THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR INGENIEUR

(Informatique)

par

Didier LANCIAUX

Ingénieur I.S.E.N.

CONCEPTION D'UN SYSTEME FRONTAL A ACCES MULTIPLE

Soutenue le 20 Juin 1973 devant la Commission d'Examen

Membres du Jury :	MM. P. BACCHUS	Président
	C. CARREZ	Examinateur
	C. KAISER	Invité
	V. CORDONNIER	Rapporteur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE.

DOYENS HONORAIRES

MM. H.LEFEBVRE, PARREAU.

PROFESSEURS HONORAIRES

M. ARNOULT, Mme BEAUJEU, MM. BEGHIN, BROCHARD, CAU, CHAPPELON, CHAUDRON, CORDONNIER, DEHEUVELS, DEHORNE, DEHORS, FAUVEL, FLEURY, P.GERMAIN, HEIM DE BALSAC, HOCQUETTE, KAMPE DE FERIET, KOURGANOFF, LAMOTTE, LELONG, Mme LELONG, MM. LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, NORMANT, PARISELLE, PASCAL, PAUTHENIER, PEREZ, ROIG, ROSEAU, ROUBINE, ROUELLE, WIEMAN, ZAMANSKY.

PROFESSEURS TITULAIRES

M.	ANGRAND Jean Pierre	Géographie et Aménagement Spatial
M.	BACCHUS Pierre	Astronomie et Calcul
M.	BEAUFILS Jean Pierre	Chimie Générale
M.	BECART Maurice	I.U.T. Lille
M.	BLOCH Vincent	Psychophysiologie
M.	BIAYS Pierre	Géographie et Aménagement Spatial
M.	BONNEMAN Pierre	Chimie Industrielle
M.	BONTE Antoine	Géologie Appliquée
M.	BOUGHON Pierre	Mathématiques
M.	BOURIQUET Robert	Biologie Végétale
M.	CAPET Marcel-Francis	Institut de Préparation aux Affaires
M.	CELET Paul	Géologie Générale
M.	CONSTANT Eugène	Electronique
M.	CORSIN Pierre	Paléobotanique
M.	DECUYPER Marcel	Mathématiques
M.	DEDECKER Paul	Mathématiques
M.	DEFRETIN René	Biologie Animale - Directeur de l'Institut de Biologie Maritime de Wimereux
M.	DELATTRE Charles	Géologie Générale
M.	DURCHON Maurice	Biologie Animale
M.	FLATRES Pierre	Géographie et Aménagement Spatial
M.	FOURET René	Physique
M.	GABILLARD Robert	Electronique
M.	GEHU Jean Marie	Institut Agricole
M.	GLACET Charles	Chimie Organique
M.	GONTIER Gérard	Mécanique des Fluides
M.	GUILLAUME Jean	Biologie Végétale
M.	HEUBEL Joseph	Chimie Minérale
Mme	LENOBLE Jacqueline	Physique
M.	MONTREUIL Jean	Chimie Biologique
M.	POUZET Pierre	I.U.T. Lille

Mme	SCHWARTZ Marie Hélène	Mathématiques
M.	TILLIEU Jacques	Physique
M.	TRIDOT Gabriel	Chimie Minérale Appliquée
M.	VIDAL Pierre	Automatique
M.	VIVIER Emile	Biologie Animale
M.	WATERLOT Gérard	Géologie et Minéralogie
M.	WERTHEIMER Raymond	Physique

PROFESSEURS A TITRE PERSONNEL

M.	BOUISSET Simon	Physiologie Animale
M.	DELHAYE Michel	Chimie Physique et Minérale 1er Cycle
M.	LEBRUN André	Electronique
M.	LINDER Robert	Biologie Végétale
M.	LUCQUIN Michel	Chimie Physique
M.	PARREAU Michel	Mathématiques
M.	PRUDHOMME Rémy	Sciences Economiques et Sociales
M.	SAVARD Jean	Chimie Générale
M.	SCHALLER François	Biologie Animale
M.	SCHILTZ René	Physique

PROFESSEURS SANS CHAIRE

M.	BELLET Jean	Physique
M.	BODARD Marcel	Biologie Végétale
M.	BOILLET Pierre	Physique
M.	DERCOURT Jean Michel	Géologie et Minéralogie
M.	DEVRAINNE Pierre	Chimie Minérale
M.	LOMBARD Jacques	Sciences Economiques et Sociales
Mle	MARQUET Simone	Mathématiques
M.	MONTARIOL Frédéric	Chimie Minérale Appliquée
M.	PROUVOST Jean	Géologie et Minéralogie
M.	VAILLANT Jean	Mathématiques

MAITRES DE CONFERENCES (et chargés de fonctions)

M.	ADAM Michel	Sciences Economiques et Sociales
M.	ANDRE Charles	Sciences Economiques et Sociales
M.	AUBIN Thierry	Mathématiques Pures
M.	BEGUIN Paul	Mécanique des Fluides
M.	BILLARD Jean	Physique
M.	BKOUCHE Rudolphe	Mathématiques
M.	BOILLY Bénoni	Biologie Animale
M.	BONNEMAIN Jean Louis	Biologie Végétale
M.	BONNOT Ernest	Biologie Végétale
M.	BRIDOUX Michel	I.U.T. Béthune
M.	BRUYELLE Pierre	Géographie et Aménagement Spatial

M. CAPURON Alfred	Biologie Animale
M. CARREZ Christian	Calcul
M. CHOQUET Marcel	I.U.T. Lille
M. CORDONNIER Vincent	Calcul
M. CORTOIS Jean	Physique
M. COULON Jean Paul	Electrotechnique
M. DEBRABANT Pierre	Sciences appliquées
M. ESCAIG Bertrand	Physique
Mme EVRARD Micheline	I.U.T. Lille
M. FAIDHERBE Jacques	Biologie Animale
M. FONTAINE Jacques	I.U.T. Lille
M. FROELICH Daniel	Sciences Appliquées
M. GAMBLIN André	Géographie et Aménagement Spatial
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel	Sciences Economiques et Sociales
M. GOUDMAND Pierre	Chimie Physique
M. GRANELLE	Sciences Economiques et Sociales
M. GRUSON Laurent	Mathématiques
M. GUIBAULT Pierre	Physiologie Animale
M. HERMAN Maurice	Physique
M. HUARD de la MARRE Pierre	Calcul
M. JOLY Robert	Biologie (Amiens)
M. JOURNEL Gérard	Sciences Appliquées
Mle KOSMANN Yvette	Mathématiques
M. LABLACHE-COMBIER Alain	Chimie Générale
M. LACOSTE Louis	Biologie Végétale
M. LANDAIS Jean	Chimie Organique
M. LAURENT François	Automatique
M. LAVAGNE Pierre	Sciences Economiques et Sociales
Mle LEGRAND Solange	Mathématiques
M. LEHMANN Daniel	Mathématiques
Mme LEHMANN Josiane	Mathématiques
M. LENTACKER Firmin	Géographie et Aménagement Spatial
M. LEROY Jean Marie	ENSCL
M. LEROY Yves	I.U.T. Lille
M. LHENAFF René	Géographie et Aménagement Spatial
M. LOCQUENEUX Robert	Physique
M. LOUAGE Francis	Sciences Appliquées
M. LOUCHEUX Claude	Chimie Physique
M. MAES Serge	Physique
Mme MAILLET Monique	Sciences Economiques et Sociales
M. MAIZIERES Christian	Automatique
M. MALAUSSENA Jean Louis	Sciences Economiques et Sociales
M. MESSELYN Jean	Physique
M. MIGEON Michel	Sciences Appliquées
M. MONTEL Marc	Physique
M. MONTUELLE Bernard	I.U.T. Lille
M. MUSSCHE Guy	Sciences Economiques et Sociales
M. NICOLE Jacques	E.N.S.C.L.
M. OUZIAUX Roger	Sciences Appliquées
M. PANET Marius	Electrotechnique
M. PAQUET Jacques	Sciences Appliquées
M. PARSY Fernand	Mécanique des Fluides
M. PONSOLLE Louis	Chimie (Valenciennes)
M. POVY Jean Claude	Sciences Appliquées
M. RACZY Ladislas	Radioélectricité
Mme RENVERSEZ Françoise	Sciences Economiques et Sociales
M. ROUSSEAU Jean Paul	Physiologie Animale

M. ROYNETTE Bernard	Mathématiques
M. SALMER Georges	Electronique
M. SEGUIER Guy	I.U.T. Béthune
M. SIMON Michel	Sciences Economiques et Sociales
M. SMET Pierre	Physique
M. SOMME Jean	Géographie et Aménagement Spatial
M. THOMAS Daniel	Chimie Minérale Appliquée
M. TOULOTTE Jean Marc	Sciences Appliquées
M. TREANTON Jean René	Sciences Economiques et Sociales
M. VANDORPE Bernard	Sciences Appliquées
M. VILETTE Michel	I.U.T. Béthune
M. WATERLOT Michel	Géologie Générale
Mme ZINN JUSTIN Nicole	Mathématiques.

Je voudrais exprimer ma profonde reconnaissance à Monsieur le Professeur P. BACCHUS, Directeur de l'U.E.R. Informatique - Electronique - Electrotechnique - Automatique de l'Université de Lille qui me fait l'honneur de présider le jury de cette thèse.

Je remercie vivement Monsieur le Professeur C. CARREZ ainsi que Monsieur C. KAISER qui se sont intéressés à ce travail et ont accepté de le juger.

Que Monsieur le Professeur V. CORDONNIER qui n'a cessé de m'encourager et m'a guidé de ses conseils soit assuré de ma plus vive reconnaissance.

Je remercie également Monsieur J.P. DELAISSE de l'aide qu'il m'a apportée en participant à l'écriture des programmes et en assurant leur exploitation.

Enfin, je remercie Madame et Monsieur DEBOCK ainsi que Mademoiselle DRIESSENS qui tous trois, avec gentillesse, ont assuré la réalisation matérielle de cette thèse.

Table des Matières

INTRODUCTION

Les systèmes "time sharing" sont souvent sous-employés en ce sens que le nombre de terminaux qu'ils peuvent admettre est réduit en regard des travaux réalisés par les usagers conversationnels et de la puissance de la machine utilisée. On remarque en effet que la plupart des accès à la machine par ces usagers concerne la création de leur programme. Ceci est particulièrement vrai lorsque la destination du système est l'enseignement. L'amélioration des performances peut donc être obtenue en éliminant certaines tâches de service du système et en les traitant à l'aide d'un calculateur annexe appelé calculateur frontal.

Le travail présenté ici est l'étude préliminaire d'un projet qui sera réalisé au Laboratoire de Calcul de l'Université des Sciences et Techniques de Lille. Celui-ci consiste en la connexion d'un calculateur frontal à la machine CII 10070 du laboratoire (exploitée sous Siris 7) dans le but de permettre l'accès à de nombreux utilisateurs"time sharing'sans pour autant diminuer la charge'batch" du système. Le système frontal que nous proposons utilise un calculateur de puissance réduite et tend à optimiser son utilisation par l'extension des tâches qu'il réalise.

Dans une première partie nous déterminons les fonctions du calculateur frontal puis l'architecture du système en nous appuyant sur une étude générale des aspects qui peuvent apparaître dans un système"time sharing! Nous avons tenté, lors de la définition de l'architecture que nous avons retenue, de nous affranchir de toute contrainte. Il apparaît alors que certaines interfaces cablées devront être réalisées. Pour en simplifier l'implantation, nous utiliserons un calculateur microprogrammable. L'expérience acquise lors de la réalisation d'un précédent projet [6,7] nous a beaucoup aidé dans ce domaine.

Dans la deuxième partie nous étudions les principes de réalisation du système frontal aussi bien du point de vue du "hardware" à mettre en place que du système d'exploitation. Ceci nous a conduit à définir certains mécanismes élémentaires dont le but est de permettre une écriture aisée du système d'exploitation.

Nous avons essayé de concevoir celui-ci en interaction avec le microprogramme ; nous avons été amenés dans ce but à définir de nouvelles instructions et à décider de simuler par microprogramme un mécanisme de segmentation et de pagination de la mémoire.

