
     for (j=0;j< nbrint(NBRLIG,BITINT);j++)
          * jpter++ = * apter++ ; * bpter++ ;

}

/*****************************************************************************/
/*****************************************************************************

                                   EXCLOR

     Effectue un OU exclusif logique entre deux vecteurs de NBRLIG bits.

*****************************************************************************/

exclor (apter,bpter,solpter)

unsigned int *apter, *bpter, *solpter ;

{
     unsigned int * jpter ;
     int j ;

     jpter = solpter ;
     for (j=0;j< nbrint(NBRLIG,BITINT);j++)
          * jpter++ = * apter++ ^ * bpter++ ;

}

/*****************************************************************************/
/*****************************************************************************

                                   EGAL

     Affecte la valeur d'un vecteur de NBRLIG bits a un autre vecteur.

*****************************************************************************/

egal (apter,solpter)

unsigned int *apter, *solpter ;

{
     unsigned int * jpter ;
     int j ;

     jpter = solpter ;
     for (j=0;j< nbrint(NBRLIG,BITINT);j++)
          * jpter++ = * apter++ ;

}

/*****************************************************************************/
/*****************************************************************************

                                  VALBIT

     Retourne la valeur 0 ou 1 d'un bit dans un entier (unsigned).

*****************************************************************************/

unsigned int valbit (pint,nbit)

unsigned int * pint ;
int nbit ;
```

```
{
     unsigned int inter ;
     extern int power() ;

     inter = power (2,nbit-1) ;
     inter = inter & *pint ;
     if (inter != 0)
         inter = 1 ;

     return (inter) ;
}

/*****************************************************************************/
/*****************************************************************************

                                    UNBIT

     Affecte la valeur 1 a un bit dans un entier (unsigned).

*****************************************************************************/

unbit (pint,nbit)

unsigned int * pint ;
int nbit ;

{
     unsigned int inter ;
     extern int power() ;

     inter = power (2,nbit-1) ;
     *pint = *pint | inter ;

}

/*****************************************************************************/
/*****************************************************************************

                                   NULBIT

     Affecte la valeur 0 a un bit dans un entier (unsigned).

*****************************************************************************/

nulbit (pint,nbit)

unsigned int * pint ;
int nbit ;

{
     unsigned int inter ;
     extern int power() ;

     inter = ~ power (2,nbit-1) ;
     *pint = *pint & inter ;

}

/*****************************************************************************/
/*****************************************************************************

                                   POWER

     Eleve un nombre entier a une puissance entiere donnee.
```

```
/*****************************************************************************/

int power (x,n)

int x,n ;

{
     int i,p ;
     extern errind () ;

     if (x == 0 & n == 0)
         errind () ;
     for (p=i=1;i<=n;++i)
          p = p * x ;

     return (p) ;
}

/*****************************************************************************/
/*****************************************************************************

                                   PASMEM

     Traitement d'un manque de mémoire .

*****************************************************************************/

pasmem ()

{
     printf ("\n\n\n\nPas de mémoire libre pour envoyer un message ") ;
     printf ("(fonction SCANNING)\n\n\n") ;
}

/*****************************************************************************/
/*****************************************************************************

                                   RETETAT

     Traitement d'une erreur dans un changement d'état.

*****************************************************************************/

retetat ()

{
     printf ("\n\n\n\nTentative de retirer une ligne d'un état où ") ;
     printf ("elle ne se trouve pas (fonction CHANGETAT)\n\n\n") ;
}

/*****************************************************************************/
/*****************************************************************************

                                   ERRIND

      Traitement d'une indétermination dans le calcul d'une puissance.

*****************************************************************************/

errind ()

{
     printf ("\n\n\n\nTentative d'élever 0 à la puissance 0 ") ;
     printf ("(fonction POWER)\n\n\n") ;
}
```

Annexe 1.2 : C O L I G.

```
/* traiter la reception d'un chiffre */
trrech();

{
  extern int sv200();
  static int svid,nmsg,chiffre;

    chiffre=0;
    svid = create (sv200,2,PRSV,sv200,1,nlgclg);
    while (nbrechiffres != 0){
          nmsg = receive();
          switch (nmsgprec){
                case '1201':switch (nmsg){
                                  case '0201': nbrechiffres--;
                                               evchiff(2303,6,nlgclg,chiffre);
                                               kill (svid);
                                               chiffre=0;
                                               break;
                                  case '1202': nmsgprec=1202;
                                               kill (svid);
                                               chiffre++;
                                               trrech();
                                               break;
                            }
                case '1202':switch (nmsg){
                                  case '0201': evmsg(2307,4,nlgclg);
                                               kill(svid);
                                               break;
                                  case '1201': nmsgprec=1201;
                                               kill(svid);
                                               chiffre++;
                                               trrech();
                                               break;
                            }
          }
    }
    return();
}

/*_____________________________________________________________________*/
/*                                                                     */
/*                  S U P E R V I S E U R                              */
/*                        C O L I G                                    */
/*_____________________________________________________________________*/

supclg();

{
 extern int *msg;

    rmsg(PCOLIG);
    switch (lgetclg(nlgclg)){      /* test de l'etat present de colig */
          case '0':
               switch(nmsgclg){
                     case '1201':
                          chetclg(nlgclg,1);
                          break;
                     case '0202':
                          chetclg(nlgclg,14);
                          break;
                     default:
                          error();
                          break;
```

```
case '1':if (nmsgclg==3201)
            chetclg(nlgclg,2)
         else
            error();
         break;
case '2':if (nmsgclg==1202)
            chetclg(nlclg,3);
            nmsgprec=1202
         else
            error();
         break;
case '3':switch (nmsgclg){
            case '0201':
                 chetclg(nlgclg,15);
                 break;
            case '3202':
                 chetclg(nlgclg,4);
                 break;
            case '1201':
            case '1202':
                 send(tab-cor[(donind(nlgclg,APPELANT))] [5]),nmsgcl
g);
                 break;
            default:
                 error();
                 break;
         }
         break;
case '4':if (nmsgclg==3203)
            chetclg(nlgclg,5)
         else
            error();
         break;
case '5':if (nmsgclg==1201)
            chetclg(nlgclg,6)
         else
             error();
         break;
case '6':if (nmsgclg==3205)
            chetclg(nlgclg,7)
         else
            error();
         break;
case '7':if (nmsgclg==1202)
            chetclg(nlgclg,8)
         else
            error();
         break;
case '8':switch(nmsgclg){
            case '3206':
                 chetclg(nlgclg,9);
                 break;
            case '1201':
                 chetclg(nlgclg,14);
                 break;
            case '3207':
                 chetclg(nlgclg,11);
                 break;
            default:
                 error();
                 break;
```

```
            case '9':if (nmsgclg==1202)
                       chetclg(nlgclg,10)
                     else
                       error();
                     break;
            case '10':chetclg(nlgclg,0);
                     break;
            case '11':if (nmsgclg==1202)
                       chetclg(nlgclg,12)
                     else
                       error();
                     break;
            case '12':chetclg(nlgclg,0);
                     break;
            case '13':if (nmsgclg==1202)
                       chetclg(nlgclg,8)
                     else
                       error();
                     break;
            case '14':/* cas a traiter plus tard */
            case '15':chetclg(nlgclg,0);
                     break;
    }
    colig(lgetclg(nlgclg));    /* declenchement de colig avec l'etat suivant */
}

/*_____________________________________________________________________*/

/*              A U T O M  A T E     C O L I G                        */
/*_____________________________________________________________________*/

colig(etat)

  int cocom();
  int etat;
  int *msg;

{
  int id-cc,id-pcc;

    switch(etat){
       case '0':
                supclg();
       case '1':
                if (n-co-cour<=n-co-max)
                {
                      id-cc=create(supcocom,1,PRCOCOM,supcocom,1,nlgclg);
                      id-pcc=pcreate(10);
                      init-elt(nlgclg,id-cc,id-pcc);
                      chgt-etat(LIBRE,PRENUM);
                      act-scan(PRENUM);
                      evmsg(2301,4,nlgclg);
                      supclg();
                 }
                 else
                      metcon(ENCOMBREMENT);
```

```
      case '2':
                chgt-etat(PRENUM,NUM);
                act-scan(NUM);
                desact-scan(PRENUM);
                metson(INVNUM);
                evmsg(2302,4,nlgclg);
                supclg;
       case '3':
                desson(INVNUM);
                tab-cor[donind(nlgclg,APPELANT)] [5] = create (trrech,0,PRREC,0
);
                supclg();
       case '4':
                chgt-etat(NUM,POSTNUM);
                act-scan(POSTNUM);
                desact-scan(NUM);
                evmsg(2304,4,nlgclg);
                supclg();
       case '5':
                chgt-etat(POSTNUM,PRECONV);
                act-scan(PRECONV);
                metson(SONAPP);
                metson(ECHOSON);
                evmsg(2305,4,nlgclg);
                supclg();
       case '6':
                desson(SONAPP);
                chgt-etat(PRECONV,CONV);
                evmsg(2306,4,nlgclg);
                supclg();
       case '7':
                changetat(nlgclg,POSTNUM,CONV);
                break;
       case '8':
                evmsg(2307,4,nlgclg);
                break;
       case '9':
                lgccpc(nlgclg);
                kill (dcocom);
                pdelete (dpcocom,dispose);
                tabcor[(donind(nlgclg,APPELANT)] [6]=0;
                changetat (nlgclg,CONV,LIBRE);
                changetat (tabcor[(donind(nlgclg,APPELANT))] [APPELE],CONV,PARK
ING);
                break;
       case '10':
                 changetat(nlgclg,PARKING,LIBRE);
                 break;
       case '11':
                 lgccpc(tabcor[(donind(nlgclg,APPELE))] [APPELANT]);
                 kill (dcocom);
                 pdelete (dpcocom,dispose);
                 tabcor [(donind(nlgclg,APPELE))] [6]=0;
                 changetat (nlgclg,CONV,LIBRE);
                 changetat (tabcor[(donind(nlgclg,APPPELE))] [APPELANT],CONV,PA
RKING);
                 break;
       case '12':
                 changetat (nlgclg,PARKING,LIBRE);
```

```
case '13':
          evmsg(2309,4,nlgclg);
          break;
case '14': /* cas momentanement en suspend */
          break;
case '15':
          lgccpc(nlgclg);
          kill (dcocom);
          pdelete (dpcocom,dispose);
          tabcort(donind(nlgclg,APPELANT)].ca]=0;
          changetat (nlgclg,NUN,LIBRE);
          break;
default: error();
         supclg();
}
```

```
extern int (*tabcor) [8];
extern int dcocom,dpcocom;
extern int indtab;
£define NBRELIGNES = 0;

/* initialiser les valeurs des elements du tableau a la  valeur 0 */

inittab()

{
  int i,j;
     for ( i=0; i != NBRELIGNES ; ++i)
         for ( j=0; j != 8; ++j)
             (*tabcor+i) [j] = 0;
}


/* donner l'indice de l'element du tableau  */

donind(numligne,col)
int numligne,col;
{
  int i;
      for(i=0; (*tabcor+i) [col] != numligne; ++i) ;
      return(i);
}
/*_____________________________________________________________________*/

/* initialisation d'un element de la table de correspondance avec  */
/* un numero de ligne, un numero de cocom et un numero de port de cocom */

initelt(numligne,numcocom,numpcocom)
  int numligne,numcocom,numpcocom;

{
  int i;

    for ( i=0; (*tabcor+i) [6] != 0; ++i) ;
    (*tabcor+i) [1]=numligne;
    (*tabcor+i) [2]=numcocom;
    (*tabcor+i) [3]=numpcocom;
    (*tabcor+i) [4]=0;
    (*tabcor+i) [6]=1;
    (*tabcor+i) [8]=0;
    ++i;
    return(i);
}
```

```
/* etablir la correspondance entre un numero de ligne et un numero de cocom */
/* et un numero de pcocom */

lgccpc(numligne)
  int numligne;

{
  int i;
    for (i=0; (*tabcor+i) [1] != numligne; ++i) ;
    dcocom=(*tabcor+i) [2];
    dpcocom=(*tabcor+i) [3];
    return(i);
}

/*_____________________________________________________________________________*/

/* etablir la correspondance entre un numero de ligne et un numero de pcocom */

lgpc(numligne)
  int numligne;

{
  int i;
    for (i=0; (*tabcor+i) [1] != numligne; ++i) ;
    return((*tabcor+i) [3]);
}

/*_____________________________________________________________________________*/

/* changer l'etat de l'automate COLIG */

chetclg(numligne,nouvetatcolig)
  int numligne,nouvetatcolig;

{
  int i;

    for (i=0; (*tabcor+i) [1] != numligne; ++i) ;
    return((*tabcor+i) [4]);
}

/*_____________________________________________________________________________*/

/* etablir la correspondance entre un numero de ligne et l'etat actuel*/
/* de l'automate COLIG pour cette ligne */

lgetclg(nligne)
  int nligne;

{
  int i;

    for (i=0; (*tabcor+i) [1] != nligne; ++i) ;
    return((*tabcor+i) [4]);
```

```
    }

/*_____________________________________________________________________*/

evmsg(nmsg,nbytes,nlig)
  int nmsg,nbytes,nlig;
  char *getmem();
  extern int *msg;

{
    msg=getmem(nbytes);
    *msg=nmsg;
    *msg+1=nlig;
    psend(tab-cor[(donind(nlig,APPELANT))] [3],*msg);
    return();
}

/*_____________________________________________________________________*/

/* prendre connaissance d'un message enregistrO dans le port de COLIG */

rmsg(pid)
  int pid;

{
  extern int *msg;

    *msg=(int *)preceive(pid);
    nmsgclg=*msg;
    nlgclg=*msg+1;
    return();
}
```

Annexe 1.3 : C O C O M.

```
ocom();
nt etprescoc;

    rmsgcoc();
    switch (etprescoc) {
           case '0' : switch ( nmsgcoc ) {
                             case ' 2301' : etprescoc = 1;
                                            cocom();
                             case ' 2310' : etprescoc = 12;
                                            cocom();
                             default : errdet = 1;
                      }
           case '1' : inifco();
                      lab2(numabo);
                      if (numabpre==0) {
                         etprescoc = 4;
                         cocom();
                         }
                      else {
                         rmsgcoc();
                         if (nmsgcoc == 2302) {
                            etprescoc = 2;
                            cocom () ;
                            }
                         else
                            errdet = 2 ;
                       }
                       clofco ();
            case '2' : resume (create (sleep24, 0, 3, "sleep24")) ;
                       rmsgcoc () ;
                       switch (nmsgcoc) {
                              case '2303' : etprescoc = 3;
                                            cocom ();
                              case '0302' : etprescoc = 16;
                                            cocom ();
                       }
            case '3' : while (nbchiff <> 0)
                             enrchif () ;
                       analchif () ;
                       traitformchif ();
            case '4' : send (3203);
                       rmsgcoc ();
                       switch (nmsgcoc) {
                              case '2305' : etprescoc = 5;
                                            cocom ();
                              case '2307' : etprescoc = 20;
                                            cocom ();
                              case '2309' : etprescoc = 21;
                                            cocom ();
                              default     : errdet = 3;
                       }
            case '5' : send (3204);
                       rmsgcoc ();
                       switch (nmsgcoc) {
                              case '2306' : etprescoc = 6;
                                            cocom () ;
                              case '2307' : etprescoc = 20;
                                            cocom ();
                              default     : errdet = 4;
                       }
            case '6' : send (3205);
                       rmsgcoc ();
                       if (nmsgcoc == 2307) {
                          numabob = *msgcoc + 1;
                          etprescoc = 7;
```

```
                else
                   errdet = 5;
case '7' : send (3401);
           rmsgcoc ();
           if (nmsgcoc == 4301) {
              etprescoc = 8;
              cocom ();
           }
           else
              errdet = 6;
case '8' : if (nmsgcoc+1 == numaba)
              send (3207);
           else
              send (3206);
case '9' : resume(pid = create(sleep90, 0, 4, "sleep90"));
           send (3402);
           rmsgcoc ();
           switch (nmsgcoc) {
                  case '2308' : etprescoc = 11;
                                cocom ();
                  case '4301' : etprescoc = 10;
                                cocom ();
                  default     : errdet = 7;
           }
case '10': send (3206);
case '11': kill (pid);
           if (nmsgcoc == 2307) {
              etprescoc = 7;
              cocom ();
           }
           else
              errdet = 8;
case '12': traitrev ();
           send (3204);
           if (nmsgcoc == 2305) {
              etprescoc = 13;
              cocom ();
           }
           else
              errdet = 9;
case '13': send (3204);
           if (nmsgcoc == 2306) {
              etprescoc = 14;
              cocom();
           }
           else
              errdet = 10;
case '14': send (3401);
           if (nmsgcoc = 4301) {
              etprescoc = 8;
              numaba = 0;
              cocom();
           }
           else
              errdet = 11;
case '15': enrnpre ();
           send (3207);
case '16': send (3207);
case '17': enrtab ();
           send (3207);
case '18': traitdet ();
           send (3202);
           switch (nmsgcoc) {
                  case '2304' : etprescoc = 4;
                                cocom ();
                  case '2307' : etprescoc = 20;
                                cocom ();
                  default     : errdet = 12;
           }
case '19': enrheur ();
```

```
                switch (nmsgcoc) {
                        case '2304' : etprescoc = 4;
                                      cocom ();
                        case '2307' : etprescoc = 20;
                                      cocom ();
                        default     : errdet = 13;
case '20': send (3206);
case '21': search ();
```

Annexe 1.4 : T A X A T.