Dans la troisième partie nous tentons d'évaluer les performances du système frontal. Les résultats obtenus permettent d'en attendre des performances intéressantes. Il nous a cependant été impossible de déterminer les améliorations de performances de la machine 10070 faute de pouvoir en mesurer les performances actuelles.

*

* *

ORGANISATION DU SYSTEME PROPOSE

Chapitre I

ASPECT CONVERSATIONNEL DES SYSTEMES TIME SHARING

1 - INTRODUCTION

Il n'est pas facile de définir ce qu'est le "Time Sharing". En effet, à quelque niveau que ce soit de l'organisation du système, il est possible de donner des exemples d'organisation totalement différents. On doit plutôt définir le "Time Sharing" par ce qu'il ambitionne de proposer à l'utilisateur. Quelle que soit la technique utilisée, le but est de fournir une possibilité de dialogue entre l'utilisateur et le système. Suivant le degré de perfectionnement du système, les possibilités de dialogue seront plus ou moins étendues.

Ceci suppose l'adaptation :

. D'une part, de l'environnement hardware du calculateur. Les terminaux utilisés comme supports physiques du dialogue sont lents et utilisent un mode de transmission série. Ils réalisent un compromis entre la vitesse de réflexion de l'utilisateur, le volume et la fréquence des échanges, et le coût de transmission. Leur utilisation nécessite la présence d'éléments spécifiques qu'il est important de rentabiliser.

. D'autre part, de son environnement software. Dans toute exécution d'un travail sur calculateur, on peut distinguer deux phases : la phase de création du fichier programme et la phase d'exécution. Les systèmes "Time Sharing" se caractérisent par le fait qu'ils sont conversationnels lors de ces deux phases.

L'évolution des systèmes "Time Sharing" s'est faite sur ces deux points. Du point de vue de leur structure, ils ne se distinguent pas des autres systèmes. On peut y trouver de nombreuses raisons. Nous remarquerons essentiellement qu'un système ne peut réaliser uniquement la fonction "Time Sharing" pour des raisons de rentabilisation. La fonction "Time Sharing" est en général obtenue par l'adjonction de modules spécifiques à un système plus classique. C'est ainsi que la plupart des

systèmes "Time Sharing" sont en réalité des systèmes "batch" utilisant la technique du partage du temps d'unité centrale. Les seuls points qui les distinguent concernent la réalisation des entrées-sorties. Il est tentant dans ces conditions de reporter à l'extérieur du système lui-même un maximum de modules spécifiques du "Time Sharing".

Nous nous proposons, dans ce chapitre, de définir ces modules, et donc d'examiner en particulier les différentes étapes de réalisation des entrées-sorties :

- acquisition de l'information élémentaire en mode série et assemblage en caractères

- assemblage des caractères reçus en messages

- interprétation des messages.

Nous examinerons ensuite l'organisation des systèmes time sharing en ce qui concerne leur aspect conversationnel. Puis nous tenterons de définir dans quelle mesure certaines fonctions de ces systèmes peuvent être prises en compte par des processeurs d'entrées-sorties que nous appellerons calculateurs frontaux.

2 - ECHANGES D'INFORMATIONS ELEMENTAIRES

2.1 - Aspects généraux

La liaison entre le terminal de l'utilisateur et le système est réalisée soit directement, soit par le biais du réseau téléphonique. Dans les deux cas la transmission est faite en mode série, c'est-à-dire que les caractères sont émis bit à bit sur la ligne. Ce principe nécessite un codage particulier : codage suivant 11-10-9-7,5 bits selon le code de départ (ASCII, ISØ7, EBCDIC ...) et selon l'utilisation d'un bit de parité ou non.

Dans une architecture classique ceci impose, ou l'utilisation d'autant d'interfaces que de principes de codage, ou l'utilisation d'une interface complexe et coûteuse. Dans l'un ou l'autre cas il n'en reste pas moins que le système doit participer au traitement des caractères reçus ou émis. Ces remarques, associées au fait que les calculateurs de petite taille voient leur vitesse augmenter en même temps que leur prix diminue, ont conduit certains constructeurs ou sociétés de

service à envisager le remplacement de ces structures cablées et figées par un calculateur programmé en fonction des terminaux utilisés.

C'est ainsi que Télésystèmes utilise un Multi 8, et que le système 9200 TS de Philips utilise également un concentrateur programmé.

2.2 - Traitement au niveau caractère

Les fonctions à réaliser au niveau du caractère sont de trois types :

2.2.1 - Contrôle de validité : il n'est réalisé que lors d'une entrée de caractère venant du terminal et doit porter sur trois points :

i) Le code reçu est-il valide ? Ce point est souvent traité succinctement en vérifiant que le code reçu est inclus dans les deux limites extrêmes du code (comparaison arithmétique).

ii) Le code reçu est-il celui qu'a émis l'utilisateur ? Ce point n'est pas toujours traité.

Deux modes de transmission peuvent être utilisés, le mode half duplex et le mode full duplex. Dans le premier cas, le caractère transmis par l'utilisateur est immédiatement imprimé sur son terminal de sortie (imprimante de la machine à écrire), l'impression étant exécutée au niveau local. Dans le second cas, le caractère est d'abord transmis au système puis renvoyé par le système vers le terminal de sortie de l'utilisateur, permettant ainsi un contrôle de validité par l'utilisateur.

iii) Le caractère émis par l'utilisateur a-t-il été pris en compte ? Ce traitement ne peut être réalisé qu'en full duplex.

Le caractère reçu n'étant renvoyé vers le terminal qu'à la demande du système, l'utilisateur a ainsi la possibilité de savoir si son message a été totalement reçu. On remarquera que ce mode de transmission impose un retard entre l'émission d'un caractère et la réception de l'écho par l'utilisateur. Ce retard qui peut sembler gênant en fin de message lorsque la transmission a lieu à la vitesse maximale ne peut en fait être évité dans aucun des deux modes de transmission.

2.2.2 - **Transcodage** : Dans tous les cas il ne peut être réalisé que par le système et il nécessite autant de tables qu'il y a de types différents de terminaux. On conçoit assez bien que ceci alourdisse fortement le système comparativement au traitement réalisé par la fonction transcodage.

2.2.3 - **Interprétation des caractères** : Ce module n'est mis en oeuvre lui aussi que lors d'une entrée de caractère en provenance du terminal.

Dans l'alphabet dont dispose l'utilisateur, un certain nombre de caractères n'a pas de rôle au niveau du message tel qu'il sera interprété ensuite par la fonction time-sharing. Ces caractères n'ont pour but que d'aider l'utilisateur à créer les messages. Les plus couramment employés sont

retour chariot : interprété comme un symbole fin de message

retour arrière : effacement du caractère précédent

annulation de la ligne.

On note une exception : il est toujours donné à l'utilisateur la possibilité d'arrêter toute exécution en cours concernant son programme. L'ordre correspondant est codé suivant un seul caractère interprété comme un message complet.

Les codes alloués à ces caractères varient avec le constructeur du système et l'alphabet disponible sur les terminaux utilisés.

Les deux premières fonctions citées suggèrent ce que peut apporter l'utilisation d'un concentrateur programmé. Il peut sembler que ces traitements sont rapides et ne justifient guère l'emploi d'un processeur d'entrées-sorties ; des mesures réalisées sur un petit système nous ont montré que le taux d'arrivée des caractères peut être de l'ordre de 50/terminal minute. En admettant qu'il n'y ait que 32 utilisateurs, ce qui représente un système moyen, le taux d'entrée serait de 27 caractères/seconde ; c'est-à-dire que toutes les 38 ms en moyenne un changement de contexte du système est nécessaire pour réaliser le traitement des caractères reçus. En admettant que le changement de contexte et le traitement du caractère ne demandent que 200 microsecondes de traitement, ce qui est rapide, le système passe 0,5% de son temps de calcul à traiter les caractères en entrée. Ce temps n'est pas prohibitif

mais compte tenu du nombre de procédures de traitement de ce type, il semble intéressant d'éliminer du calculateur central celles qui peuvent l'être.

2.3 - Principe des concentrateurs programmés

Le concentrateur est un calculateur de petite taille (4 K mots de mémoire) et ne dispose que d'un seul type de périphérique : des coupleurs, par l'utilisation desquels il gère des terminaux du type machine à écrire. Il est relié au calculateur de traitement que nous appellerons calculateur central par le biais d'une ligne d'une unité d'échange ou par le canal normal (entrées-sorties directes). La configuration type d'un tel système serait celle de la figure I_1.

Exemple :

Nous nous baserons sur un exemple dans lequel la vitesse de transfert des terminaux est de dix caractères/seconde et le code suivant onze bits (2 bits départ, 8 bits d'information, 1 bit arrêt). Le nombre de terminaux est de trente deux et les mots mémoire sont de seize bits.

Le software du concentrateur est un superviseur gérant deux types d'interruption :

- les interruptions de rythme associées aux transferts série entre les terminaux et le concentrateur

- les interruptions de rythme associées aux transferts parallèles entre le concentrateur et le calculateur central.

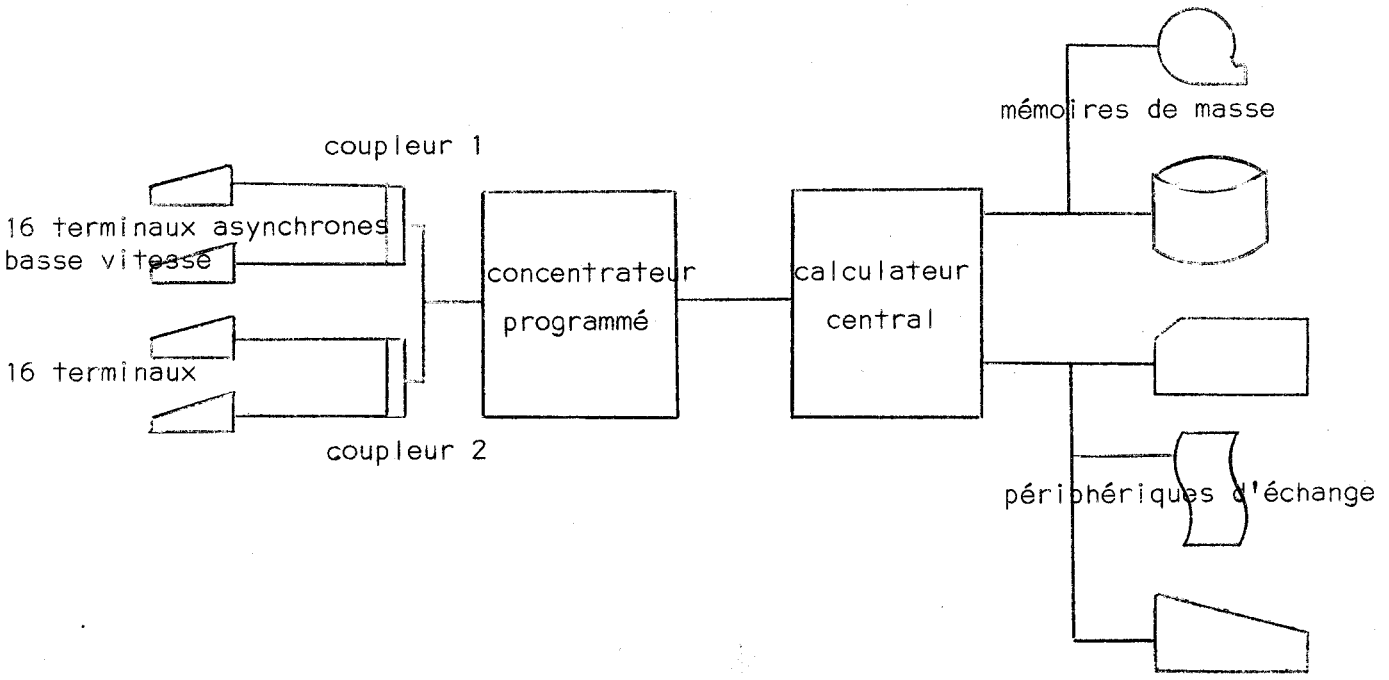


figure I_1 : Configuration type d'un système utilisant un concentrateur programmé.

2.3.1 - Entrée des informations en provenance des terminaux

Le bus d'échange du calculateur d'entrée-sortie ne comportant que seize lignes, on utilise deux coupleurs gérant chacun seize terminaux.

Afin d'assurer un échantillonnage correct, les lignes sont interrogées sept fois par période de base, une période de base correspondant à la vitesse de transmission sur la ligne. Cet échantillonnage est donc réalisé à une fréquence de 770 Hz. On assure ainsi que quel que soit l'instant d'initialisation du transfert, compte tenu de la largeur des signaux transmis sur la ligne, l'information sera sûrement acquise (figure I_2).

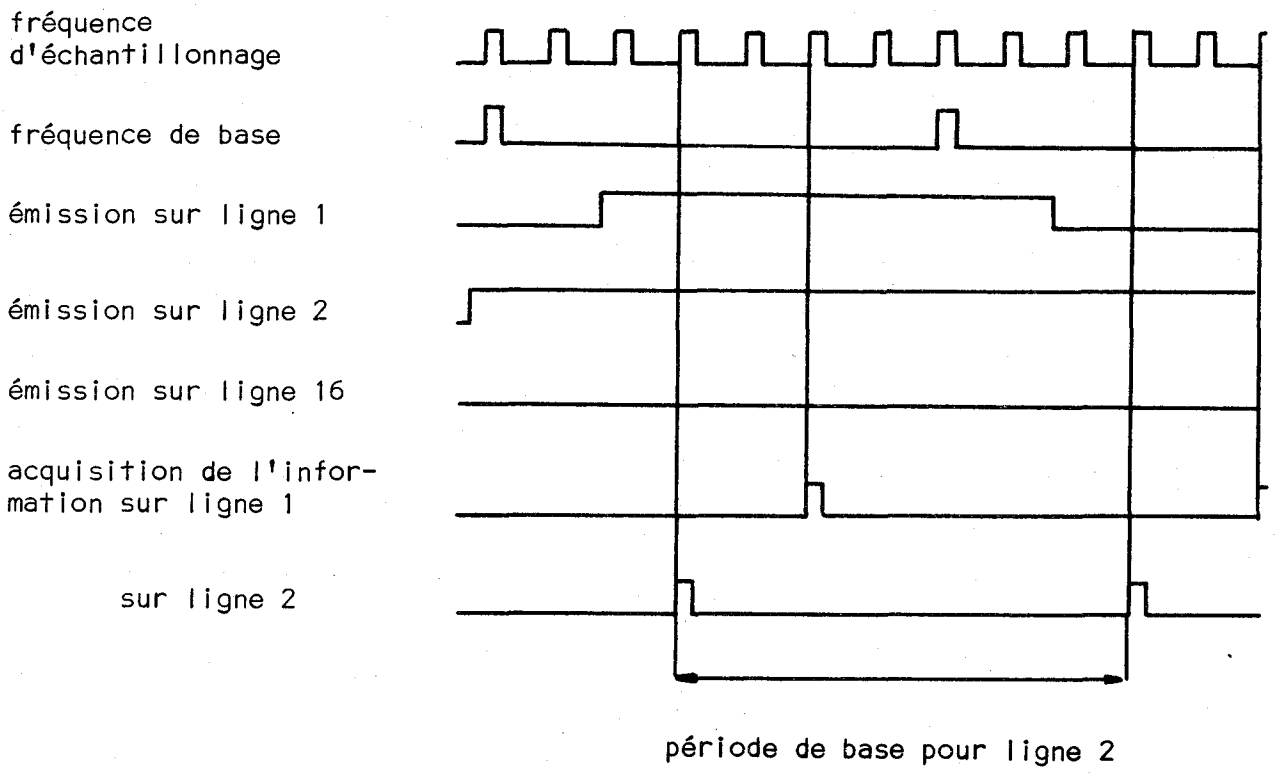


figure I_2 : Principe d'acquisition de l'information.

A chaque top d'échantillonnage (770 fois par seconde) le coupleur est interrogé et les seize bits lus correspondants aux terminaux 1,2,...,16 sont chargés dans un registre de seize bits. Il en est de même ensuite pour le coupleur 2. Au top suivant le même processus est répété et l'information acquise est superposée au contenu du registre déjà chargé. A la fin de la période de base (110 fois par seconde) on est assuré d'avoir reçu toute l'information émise sur les lignes et chaque bit des registres 1 et 2 est ventilé vers un buffer de 16 bits dont les 8 bits de poids fort représentent le numéro de la ligne et les 8 autres le caractère reçu bit à bit sur la ligne. A chaque buffer est associé un mot dans lequel on trouve un compteur de bits permettant de détecter la présence des 2 bits départ puis celle du bit arrêt. Lorsque dans un buffer les 11 bits ont été reçus, le caractère est complet et donc prêt au transfert vers le calculateur central.

Tous les caractères complets sont chargés dans la file d'attente associée à une procédure de contrôle de validité et de transcodage qui, à son tour, les charge dans la file d'attente de la procédure de transfert vers le calculateur central.

On remarquera que le calculateur gérant les lignes de transmission est ainsi nettement sous-utilisé. En effet, supposons qu'à chaque interruption de rythme un bit d'information soit reçu et que le traitement correspondant nécessite un temps d'unité centrale de cent microsecondes, ce qui semble élevé, le taux d'oisiveté du calculateur serait de 67 pour cent. Le transcodage et le contrôle de validité du code reçu ne peuvent suffire à compléter l'utilisation du calculateur.

2.3.2 - Sortie des informations vers les terminaux

Périodiquement, le calculateur central interroge le concentrateur afin que celui-ci lui fournisse les caractères reçus. Dans le cas présent cette période serait de 100 ms correspondant à la fréquence d'arrivée de caractères complets. Les caractères sont transmis suivant 16 bits, 8 bits étant utilisés pour indiquer la ligne d'origine. Sitôt cette phase réalisée, le calculateur central fournit au concentrateur l'ensemble des caractères à émettre en sortie. Ceux-ci sont rangés chacun dans un buffer à partir duquel est réalisée la sérialisation. A chaque top d'horloge correspondant à une période de base (110 fois par seconde), après que le contenu des lignes d'entrée ait été acquis pour chacun des coupleurs, un bit de chaque buffer est présenté dans le registre de sortie. La sortie est alors réalisée successivement pour chaque groupe de 16 lignes.

2.3.3 - Remarque : Le groupement par 16 des lignes permet au besoin l'utilisation de deux types différents de terminaux, chaque type étant attaché à un contrôleur spécifique.

On peut résumer le fonctionnement du concentrateur à l'aide des diagrammes des figures I_3 et I_4.

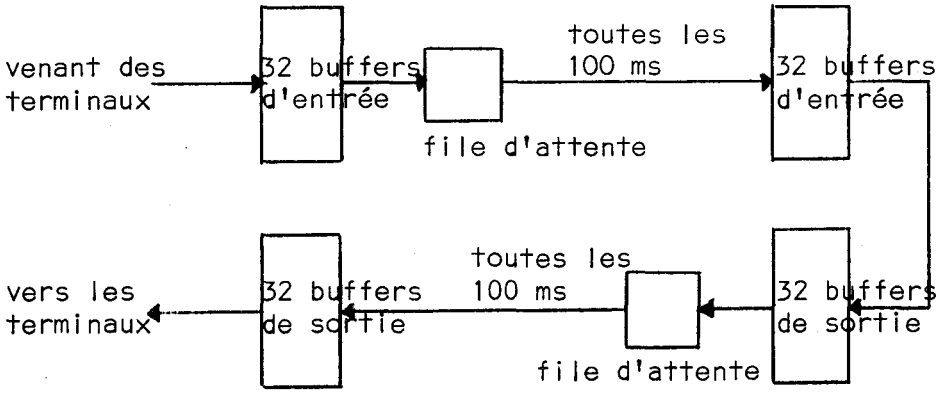


figure I_3 : Liaison entre calculateur central et concentrateur programmé

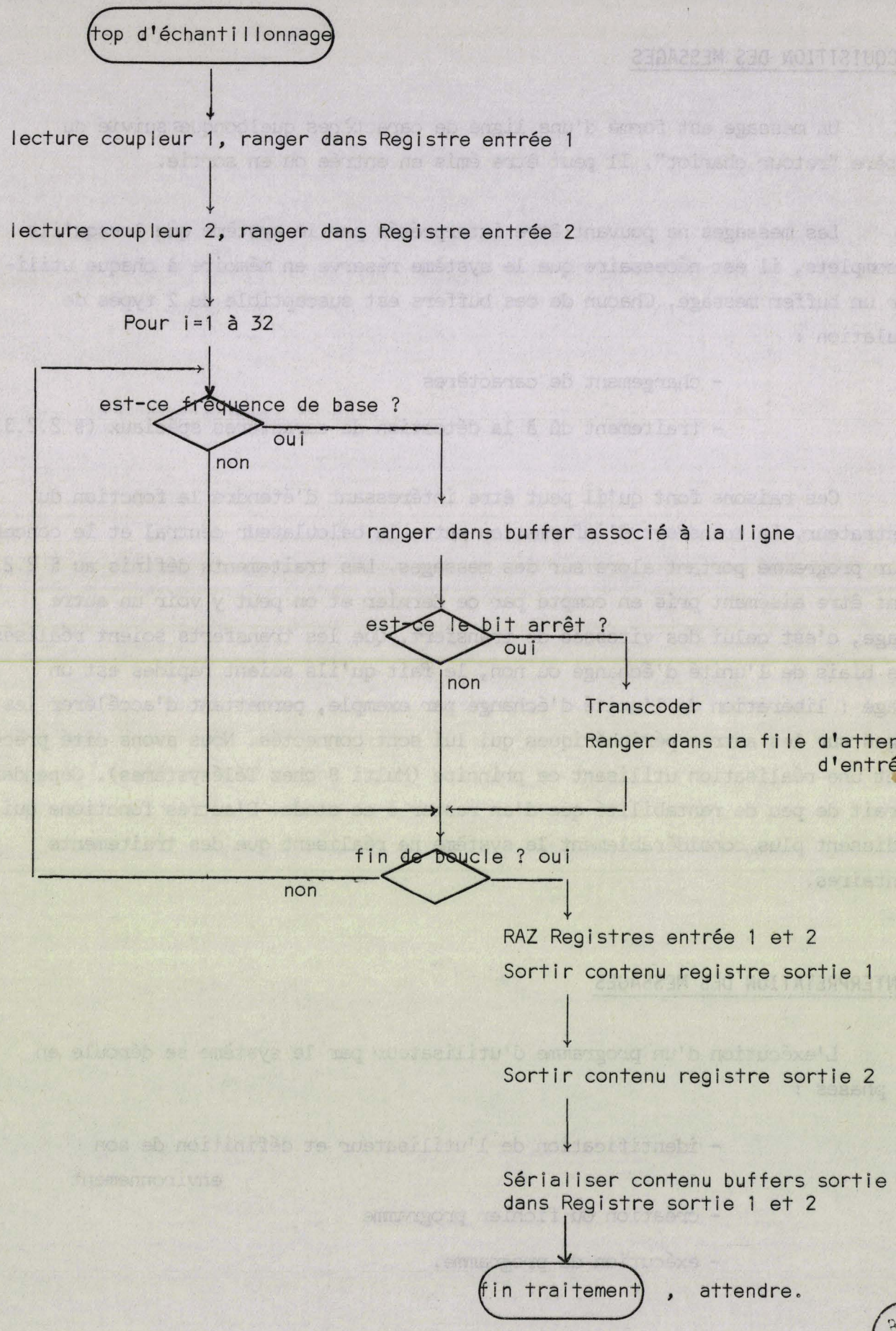


figure I_4 : Traitement caractère au niveau du concentrateur.

3 - ACQUISITION DES MESSAGES

Un message est formé d'une ligne de caractères quelconques suivie du caractère "retour chariot". Il peut être émis en entrée ou en sortie.

Les messages ne pouvant être interprétés par le système que lorsqu'ils sont complets, il est nécessaire que le système réserve en mémoire à chaque utilisateur un buffer message. Chacun de ces buffers est susceptible de 2 types de manipulation :

- chargement de caractères
- traitement dû à la détection de caractères spéciaux (§ 2.2.3).