```
/*=============================================================================*/

/* igeocom.c :    [p42]
   ----------

     indiquer le type geographique de communication sur base des principes
suivants :

type 1 : communication zonale

type 2 : communication etablie entre deux abonnes appartenant a deux zones
         geographiques petites contigues

type 3 : communication etablie entre deux abonnes appartenant a deux zones
         geographiques contigues dont l'une au moins est une grande zone ou
         appartenant a deux zones petites separees par une seule zone
         geographique petite

type 4 : autres communications

*/

igeocom(ptrmsg,tgeocom)
extern struct btxcom *ptrmsg ;
extern int tgeocom ;
{
        int izonea izoneb ;
        int i j ;
        int clerel ;
        tabo abonne ;
        tzone zone zonea zoneb ;
        labocle(ptrmsg->numaboa) ;
        izonea = abonne.igeozone ;
        labocle(ptrmsg->numabob) ;
        izoneb = abonne.igeozone ;
        lzonecle(izonea) ;
        zonea = zone ;
        lzonecle(izoneb) ;
        zoneb = zone ;
        if (izonea == izoneb)
           { tgeocom = 1 ;
             return(tgeocom) }
        for (i = 1 ; i <= j ; i++)
           if (zonea.izonecont[i] == izoneb)
              if ((zonea.taillezone == 1) & (zoneb.taillezone == 1))
                 { tgeocom = 2 ;
                   return(tgeocom) }
              else
                 { tgeocom = 3 ;
                   return(tgeocom) }
        for (i = 1 ; i <= 5 ; i++)
           for (j=1 ; j <= 5 ; j++)
              if (zonea.izonecont[i] == zoneb.izonecont[j])
                 { lzonecle(zonea.izonecont[i])
                   if (zone.taillezone == 1)
                      { tgeocom = 3 ;
                        return(tgeocom) }}
        tgeocom = 4 ;
        return(tgeocom)

} /* fin de igeocom */
```

```
/* calcut.c :    [p43]
   ---------

      calcul du nombre d'unites de taxation en tarif de nuit et de jour

*/
calcut(ptrmsg,nbuttn,nbuttr)
struct btxcom *ptrmsg ;
int nbuttn nbuttr ;
{       int h1 h2 nbut;
        nbuttn = nbuttr = 0 ;
        h1 = hdeb ;
        j1 = jdeb ;
        while ((h1 != hfin) | (j1 != jfin))
           { isoltranche(h1,j1,nbut,tt);
             if (tt = 1)
                nbuttn = nbuttn + nbut
             else
                nbuttr = nbuttr + nbut ) ;
        return()

}

/*=============================================================================*/

/* calcfact.c :    [p44]
   -----------

        calcul de la facture

*/
calcfact(nbuttn,nbuttr,tgeocom,montfact)
int nbuttn nbuttr tgeocom montfact ;
{
     extern struct ttarif tarif[] ;
     montfact = nbuttn * tarif[tgeocom].jour + nbuttr * tarif[tgeocom].nuit ;
     return()

}

/*=============================================================================*/

/* factur.c :    [p45]
   ---------

      facturation d'une communication a un abonne

*/

factur(ptrmsg,montfact)
struct btxcom *ptrmsg ;
int montfact ;
{     extern struct abonne ;
      extern struct zone ;
      if (ptrmsg->indchrg == 1)
         { labocle(ptrmsg->numaboa) ;
           abonne.montfact += montfact )
      else if (ptrmsg->indchrg == 2)
         { labocle(ptrmsg->numabob) ;
           abonne.montfact += montfact ) ;
      lfwrite(devfab,&abonne,sizeof(abonne)) ;
      return()

}
```

```
/* clasfgcom.c :    [p58]
   ------------

       enregistrement de tous les renseignements relatifs a une communicatio
dans le fichier general des communications fhistcom

*/
clasfgcom(ptrmsg,nbuttn,nbuttr,montfact)
extern struct btxcom *ptrmsg ;
int nbuttn nbuttr montfact ;
{
     lfwrite(devfhist,&histcom,sizeof(histcom)) ;
     return()

}

/*=============================================================================*/

/* bissext :    [p33]
   --------

           = 0 si annee est bissextile

*/
bissext(annee)
int annee ;
{
     int bis ;
     if (((annee%4 == 0)&&(annee%100 != 0))||(annee%400 == 0))
        { bis = 0 ;
          return(bis) }
     else
        { bis = 1 ;
          return(bis) }

}

/*=============================================================================*/
```

```
}

/* isoltranche :    [p59]
   ------------

          isoler la tranche de periode de communication comprise entre une
heure donnee et la fin de periode jour/nuit

*/
isoltranche(h1,h2,j1,j2,nbut,tt)
int h1,h2,j1,j2,nbut,tt ;
{
        if (ferie(j1))
           { if ((h2<minuit)&(j1==j2))
                { calcint(h1,h2,nbut) ;
                  h1 = h2 }
             else
                { calcint(h1,minuit,nbut) ;
                  h1 = 0 } ;
             tt = 2 ;
             return() }
        if (h1 < hdjour)
           { if ((h2 < hdjour) & (j1 == j2))
                { calcint(h1,h2,nbut) ;
                  h1 = h2 }
             else
                { calcint(h1,hdjour,nbut) ;
                  h1 = hdjour } ;
             tt = 2 ;
             return() }
        if (h1 < hfjour)
           { if ((h2 < hfjour) & (j1 == j2))
                { calcint(h1,h2,nbut) ;
                  h1 = h2 }
             else
                { calcint(h1,hfjour,nbut) ;
                  h1 = hfjour } ;
             tt = 1 ;
             return() }
        if ((h2 < minuit) & (j1 == j2))
           { calcint(h1,h2,nbut) ;
             h1 = h2 }
        else
           { calcint(h1,minuit,nbut) ;
             h1 = 0 ;
             j1 = joursuiv(j1) }
        tt = 2 ;
        return() ;

} /* fin de isoltranche */

/*=============================================================================*/
```

```
/* joursuiv :    [p40]
   ---------

        donne le jour suivant un jour donne

*/
int joursuiv(jprec)
int jprec ;
{       jour = (jprec/100 - (int)(jprec/100)) * 100 ;
        mois = ((jprec - jour) / 10000 - (int)((jprec - jour)/10000))*10000 ;
        annee = (int) (jprec/10000) ;
        if ((jour == 28) & (mois == 2) & (bissext(annee) != 0))
           { moissuiv(annee,mois,jour) ;
             return(jprec) } ;
        if ((jour == 29) & (mois == 2) & (bissext(annee) == 0))
           { moissuiv(annee,mois,jour) ;
             return(jprec) } ;
        if((jour==30)&&((mois==4)||(mois==6)||(mois==9)||(mois==11)))
           { moissuiv(annee,mois,jour) ;
             return(jprec) } ;
        if((jour==31)&&((mois==1)||(mois==3)||(mois==5)||(mois==7)||(mois==8)||(m
ois==10)||(mois==12)))
           { moissuiv(annee,mois,jour) ;
             return(jprec) } ;
        jprec += 1 ;
        return(jprec)

}

/*=============================================================================*/

/* moissuiv :    [p41]
   ---------

        passer au mois suivant

*/
int moissuiv(annee,mois,jour)
int annee mois jour ;
{
    int jsuiv ;
    if (mois == 12)
        { mois = 1 ;
          annee ++ }
     else mois ++ ;
     jour = 1 ;
     jsuiv = annee*10000 + mois*100 + jour ;
     return(jsuiv)

}

/*=============================================================================*/
```

Annexe 1.5 : S G F I.

```
}
```

```
/*****************************************************************************
 *                                                                           *
 *                              F S G F I                                    *
 *                              = = = = =                                    *
 *                                                                           *
 *       PRIMITIVES DE GESTION DE FICHIERS INDEXES PAR CLE RELATIVE          *
 *       ========== == ======= == ======== ======= === === ========          *
 *                                                                           *
 ****************************************************************************/

/* cleabo :    [p17]
   -------

      obtenir la cle relative d'un abonne a partir de son numero d'abonne

*/
cleabo(numabo,offset,indabo)
int numabo offset indabo ;
{     extern int nbabo ;
      extern struct index tabiabo[nblig] ;
      int i j m ;
      cleabo = 0 ;
      i = 1 ;
      j = nbabo ;
      while (i <= j)
         { m = (i+j)%2 ;
           if (numabo = tabiabo[m].cle)
              { offset = tabiabo[m].offset ;
                indabo = m ;
                return() } ;
           if (numabo < tabiabo[m].cle)
              j = m - 1
           else
              i = m + 1
         } ;
      indabo = j ;
      return()
}

/*============================================================================*/
```

```
/* labo :    [p9]
   -----

        lire un enregistrement abonne

           input : - numabo = 0 : lecture sequentielle
                   - numabo > 0 : lecture du record de cle numabo
                   - numabo < 0 : erreur

           output: - numabo = -1: signale qu'il y a eu une erreur
                   - sinon le resultat se trouve dans abonne

*/
labo(numabo)
int numabo ;
{
      int offset indabo ;
      extern struct tabo abonne ;
      extern struct devsw *devfab ;
      if (numabo < 0) {
         numabo = -1 ;
         return() }
      if (numabo > 0) {
         cleabo(numabo,offset,indabo) ;
         if (offset == 0) {
            numabo = -1 ;
            return() } ;
         lfseek(devfab,offset) }
      lfread(devfab,abonne,sizeof(abonne)) ;
      return()

}

/*=============================================================================*/

/* iniftra :    [p52]
   --------

        initialisation et ouverture des fichiers utilises par tratax

*/
iniftra()
{     char *mode ;
      lfinit(devfab) ;
      lfinit(devfgeo) ;
      lfinit(devfhist) ;
      *mode = "rwo" ;
      dsopen(devfab,fabotra,mode) ;
      *mode = "ro" ;
      dsopen(devfgeo,fgeotra,mode) ;
      dsopen(devfhist,fhistcom,mode) ;
      return()

}
```

```
}

/* clezone :    [p18]
   --------

      obtenir une cle relative sur fgeozone a partir d'un numero de zone

*/
clezone(igeozone,offset,indzone)
int igeozone offset indzone ;
{       extern int nbzone ;
        extern struct index tabizone[nbmzone] ;
        int i j m ;
        offset = 0 ;
        i = 1 ;
        j = nbzone ;
        while (i <= j)
           { m = (i + j) / 2 ;
             if (igeozone = tabizone[m].cle)
                { offset = tabizone[m].offset ;
                  indzone = m ;
                  return() } ;
             if (igeozone < tabizone[m])
                j = m - 1
             else
                i = m + 1 ;
           } ;
        indzone = j ;
        return()
}

/*=============================================================================*/

/* lzone :    [p13]
   ------

       lire un enregistrement de zone a partir de l'indicatif de zone

*/
lzone(igeozone)
int igeozone ;
{      int offset ;
       clezone(igeozone,offset,indzone) ;
       if (offset == 0) {
          igeozone = -1 ;
          return() } ;
       lfseek(devfgeo,offset) ;
       lfread(devfgeo,&zone,sizeof(zone)) ;
       return()

}

/*=============================================================================*/
```

```
    }
/* inifco :    [p54]
   -------

        initialisation et ouverture des fichiers accedes par cocom et colig

*/
inifco()
{       char *mode ;
        lfinit devfab2 ;  /* devfab2 est le devptr du fabonne de cocom/colig */
        *mode = "rwo" ;
        dsopen(devfab2,fabococ,mode) ;
        return()

}

/*=============================================================================*/

/* clofco :    [p55]
   -------

        cloture des fichiers accedes par cocom/colig

*/
clofco()
{
        lfclose(devfab2) ;
        return()

}

/*=============================================================================*/

/* lab2 :    [p10]
   -----

        lire un enregistrement d'abonne dans le fabonne de cocom/colig a partir
du numero d'abonne

*/
lab2(numabo)
int numabo ;
{       int offset ;
        cleabo(numabo,offset,indabo) ;
        if (offset == 0) {
           numabo = -1 ;
           return() }
        lfseek(devfab2,offset) ;
        lfread(devfab2,&abonne,sizeof(abonne)) ;
        return()

}

/*=============================================================================*/
```

```
}

/* chgiabo :    [p49]
   --------

        chargement de l'index des numeros d'abonne

*/
chgiabo()
{
      char *mode ;
      int i ;
      extern int nbabo ;
      *mode = "ro" ;
      dsopen (deviabo,ifabonne,mode) ;
      for (i=1 ; ; i++) {
         lfread(deviabo,tabiabo[i],sizeof(tabiabo[i])) ;
         if (eof) break }
      nbabo = j - 1 ;
      lfclose(deviabo) ;
      return()

}

/*=============================================================================*/

/* chgilig :    [p50]
   --------

        chargement de l'index des numeros de ligne

*/
chgilig()
{
      char *mode ;
      int i ;
      extern int nbabo ;
      *mode = "ro" ;
      dsopen(devilig,ifablig,mode) ;
      for (i = 1 ; i <= nbabo ; i ++)
         lfread(devilig,tabilig[i],sizeof(tabilig[i])) ;
      lfclose(devilig) ;
      return()

}

/*=============================================================================*/
```

```
}

/* chgizone :    [p51]
   ---------

          chargement de l'index des indicatifs de zone

*/
chgizone()
{
      char *mode ;
      int i ;
      extern int nbzone ;
      *mode = "ro" ;
      dsopen(devizone,ifgeozone,mode) ;
      for (i = 1 ; ; i ++) {
         lfread(devizone,tabizone[i],sizeof(tabizone[i])) ;
         if (eof) break }
      nbzone = i - 1 ;
      lfclose(devizone) ;
      return()

}

/*=============================================================================*/

/* iniiabo :    [p27]
   --------

         initialiser et ouvrir l'index de numeros d'abonnes

*/
iniiabo()
{
      lfinit(deviabo) ;
      return()

}

/*=============================================================================*/

/* iniilig :    [p28]
   --------

          initialiser et ouvrir l'index des numeros de ligne

*/
iniilig()
{
      lfinit(devilig) ;
      return()

}
```

```
/* iniizone :    [p29]
   ---------

          initialiser et ouvrir l'index des indicatifs de zone

*/
iniizone()
{
     lfinit(devizone) ;
     return()

}

/*=============================================================================*/