Ces raisons font qu'il peut être intéressant d'étendre la fonction du concentrateur. Le transfert d'information entre le calculateur central et le concentrateur programmé portant alors sur des messages. Les traitements définis au § 2.2 peuvent être aisément pris en compte par ce dernier et on peut y voir un autre avantage, c'est celui des vitesses de transfert. Que les transferts soient réalisés par le biais de l'unité d'échange ou non, le fait qu'ils soient rapides est un avantage : libération de l'unité d'échange par exemple, permettant d'accélérer les échanges sur les autres périphériques qui lui sont connectés. Nous avons cité précédemment une réalisation utilisant ce principe (Multi 8 chez Télésystèmes). Cependant ce serait de peu de rentabilité que d'en rester à ce stade. D'autres fonctions qui alourdissent plus considérablement le système ne réalisent que des traitements élémentaires.

4 - INTERPRETATION DES MESSAGES

L'exécution d'un programme d'utilisateur par le système se déroule en trois phases :

- identification de l'utilisateur et définition de son environnement
- création du fichier programme
- exécution du programme.

Ces trois phases définissent trois fonctions du système time sharing :

- le moniteur time sharing qui permet à l'utilisateur de dialoguer avec le système, afin de définir une tâche à exécuter.
Il est activé par les commandes

  . d'identification

  . d'appel au compilateur

  . d'exécution

  . de création de fichiers

  . de sauvegarde etc...

- l'éditeur de texte par lequel l'utilisateur crée son fichier programme à l'aide de commandes du type :

  . effacer

  . insérer

  . modifier

  . commandes implicites de rangement de lignes de programme

- le processeur virtuel sur lequel le programme est compilé puis exécuté. Il donne lieu à des messages :

  . d'entrée de données

  . de sortie de résultats

  . d'erreurs à l'analyse syntaxique, à la compilation, à l'exécution.

On constate qu'un nombre important d'opérations porte en fait sur le fichier programme. Lorsque celui-ci est créé, le système n'a plus d'aspect conversationnel que dans la mesure où il permet l'échange de données et de résultats. Si la création du fichier programme est prise en compte par un processeur externe (calculateur frontal), les tâches réalisées par le système sont alors très proches d'une organisation de type "batch processing".

En fait ceci n'est pas vraiment exact et varie avec le niveau auquel on définit que le fichier programme est créé.

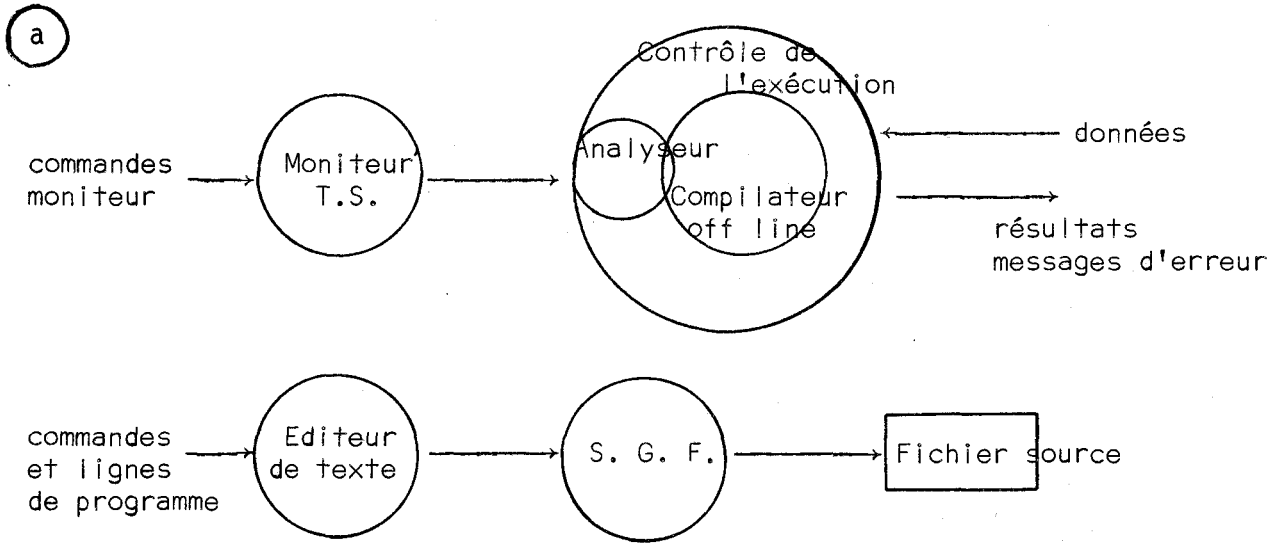
5 - ORGANISATION DES SYSTEMES TIME SHARING CLASSIQUES

Par systèmes classiques nous entendons systèmes dans lesquels n'intervient pas de calculateur frontal mais où l'acquisition des informations en provenance des terminaux conversationnels est réalisé à l'aide d'un concentrateur aussi bien cablé que programmé.

Sans que la stratégie employée par le système intervienne dans cette distinction, on peut définir deux classes de systèmes time sharing.

5.1 - Certains systèmes ne réalisent en fait que des traitements de type batch multiprogrammé ou non, parmi lesquels les programmes émis par les utilisateurs conversationnels sont de plus haute priorité. C'est le cas en général des petits systèmes proposés par les sociétés de service des constructeurs (exemple : Philips 9200 TS, GE 235).

Dans ce type d'organisation, le fichier programme source est créé par l'utilisateur à l'aide de l'éditeur de texte, puis sur sa demande,par l'émission d'une commande EXECUTER, le programme est analysé, compilé et enfin exécuté. Ce mode de fonctionnement est schématisé par le diagramme de la figure I_{5a}, il utilise ce que nous appellerons un compilateur "off line".



b

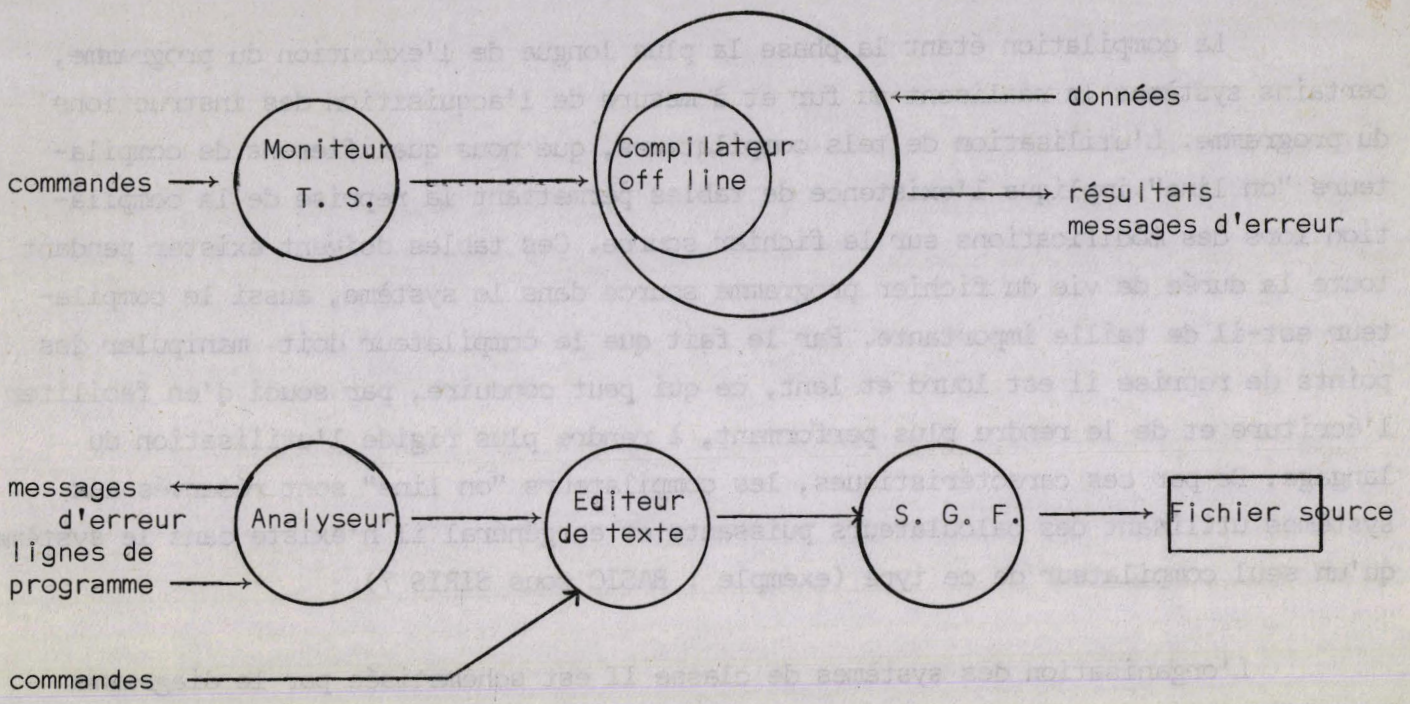


figure I_5 : Organisation des systèmes de classe I.

Dans un autre mode (figure I_{5b}), la compilation est réalisée "off line" mais le système dispose d'un analyseur conversationnel utilisé au fur et à mesure de l'acquisition des instructions. C'est généralement le cas lorsque le calculateur est plus puissant et pour des langages n'ayant pas une structure par blocs (Basic en général). Ce mode permet de ne mettre en oeuvre le compilateur que pour des programmes syntaxiquement corrects, c'est un moyen de décharger le système : des mesures globales réalisées pendant un mois sur les passages réalisés par des programmeurs ayant en moyenne un an d'expérience, nous ont montré que 30% des programmes soumis étaient rejetés pour erreurs détectées au cours de l'analyse syntaxique. C'est en outre un moyen de fournir à l'utilisateur une réponse rapide sur la validité des messages transmis, donc un "service rendu" supplémentaire.

5.2 - On constate que les systèmes de classe I imposent un temps de réponse à l'exécution important. Nous appellerons temps de réponse à l'exécution, le temps écoulé entre la frappe par l'utilisateur du caractère "fin de message" du message EXECUTER et la sortie du premier résultat fourni par son programme sur l'imprimante du terminal. Ce temps peut varier de quelques dizaines de secondes à quelques minutes

suivant la charge du système et la taille du programme à compiler et à exécuter.

La compilation étant la phase la plus longue de l'exécution du programme, certains systèmes la réalisent au fur et à mesure de l'acquisition des instructions du programme. L'utilisation de tels compilateurs, que nous qualifierons de compilateurs "on line" implique l'existence de tables permettant la reprise de la compilation lors des modifications sur le fichier source. Ces tables doivent exister pendant toute la durée de vie du fichier programme source dans le système, aussi le compilateur est-il de taille importante. Par le fait que le compilateur doit manipuler des points de reprise il est lourd et lent, ce qui peut conduire, par souci d'en faciliter l'écriture et de le rendre plus performant, à rendre plus rigide l'utilisation du langage. De par ces caractéristiques, les compilateurs "on line" sont réservés aux systèmes utilisant des calculateurs puissants et en général il n'existe dans le système qu'un seul compilateur de ce type (exemple : BASIC sous SIRIS 7).

L'organisation des systèmes de classe II est schématisée par le diagramme de la figure I_6.

5.3 - Remarque

Nous avons défini en classes I et II des systèmes "monolangages" . En réalité, il convient de créer une classe III qui pourrait s'intituler systèmes "multilangages". Cette distinction est particulièrement utile pour ce qui concerne les systèmes de classe II. En effet, nous avons indiqué que ces systèmes étaient réalisés, de par la lourdeur des compilateurs "on line", sur des calculateurs puissants. Souvent ces systèmes laissent à l'utilisateur la disposition de plusieurs langages. Il ne présente guère d'intérêt que tous utilisent un compilateur "on line", aussi dans la fonction Time Sharing de ces systèmes, les deux types de compilateur voisinent-ils. Lorsque la taille du calculateur le permet, aux compilateurs "off line" sont associés des metteurs au point permettant la modification de la version compilée du programme de l'utilisateur d'une manière conversationnelle (figure I_7). On peut citer à ce propos FORTRAN sous SIRIS 8.

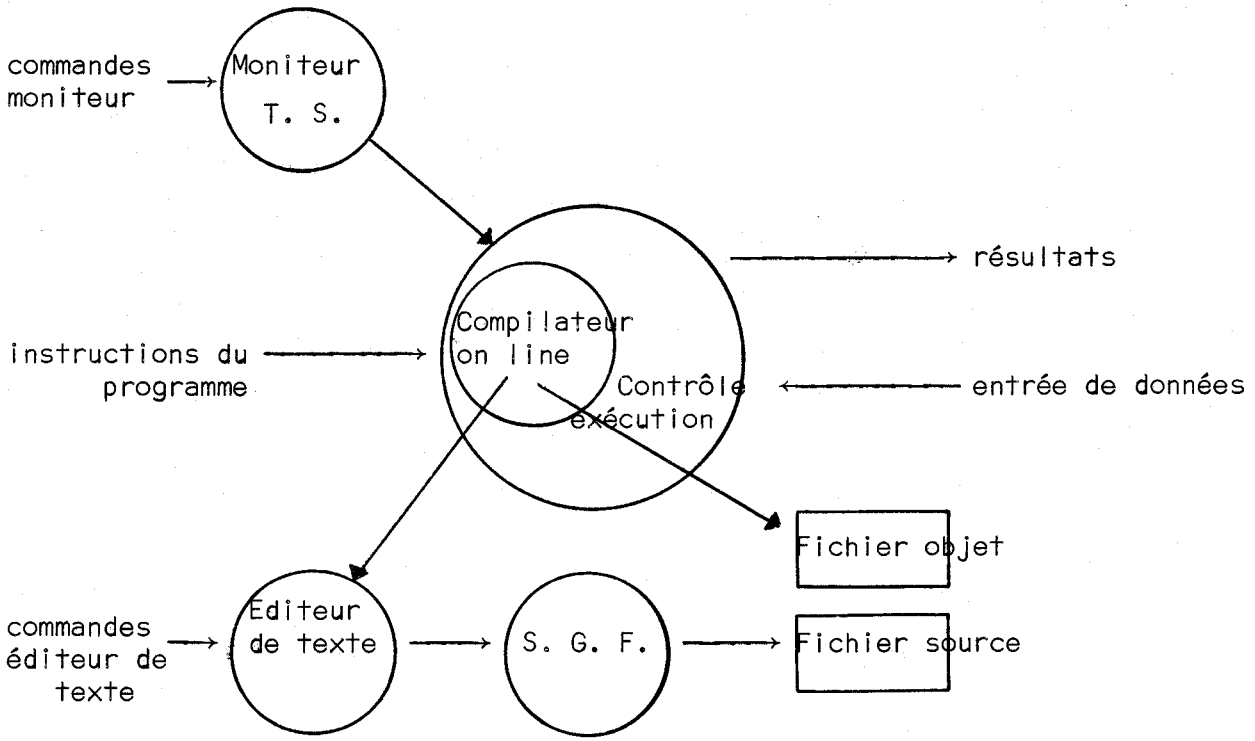


figure I_6 : Organisation des systèmes de classe II.

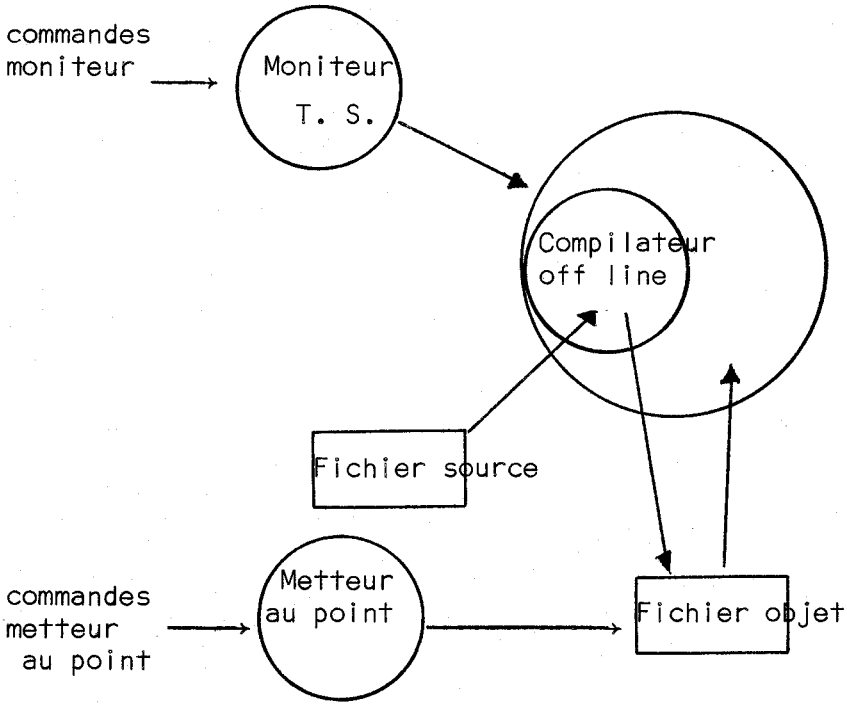


figure I_7 : Utilisation d'un metteur au point.

6 - UTILISATION D'UN CALCULATEUR FRONTAL

L'utilisation qui peut être faite dépend essentiellement du type de service que réalise le système. Suivant que le système est de classe I ou II, le processeur d'entrées-sorties peut participer plus ou moins à l'exécution des tâches de service.

Si l'on excepte les messages échangés avec le processeur virtuel de l'utilisateur (§ 4) et les lignes d'instructions, on voit que les messages susceptibles d'être échangés sont des commandes émises par l'utilisateur. Devant être compréhensibles pour ce dernier, ce sont des messages qui nécessitent, avant toute utilisation par le système, une analyse syntaxique et une interprétation. Un rôle important du calculateur frontal sera de réaliser la première de ces fonctions afin d'assurer leur transcodage dans un langage plus directement assimilable par le système.

En associant au calculateur frontal une mémoire secondaire, les tâches qu'il réalise peuvent être étendues sans qu'il soit besoin d'augmenter la taille de sa mémoire, en particulier il peut être chargé de la gestion des catalogues des utilisateurs conversationnels. Si cette mémoire secondaire est un disque de taille suffisante, les fichiers source de l'utilisateur peuvent y être créés à l'aide de l'éditeur de texte qui est alors implanté dans le calculateur frontal.

Les réalisations appliquant ces principes sont peu nombreuses. On peut citer :

- Diamag 2 réalisé à l'I.M.A.G. qui se rattache à la classe II

- le projet M.C.S.S. d'I.B.M. sur lequel les informations sont peu précises mais qui semble appartenir à la classe I.

6.1 - Diamag 2 [1]

Diamag 2 est formé d'un PDP 8 (Digital instruments) couplé par entrée-sortie directe à un IBM 7044. Les terminaux conversationnels sont reliés au PDP 8 par un multiplexeur. La configuration du système est schématisée sur la figure I_8.

Diamag 2 est un système de classe II "monolangage". Ce langage est un ALGOL conversationnel. Ceci impose, afin de faciliter la génération de code, des contraintes à l'utilisateur lors de la manipulation du fichier source. En particulier les lignes d'instruction sont numérotées par le système et lorsqu'une rupture de séquence doit avoir lieu lors de la création du Fichier source, celle-ci nécessite l'émission par l'utilisateur d'une commande particulière d'appel de l'éditeur de texte (commande MØDIFIER). Afin d'assurer le contrôle de l'exécution, celle-ci se fait suivant une méthode interprétative du code généré à la compilation.

Les deux calculateurs ont leur organisation propre et travaillent en parallèle.

Le PDP 8
- . gère les terminaux
- . tient à jour la table des utilisateurs et la table des fichiers
- . décodifie les commandes et vérifie leur validité
- . gère les communications avec le 7044.

Le 7044
- . manipule les fichiers
- . exécute les phases de génération et d'interprétation conversationnelles

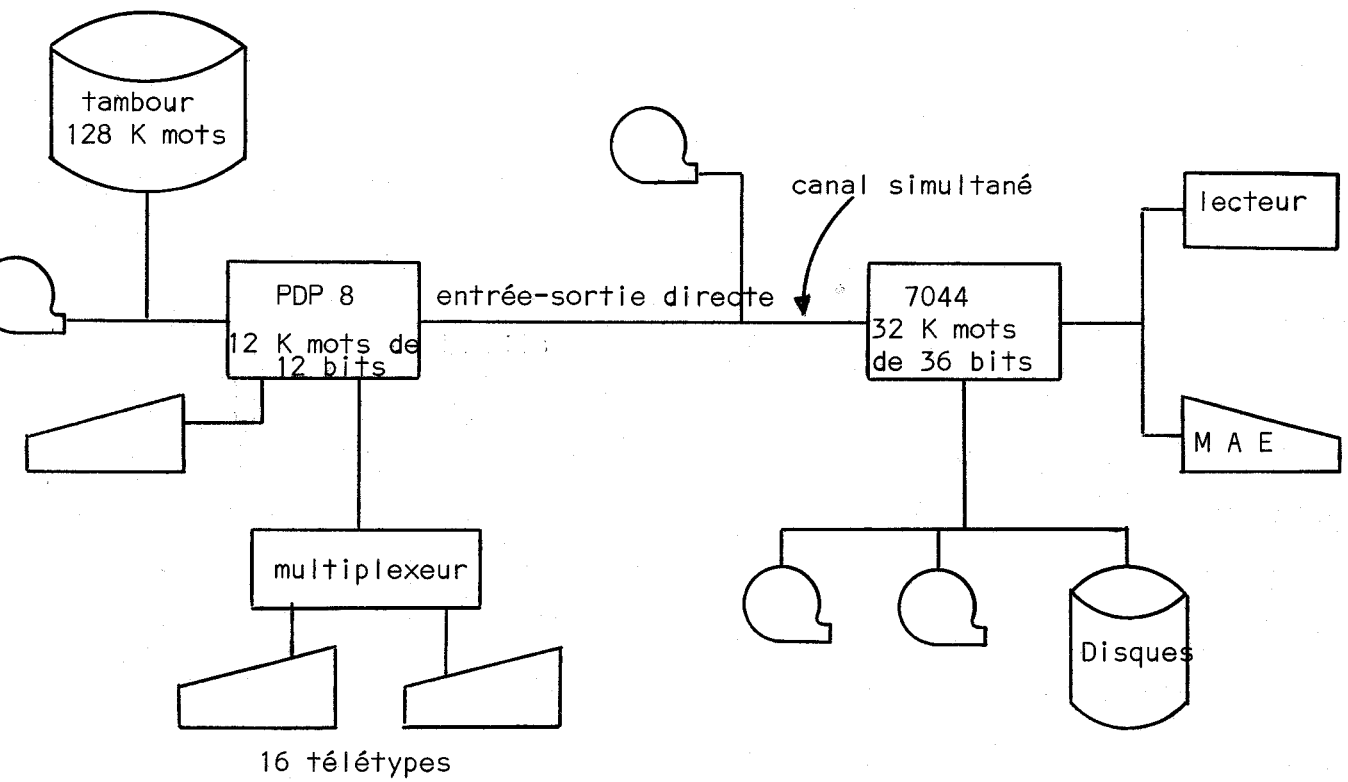


figure I_8 : Configuration de Diamag 2.

Le traitement au niveau du 7044 respecte les files d'attente créées par le PDP 8 et correspondant aux trois classes de travaux :

. génération de code
. interprétation du code généré
. travail sur fichier.

6.2 - Modular Computer Sharing System (figure I_9) [2]

Ce système utilise un réseau formé de M calculateurs de traitement (C.T. 1 à M) et d'un calculateur de contrôle (C.C.). Dans la définition de ce projet tous ces calculateurs sont de type IBM 1130. Les auteurs se basent sur le fait que les besoins des utilisateurs de systèmes Time Sharing sont relativement faibles au regard des travaux qu'ils réalisent et qu'au besoin le même réseau peut être réalisé avec des calculateurs plus puissants. Le réseau gère N terminaux et P disques ($P \geq N$). Le principe de MCS est d'attribuer un calculateur de traitement à chaque utilisateur émettant un ordre d'exécution. Le calculateur de contrôle réalise toutes les fonctions d'édition et gère les terminaux (*), il joue le rôle de "Scheduler" pour l'ensemble des calculateurs de traitement. Ceux-ci sont utilisés en monoprogrammation.