/* cloiabo :    [p30]
   --------

          cloturer ifabonne

*/
cloiabo()
{
     lfclose(deviabo) ;
     return()
}

/*=============================================================================*/

/* cloilig :    [p31]
   --------

          cloturer ifablig

*/
cloilig()
{
     lfclose(devilig) ;
     return()
}

/*=============================================================================*/

/* cloizone :    [p32]
   ---------

           cloturer ifgeozone

*/
cloizone()
{
     lfclose(devizone) ;
     return()
}

/*=============================================================================*/
```

```
/* aabo :    [p60]
   -----

         ajouter un abonne de cle numabo

*/
aabo(succes)
int succes ;
{
/* declarations */
     extern int nbabo ;
     succes = 1 ;

/* positionnement en fin de fichier */
     lfseek(devfab,(flptr->fl_dent)->fdlen) ;
     lfseek(devfab2,(flptr->fl_dent)->fdlen) ;

/* incrementation du nombre d'abonnes */
     nbabo += 1 ;

/* insertion des index */
     cleabo(abonne.numabo,offset,indabo) ;
     if (offset != 0)
        { nbabo -= 1 ;
          succes = 0 ;
          return() }
     clelig(abonne.numlig,offset,indlig) ;
     if (offset != 0)
        { nbabo -= 1 ;
          return(syserr) }
     insiabo(abonne.numabo,(flptr->fl_dent)->fdlen + 1,indabo) ;
     insilig(abonne.numlig,(flptr->fl_dent)->fdlen + 1,indlig) ;

/* ecriture du record */
     eabo(abonne.numabo) ;
     eab2(abonne.numabo) ;

     return()

}

/*=============================================================================*/

/* sviabo :    [p46]
   -------

         sauver l'index des abonnes

*/
sviabo()
{
     char *mode ;
     int i ;
     extern int nbabo ;
     extern struct index tabiabo[] ;
     *mode = "wo" ;
     dsopen(deviabo,ifabonne,mode) ;
     for (i = 1 ; i <= nbabo ; i ++)
        lfwrite(deviabo,&tabiabo[i],sizeof(tabiabo[i])) ;
     lfclose(deviabo) ;
     return()

}
```

```
/* svilig :   [p47]
   -------

        sauver l'index des lignes

*/
svilig()
{
     char *mode ;
     int i ;
     extern int nbabo ;
     extern struct index tabilig[] ;
     *mode = "wo" ;
     dsopen(devilig,ifablig,mode) ;
     for (i = 1 ; i <= nbabo ; i ++)
        lfwrite(devilig,&tabilig[i],sizeof(tabilig[i])) ;
     lfclose(devilig) ;
     return()

}

/*=============================================================================*/

/* svizone :   [p48]
   --------

        sauver l'index des zones geographiques

*/
svizone()
{
     char *mode ;
     int i ;
     extern int nbzone ;
     extern struct index tabizone[] ;
     *mode = "wo" ;
     dsopen(devizone,ifgeozone,mode) ;
     for (i = 1 ; i <= nbzone ; i ++)
        lfwrite(devizone,&tabizone[i],sizeof(tabizone[i])) ;
     lfclose(devizone) ;
     return()

}

/*=============================================================================*/

/* cloftra :   [p53]
   --------

         cloture des fichiers utilises par tratax

*/
cloftra()
{
     lfclose(devfab) ;
     lfclose(devfgeo) ;
     lfclose(devhist) ;
     return()

}
```

```
/* dtriabo :    [p37]
   --------

            detruire un...

*/
dtriabo(indabo)
int indabo ;
{
     int i ;
     extern int nbabo ;
     extern struct index tabiabo[] ;
     for (i = indabo ; i < nbabo ; i --)
        tabiabo[i] = tabiabo[i+1] ;
     return()

}

/*=============================================================================*/

/* dtrilig :    [p38]
   --------

          detruire un index ligne

*/
dtrilig(indlig)
int indlig ;
{
     int i ;
     extern int nbabo ;
     extern struct index tabilig[] ;
     for (i = indlig ; i < nbabo ; i --)
        tabilig[i] = tabilig [i+1] ;
     return()

}

/*=============================================================================*/

/* dtrizone :    [p39]
   ---------

              detruire un index zone

*/
dtrizone(indzone)
int indzone ;
{
     int i ;
     extern int nbzone ;
     extern struct index tabizone[] ;
     for (i = indzone ; i < nbzone ; i --)
        tabizone[i] = tabizone[i+1] ;
     return()

}
```

```
/* eabo :    [p11]
   -----

          ecrire un record abonne de cle numabo dans fabonne

             input : - numabo = 0 : ecriture en fin de fichier
                         verifier que numabo n'est pas double
                     - numabo > 0 : reecriture de l'abonne de cle numabo
                         pas de verification sur numabo
                     - numabo < 0 : ecriture sequentielle
                         la fin du fichier est positionnee a la fin de
                         l'article ecrit
                     - le record se trouve dans abonne

             output : - numabo = -1 : erreur

*/
eabo()
{
     int offset,indabo,numabo ;
     extern struct tabo abonne ;
     extern struct devsw *devfab ;
     extern struct flblk *fabptr ;
     numabo = abonne.numabo ;
     if (numabo == 0) {
        cleabo(numabo,offset,indabo) ;
        if (offset != 0) {
           numabo = -1 ;
           return() }
        lfseek(devfab,(fabptr->fl_dent)->fdlen) }
     else if (numabo > 0) {
        cleabo(numabo,offset,indabo) ;
        if (offset == 0) {
           numabo = -1 ;
           return() } ;
        lfseek(devfab,offset) }
     else (fabptr->fl_dent)->fdlen = fabptr->fl_pos ;
     lfwrite(devfab,abonne,sizeof(abonne)) ;
     return()

}

/*=============================================================================*/
```

```
/* eab2 :    [p12]
   -----

              ecrire un record abonne de cle numabo dans fab2
                memes regles que pour eabo

*/
eab2()
{
     int offset indabo numabo ;
     extern struct tabo abonne ;
     extern struct devsw *devfab2 ;
     extern struct flblk *fab2ptr ;
     numabo = abonne.numabo ;
     if (numabo == 0) {
        cleabo(numabo,offset,indabo) ;
        if (offset != 0) {
           numabo = -1 ;
           return() }
        lfseek(devfab2,(fab2ptr->fl_dent)->fdlen) }
     else if (numabo > 0) {
        cleabo(numabo,offset,indabo) ;
        if (offset == 0) {
           numabo = -1 ;
           return() }
        lfseek(devfab2,offset) }
     else (fab2ptr->fl_dent)->fdlen = fab2ptr->fl_pos ;
     lfwrite(devfab2,abonne,sizeof(abonne)) ;
     return()

}

/*=============================================================================*/

/* ezone :    [p14]
   ------

              ecrire une zone geographique
                memes regles que pour eabo

*/
ezone()
{
     int offset indzone ;
     extern struct tzone zone ;
     extern struct devsw *devfgeo ;
     extern struct flblk *fgeoptr ;
     igeozone = zone.igeozone ;
     if (igeozone == 0) {
        clezone(igeozone,offset,indzone) ;
        if (offset != 0) {
           igeozone = -1 ;
           return() }
        lfseek(devfgeo,(fgeoptr->fl_dent)->fdlen) }
     else if (igeozone > 0) {
        clezone(igeozone,offset,indzone) ;
        if (offset == 0) {
           igeozone = -1 ;
           return() }
        lfseek(devfgeo,offset) }
     else (fgeoptr->fl_dent)->fdlen = fgeoptr->fl_pos ;
     lfwrite(devfgeo,zone,sizeof(zone)) ;
     return()

}
```

```
/* dabo :    [p15]
   -----

         destruction d'un abonne de cle numabo

*/
dabo(numabo)
int numabo ;
{
/* declarations */
     int offset indabo ;
     extern struct tabo abonne abo2 ;
     extern struct devsw *fabptr *fab2ptr ;

/* positionnement sur le dernier record */
     lfseek(devfab,(fabptr->fl_dent)->fdlen-sizeof(abonne)) ;
     lfseek(devfab2,(fab2ptr->fl_dent)->fdlen-sizeof(abo2)) ;

/* lecture du dernier record */
     lfread(devfab,abonne,sizeof(abonne)) ;
     lfread(devfab2,abo2,sizeof(abo2)) ;

/* recherche de l'offset du record numabo */
     cleabo(numabo,offset,indabo) ;
     if (offset == 0) {
        numabo = -1 ;
        return() }

/* positionnement sur le record numabo */
     lfseek(devfab,offset) ;
     lfseek(devfab2,offset) ;

/* ecriture du dernier record a la place de celui d'index numabo */
     lfwrite(devfab,&abonne,sizeof(abonne)) ;
     lfwrite(devfab2,&abo2,sizeof(abo2)) ;

/* diminution de la longueur du fichier */
     (fabptr->fl_dent)->fdlen -= sizeof(abonne) ;
     (fab2ptr->fl_dent)->fdlen -= sizeof(abo2) ;

/* destruction de l'index */
     dtriabo(indabo) ;

     return()

}

/*=============================================================================*/
```

```
/* dzone :    [p15]
   ------

         destruction d'une zone

*/
dzone(igeozone)
int igeozone ;
{
     int offste indzone ;
     extern struct tzone zone ;
     extern struct devsw *devfgeo ;
     extern struct flblk *fgeoptr ;
     lfseek(devfgeo,(fgeoptr->fl_dent)->fdlen - sizeof(zone)) ;
     lfread(devfgeo,zone,sizeof(zone)) ;
     clezone(igeozone,offset,indzone) ;
     if (offset == 0) {
        igeozone = -1 ;
        return }
     lfseek(devfgeo,offset) ;
     lfwrite(devfgeo,&zone,sizeof(zone)) ;
     (fgeoptr->fl_dent)->fdlen -=sizeof(zone) ;
     dtrizone(igeozone) ;
     return()

}

/*=============================================================================*/

/* insiabo :    [p34]
   --------

         inserer un index abonne

*/
insiabo(numabo,offset,indabo)
int numabo offset indabo ;
{
     extern struct index tabiabo[] ;
     int i ;
     extern int nbabo ;
     for (i = indabo + 1 ; i <= nbabo ; i ++)
        tabiabo[i] = tabiabo[i-1] ;   /* nbabo suppose augmente de 1 */
     tabiabo[indabo].cle = numabo ;
     tabiabo[indabo].offset = offset ;
     return()

}

/*=============================================================================*/
```

```
}

/* insilig :    [p35]
   --------

            inserer un index ligne

*/
insilig(numlig,offset,indlig)
int numlig offset indlig ;
{
     extern struct index tabilig[] ;
     int i ;
     extern int nbabo ;
     for (i = indlig + 1 ; i <= nbabo ; i++)
        tabilig[i] = tabilig[i-1] ;  /* nbabo suppose augmente de 1 */
     tabilig[indlig].ligne = numlig ;
     tabilig[indlig].offset = offset ;
     return()

}

/*=============================================================================*/

/* insizone :    [p36]
   ---------

            inserer un index zone

*/
insizone(igeozone,offset,indzone)
int igeozone offset indzone ;
{
     extern struct index tabizone[] ;
     int i ;
     extern int nbzone ;
     for (i = indzone + 1 ; i <= nbzone ; i++)
        tabizone[i] = tabizone[i-1] ;  /* nbzone suppose augmente de 1 */
     tabizone[indzone].zone = igeozone ;
     tabizone[indzone].offset = offset ;
     return()

}

/*=============================================================================*/
```

```
/* azone :   [p61]
   ------

          ajouter une zone geographique

*/
azone(succes)
int succes ;
{
/* declarations */
     extern int nbzone ;
     succes = 1 ;

/* positionnement en fin de fichier */
     lfseek(devfgeo,(fgeoptr->fl_dent)->fdlen) ;

/* incrementation du nombre de zones */
     nbzone += 1 ;

/* insertion d'un index */
     clezone(zone.igeozone,offset,indzone) ;
     if (offset != 0) {
        nbzone -= 1 ;
        succes = 0 ;
        return() } ;
     insizone(zone.igeozone,(fgeoptr->fl_dent)->fdlen+1,indzone) ;

/* ecriture du record */
     ezone(zone.igeozone) ;

     return()

}

/*=============================================================================*/

/* llig :   [p19]
   -----

          lecture aleatoire d'un abonne de ligne donnee ; numlig <= 0
          donne lieu a une erreur ; output : erreur ==> numlig == 0 ;
          lecture dans fabonne

*/
llig(numlig)
int numlig ;
{
     int offset indlig ;
     extern struct tabo abonne ;
     extern struct devsw *devfab ;
     if (numlig <= 0) {
        numlig = -1 ;
        return } ;
     clelig(numlig,offset,indlig) ;
     if (offset == 0) {
        numlig = -1 ;
        return } ;
     lfseek(devfab,offset) ;
     lfread(devfab,abonne,sizeof(abonne)) ;
     return()

}
```

```
/* llig2 :    [p20]
   ------

          lire un abonne de ligne donnee dans fab2 ; memes regles que
          dans llig

*/
llig2(numlig)
int numlig ;
{
     int offset indlig ;
     extern struct tabo abonne ;
     extern struct devsw *devfab2 ;
     if (numlig <= 0) {
        numlig = -1 ;
        return() } ;
     clelig(numlig,offset,indlig) ;
     if (offset == 0) {
        numlig = -1 ;
        return() } ;
     lfseek(devfab2,offset) ;
     lfread(devfab2,abonne,sizeof(abonne)) ;
     return()

}

/*=============================================================================*/

/* elig :    [p21]
   -----

           ecrire un article de ligne donnee ; memes regles que pour eabo

*/
elig()
{
     int offset indlig numlig ;
     extern struct tabo abonne ;
     extern struct devsw *devfab ;
     extern struct flblk *fabptr ;
     numlig = abonne.numlig ;
     if (numlig == 0) {
        clelig(numlig,offset,indlig) ;
        if (offset != 0) {
           numlig = -1 ;
           return() }
        lfseek(devfab,(fabptr->fl_dent)->fdlen) }
     else if (numlig > 0) {
        clelig(numlig,offset,indlig) ;
        if (offset == 0) {
           numlig = -1 ;
           return() } ;
        lfseek(devfab,offset) }
     else (fabptr->fl_dent)->fdlen = fabptr->fl_pos ;
     lfwrite(devfab,abonne,sizeof(abonne)) ;
     return()

}

/*=============================================================================*/
```

```
/* elig2 :   [p22]
   ------

        ecrire un abonne de ligne donnee dans fab2 ; memes regles que
        pour sebo

}
elig2()
{
    int offset indlig numlig ;
    extern struct tabo abonne ;
    extern struct devsw *devfab2 ;
    extern struct flblk *fab2ptr ;
    numlig = abonne.numlig ;
    if (numlig == 0) {
       clelig(numlig,offset,indlig) ;
       if (offset != 0) {
          numlig = -1 ;
          return() }
       lfseek(devfab2,offset) }
    else if (numlig > 0) {
       clelig(numlig,offset,indlig) ;
       if (offset == 0) {
          numlig = -1 ;
          return() }
       lfseek(devfab2,offset) }
    else (fab2ptr->fl_dent)->fdlen = fab2ptr->fl_pos ;
    lfwrite(devfab2,abonne,sizeof(abonne)) ;
    return()

}

/*=============================================================================*/

/* clelig :   [p23]
   -------

        trouver l'offset et l'indice de la cle ligne donnee ; si la cle
        n'existe pas , l'indice est celui de la cle immediatement
        inferieure a celle demandee et offset = 0

*/
clelig(numlig,offset,indlig)
int numlig offset indlig ;
{
    extern int nbabo ;
    extern struct index tabilig[] ;
    int i j m ;
    i = 1 ;
    j = nbabo ;
    while (i <= j) {
       m = (i+j)%2 ;
       if (numlig = tabilig[m].cle) {
          offset = tabilig[m].offset ;
          indlig = m ;
          return() } ;
       if (numlig < tabilig[m].cle) {
          j = m - 1
       else
          i = m + 1 } ;
    offset = 0 ;
    indlig = j ;
    return()