A la mise en connexion d'un utilisateur, le calculateur de contrôle lui attribue un disque. Ceux-ci sont distribués à raison de un par utilisateur. Les lignes de programme reçues par le calculateur de contrôle sont chargées sur ce disque. A la réception d'une commande d'exécution le calculateur de contrôle examine l'état de chaque calculateur de traitement dans une table d'état du système qu'il possède en mémoire. Si l'un d'eux est libre, la commande lui est transmise et le disque de l'utilisateur lui est connecté. Sinon le calculateur de contrôle détermine si l'une des exécutions en cours ne dépasse pas le quantum de temps qui lui a été alloué ; lorsque c'est le cas, le calculateur de traitement en cause reçoit l'ordre de "swapper" le contenu de sa mémoire sur le disque de l'utilisateur pour lequel il travaille. La commande d'exécution lui est ensuite transmise et le disque du nouvel utilisateur lui est connecté.

(*) C'est le rôle qui est attribué en partie au Datanet 30 dans certaines réalisations de General Electric.

Tous les échanges sont réalisés par le calculateur de contrôle seul. Ainsi lorsqu'une entrée de donnée est demandée par un programme, le calculateur de traitement sur lequel il s'exécute transmet la demande et toutes les informations nécessaires à cette acquisition au calculateur de contrôle qui dialogue alors avec le terminal.

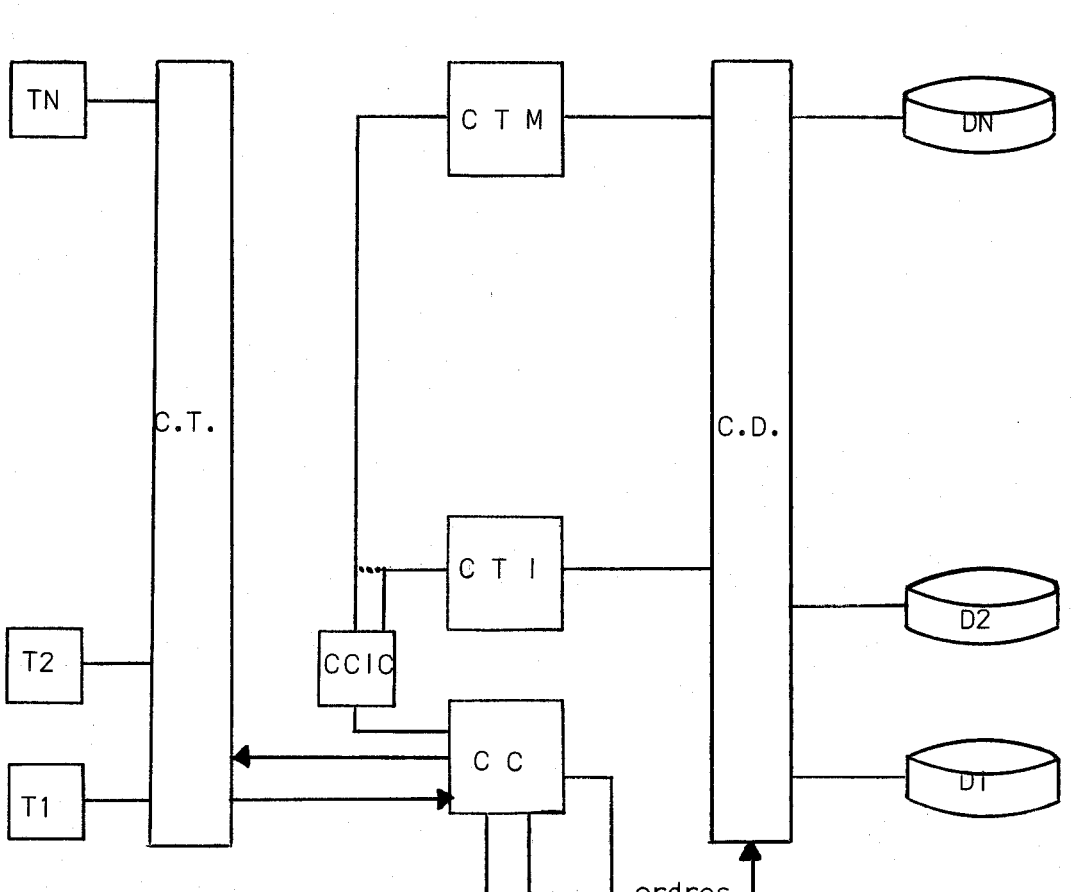


figure I_9 : Configuration du système M.C.S.

Ces différentes fonctions nécessitent un environnement hardware particulier tel que :

. commutateur (C.D.) permettant la mise en connexion simultanée de M+1 disques et calculateurs, sous contrôle du calculateur C.C.

. contrôleur de communication intercalculateurs (C.C.I.C.)

. contrôleur de transmissions réalisant le multiplexage des informations échangées entre le calculateur de contrôle et les terminaux.

On peut noter que le système peut être aussi bien "monolangage" que "multilangage" suivant la structure de base des calculateurs composant le réseau. Son architecture est intéressante dans la mesure où étant totalement modulaire elle résout les problèmes de reconfiguration pour cause de panne.

On remarquera cependant que cette structure en réseau hiérarchisé rend peu pratique l'utilisation de compilateurs "on line". De plus, si cette architecture semble optimale pour ce qui concerne les systèmes de classe I, on peut montrer en fait que l'utilisation de plusieurs calculateurs moins puissants est dans une certaine mesure moins intéressante du point de vue du temps de réponse du système que l'utilisation d'un seul calculateur très puissant.

Soit un système (C,M) où C est la capacité d'exécution du calculateur (nombre d'opérations élémentaires exécutables par seconde) et M le nombre de terminaux ; son comportement peut être décrit par le rapport du temps de réponse moyen au temps de service moyen demandé [3]. La comparaison de ce système à des systèmes formés de N calculateurs indépendants (C/N, M/N) fournit les courbes de la figure I_{10} où la pente des asymptotes est N. On constate ainsi que les performances se dégradent d'un système à l'autre.

Dans la suite, nous ne considèrerons donc plus que des systèmes comportant un seul calculateur de traitement. La présence du calculateur frontal se justifie alors non par sa participation au traitement mais par l'aide qu'il apporte à la réalisation des tâches de gestion. En conséquence cette configuration ne peut être intéressante que dans la mesure où le calculateur frontal est de puissance réduite par rapport au calculateur de traitement.

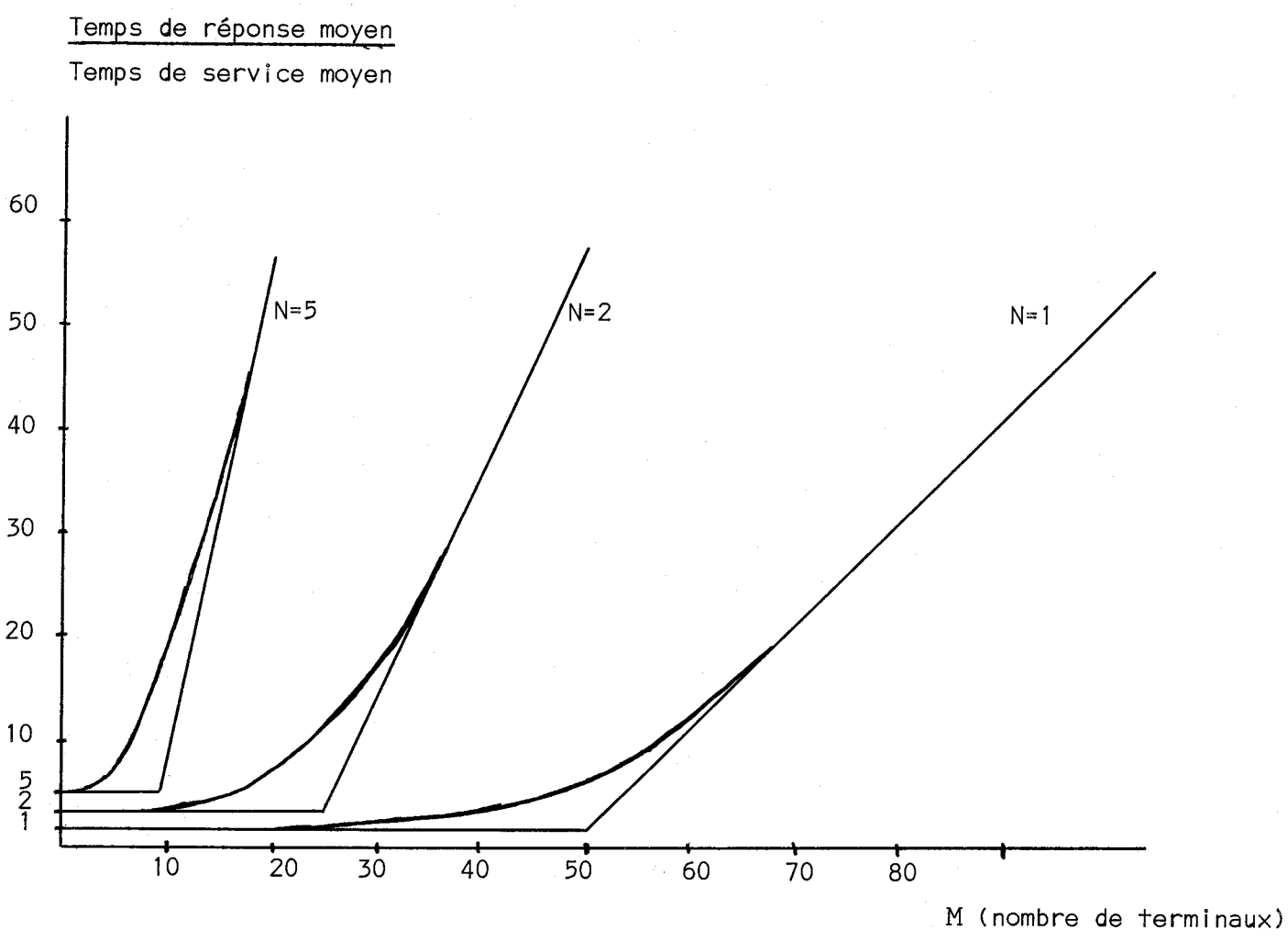


figure I_{10} : Performances comparées des systèmes à N calculateurs (C/N, M/N)

Chapitre II

RECHERCHE D'UNE STRUCTURE ADAPTEE AU TIME SHARING

I - INTRODUCTION

Différents points de vue peuvent être adoptés dans la recherche d'une structure de système time sharing utilisant des calculateurs frontaux. Ainsi qu'il est apparu précédemment il est intéressant de rejeter à l'extérieur du système central le traitement des caractères, d'autant que celui-ci s'insère mal dans ce système. L'utilisation d'un calculateur frontal permet alors de gérer les terminaux conversationnels, de la même manière que des imprimantes ou des lecteurs de cartes, c'est-à-dire comme des fichiers logiques dans lesquels l'accès se fait article par article. C'est cette solution qui a été retenue à la définition de Diamag 2. Pourtant l'utilisation des calculateurs frontaux se fait souvent en fonction du matériel disponible et de telles structures ne sont en général adoptées que pour simplifier l'organisation d'un système mal adapté. La recherche d'une architecture porte rarement sur cette architecture elle-même, celle-ci est davantage considérée comme un support permettant la mise en place, d'une manière simple, d'un système d'exploitation donné. Les contraintes de matériel sont alors importantes. C'est ainsi que Diamag 2 se limite au traitement des messages parce que les mémoires secondaires du PDP 8 sont de taille restreinte (128 K mots). Nous avons tenté dans cette étude de nous affranchir de ces contraintes. Il apparaît alors que, sans remettre en cause les principes d'exploitation du calculateur central, l'on peut étendre assez fortement les fonctions du calculateur frontal. Seules quelques interfaces simples doivent être mises en place.

Dans ce chapitre, nous définirons le rôle qui doit être attribué au calculateur frontal, puis les objectifs étant connus, nous rechercherons une architecture qui permette leur réalisation. Enfin nous préciserons les fonctions attribuées à chaque partie du système et leurs articulations.

2 - ROLE DU CALCULATEUR FRONTAL

2.1. - Définitions

. Nous appellerons utilisateur toute personne ayant une possibilité d'accès au système, c'est-à-dire à laquelle a été attribué un numéro de compte.

. Un usager sera un utilisateur ayant une possibilité de dialogue avec le système, donc relié à lui par un terminal.

. On définiera par Session, la tranche de temps pendant laquelle un utilisateur est usager.

. Une tâche sera définie par le traitement particulier qu'elle réalise, par exemple le traitement caractère ou l'analyse.

. Une phase ou interaction englobe plusieurs traitements de base correspondant à des tâches interactives. On parlera par exemple de phase d'acquisition ou de phase de compilation etc...

2.2 - Gestion des lignes

La gestion des lignes consiste en trois fonctions distinctes.

2.2.1 - Le repérage des demandes d'entrée en Session des utilisateurs

Lorsqu'un utilisateur demande à entrer en session, le système doit établir si la connexion est réalisable. Ceci est réalisé en deux temps dans la phase d'initialisation :

- Par comparaison avec l'état de charge actuel du système. C'est en particulier le cas sur les gros systèmes d'exploitation où la charge time sharing du système peut être définie par l'opérateur. Celle-ci est en général fixée par le nombre d'usagers admis. On peut encore imaginer de fournir une priorité d'accès plus élevée à certains terminaux. Ce traitement est réalisé dès que la connexion téléphonique est établie.

- Si la connexion avec le système est réalisable, l'usager éventuel doit prouver au système qu'il est un utilisateur répertorié en fournissant son numéro de compte.

D'autres opérations plus précises restent alors à réaliser et sont fonction de l'organisation du système. En particulier l'attribution éventuelle d'un tampon et la création de tables établissant la connexion logique du terminal de l'utilisateur au système , doivent être réalisées dans la phase d'initialisation.

2.2.2 - La surveillance des lignes de transmission

Les intermédiaires hardware entre l'usager et le système sont nombreux, leur structure particulière fait que seul le système peut vérifier leur bon fonctionnement. C'est ainsi que doit être pris en charge le repérage de codes invalides. Leur présence implique un mauvais fonctionnement et doit être communiquée à l'opérateur et si possible à l'usager. On peut aller plus loin et envisager le repérage des coupures de transmission afin de permettre dans la mesure du possible la restauration du traitement interrompu si celle-ci est demandée par la suite par l'usager.

2.2.3 - La gestion des terminaux

Nous entendons par là la gestion du dialogue en entrée ou en sortie de l'usager avec le système. Ceci a déjà été envisagé au chapitre précédent.

Cette fonction comprend le transcodage, la réalisation de la tabulation, l'édition au niveau de la ligne c'est-à-dire la prise en compte de caractères spéciaux de suppression, de fin de message etc...

Ces trois fonctions peuvent aisément être réalisées par le calculateur frontal, c'est d'ailleurs souvent le cas dans les réalisations existantes. Cependant, on peut noter que la création des fichiers des usagers se fait indépendamment du reste du traitement exécuté pendant la session, aussi il est envisageable que le calculateur frontal prenne à son compte cette fonction.

2.3 - Création des fichiers source

2.3.1 - Définitions

. On définit par fichier source tout fichier propre à un utilisateur et dont le contenu est un programme en langage source. Chaque article est formé d'une ou plusieurs instructions en langage symbolique.

. Un fichier est dit courant lorsque des accès peuvent y être effectués. C'est-à-dire lorsque celui-ci est connu du système. Ceci implique que le fichier ait été nommé et que la liaison fichier logique - fichier physique ait pu être assurée par le système.

. L'article courant sera donc celui dont l'adresse physique est connue du système. C'est donc le dernier article auquel l'usager a accédé ou éventuellement l'article suivant si le fichier est séquentiel.

2.3.2 - Remarques

- Selon la classe du système, la création des fichiers source peut se faire de deux manières distinctes :

. en indépendance totale avec tout autre traitement, c'est le cas des systèmes de classe I

. en interaction directe avec la compilation, c'est le cas des systèmes de classe II.

Dans les deux cas, elle se fait par appel implicite à l'éditeur de texte. C'est ainsi que les remplacements et insertions d'instructions ne sont pas déclarés, le fichier est créé dans le désordre par l'usager. Ceci implique que la méthode d'accès aux fichiers et leur structure soient figées par le système, aussi aucune déclaration de fichier source n'est en général faite par l'utilisateur. D'une manière générale, la création du fichier source dans ces conditions se fait par appel au processeur de compilation qui, précédé d'un analyseur de commandes, interagit plus ou moins suivant la classe du système avec l'éditeur de texte.

Suivant la classe du système, cet appel pourra être immédiat (système de classe II) ou différé (système de classe I). Dans ce dernier cas l'usager est en réalité en connexion directe avec l'éditeur de texte, l'appel du compilateur n'étant effectué par le système qu'à la suite de l'abandon de l'éditeur de texte par l'usager. Sous Siris 7 ceci peut être réalisé par l'utilisation de commandes cataloguées.

- Puisque l'on envisage d'affecter la fonction de création du fichier au calculateur frontal, il est nécessaire qu'il réalise les éditions de texte de ce type. La structure des fichiers doit alors être choisie de sorte qu'ils soient manipulables par le calculateur frontal dont la mémoire principale sera de taille réduite, mais également par le système central. Ils doivent en particulier permettre un accès séquentiel pour la compilation et rendre l'édition de texte performante.

Lorsque le système est de classe II, il ne s'agit pas seulement de réorganiser le fichier au fur et à mesure de l'émission des messages par l'usager, encore faut-il indiquer au compilateur les ruptures de séquence de sorte qu'il puisse réorganiser ses tables et reprendre le traitement. On résoudra d'une manière simple ce problème en réalisant la réorganisation du fichier au niveau du calculateur frontal puis en communiquant au compilateur l'adresse physique dans le fichier de l'article courant. En procédant de cette manière, on évite l'utilisation du système de Gestion des Fichiers par le compilateur, ce qui aurait pu être source d'un ralentissement important du traitement par rapport à un système classique où le compilateur possède, dans un tampon associé au terminal utilisé par l'usager, l'article courant émis par celui-ci. De plus le compilateur dispose ainsi d'un fichier toujours séquentiel dont il possède l'adresse courante, et les interactions avec l'éditeur de texte sont supprimées au niveau du système central. Vis-à-vis de ce dernier l'usager est réellement "on line" avec le compilateur.

- On doit noter alors qu'il ne serait pas très judicieux de disposer d'un éditeur de texte simplifié au niveau du calculateur frontal et d'un éditeur de texte plus élaboré au niveau du calculateur central pour réaliser la création en mode conversationnel de fichiers de type quelconque. Le calculateur frontal devra prendre en compte toutes les éditions de texte. L'édition de texte demandant peu de traitement en unité centrale et peu d'accès à la mémoire secondaire lorsque la méthode d'accès est adaptée, le calculateur frontal devrait pouvoir la réaliser avec des performances correctes.

2.4 - Analyse syntaxique

Nous avons déjà fait la remarque qu'une mesure nous avait permis de constater que dans une utilisation de type time sharing, 30% des programmes sont rejetés au cours de la compilation pour erreurs de syntaxe. Ceci justifie le choix que nous faisons d'analyser syntaxiquement les messages émis par l'usager au niveau du calculateur frontal. Il n'est pas souhaitable que cette analyse soit complète car elle serait assez longue et ferait double emploi avec la compilation. On peut se contenter d'une analyse grossière portant sur la syntaxe des mots clés, la place des parenthèses et des séparateurs. Une telle analyse peut être rapide et apporte un confort supplémentaire à l'usager dans la mesure où celui-ci est assuré d'une réponse immédiate quant à la validité de son message, quelle que soit la classe du système. De plus elle assure une décharge partielle du système car la compilation est lourde à mettre en oeuvre, aussi vaut-il mieux la faire avec une probabilité de réussite maximale. La réalisation d'une analyse syntaxique immédiate par le calculateur frontal augmente cette probabilité. Nous n'éliminerons pas ainsi les 30% de pertes que nous avons mesurés , mais certainement une part importante de celles-ci.

On peut peut être penser que si le calculateur frontal réalise cette analyse, il est envisageable d'aller plus loin et de lui confier une participation à la compilation. Ce serait alors un moyen de donner un aspect plus conversationnel aux systèmes de classe I. Cependant ceci serait peu efficace dans le cadre des systèmes de classe II car les interactions avec le traitement effectué par le système central seraient très fréquentes. De plus, la puissance du calculateur frontal devant être limitée, le temps de réponse de l'analyseur serait alors trop important pour que cette solution soit interessante.

3 - CHOIX D'UNE ARCHITECTURE ADAPTEE DU SYSTEME AVEC CALCULATEUR FRONTAL

3.1 - Critère de choix

De par les fonctions que nous envisageons de déléguer au calculateur frontal, les critères de choix relatifs à l'architecture du système sont les suivants :

- Le calculateur frontal doit assurer la concentration et la diffusion des données, il doit donc être doté des dispositifs classiques propres à une utilisation en temps réel.

- Il doit disposer d'une voie de dialogue avec le calculateur central afin d'assurer la synchronisation des traitements par l'un et par l'autre, et la transmission des messages de l'usager au système time sharing. Ces messages sont les données à l'exécution et les commandes moniteur. Par cette même voie, le calculateur frontal recevra les messages à transmettre à l'usager (résultats, messages d'erreurs, messages de synchronisation du dialogue usager-système).

- Le calculateur frontal doit avoir accès aux fichiers des usagers.

- Le calculateur frontal doit être d'une puissance réduite (§ I.6.2).

3.2 - Les moyens d'échanges des calculateurs de taille moyenne

Ces calculateurs se distinguent par l'utilisation de processeurs d'entrée sortie qualifiés en général d'unités d'échange. Chacune d'elles est connectée à un canal d'accès direct à la mémoire. Dans ce but, la mémoire dispose de plusieurs voies d'accès de priorité variable, l'une d'entre elles étant réservée à l'unité centrale, c'est en général la moins prioritaire.

Ce mode d'échange par vol de cycle ralentirait fortement l'unité centrale aussi la mémoire est structurée en blocs physiquement indépendants et possédant chacun leur dispositif d'adressage. Ces blocs sont imbriqués de sorte que deux mots d'adresses consécutives n'appartiennent pas au même bloc physique, on tend ainsi à minimiser la probabilité pour que l'unité centrale et une unité d'échange tentent d'accèder au même instant à des informations se trouvant dans le même bloc de mémoire. Les zones mémoire utilisées par chacune des unités de traitement (unité centrale ou unité d'échange) sont fonction du système d'exploitation du calculateur ; aussi dans certains calculateurs (C.I.I. 10070 par exemple), l'imbrication peut être définie par cablage amovible au moment de l'installation du système. Nous tenterons d'utiliser cette particularité.

3.3 - Quelques architectures envisageables

3.3.1 - Le calculateur frontal possède une voie d'accès au calculateur central et gère sur un disque qui lui est propre les fichiers des usagers du Time sharing (figure II_1).

La création des fichiers sources courants se fait alors en indépendance totale du reste du traitement. Vis-à-vis du système chaque usager possède deux fichiers, un fichier d'entrée et un fichier de sortie, dont le support est identique (la mémoire externe du calculateur frontal) et dans lesquels l'accès se fait par blocs de longueur variable définis au cours d'un dialogue préalable entre les deux calculateurs.

On peut noter deux inconvénients dans ce type d'architecture :

i) au niveau du dialogue entre les calculateurs :

Puisque l'on doit prévoir de fournir au système central le contenu du fichier source courant afin qu'il en exécute la compilation, les blocs d'information échangés seront de taille importante (du même ordre de grandeur qu'une page de la mémoire principale du calculateur central). Cet échange ne peut se réaliser que par l'utilisation d'un accès direct à chaque mémoire. Une telle connexion entre les deux calculateurs est délicate, essentiellement en ce qui concerne l'initialisation des échanges.