}

/*=============================================================================*/
```

Annexe 2 :

Caractéristiques essentielles des systèmes étudiés.

Annexe 2.1 : RTM-80.

I. GESTION DES TACHES

Une tâche est un programme connu du système exécutif par :

- une adresse de départ (première instruction de la tâche),
- une adresse de stack (chaque tâche ayant son stack propre)
- un niveau de priorité (numéroté de 1 à 16).

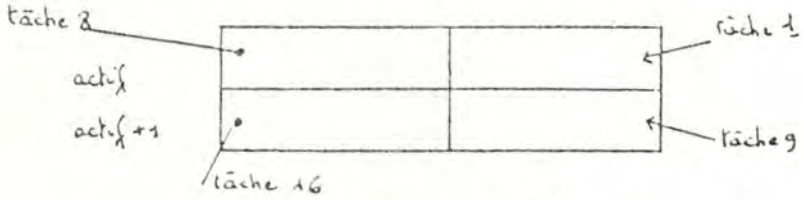
En outre, la dernière instruction exécutable d'une tâche est nécessairement :

```
JMP        RTMON
```

(voir § 4.).

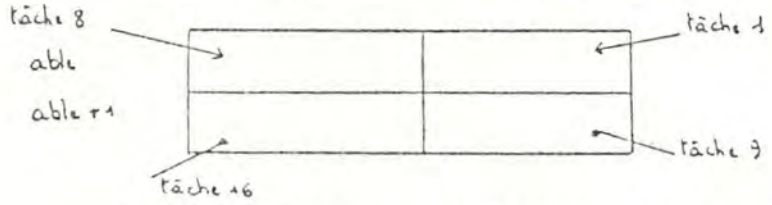
1. PRINCIPES

Une tâche peut se trouver active (en train de tourner) ou non active (au repos). Le mot "actif" a été créé pour mémoriser l'état des 16 tâches.

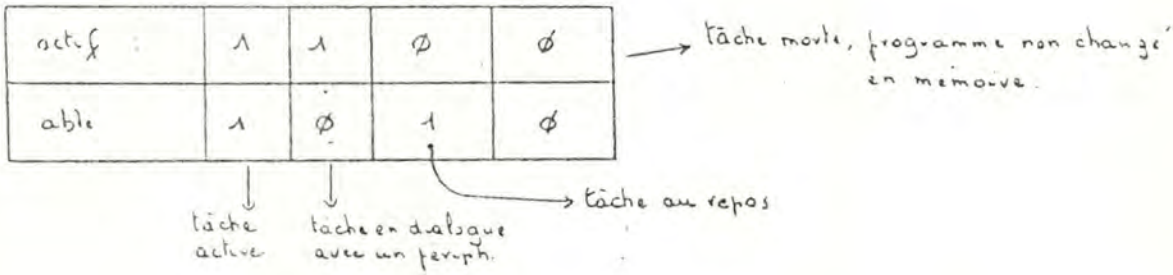


inscrire un bit = 1 — acti
un bit = ø — repo

Mais, une tâche active peut dialoguer avec un périphériqu
Nous sommes donc obligés de créer un deuxième mot : "able
représentant l'état des 16 tâches vis-à-vis du périphériq
Une tâche en dialogue avec un périphérique aura : able =



Quatre états définissant la tâche sont possibles :



Une tâche au repos est able = 1

Pour savoir quand il faut démarrer une tâche, il suffit d tester quand actif et able seront égaux à 1 en même temps C'est le rôle du scheduler.
Toutes les 20 ms, le scheduler testera les 16 tâches.

(Il est possible d'initialiser le système pour une périod différente, par exemple de 10 msec.)

2. QU'ENTEND-ON PAR DEMARRER UNE TACHE ?

Si une tâche est interrompue (par un interrupt), elle sauve l'état de ses registres et son PC sur un stack. Nous appelons cela le sauvetage du contexte.
A la fin de l'interrupt, la tâche reprend son contexte du stack et redémarre là où elle a été interrompue.

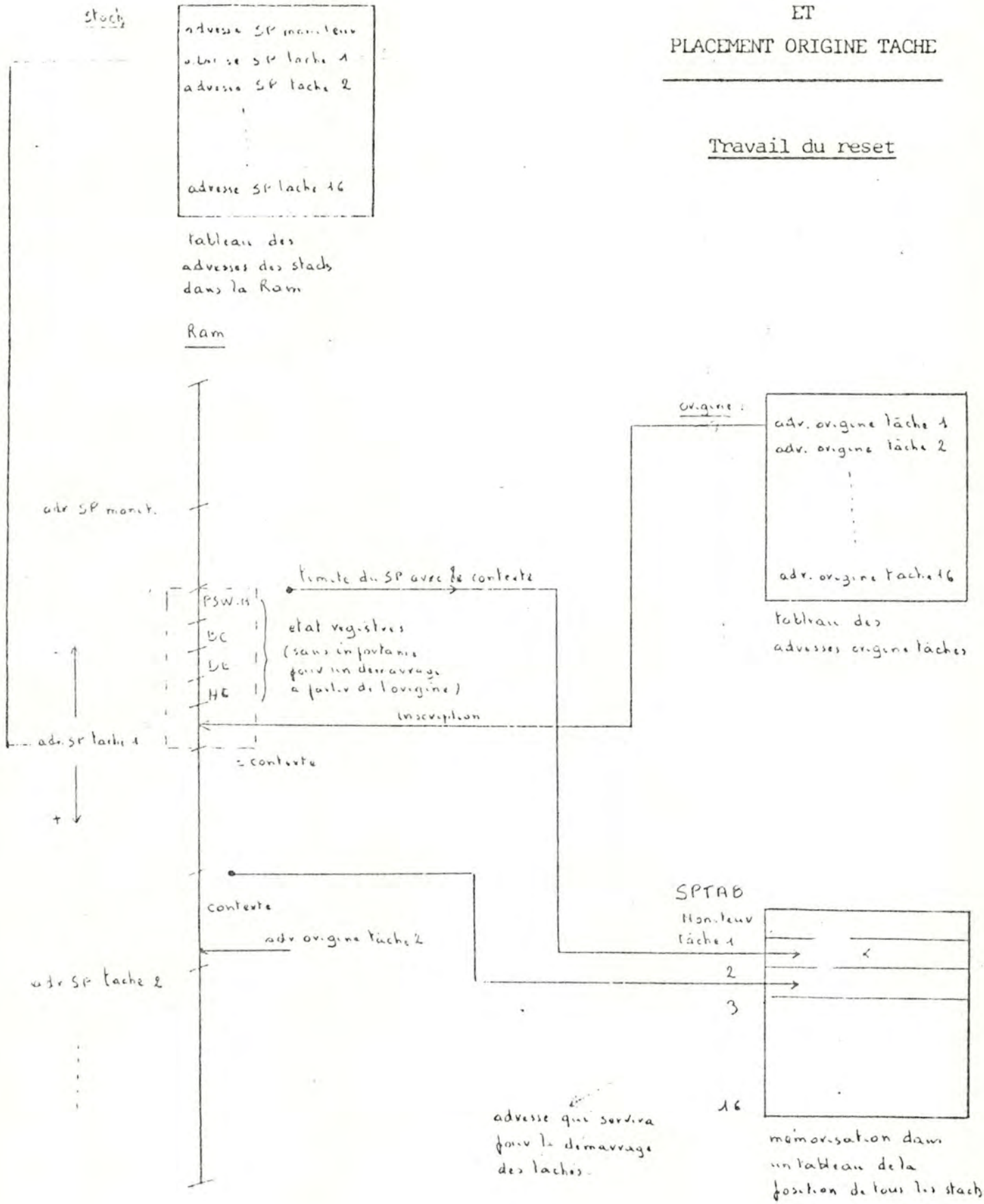
Si une tâche doit démarrer à partir de l'origine, il suffit de suivre le même principe et de placer un contexte sur un stack comprenant un PC de retour = adresse origine de la tâche.

La phase d'initialisation du moniteur (appelée dans le programme "reset") s'occupe du placement des adresses origine tâches dans les stack des tâches.
(Voir schéma page suivante.)

Chaque tâche possède son propre stack.

INITIALISATION DES STACK ET PLACEMENT ORIGINE TACHE

Travail du reset



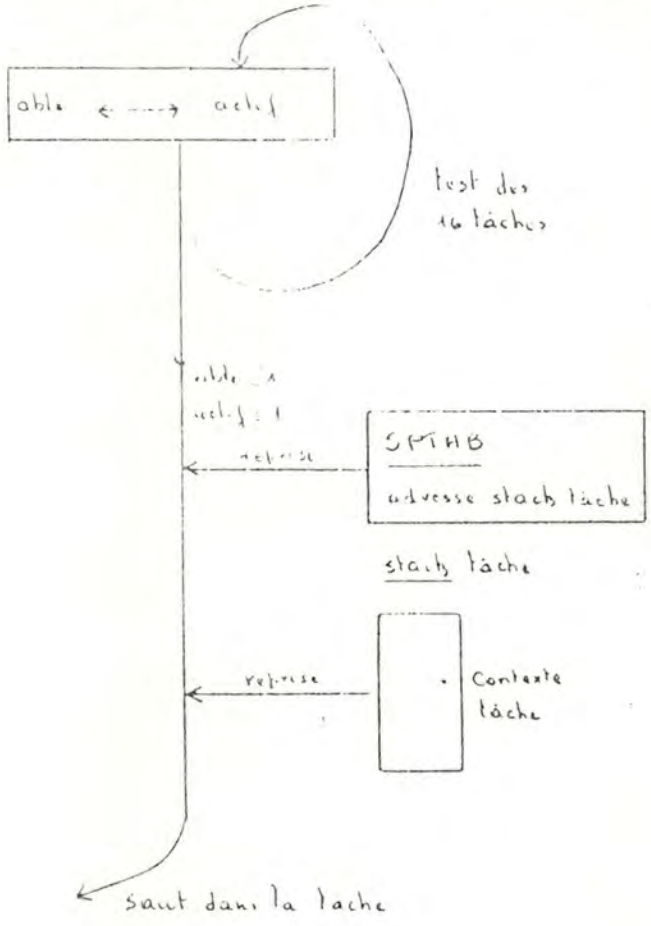
Dès que les origines des tâches sont placées, il suffit de descendre le stack de 8 pour trouver l'adresse limite du contexte.

```
PSW-A   -8 ←——
 BC     -6
 DE     -4
 HL     -2
 ←——
 PC  (origine tâche)
adresse stack → X
```

C'est une adresse qu'il faudra mémoriser, car elle servira à faire la restitution du contexte (registres + PC).
Cette adresse est sauvée dans le tableau SPTH

3. LE SCHEDULER

Dès lors, pour démarrer une tâche, nous aurons :



Schéduler
Idée générale

Or, l'initialisation des tables (le reset), l'appel des routines, s'effectue à l'aide du stack du moniteur. Le scheduler, pour le test de able et actif utilise le stack du moniteur. Il y a donc un changement de stack pour le démarrage d'une tâche (abandon avec mémorisation du SP moniteur et reprise SP tâche).

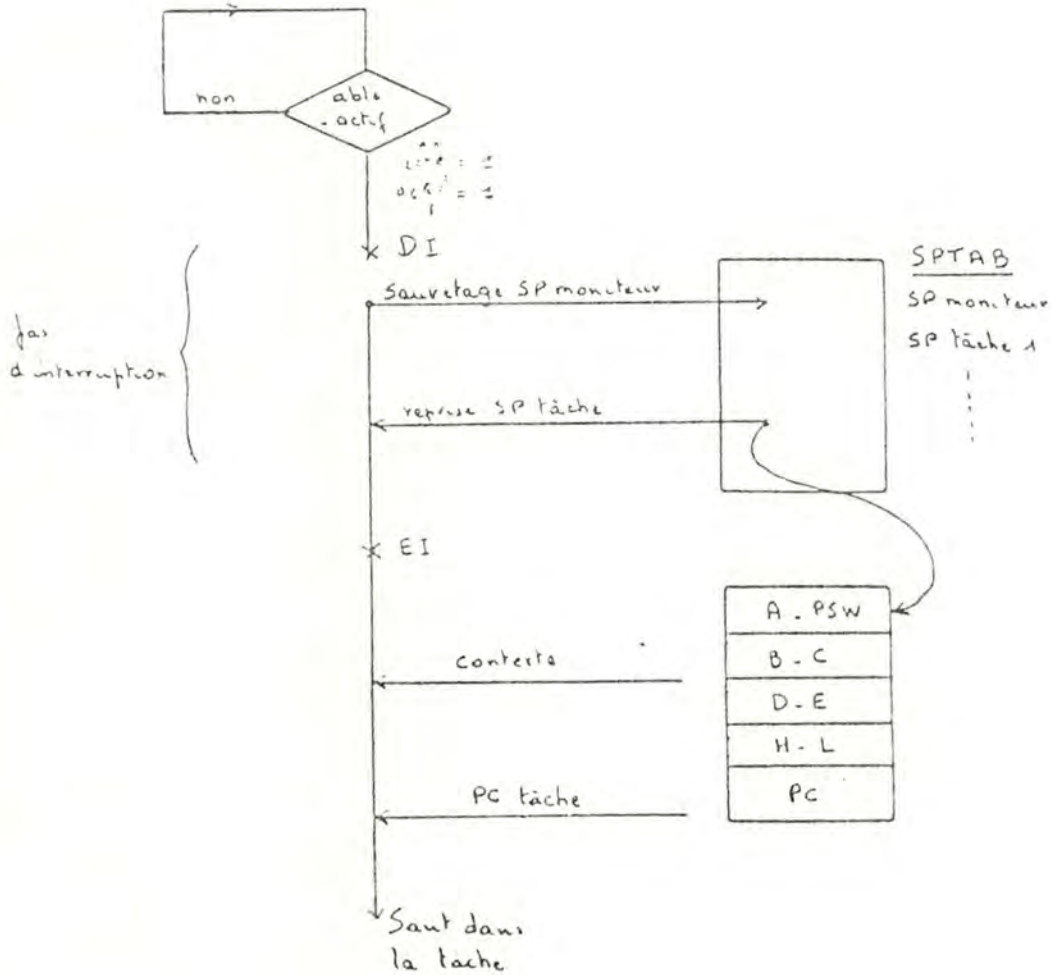
A tout instant, pour connaître le numéro de la tâche en cours, on a créé le byte "taska".

Taska contient donc en binaire le numéro de la tâche acti

Par définition, { une tâche est représentée par son numéro de tâche dans taska ;
le moniteur est représenté par taska = ø.

On peut donc schématiser le scheduler comme suit :

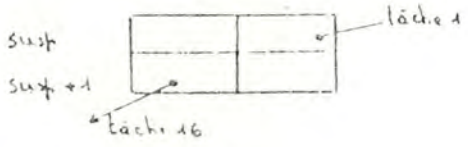
Scheduler



En outre, on met à la disposition du software d'application fonctions suspension et désuspension de tâches.

La suspension d'une tâche étant directement liée au démarrage de la tâche, on a décidé de tester également cette suspension dans le scheduler.

Le mot "susp" a été créé. Suspendre une tâche revient à mettre le bit à 1.



Pour démarrer une tâche, on fait une opération logique ("et" logique) entre able et actif.

On placera le susp dans cette opération logique et on fera

"et" logique { able, actif, $\overline{\text{susp}}$ (complément par rapport à 1)

Le diagramme du scheduler est presque complet. Il doit cependant être complété pour satisfaire les propriétés des interrupts (voir § II.).

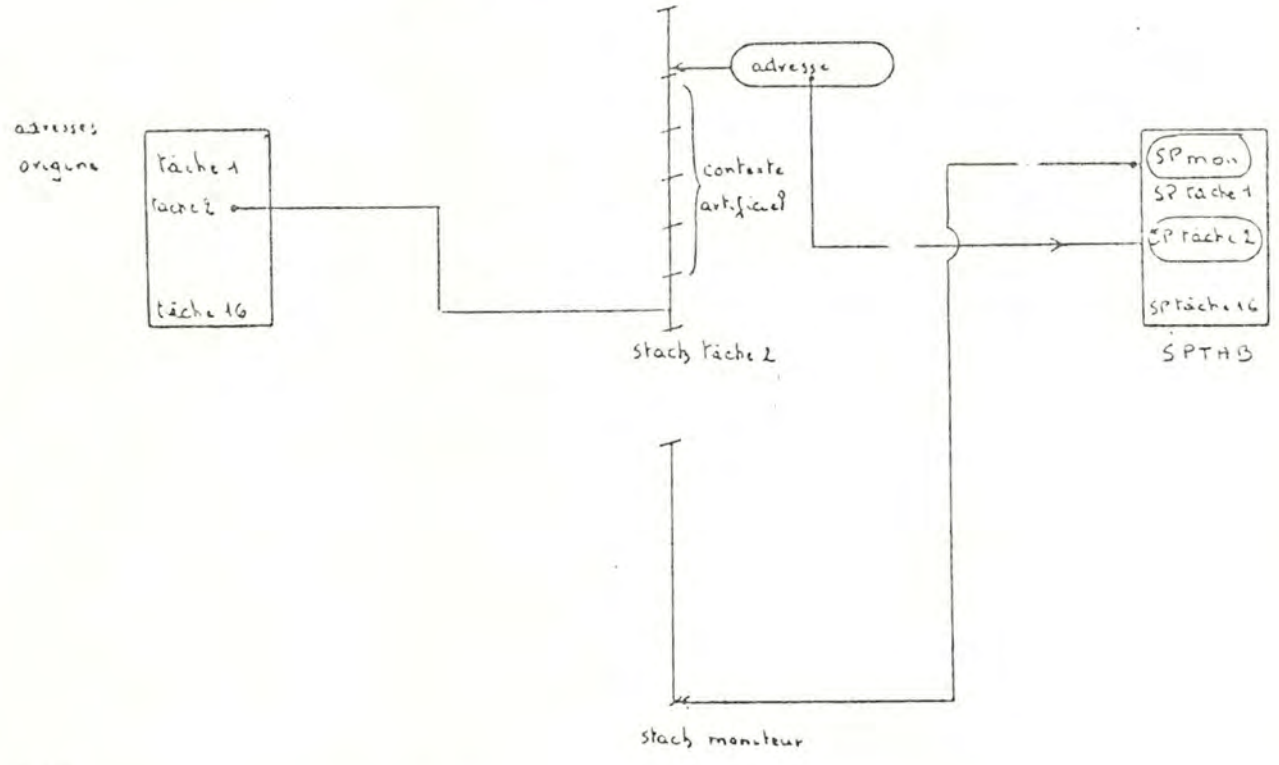
. FIN DE TACHE

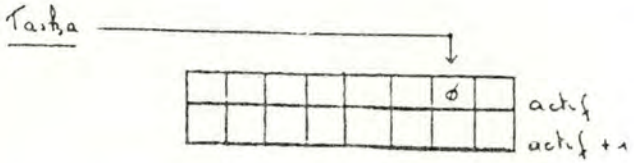
A la fin de chaque tâche, il faut redonner la main au scheduler. Cette opération est faite par la routine du moniteur RTMON.

Cette routine comporte 7 phases :

1) rechercher l'origine de la tâche qui se termine et placer cette origine dans le stack ;
2) rechercher le stack pointer à sauver dans le SPTAB ;
3) désactiver la tâche ;
4) sauver la SP tâche dans le SPTAB ;
5) reprendre les paramètres moniteur : taska = ø ;
6) reprise du SP moniteur de SPTAB ;
7) saut dans le scheduler.

Exemple : fin tâche n° 2.





Tasha → A

B ← A

Recherche origine tache → DE

PUSH D

Recherche SP tâche à sauver
HL = -8
DAD SP
XCHG ; DE = SP tâche

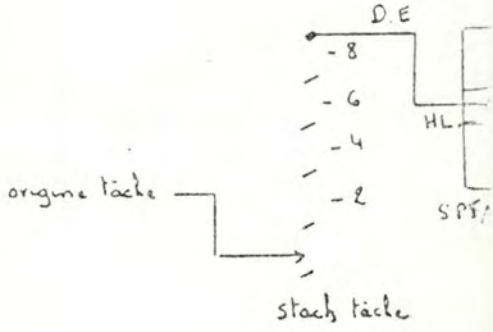
Recherche dans HL
SPTAB + 2. tasha ; HL = SPTAB + 2. tasha

DI

A ← B

Désactivation tâche

Mise en mémoire dans SPTAB
M ← E
HL + 1
M ← D



tasha = ∅

LHLD STAB
SPHL
} Reprise stack moniteur

EI

jmp SCHED

RTMON

Annexe 2.2 : iRMX 80/88.

Les systèmes en temps réel iRMX 80/86.

Les systèmes iRMX fournissent des fonctions nécessaires au développement d'applications destinés à tourner sur des architectures mono-processeur.

Le iRMX 80 est utilisé avec des architectures basées sur les micro-processeur 8080 et 8085. Le iRMX 88 est utilisé avec une architecture iAPX 88 et iAPX 86.

Voici un bref synopsis des fonctions d'un système iRMX:

- le noyau est en quelque sorte un mélange de policier de carrefour et d'arbitre, il coordonne toutes les activités concurrentes dans un système applicatif basé sur iRMX.
- le gérant du terminal réalise les I/O entre un terminal opérateur et les tâches schedulées par le noyau.
- le gérant des espaces libres maintient un pool des espaces de mémoire vive libre, alloue des blocs de cette mémoire aux tâches demanderesses et récupére les blocs libérés par les tâches.
- le disk system file fournit des possibiltés de gestion de fichier.
- le bootstrap loader se charge du chargement d'une application basée sur le système iRMX et des fichiers qui lui sont propres.
- le analog handler réalise la conversion analogique-digital pour les tâches faisant des inputs et la conversion digital-analogique pour les tâches faisant des outputs.

Table 1-1. iRMX 80 and iRMX 88 Components

Component	IRMX 80	IRMX 88
Nucleus	x	x
Terminal Handler	x	x
Free Space Manager	x	x
Disk File System	x	future
Bootstrap Loader	x	future
Analog Handlers	x	future
Command Line Interpreter	x	future
Debugger	x	future

Nous donnons ici une présentation des différentes procédures des systèmes iRMX qui auraient pu soutenir notre application si nous avions pu utiliser pareil système.
Le lecteur désirant approfondir ses connaissances sur iRMX pourra toujours se référer au manuel de l'utilisateur.

A. La notion de tâche.

Une tâche est un élément actif dans un système iRMX
Elle peut perçu de deux manières:

- du point de vue du programmeur, une tâche est une entité qui exécute un programme. Lorsque plusieurs tâches coexistent dans un système, elles se partagent le CPU et même peuvent se partager le code.
- du point de vue du noyau, une tâche est un ensemble de valeurs de registres. Au moyen de ces valeurs, le noyau peut à tout moment exécuter une tâche.

Les états des tâches.

Une tâche est toujours dans un des quatre états d'exécution suivants:

- Running: la tâche est exécutée, à tout moment une seule tâche est dans cet état,
- Ready : la tâche est éligible à l'état Running remarquons que la tâche à l'état Running est la tâche à l'état Ready de plus haut priorité.
- Waiting: la tâche attend à un point d'échange, un message que doit lui envoyer une autre tâche.
- Suspended: une tâche peut se suspendre elle-même ou l'être par une autre tâche.

B. Procédures du noyau.

B.1 La procédure RQSUP.

Cette procédure a pour spécification de suspendre une tâche, donc de l'empêcher de pouvoir acquérir les ressouces du système.

Syntaxe: CALL RQSUSP(task-descriptor);

avec task-descriptor: une adresse conteneant l'adresse du descripteur de tâche de la tâche suspendue.

Description: RQSUP est réentrante et peut être appelée à n'importe quel moment par la tâche en état Running.

B.2 La procédure RQCTSK.

Cette procédure accepte un descripteur de tâche pour la tâche désignée, installe le stack de la tâche et place cette tâche dans l'état Ready.

Syntaxe: CALL RQCTSK(static-task-descriptor)

avec static-task-descriptor: un pointeur contenant l'adresse du descripteur de tâche de la tâche qui vient d'être crée.

Description: idem que B.1

B.3 La procédure RQDSTK.

Cette procédure détruit une tâche dans le système.

Syntaxe: CALL RQDTSK(task-descriptor)

avec task-descriptor: une adresse contena nt l'adresse du descripteur de tâche de la tâche à détruire.

Description: idem que B.1

B.4 Les priorités de la tâche.

Chaque tâche a une priorité dans l'intervalle 0-255.
Le système iRMX peut donc gérer jusqu'à 256 tâches.
Il existe une correspondance entre la priorité d'une tâche et les niveaux d'interruption masquables.

C. Les échanges.

Les tâches doivent souvent interagir, et pour ce faire ilest nécessaire de leur permettre des échanges mutuels.

La méthode d'interaction est qu'une tâche envoie un message et qu'une autre tâche reçoit ce message à un échange.

Chaque échange a deux files d'attente, une pour les tâches qui sont en attente de réception, l'autre pour les messages envoyés et non encore réceptionnés. Le

Une tâche envoie un message à un échange au moyen de la procédure RQSEND. A cet instant deux choses peuvent se passer: Si aucune tâche n'est en attente à cet échange, le message est placé à l'arrière de la file. La gestion de cette file est FIFO, D'autre part, s'il existe des tâches en attente; le message est remis à la tâche concernée et son état devient Ready.
Il est à remarquer que si la tâche réceptrice a un priorité plus élevée que la tâche émettrice, ce sera la première qui aura la possibilité d'être exécutée avant la seconde tâche.

Avant d'envoyer un message, une tâche doit construire le message dans la RAM. Chaque message est constitué d'un header section(de 5 à 9 bytes).

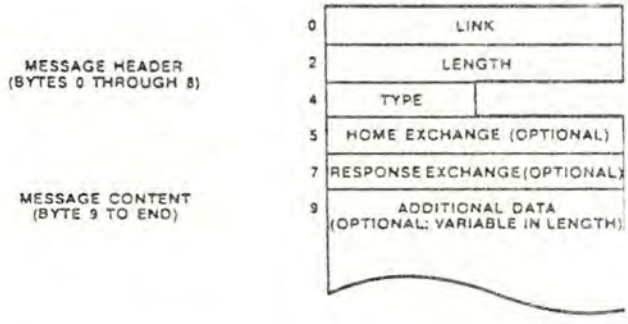


Figure 1-2. iRMX 88 Message Format

The LINK field of the message is used by the Nucleus to link together multiple messages waiting at the same exchange. This field should not be changed by sending tasks.

The sending task must set the LENGTH field equal to the length, in bytes, of the entire message, including both the header section and, if present, the message content section.

TYPE is a one-byte field containing a code identifying the type or purpose of the message. Type values 0 to 63 are reserved for iRMX 88-defined message types, while values 64 through 255 are available to you to define for your own purposes. A list of all currently defined iRMX 88 message types is given in Appendix C. For the convenience of PL/M users, iRMX 88 supplies an INCLUDE file named MSGTYP.LIT, which contains symbolic names for the message types.

The HOME EXCHANGE field, when used, contains the address of the exchange to which the message is to be sent when it is no longer needed. Even if this field is not used in messages longer than 5 bytes, space must be reserved for it. If the message is being sent to any of certain iRMX 88 exchanges, the field must be set as required by the exchange. In the descriptions of specific exchanges later in this manual, necessary information regarding this field is provided.

The RESPONSE EXCHANGE field, when used, contains the address of the exchange to which the sending task wants the receiving task to send a response message. Note that this implies that tasks receiving messages should, as a matter of routine, check this field and act accordingly. By using this field in this manner, tasks can synchronize their activities. Even if this field is not used in messages longer than 7 bytes, space must be reserved for it. If the message is being sent to any of certain iRMX 88 exchanges, the field must be set as required by the exchange. In the descriptions of specific exchanges later in this manual, necessary information regarding this field is provided.

C.1 La procédure RQSEND.

Cette procédure envoie un message à un échange.

Syntaxe: CALL RQSEND(exchange-address,message-adress)

avec exchange-address: l'adresse du descripteur de l'échange

message-address: l'adresse du début de message.

Description: RQSEND est réentrant et peut être appelé à n'importe quel moment.

C.2 La procédure RQWAIT.

Cette procédure retourne l'adresse d'un message à la tâche appelante.

Syntaxe: message-address = RQWAIT(exchange-adress, time-limit)

avec message-address: l'adresse dans laquelle est inscrite l'adresse du message

exchange-address: l'adresse du descripteur de l'échange où la tâche appelante veut recevoir le message

time-limit: un mot contenant soit zéro ou le nombre d'unités de temps que la tâche appelante attendra; le zéro signifie que la tâche attendra indéfiniment.

C.3 La procédure RQACPT.

Cette procédure accepte un message; soit elle retourne zéro auquel cas il n'ya pas de message destiné à cette tâche, soit un message.

Syntaxe: message-address= RQACPT(exchange-adress)

avec message-address: l'adresse égal soit à zéro soit à l'adresse du message

exchange-address:l'adresse du descripteur d'échange de la tâche appelante.

C.4 La procédure RQCXCH.

Cette procédure crée un échange en acceptant un descripteur d'échange.

Syntaxe : CALL RQCXCH(exchange-adress)

C.5 La procédure RQDXCH.

Cette procédure permet la destruction d'un échange.
La destruction n'est effectuée que s'il n'y a plus d'éléments dans les deux files constituantes de l'échange.

Syntaxe: CALL byte-variable=RQDXCH(exchange-address)

avec byte-variable: un byte dans lequel la valeur
OFFH est retournée si la
destruction est effectuée,
0 est retourné dans tout autre
circonstance.

Annexe 2.3 : Xinu.

INTRODUCTION

XINU est un système d'exploitation conçu selon le principe d'ordonnancement hièrarchique qui permet de construire des structures par niveaux et classes.

XINU incorpore les concepts de UNIX dans un système simple, facile à comprendre, à modifier, à porter parce qu'il est proprement partitionné en couches.

XINU peut tourner avec 4000 bytes (moins de 4 K) de mémoire centrale. L'ensemble des fichiers sources, avec les commentaires, représente à peine 5850 lignes de code en langage C et 650 lignes de code assembleur. Sans les commentaires, nombreux et clairs, ces chiffres tombent à 4300 lignes de C et 550 lignes d'assembleur.

XINU est conçu comme un système en time sharing c'est-à-dire permettant la multiprogrammation avec une politique de partage du processeur égal pour chaque utilisateur. Il fournit également les moyens de communication entre machines grâce à un réseau en anneau.

Ce système d'exploitation fut initialement implémenté pour des processeurs LSI 11 équipant les ordinateurs PDP 11.

ORGANISATION HIERARCHIQUE

Les composants du système sont organisés en une hiérarchie de couches rendant les interconnexions entre elles claires.

(FIGURE 1)

Au coeur de la structure, se trouve le hardware. A partir de là, des couches de logiciel fournissent des primitives de plus en plus puissantes et cachent la machine aux utilisateurs. Chaque couche du système fournit un service abstrait, implémenté en termes des services abstraits fournis par les couches de niveaux inférieurs.

L'essence d'un système d'exploitation se trouve dans les services qu'il peut fournir aux programmes utilisateurs. Les programmes accèdent à ces services en faisant des "appels système". Ces appels système ressemblent à des appels de procédure et sont connus sous le nom de "primitive". Les primitives définissent les services que le système fournit et l'interface à ces services.

Sans les couches 7 et 8, c'est-à-dire le système de gestion de fichiers et le réseau, on obtient un système minimal qui sera décrit dans le cadre de cette étude. Chaque niveau sera quelque peu présenté. Suivent ensuite une éventuelle description des primitives ainsi que des quelques routines dépendantes de la machine et qui sont écrites en assembleur.

COUCHE 0 : LA MACHINE.

XINU fut implémenté par Douglas Comer sur un micro-processeur 16 bits de chez Digital Equipment Corporation : le LSI 11/02. Celui-ci équipe le mini-ordinateur PDP 11.

Le LSI 11/02 est construit à partir de 3 cartes de circuit intégré reliées par un bus :

- le processeur
- la mémoire
- l'interface terminal.

Les sorties (entrées) sont effectuées en écrivant (lisant) à des adresses comprises dans les 8 derniers K bytes de l'espace d'adressage.

Quand un device reconnait son adresse sur le bus, soit il transmet des données au CPU, soit il réceptionne des données en provenance du CPU, soit il contrôle le device. Les transmissions sont effectuées par DMA (Direct Memory Access) c'est-à-dire que le device utilise le bus pour communiquer directement avec la mémoire.

routines Assembleur

En cas d'appel de procédure, la procédure appelée sauve et restaure les registres respectivement avant et après l'exécution du corps de la procédure.

CSV

Routine de sauvetage des registres.

CRET

Routine de restauration des registres.

COUCHE 1 : GESTION MEMOIRE DE BAS NIVEAU.

Cette couche gère une liste chaînée de tous les espaces mémoire libres, allouant et désallouant des blocs à la demande.

La mémoire est allouée à partir du début de la mémoire tandis que les stacks de processus se trouvent dans la fin de la mémoire.

A ce niveau, les demandes qui ne peuvent être satisfaites sont simplement rejetées.

primitives

GETMEM (nbytes)

Cette routine alloue un bloc de nbytes (arrondi) bytes de mémoire au processus appelant à partir de la fin de l'espace utilisé par le programme et retourne l'adresse de début de bloc.

GETSTK (nbytes)

Cette routine alloue un bloc de nbytes (arrondi) bytes de mémoire au processus appelant à partir de la fin de la mémoire et en remontant. GETSTK retourne l'adresse de fin du bloc.

COUCHE 2 : GERANT DES PROCESSUS.

Le gérant des processus est pratiquement conçu en trois sous-couches répondant au même principe d'ordonnancement hiérarchique.

La première couche implémente les manipulations de listes et de files.

Le scheduling et le changement de contexte feront l'objet de la sous-couche numéro deux.

La dernière couche fournit les primitives de manipulation des processus.

COUCHE 2.1 : MANIPULATION DE LISTES ET DE FILES.

Les routines décrites ici sont utilisées pour gérer des files par ordre chronologique d'insertion (FIFO) ou par priorité. Ces routines permettent d'insérer un élément à la queue d'une liste, d'insérer un élément dans une liste ordonnée, d'enlever un élément à la tête d'une liste et d'allouer une nouvelle liste.

Toutes les listes ont le même format : elles sont chaînées en avant et en arrière, elles ont une tête et une queue, chaque élément de la liste possède une clé (entier) utilisée en cas de file organisée par priorité.

COUCHE 2.2 : SCHEDULING ET CHANGEMENT DE CONTEXTE.

Le scheduling et le changement de contexte sont des activités très proches rendant possible l'exécution concurrente. Le scheduling consiste à choisir un processus parmi ceux prêts à être exécutés. Le changement de contexte consiste à arrêter l'exécution d'un processus et à en relancer un nouveau. Pour garder trace des processus, le système utilise une structure de données globale appelée **table de processus.** Chaque fois que le scheduler suspend temporairement un processus, il sauve toute l'information pertinente concernant ce processus, en même temps que l'état du processus.

routine Assembleur

CTXW

Routine de changement de contexte : sauve et charge les registres.

COUCHE 2.3 : GESTION DES PROCESSUS.

Ce paragraphe étend la notion de gérant de processus en ajoutant une couche de logiciel au-dessus du scheduling et du changement de contexte. Cette nouvelle couche inclut des routines pour suspendre et reprendre une exécution ainsi que des routines qui créent des processus et les tuent. Sont développées également deux procédures utilitaires qui permettent d'obtenir l'identificateur d'un processus et de changer la priorité d'un processus.

primitives

RESUME (pid)

Cette primitive autorise un processus suspendu dont l'identificateur est pid à reprendre son exécution. Si la demande est valide, RESUME retourne la priorité du processus.

SUSPEND (pid)

Cette primitive place le processus dont l'identificateur est pid en état d'hibernation. Si la demande est valide, SUSPEND retourne la priorité du processus. Un processus peut suspendre soi-même son exécution.

KILL (pid)

Cette primitive arrètera immédiatement le processus dont l'identificateur est pid et l'expulsera du système. On peut tuer un processus quel que soit son état. Une fois tué, il ne pourra être rescusité.

CREATE (caddr, ssize, prio, name, nargs °,argument§)

Cette primitive crée un processus qui débutera son exécution à l'adresse caddr avec un stack de ssize mots, une priorité initiale prio et comme nom identifiant name. Si la demande est valide, CREATE retourne l'identificateur du processus.

Le processus est créé en état suspendu.

Un nombre variable d'arguments auquels on accède via des paramètres formels peut être passé au nouveau processus.

CHPRIO (pid, newprio)

Cette primitive donne newprio comme nouvelle priorité de scheduling au processus dont l'identificateur est pid.

GETPID ()

Cette primitive retourne l'identificateur du processus en cours d'exécution.

COUCHE 3 : COORDINATION DES PROCESSUS

Les primitives de synchronisation de cette couche sont basées sur le principe des sémaphores de comptage. A côté des procédures de création et de destruction des sémaphores, les primitives **WAIT** et **SIGNAL** sont utilisées pour suspendre et reprendre des processus en fonction de la valeur d'un sémaphore. Cette technique permet de coordonner deux processus de manière à garantir que le consommateur reçoive chaque valeur émise par le producteur. De même, un ensemble de processus peut utilisé un sémaphore pour obtenir l'accès exclusif à une structure de données qu'ils partagent.

primitives

WAIT (sem)

Cette primitive décrémente la valeur du sémaphore **sem** et bloque le processus appelant si la valeur du sémaphore devient négative.

SIGNAL (sem)

Cette primitive incrémente la valeur du sémaphore **sem** et libère éventuellement un processus en attente du sémaphore.

SCREATE (count)

Cette primitive crée un sémaphore de comptage et l'initialise avec la valeur **count**. Si la demande est valide, SCREATE retourne l'identificateur du sémaphore.

SDELETE (sem)

Cette primitive retire du système le sémaphore **sem** et libère les processus éventuellement en attente de ce sémaphore.

COUCHE 4 : ENVOI DE MESSAGES

L'envoi de messages de processus à processus est une forme simple et efficiente de communication. Chaque processus possède une file d'attente de messages de longueur un. De plus, la longueur d'un message est limitée à un mot.

primitives

SEND (pid, msg)

Cette primitive envoie un message au processeur dont l'identificateur est pid. Si ce processus possède déjà un message en attente d'être lu, la demande d'envoi est annulée.

RECEIVE ()

Cette primitive retourne le message envoyé au processus appelant. S'il n'y a aucun message en attente, le processus est bloqué jusqu'à l'apparition d'un message.

RECVCLR ()

Cette primitive vérifie si il y a un message en attente et le retourne si c'est le cas.

COUCHE 5 : GERANT D'HORLOGE TEMPS REEL.

Avant la couche d'implémentation proprement dite de l'horloge, vient s'intercaler une couche intermédiaire, mais très importante, gérant le traitement des interruptions.

COUCHE 5.1 : TRAITEMENT DES INTERRUPTIONS.

Le système d'interruption est un mécanisme puissant sur lequel se repose une bonne partie du système d'exploitation.

Un appareil demande une interruption en envoyant un signal au CPU via un bus. Avant d'exécuter une instruction, le CPU contrôle le bus d'interruption et appelle une procédure de traitement de l'interruption en cas de demende. Quand cette procédure est terminée, le contrôle revient normalement au processus qui fut interrompu.

Cette couche implémente les routines de dispatching écrites en assembleur sur lesquelles viennent se gerffer les gérants d'interruptions écrits en langage C. Les dispatchers détectent les interruptions, sauvent les registres machine et passent le contrôle au gérant des interruptions identifié grâce à une table de dispatching. Quand le gérant a terminé son travail, le contrôle est rendu au dispatcher qui recharge les registrese et retourne au programme interrompu après avoir restauré son état.

<u>routine</u> <u>Assembleur</u>

ININT, OUTINT

Les opérations d'entrées/sorties sont initialement traitées par ces routines. Pendant que ces dernières exécutent les fonctions déjà décrites, les interruptions ne sont plus autorisées.

COUCHE 5.2 : GERANT D'HORLOGE TEMPS REEL.

Une horloge el hardware émettant avec une grande précision des impulsions à intervalle régulier. A côté de l'horloge centrale, le système gère une horloge temps réel et une horloge calendrier.

Cette dernière possède un compteur du nombre d'impulsions émises et fait progresser l'heure et la date toutes les secondes.

L'horloge temps réel est sensiblement différente. Chaque impulsion de l'horloge hardware provoque une interruption du CPU et, au lieu d'effectuer systématiquement les traitements correspondant à la progression de l'horloge, incrémente un compteur. Seulement après un certain nombre d'interruptions hardware, la routine de traitement est effectivement exécutée.

Le gérant d'horloge temps réel est utilisé pour programmer l'exécution d'évènements dans le futur et pour les lancer effectivement au moment voulu.

La préemption qui force l'appel au scheduler après qu'un processus ait été exécuté un certain temps, est implémenté de cette manière.

Certains processus peuvent de même s'endormir pendant un certain temps sur une liste de processus endormis et, une fois leur délai expiré, être réveillés et relancés par le gérant d'interruptions.

primitives

SLEEP (secs)

Cette routine arrête et diffère l'exécution du processus en cours un nombre donné secs de secondes.

SLEEP10 (dix)

Cette routine arrête et diffère l'exécution du processus en cours un nombre donné dix de dixièmes de seconde.

routines Assembleur

SETCLKR

Cette routine initialise l'horloge temps réel au (re)démarrage du système.

CLKINT

Cette routine de traitement de l'interruption d'horloge met à jour le compteur d'interruptions. CLKINT réveille, si nécessaire, un processus endormi et met à jour le compteur de préemption.

COUCHE 6 : GERANT DES PERIPHERIQUES.

Deux couches composent le gérant des périphériques. La première reprend les device drivers correspondant à chaque appareil périphérique physique. La seconde implémente une couche de primitives d'entrée/sortie indépendante de tout appareil réel.

La découpe de cette couche en deux sous-couches distinctes répond à trois raisons essentielles.

Tout d'abord, l'interface hardware de la plupart des périphériques est lourd et complexe à manipuler.

Ensuite, ces appareils sont des ressources partagées dont la sécurite d'accès doit être contrôlée par le système d'exploitation.

Enfin, XINU fournit aussi un interface cohérent, uniforme et flexible à tous les périphériques permettant aux utilisateurs d'effectuer des entrées/sorties sans rien connaître de la configuration de la machine.

COUCHE 6.1 : UN DEVICE DRIVER POUR TERMINAL AVEC CLAVIER.

Un **device driver** consiste en un ensemble de procédures qui contrôlent un périphérique hardware. Cet ensemble est composé effectivement de deux sous-groupes : un groupe supérieur contenant les routines appelées à partir des programmes utilisateurs et un groupe inférieur contenant les routines de traitement d'interruptions. Ces deux sous-groupes communiquent au travers une structure de données appelée **bloc de contrôle de device.**

Le device driver fournit dans cette version est appelé **driver tty.** Il gère les sorties sur un écran et les entrées à partir d'un clavier en utilisant un interface de ligne série asychrone.

Le groupe supérieur du driver tty contient les routines qui implémentent les opérations READ, WRITE, GETC, PUTC et CONTROL ; un utilisateur les appelle indirectement avec les procédures d'entrée/sortie indépendantes développées dans la couche 6.2.

La procédure de sortie du groupe inférieur, appelée à chaque interruption du transmetteur, envoie des caractères pris dans une file d'attente. Quand un caractère en entrée arrive, la procédure d'entrée du groupe inférieur le dépose dans une file de caractères entrant où il peut être récupéré par le groupe supérieur.

COUCHE 6.2 : GERANT D'UN PERIPHERIQUE VIRTUEL.

Le système d'exploitation fournit un environnement de haut niveau aux programmes utilisateurs en cachant les détails des périphériques sous une couche de routines d'entrée/sortie indépendantes de tout appareil. Les programmes utilisateurs accèdent aux périphériques par leur nom en utilisant des opérations de haut niveau. Le système opère de manière synchrone, stoppant l'exécution du processus appelant jusqu'à ce que les données aient été transférées.

Pour maintenir les informations concernant les périphériques indépendantes des appareils hardwares et de leurs adresses, le système relie des noms abstraits comme **CONSOLE** à des nombres entiers appelés **descripteurs d'appareils.** Ces descripteurs, lors de l'exécution, correspondent à des appareils spécifiques grâce à une **table d'appareils.** Cette table contient une entrée par appareil avec son adresse et l'ensemble des routines de contrôle. Les opérations d'entrée/sortie de haut niveau accèdent à la table d'appareils pour déterminer la routine de contrôle qui réalise l'opération spécifique au périphérique mentionné. Les routines individuelles interprètent ces appels en fonction d'un périphérique particulier.

primitives

READ (per, buffer, nombcar)

Cette routine va lire un nombre maximum de **nombcar** bytes en provenance du périphérique déterminé par **per.** Ces bytes seront transférés dans une zone pointée par **buffer.** READ retourne le nombre de caractères lus. En effet, le nombre réel de bytes retournés dépendra du type du périphérique.

CONTROL (per, fonction, arg1, arg2)

Ce mécanisme est utilisé pour envoyer des informations de contrôle à des périphériques ou pour interroger leur statut. Les valeurs retournées sont dépendantes des appareils.

GETC (per)

Cette routine lis le caractère suivant en provenance du périphérique déterminé par per. GETC retourne le caractère lu, éventuellement étendu à un entier.

PUTC (per, car)

Cette routine écrit le caractère car sur le périphérique déterminé par per.

SEEK (per, buffer, pos)

Cette routine positionne le périphérique déterminé par per après le posième caractère. SEEK ne peut être utilisé avec des périphériques connectés à des terminaux.

WRITE (per, buffer, compt)

Cette routine écrit compt caractères vers le périphériquedéterminé par per, lus séquentiellement à partir de la zone mémoire pointée par buffer. Une fois le transfert terminé, la routine revient au programme utilisateur pour qu'il puisse changer le contenu du buffer.

CLOSE (per)

Cette routine met fin au flux E/S de (vers) le périphérique déterminé par per.

OPEN (per)

Cette routine établit une connexion avec le périphérique déterminé par per.

Annexe 2.4 : Autres systèmes.

Autres systèmes.

Nous voudrions mentionner ici rapidement un certain nombre de systèmes que nous avons envisagés pour l'implémentation du prototype.

Rappelons que les critères principaux de choix d'un système étaient les suivants :

- existence d'un mécanisme d'interruption élaboré permettant la gestion des diverses temporisations,
- possibilité d'exécuter plusieurs processus en quasi-simultanéité,
- disponibilité d'un système.

Les systèmes suivants ont été pris en considération, mais contrairement à ceux exposés plus haut, ils n'ont pas été étudiés en détail. En effet, un ou plusieurs des critères ci-dessus n'étaient pas pleinement satisfaits.

1) Xenix : gestion multiprocessus insuffisante.

2) Concurrent DOS : limité à 4 processus simultanés, et impossibilité de gérer des temporisations aussi courtes que voulu.

3) MS-DOS : disponible mais totalement insuffisant sur les autres points.

4) iRMX86 : gestion multiprocessus insuffisante.

Annexe 3 :

Caractéristiques essentielles des langages étudiés.

Annexe 3.1 : Le langage Chill.

LE LANGAGE CHILL.

Chill (CCITT high-level language) a été développé par le CCITT tout spécialement pour les applications de télécommunications. Il est décrit dans la recommandation Z200.

On constate en effet que la programmation devient une part de plus en plus grande de ces systèmes à cause de l'usage croissant du Contrôle par Programme Enregistré (CPE, ou Stored Program Control : SPC).

Chill est un langage offrant de nombreuses facilités pour ce domaine, tout en étant qu'il est complètement indépendant d'un quelconque système commercial particulier.

1. Bases du langage.

1.1. Expressions et instructions d'assignation.

cf. langage Pascal.

1.2. Objets de données et modes.

Les objets de données ont un mode, ce qui correspond à un type en Pascal.

1.2.1. Modes discrets.

Les modes discrets sont ceux attachés à des objets ayant un ensemble fini ordonné de valeurs indivisibles. Les modes discrets suivants sont standards: INT, BOOL et CHAR.

1.2.2. Déclaration de variables.

DCL (nom de variable)*(mode);

1.2.3. Initialisation.

On peut assigner une valeur au départ à une variable dès sa déclaration.

DCL (nom de variable)* (mode) := (valeur);

1.2.4. Variables "read-only".

Elles sont obligatoirement initialisées.

DCL (nom de variable)* READ (mode) := (valeur);

1.3. Instructions et programmes.

Un programme Chill commence par le mot MODULE, suivi dans l'ordre,des définitions (voir plus loin), des déclarations et des instructions, le tout terminé par END.

1.3.1. Entrées/Sorties.

Il n'y a pas d'instructions standard d'entrée:sortie en Chill. Il faut au minimum écrire les routines suivantes: INBOOL, ININT, INCHAR, OUTBOOL, OUTINT, OUTCHAR.

1.4. Opérateurs et routines standards.

* arithmétiques: +, -, -, *, /, REM, MOD

* relationnels: =, /=, <, <=, >, >=

* booléens: AND, OR, XOR, NOT

* autres: ABS, PRED, SUCC, NUM

1.5. Objets de données composites.

Les objets de données composites peuvent être construits par agrégation de objets plus simples.

1.5.1. Tableaux.

* Déclaration:

```
DCL (nom du tableau) ARRAY(borne inf. : borne sup.)
                                    (mode des éléments);
```

* Plusieurs dimensions:

On peut répéter la séquence ARRAY(b.i.:b.s.)

* Valeur des éléments:

- nuple simple:

ex: si A est un ARRAY(1:5), on peut lui assigner le nuple (:6,15,3,41,7:).

ex: si B est une matrice ARRAY(1,2) ARRAY(1,3), ce qui peut s'écrire MATRIX(2)(3), on peut lui assigner le nuple (:(:7,3,1:),
(:2,2,8:):).

- nuple indicés:

ex: A:=(:(1):5,(2:4):I,(6:8):I+J,(ELSE):0:);

ce qui signifie qu'on assigne la valeur 1 à A(1), I à A(2),A(3) et A(4), I+J à A(6),A(7) et A(8), et 0 à tous les autres éléments.

ex: A(2:4) ou A(2 UP 4) considère un sous-tableau de A.

1.5.2. Structures.

Une structure est composée d'objets dont les modes peuvent être différents. Chaque composant est un champ.

ex: DCL A STRUCT(I:INT,J:INT,TRUE:BOOL);

On peut assigner un nuple à une structure.

ex: A:=(:3,6,FALSE:);

ou bien: A:=(:.I:3,.J:6,.TRUE:FALSE:); ce qui s'appelle un nuple avec labels.

On référence un champ de la manière suivante:

A.J : champ J de la structure A.

1.5.3. Variables composites read-only.

Une variable composite entière ou seulement certains champs peuvent être déclarés read-only par le mot READ.

1.6. Instructions conditionnelles.

1.6.1. Deux branches.

Instruction IF (booléen) THEN (instructions) ELSE(instr.)

1.6.2. Conditions à branches multiples: ELSIF.

IF THEN ELSIF THEN ELSIF... THEN ELSE.

1.6.3. Conditions à branches multiples: CASE

```
CASE (variable) OF
     (valeur ou range) : (instructions);
       ...
     ELSE : instructions;
ESAC;
```

1.6.4. Table de décision avec CASE.

```
ex: CASE I,       C,          B          OF
          (1),    ('A'),      (TRUE):    X:=1;
          (2:5),  ('D':'F'),  (FALSE):   X:=2;
          (ELSE), ('G':'Z'),  (*):       X:=3;
          ELSE                           X:=4;
    ESAC;
```

note: "*" indique une valeur quelconque.

1.7. Instruction de boucle.

1.7.1. Forme WHILE.

```
DO WHILE (booléen);
   (instructions);
OD;
```

1.7.2. Forme FOR.

1.7.2.1. Boucle infinie.

```
DO FOR EVER;
  (instructions);
OD;
```

1.7.2.2. Boucles avec compteurs.

ex: DO FOR I IN INT(1:100);

ex: DO FOR I:=1 BY 3 TO 100;

ex: DO FOR X IN A; avec A un tableau.

1.8. Instruction EXIT.

EXIT permet de sortir prématurément d'un boucle.

1.9. Procédures.

1.9.1. Appel de procédure.

Nom de la procédure suivi des paramètres entre parenthèses.

1.9.2. Définition de procédure.

```
ex: CHECK:
    PROC(); ou bien              (pas de paramètres)
    PROC(I,j,k); ou bien         (paramètres)
    PROC(L,M) (INT);             (fonction à valeur entière)
```

1.9.3. Passage de paramètres.

Dans le cas général, le passage se fait par valeur. Il est cependant possible de passer par référence en plaçant le mot LOC dans la définition de procédure après le mode du paramètre concerné.

1.9.4. Instructions RETURN et RESULT.

A la fin d'une procédure, le contrôle revient normalement à l'appelant. Cependant, le retour peut se faire plus tôt grâce à l'instruction RETURN.

Pour une fonction, le résultat est renvoyé par l'une des deux manières suivantes:

```
RETURN(valeur)  ou bien

RESULT(valeur);
RETURN
```

1.9.5. Mode procédure.

On peut déclarer des objets de mode procédure. Ceci permet d'appeler la même procédure avec des noms différents.

1.10. Instruction ASSERT.

ASSERT (condition booléenne); cela permet de tester des valeurs en cours d'exécution. Une erreur se produit si la condition est fausse.

2. Description des objets de données.

2.1. Synonymes.

SYN (nom symbolique) = (valeur littérale)

Ceci revient à la notion de constante Pascal.

2.2. Modes.

2.2.1. Définition de modes.

2.2.1.1. Nouveaux modes.

NEWMODE (nouveau nom) = (expression de modes)

2.2.1.2. Modes synonymes.

SYNMODE (nouveau nom) = (expression de types)

2.2.2. Modes ensemble.

NEWMODE (nom de l'ensemble) = SET(valeur1,valeur2,...);

On définit ainsi un ensemble de valeurs sur lequel on peut faire des tests.

ex: IF (nom d'objet) IN (nom d'ensemble) THEN ...

2.2.3. Modes intervalle.

On indique un mode déjà défini suivi d'un intervalle de valeurs sur ce mode.

ex: INT(8:255)
CHAR(A:L)

2.3. Modes string.

2.3.1. Strings de caractères.

La définition du mode est composée du mot CHAR suivi d'une longueur de string entre parenthèses.

2.3.2. Strings de bits.

La définition du mode est composée du mot BIT suivi d'une longueur de string entre parenthèses. Les littéraux sont écrits entre apostrophes('), précédés de la lettre H, B ou O pour indiquer s'il s'agit d'un hexadécimal, d'un binaire ou d'un octal.

2.3.3. Opérations sur les strings.

* Accès à un élément du string en indiquant un indice entre parenthèses après le nom du string.

* Extraction d'un sous-string soit en indiquant un intervalle (borne inf.: borne sup.), soit en indiquant une position de départ et une longueur (pos.dép. UP lg.).

* Opérateurs: //(concaténation), AND, OR, XOR,NOT. Les quatre derniers concernent les bits.

2.4. Modes référence.

On utilise la flèche à droite → :

PTR→ représente l'objet pointé par PTR

→OBJ représente le pointeur pointant vers OBJ

Il y a deux sortes de pointeurs:

- les pointeurs liés (bound reference) sont définis par le mode de l'objet vers lequel ils pointent. Ils sont indiqués par le mot REF.

- les pointeurs libres (free reference) ne font que pointer vers une mémoire sans tenir compte de ce qui s'y trouve. Ils sont indiqués par le mot PTR.

Enfin, on peut déclarer un objet et son mode, en spécifiant qu'il ne peut être accédé que via un pointeur. Ceci se fait en indiquant BASED (pointeur). Ces objets sont en effet appelés "objets basés".

2.5. Complément sur les modes composites.

2.5.1. L'instruction DO WITH.

Identique au WITH Pascal.

2.5?2. Structures variables.

Identique aux records variables avec CASE en Pascal.

2.5.3. PACK/NOPACK.

Identique au PACK/UNPACK Pascal.

2.5.4. Description de la disposition.

On peut spécifier très précisément l'emplacement des différents champs d'une structure dans les mots de la mémoire d'un ordinateur. Ceci est utile dans le cas où il faut traiter des structures de données créées en dehors de Chill.

```
ex: DCL A STRUCT ( I INT POS(Ø),                  Ø:mot
                   J INT(Ø:1ØØ) POS(1,Ø:6),       1:mot,Ø:6:bits
                   K BOOL POS(1,7));              1:mot,7:bit
```

3. Structure du programme.

Nous allons examiner la modularité des programmes au moyen des cinq délimiteurs ("brackets") suivants :

```
MODULE  END
REGION  END
BEGIN   END
PROC    END
PROCESS END
```

Un programme Chill consiste en une séquence de modules et/ou de régions.

3.1. Visibilité et temps de vie.

Visibilité : Un nom est visible lorsqu'il peut être utilisé à un endroit donné de programme.

Temps de vie : C'est le temps durant lequel l'objet existe dans le programme.

Un objet peut exister sans être visible.

3.2. Blocs.

Un bloc est délimité par BEGIN et END. Les noms déclarés dans le bloc sont invisibles de l'extérieur mais visibles partout à l'intérieur, même dans les sous-blocs. Ils sont donc locaux pour le bloc et globaux pour les sous-blocs.

Le temps de vie des noms déclarés dans le bloc est le temps d'exécution du bloc, sauf lorsque la variable est déclarée STATIC auquel cas elle est permanente.

3.3. Modules.

Les noms définis dans un module sont locaux à ca module. Cependant, les noms globaux, c'est-à-dire les noms définis en dehors du module ne sont pas automatiquement visibles à l'intérieur du module. De plus, les noms locaux d'un module peuvent être rendus visibles hors du module. Ainsi un module est un moyen de contrôler la visibilité, pas le temps de vie. Ce dernier s'étend à celui du premier bloc entourant.

On utilise SEIZE pour rendre un nom global visible dans le module et GRANT pour rendre un nom local visible hors du module.

D'autres instructions et paramètres apportent encore des nuances à la manipulation des noms.

3.5. Exécution concurrente.

3.5.1. Le concept de processus.

Processus: unité d'exécution concurrente. C'est une partie dynamique du programme qui peut être exécutée de manière concurrente avec d'autres processus du programme.

3.5.1.1. Instruction de définition de processus.

Même structure qu'une définition de procédure, en remplaçant PROC par PROCESS.

3.5.1.2. Instruction de démarrage.

START nom-de-processus (paramètres);

3.5.1.3. Instruction d'arrêt.

STOP arrête le processus.

3.5.1.4. Modes instance.

Il faut pouvoir adresser un processus car, une fois qu'il est lancé, il continue tout seul. Chaque création de processus est enfait la création d'une instance de la définition du processus. On peut déclarer des variables de mode INSTANCE qui référenceront des processus en cours d'exécution.

3.5.2. Coordination entre processus.

Trois aspects:
- synchronisation,
- communication par messages,
- exclusion mutuelle pour l'accès à des données.

3.5.2.1. Exclusion mutuelle.

Utilisation de la REGION. Une région ressemble à un module mais ne peut contenir que des instructions de définition et de déclaration. Les objets communs pour lesquels les processus entrent en compétition sont déclarés locaux à la région et ne peuvent être manipulés que par les procédures définies dans la région. Ces procédures doivent être rendues visibles de l'extérieur par GRANT. C'est en fait une sorte de type abstrait. Lorsque un processus accède à la région, les autres processus doivent attendre s'ils veulent aussi y accéder.

3.5.2.2. Modes événements.

Le mode EVENT permet de synchroniser les processus. Soit X déclaré EVENT, DELAY X retarde unprocessus jusqu'à l'apparition de X, et CONTINUE X provoque l'événement X.

3.5.2.3. Modes buffer.

Les processus peuvent s'envoyer des messages via des buffers. Si X est déclaré BUFFER, on peut y placer un message par l'instruction SEND et en retirer un message par l'instruction RECEIVE. Un buffer est persistant, contrairement à un événement.

3.5?2.4. Signaux.

Les signaux ont ou n'ont pas de partie message. Dans le premier cas, ils sont comme les événements, mais sont persistant, et dans le second cas, ils sont comme les buffers mais la gestion du buffer lui-même est automatique. On utilise SEND et RECEIVE.

Annexe 3.2 : Le langage PLEX.

LE LANGAGE PLEX.

PLEX est langage développé par la Telefonaktiebolaget Ericsson de Stockholm spécialement pour ses applications de téléphonie.

Il s'agit en fait d'une version améliorée, appelée PLEX-C et utilisée actuellement dans les commutateurs AXE 10 de Ericsson.

PLEX-C est un langage de très haut niveau et ne pouvait par conséquent pas être utilisé seul dans un système comme un commutateur où les contraintes matérielles sont fortes. Il n'intervient en fait que dans le software central, et pour environ 70% du code. Le reste,est codé en assembleur ASA 210 C et R.

Bien que PLEX-C soit un langage puissant, comprenant toutes les caractéristiques générales d'un langage de haut niveau, il porte l'empreinte de la structure du système AXE 10 pour lequel il a été conçu.

Cette spécificité lui permet d'être beaucoup plus complet que Chill ou C dans le domaine des entrées-sorties notamment.

Il est conseillé de lire cette annexe en faisant référence à la section 2.2.2.2.B.4. et aux sections 2.2.2.2.C.2. et 2.3. du premier tome.

1. Déclarations.

```
DECLARE;
 (instructions de déclaration)
END DECLARE;
```

1.1. Déclarations de variables.

Les variables appartiennent à un CP donné. Elles peuvent être temporaires ou stockées en DS. Il y en a de trois types:

- champ : nombre sous forme binaire
- symbole : valeurs symboliques
- string : chaînes de caractères

VARIABLE (déclarations de variables) (propriétés)

1.1.1. Variables de champ.

VARIABLE nom longueur propriétés;

La longueur est un nombre de bits. Pour les propriétés voir 1.5.

1.1.2. Tableaux à une dimension.

VARIABLE nom(dimension) longueur-des-éléments propriétés;

1.1.3. Tableaux à deux dimensions.

```
VARIABLE nom(dimension 1, dimension 2) longueur-des-
                    éléments propriétés;
```

1.1.4. Variables symboliques.

SYMBOL VARIABLE nom propriétés;

1.1.5. Variables string.

STRING VARIABLE nom longueur-maximum propriétés;

1.1.6. Buffers de string.

Ils permettent de stocker des strings plus longs que les 255 maximum autorisés dans les variables de string.

STRING BUFFER nom;

1.1.7. Structures de variables.

Elles sont exprimées dans une forme similaire à celle de COBOL.

```
STRUCTURE nom1 = niveau nom2 longueur,
                 niveau nom3 longueur,
                 ...;
```

1.1.8. Structures de records et de fichiers.

Les fichiers contiennent des records d'un type unique par fichier. Les records peuvent contenir tout sauf des tableaux bidimensionnels. Les recordssont numérotés à partir de Ø dans un fichier et un pointeur indique le record courant.

1.1.9. Records.

```
RECORD nom;
  (déclarations de variables)
END RECORD;
```

Juste après cette déclaration suit une déclaration de pointeur sur ce record:

```
POINTER nom(nom du record associé);
```

1.2. Déclarations de symboles.

Ce sont les constantes du programme qui peuvent être de deux types:

- symboles locaux: valeur assignée directement dans la déclaration,

-symboles globaux : valeur assignée dans une liste de paramètres extérieure au programme.

1.2.1. Symboles locaux numériques.

```
NSYMB nom = valeur-numérique;
```

1.2.2. Symboles locaux de string.

```
STRING nom = "texte";
```

1.2.3. Symboles globaux numériques.

```
GLOBAL NSYMB nom (ensemble de valeurs permises);
```

1.2.4. Symboles globaux de string.

```
GLOBAL STRING nom (longueur);
```

1.3. Propriétés des variables.

Suit la liste des propriétés pouvant être spécifiées pour les variables, et leur signification:

DS:
- La variable est stockée dans DS
- La variable est effacée lors de l'allocation

BUFFER:
- -Une mémoire doit être commandée et libérée lors de son utilisation
- Les variables allouées ont une valeur indéfinie
- L'allocation est faite élément par élément
- Un buffer peut être contenu dans un signal transmis
- Au redémarrage la zone de données allouée est libérée

TEMPORARY VARIABLE:
- La variable est stockée dans un register
- La variable a un contenu indéfini lorsque le programme y fait accès
- Une valeur définie cesse de l'être après EXIT, un ordre d'entrée/sortie ou SEND avec WAIT.
-

NO ACTION:
- La valeur de la variable après relancement du système avec rechargement est indéfinie, mais dans les autres cas de relancement,elle reste inchangée.

CLEAR:
- Il y a effacement dans tous les cas de lancement et de relancement

RELOAD:
- A certains intervalles, la variable est perforée sur une bande de rechargement
- Au relancement avec rechargement, une valeur est obtenue de la bande de rechargement
- Après lancement la valeur de la variable = Ø ou celle contenue dans le secteur de données (lancement initial) ou celle de la bande de rechargement (démarrage avec relancement).

DUMP:
- A chaque relancement, la valeur de la variable est transférée sur un support externe.

STATIC:
- Dans les changements de fonction, la variable n'a pas sa valeur transférée du software d'origine

2. Instructions de PLEX-C.

2.1. Opérateurs et expressions.

* arithmétiques: +, -, (-), *, /

* booléens: (-):négation, (*):AND, (=):XOR, (+):OR

* strings: //:concaténation

2.2. Instructions d'interaction (interwork).

Les blocs interagissent par signaux simples et combinés. Les secteurs à l'intérieur d'un même programme interagissent par signaux de secteur, appels au secteur assembleur ou appels de procédures.

Les signaux simples ne requièrent aucun signal d'accusé de réception. Les signaux combinés sont des signaux en avant pour lancer l'exécution dans le bloc récepteur, et des signaux en arrière pour retourner au bloc émetteur lorsque l'exécution est terminée.

2.2.1. Instruction de transmission d'un signal simple.

```
SEND signal REFERENCE variable de champ
            WITH données,
            BUFFER ou
            HURRY ou
            DELAY nombre unité-de-temps  ou
            DELAY UNTIL mois,jour,heure,minute
```

Reference est utilisédans le cas où il s'agit d'un signal d'RP. Il permet d'accéder à la table d'RP.
With contient les données du message.
Buffer met le signal dans un buffer temporaire.
Hurry indique qu'un signal doit être envoyé immédiatement.
Delay indique un délai à respecter avant l'envoi.

2.2.2. Instruction de réception d'un signal simple.

```
ENTER signal WITH données;
```

2.2.3. Instruction de transmission d'un signal en avant combiné.

```
SEND signal REFERENCE variable de champ
            WITH données,
            WAIT FOR signal arrière IN label
                  OR signal arrière 2 IN label...;
```

Wait for indique tous les messages recevables possibles.

2.2.4. Instruction de réception d'un signal en avant combiné.

```
RECEIVE signal WITH données;
```

2.2.5. Instruction de transmission d'un signal en arrière combiné.

RETURN signal WITH données;

2.2.6. Instruction de réception d'un signal en arrière combiné.

label) RETRIEVE signal WITH données;

Le label est celui mentionné en 2.2.3.

2.2.7. Instruction d'envoi pour un signal de secteur.

TRANSFER signal WITH données;

2.2.8. Instruction de réception pour un signal de secteur.

ENTRANCE signal WITH données;

2.2.9. Exit.

EXIT; désactive une séquence d'instructions.

2.3. Instructions d'assignation.

SET variable = expression de champ;ou de symbole;ou de string;

2.4. Instruction de saut inconditionnel.

GOTO label;

2.5. Instructions de saut conditionnel.

2.5.1. IF.

IF (NOT) condition (PROCEED ELSE) GOTO label;

Proceed else renverse la condition tout comme Not.

2.5.2. Instruction de branchement de champ

```
BRANCH ON expression de champ
          TO label IF numeral
          TO ...
          ELSE TO label;
```

Numeral est une valeur de l'expression de champ.

2.5.3. Instruction de branchement de symbole.

```
BRANCH ON variable symbolique
          TO label IF valeur symbolique
          TO ...
          ELSE TO label;
```

2.6. Instructions conditionnelles.

2.6.1. IF.

```
IF (NOT) condition THEN séquence d'instructions
ELSIF (NOT) condition THEN séquence
ELSIF ...
ELSE séquence
FI;
```

2.7. Instruction de sélection.

```
CASE expression IS
   WHEN valeurs DO séquence
   WHEN ...
   OTHERWISE    DO séquence
ESAC;
```

2.8. Instructions de répétition.

2.8.1. FOR.

```
FOR (FIRST/ALL) variable
     FROM  expression UNTIL expression
     WHERE NOT condition      ou bien
     WHERE variable IS CHANGED TO expression
     GOTO label       ou bien
     DO instruction;
```

2.8.2. ON.

```
ON variable FROM expression
            UPTO/DOWNTO expression
            DO séquence
```

2.9. Instructions d'allocation de mémoire.

2.9.1. Allocation d'une mémoire.

```
ALLOCATE buffer(taille),
    ABRANCH IS label,
    POINTER IS pointeur;
```

Abranch est un branchement vers une séquence particulière dans le cas où l'allocation est impossible.

2.9.2. Libération d'une mémoire.

```
FREE buffer, POINTER IS pointeur
```

2.10. Instructions d'entrée-sortie.

2.10.1. Instructions pour E/S orientées fichiers.

2.10.1.1. Saisie d'un fichier.

SEIZE FILE fichier FOR BINARY/ISO INPUT/OUTPUT;

Iso est un format alternatif au binaire.

2.10.1.2. Ecriture d'un bloc sur fichier à partir d'un buffer dynamique.

WRITE BLOCK BUFFER buffer;

2.10.1.3. Ecriture d'un bloc sur fichier à partir d'une adresse absolue.

WRITE BLOCK ADDRESS;

2.10.1.4. Lecture d'un bloc d'un fichier vers un buffer dynamique.

READ TO BLOCK BUFFER buffer;

2.10.1.5? Lecture d'un bloc d'un fichier vers une adresse absolue.

READ TO BLOCK ADDRESS;

2.10.1.6. Libération d'un fichier.

RELEASE FILE connection E/S;

2.10.2. Instructions pour les E/S alphanumériques.

2.10.2.1. Réception d'une commande.

COMMAND nom TYPE numéro de catégorie de la commande

Cette instruction permet de recevoir une commande du terminal.

2.10.2.2. Saisie d'un appareil d'E/S.

SEIZE DEVICE nom OR STANDBY;

2.10.2.3. Saisie fonctionnelle d'un appareil d'E/S.

SEIZE DEVICE FOR code de fonction;

Cette instruction permet de exécuter directement une fonction.

2.10.2.4. Libération d'un appareil d'E/S.

RELEASE DEVICE;

2.10.2.5. Instruction de rejet des commandes.

BUSY;

2.10.2.6. Lecture de données à partir d'un appareil d'E/S vers un buffer de ligne.

READ ;

2.10.2.7. Lecture de données à partir d'un appareil d'E/S vers un buffer dynamique.

READ TO BUFFER buffer;

2.10.2.8. Ecriture de données sur un appareil d'E/S à partir d'un buffer de ligne.

WRITE AFTER/BEFORE numéroNL/NEWPAGE ;

NL signifie nouvelle ligne et le numéro indique combien de nouvelles lignes il y a avant ou après.

2.10.2.9. Ecriture de,données sur appareil d'E/S à partir d'un buffer dynamique.

WRITE BUFFER buffer;

2.10.2.10. Autres instructions.

INSERT: insertion de données dans le buffer de ligne
FETCH: obtention de données d'un buffer de ligne
FIND: recherche d'un string de caractères
CHECK: vérification du buffer de ligne
FETCH PARAMETER: recherche d'une valeur de paramètre

2.11. Appels de blocs d'instructions et de secteurs assembleurs.

DO nom de bloc ou de secteur

2.12. Appels de procédures.

nom de la procédure (liste de paramètres);

3. Procédures.

Les procédures suivantes sont disponibles dans les librairies:

CHAR: Insertion et lecture de caractères dans des strings.

CONVERT: Conversion de chiffres.

DISABLE: Empêcher les interruptions.

ENABLE: Autoriser les interruptions.

FCHCHAR: Extraire deux caractères consécutifs d'un string.

FETCHC: Extraire un caractère d'un string.

FILENUMBER: sauvetage temporaire du numéro de fichier.

INTERVAL: lancement d'un intervalle de temps inter-job.

JOBTABLE: Insertion et destruction de signaux de la jobtable.

LOAD: Sauvetage temporaire d'une référence de bloc récepteur.

LOADREF: Sauvetage temporaire de sa propre référence de bloc.

MOVE: Transfert d'un pointeur de buffer.

SIGGRLOC: sauvetage temporaire de la mémoire de groupe signal.

STRCHAR: Sauvetage de 2 caractères consécutifs d'un string.

TRANSFORM: Transformation d'un numéro de bloc.

VARIABLENUMBER: Sauvetage temporaire de BN et BAddress.

Annexe 3.3 : le langage C.

LE LANGAGE C.

C est un langage de programmation d'application générale dont les caractéristiques sont une économie d'expression, un flux de contrôle et des structures de données modernes, et un ensemble riche d'opérateurs.

A l'origine, C fut conçu pour et implémenté sur le système d'exploitation UNIX sur le DEC PDP-11. C et UNIX sont des propriétés des BELL Laboratories. Le système d'exploitation, le compilateur C, et essentiellement tous les programmes d'application de UNIX sont écrits en C. Des compilateurs existent aussi pour plusieurs autres machines: IBM System/370, Honeywell 6000, et Interdata 8/32.

C n'est pas lié à un quelconque hardware ou système, et il est aisé d'écrire des programmes qui tourneront sans modifications sur toute machine supportant C.

C n'est pas un langage de "très haut niveau" et il n'est pas spécialisé pour un quelconque domaine particulier d'application. Mais son absence de restrictions et sa généralité font qu'il est plus convenable et plus efficace pour beaucoup de tâches que des langages soi-disant plus puissants.

Aperçu de C.

La structure générale d'un programme C ressemble fortement à celle d'un langage comme Pascal. La puissance de C réside surtout dans un certain nombre de possibilités de traitement spécifique de données.

Pour ces raisons, nous ne détaillerons pas complètement le langage, mais nous mettrons en évidence les particularités remarquables.

1. Tableaux.

Ils sont dynamiques. Ceàa veut dire que l'on peut déclarer un tableau sans en indiquer les bornes, et que ce tableau s'accroîtra dynamiquement au cours d'une exécution.

ex: CHAR S[]; déclare un tel tableau de caractères.

D'autre part, il y a une forte ijteraction entre tableaux et pointeurs. Si l'on déclare un pointeur vers un objet de même type que les éléments d'un tableau, on peut parcourir,le tableau au moyen du pointeur au lieu d'utiliser l'indiçage.

2. Structures.

C'est la même notion que le record Pascal. Il faut noter que le même mécanisme de pointeurs que celui des tableaux joue ici.

3. Pointeurs.

La gestion des pointeurs est d'une souplesse infinie. Il y a des pointeurs vers tout ce que l'on veut, y compris des fonctions, des tableaux et des pointeurs.

*ptr est l'objet pointé par ptr

&obj est le pointeur pointant vers obj

4. Propriétés des variables.

On dispose de quatre sortes de variables selon leur mode de stockage:

- Externe: L'objet est déclaré avec le mot-clé EXTERN. Iln'est pas redéfini dans le fichier source. On ne lui réalloue pas de place car il est supposé en avoir dans le fichier où il est déclaré. Il faut donc linker ce fichier :avec le fichier source.

- Statique: L'objet est déclaré avec le mot-clé STATIC. La place lui est allouée dans la zone de données du programme objet. Elle initialisée à zéro sauf si une instruction particulière d'initialisation est contenue dans la déclaration.

- Auto: L'objet est déclaré avec le mot-clé AUTO. La place est allouée sur un stack pendant l'exécution de la fonction où il est déclaré.

- Formelle: L'objet est un paramètre formel d'une des fonctions du fichier source. La place est allouée à l'exécution lors de l'appel de la fonction contenant ce paramèter.

5. Structure des programmes.

Un programme C consiste en un ensemble d'objets externes qui sont soit des variables soit des fonctions.

L'adjectif "externe" est utilisé principalement en contraste avec "interne" qui décrit les arguments et les variables automatiques définis à l'intérieur des fonctions.

Les variables externes sont définies en dehors de toute fonction et sont ainsi potentiellement accessibles à toutes les fonctions.

Les fonctions elles-mêmes sont toujpurs externes parce que C ne permet pas aux fonctions d'être définies à l'intérieur d'autres fonctions.

6. Manipulation de bits.

Une manipulation de bits très élaborée rapproche C des capacités du langage assembleur.

Annexe 4 :

Errata (concernant les deux premiers tomes).

Errata.

- T1, Table des matières : lire 3.2.1.3. au lieu de 3.2.1.4.
- T1, p127 : commencer la page par 2.2.2.2.C. Hardware.
- T2, p192 : commencer la page par 3.3.2. Système d'exploitation.
- T2, p217 : commencer la page par 3.3.3. Application.
- T2, p244 : commencer la page par 3.4. Implémentation.
- T2, p244 : lire iRMX 80-88 au lieu de iRMX 86-88.
- T2, p245 : commencer la page par 3.4.2. Système d'exploitation.
- T2, p253 : commencer la page par 3.4.3. Application.

Annexe 5 :

Complément de bibliographie.

Bibliographie.

1. DAVENPORT, John, Divers articles et études concernant la norme ISDN du CCITT, les recommandations du CEPT, et le développement du réseau ALVEY de la firme Logica à Londres.

2. BRUNIN, Jean, Logique binaire des circuits câblés et des programmes enregistrés, Tome II-a : systèmes séquentiels, Presses Universitaires de Namur, 1980.

3. ITU/CCITT, Introduction to Chill, Geneva, 1983.

4. ERICSSON, PLEX-C : Language Description, Telefonaktiebolaget Ericsson, 1983.

5. KERNIGHAN B.W.,RITCHIE D.M., The C programming language, Prentice-Hall, 1978.

6. ERICSSON, AXE 10 : System survey, Telefonaktiebolaget Ericsson.

7. ERICSSON, Introduction to AXE, Training document, Telefonaktiebolaget Ericsson.

8. OLIVETTI, Manuel d'utilisation du système d'exploitation MS-DOS.

9. LIFEBOAT, Lattice 8086/8088 C compiler manual, 1984.

10. ACEC, Description du système RTM-80.

11. INTEL, iRMX80/88 system description.