En effet chacun des calculateurs peut prendre l'initiative du transfert. Même si la décision de l'initialiser appartient au calculateur central, dans certains

cas (transfert d'une commande moniteur par exemple) le transfert est demandé par le calculateur frontal. Il faut disposer d'un moyen de dialogue annexe entre les deux calculateurs, afin de déterminer le sens du transfert. La connexion entre les deux mémoires ne pouvant se réaliser qu'à l'aide de deux canaux d'accès direct mis en opposition, elle nécessite l'initialisation séparée de chaque canal par l'unité centrale correspondante. L'absence d'un dialogue préalable au transfert de l'information pourrait donc occasionner des blocages dûs à des transferts contraires sur chaque canal.

ii) au niveau de l'utilisation du disque :

On constate que le calculateur frontal a en réalité deux rôles bien distincts. D'une part, il réalise la gestion des fichiers des usagers ; d'autre part il constitue pour le calculateur central un contrôleur de disque. Afin d'accélérer les transferts, les disques utilisés sur les gros systèmes sont en général découpés en secteurs dont la capacité est du même ordre de grandeur que celle d'une page de mémoire centrale. Or la taille des pages de mémoire centrale est choisie de façon à réaliser un compromis entre le nombre de vidages sur disque nécessaires à la réallocation de mémoire aux usagers concurrents, et le taux de remplissage des pages de mémoire. Leur taille est en général de 1024 à 2048 octets. Il est donc nécessaire de disposer au niveau du calculateur frontal de buffers de cette taille. C'est une occupation mémoire importante pour un calculateur qui, compte tenu du taux d'occupation de son unité centrale, doit être choisi d'une taille minimale.

Enfin, les transferts sont lents parce qu'ils sont doublés et parfois triplés. Le mode de réalisation des transferts allonge le temps d'accès qui devient égal au temps d'accès sur le disque augmenté du temps de transfert entre le disque et le calculateur frontal.

Une telle architecture présente trop d'inconvénients pour être acceptable. Ceux-ci sont éliminés dans l'exemple suivant.

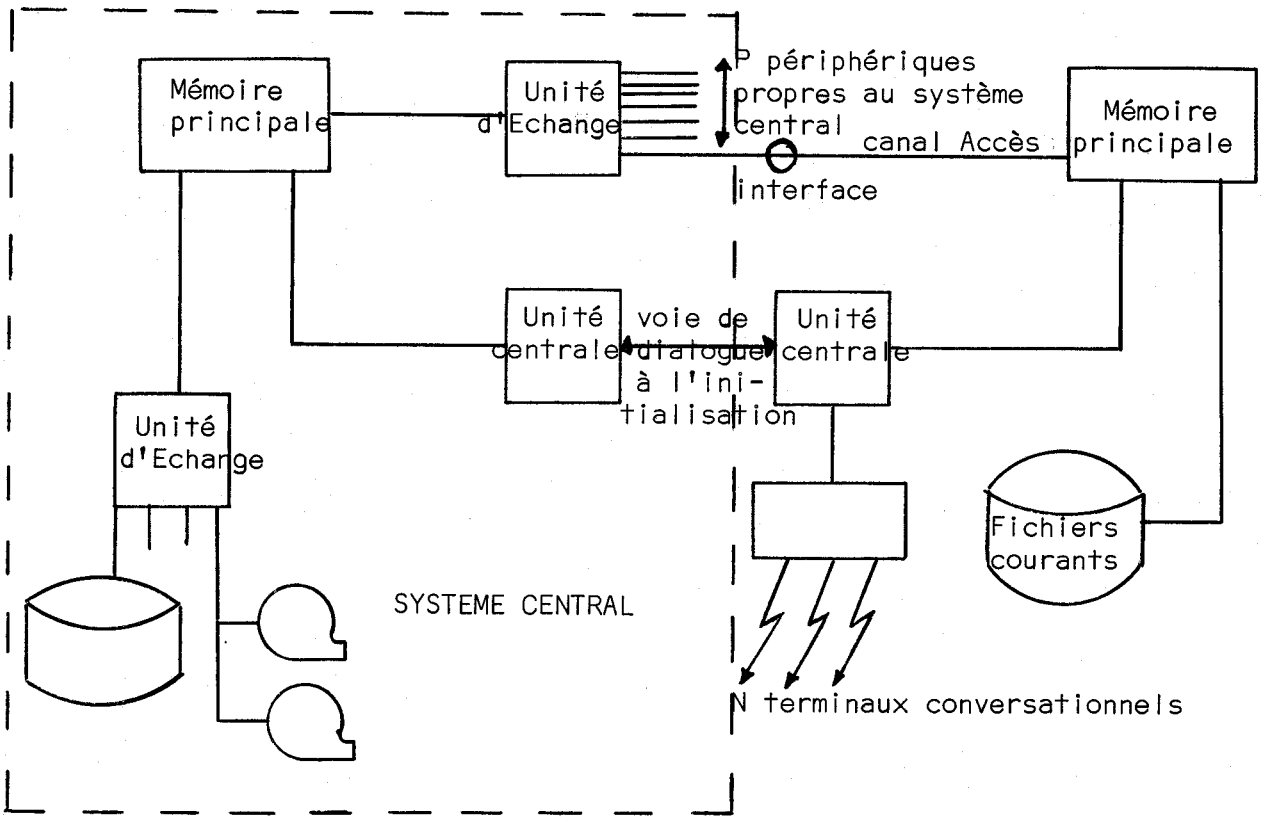


figure II_1 : Le calculateur frontal gère les fichiers courants des usagers et joue le rôle de contrôleur disque

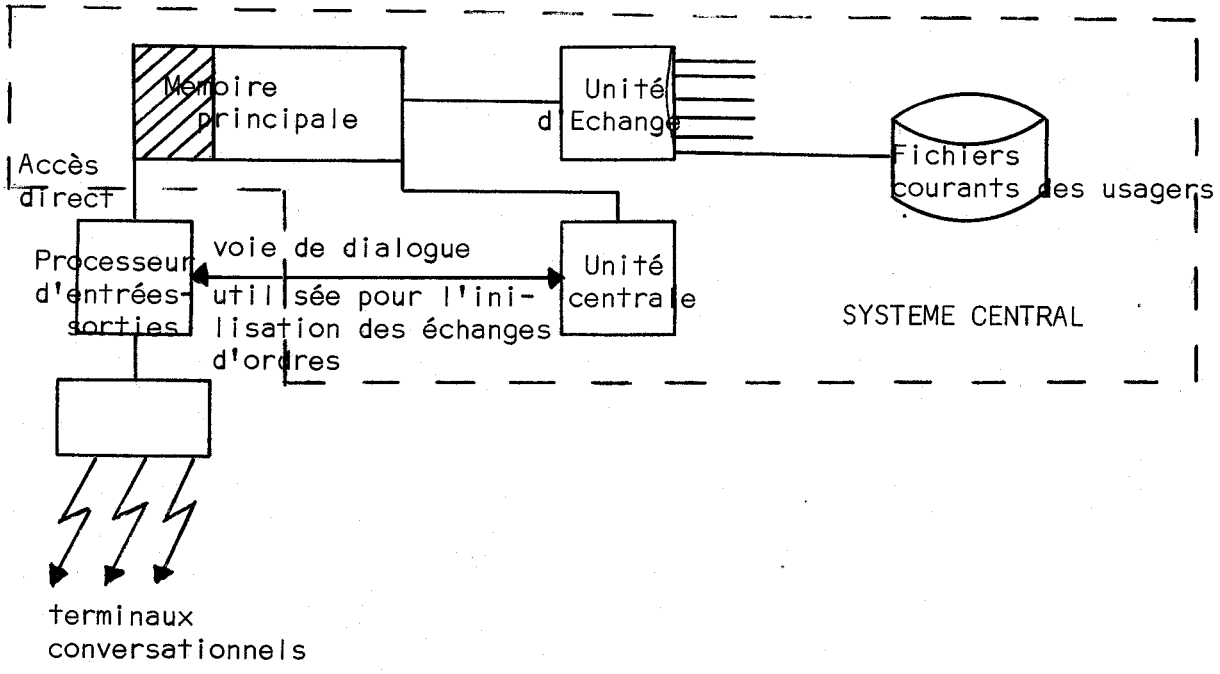


figure II_2 : Le calculateur frontal partage la mémoire du calculateur central

3.3.2 - Le calculateur frontal est un processeur sans mémoire. Il utilise celle du calculateur central (figure II_2).

Nous avons constaté, à l'étude de l'exemple précédent, que la présence d'une voie de transmission rapide entre les deux calculateurs était nécessaire mais difficile à réaliser, et que de plus, il était préférable que les deux calculateurs partagent les mêmes mémoires secondaires afin d'éviter de pénaliser le système par des transferts prohibitifs en nombre et en durée.

Ces critiques conduisent à imaginer que les mémoires secondaires utilisées soient celles du calculateur central ; et que, puisque les transferts doivent se réaliser alors depuis la mémoire principale de ce dernier, elle soit partagée par les deux calculateurs.

Pourtant il ne peut s'agir d'un biprocesseur réel dans lequel chaque unité centrale puisse initialiser à son gré les unités d'échange. Ceci poserait des problèmes de conflit et nécessiterait que le calculateur frontal soit capable d'utiliser le système de gestion de fichiers et les procédures d'accès du calculateur central. Il devrait en particulier posséder un bloc de commande, au moins en partie identique à celui du calculateur central, de façon à pouvoir exécuter les mêmes instructions. Le calculateur frontal travaille ici indépendamment du calculateur central avec lequel il ne communique que pour lui demander d'initialiser des transferts disque que, de toute manière, ce dernier aurait dû réaliser dans un système d'organisation classique (accès aux fichiers sources courants), ou pour échanger des ordres. A cet effet, les deux unités centrales disposent d'une voie de dialogue directe utilisée pour synchroniser les échanges d'ordres stockés à des adresses fixes en mémoire centrale.

Remarquons que le calculateur frontal ne doit pas disposer d'un accès à l'ensemble de la mémoire. D'une part, il n'a pas besoin d'une mémoire importante (c'est un critère de choix que nous imposons), d'autre part, si toute la mémoire était partagée, des problèmes de conflits apparaîtraient. Aussi le calculateur frontal ne dispose que d'une zone fixe en mémoire. L'utilisation d'une zone dynamique de mémoire imposerait une surcharge du système (gestion de l'implantation de cette zone) et des vidages sur disque sans doute fréquents. De plus, le calculateur frontal devrait disposer d'un mécanisme d'adressage plus important qu'il n'est nécessaire. L'implantation de cette zone fixe de mémoire utilise les particularités de l'imbrication. Si on considère que la

mémoire est formée de N blocs, les N-1 premiers blocs sont imbriqués entre eux suivant un schéma particulier au système central qui est le seul à les utiliser. Le dernier bloc est partagé entre le calculateur frontal et le calculateur central et contient M mots d'adresses consécutives. Ceci permet de simplifier l'adressage par le calculateur frontal puisqu'il suffit de réaliser par cablage une translation des adresses qu'il fournit.

Une telle architecture semble particulièrement adaptée, nous y ferons pourtant deux critiques :

. la non imbrication d'un bloc mémoire et la réservation de celui-ci au calculateur frontal limite les possibilités d'extension du système central. La charge maximale acceptable par ce dernier est donc plus faible qu'en cas normal.

. le système central réalise encore la gestion des fichiers et doit de plus réaliser les transferts propres au système du calculateur frontal.

3.3.3. - Le calculateur central et le calculateur frontal partagent la même mémoire secondaire et disposent d'une voie de dialogue leur permettant d'échanger des ordres (figure II_3).

Nous avons constaté dans les exemples précédents que le calculateur frontal devait disposer d'un accès aux fichiers courants des usagers mais que ceux-ci devaient rester accessibles au calculateur central.

Un moyen de respecter ces contraintes est d'utiliser la configuration de la figure II_3. Cependant ceci signifie qu'éventuellement le calculateur frontal doit avoir accès à l'ensemble des mémoires secondaires et en particulier aux disques amovibles de certains utilisateurs. Une telle réalisation est très lourde. Elle nécessite un coupleur à double accès pour chacun des disques partageables. De plus le calculateur frontal doit disposer d'un système de gestion de fichiers (SGF) lui permettant l'accès à tous les types de fichiers admis par le système central. Outre que le S.G.F. fait double emploi avec celui du calculateur central, il est lourd à manipuler et encombrant pour le calculateur frontal. D'autre part, on remarquera que l'optimisation des accès réalisée séparément par chacun des systèmes peut s'avérer inopérante. Cette optimisation est en général réalisée en effectuant un tri des demandes présentes dans la file d'attente de sorte à réaliser séquentiellement les accès à des secteurs d'adresses croissantes. Si

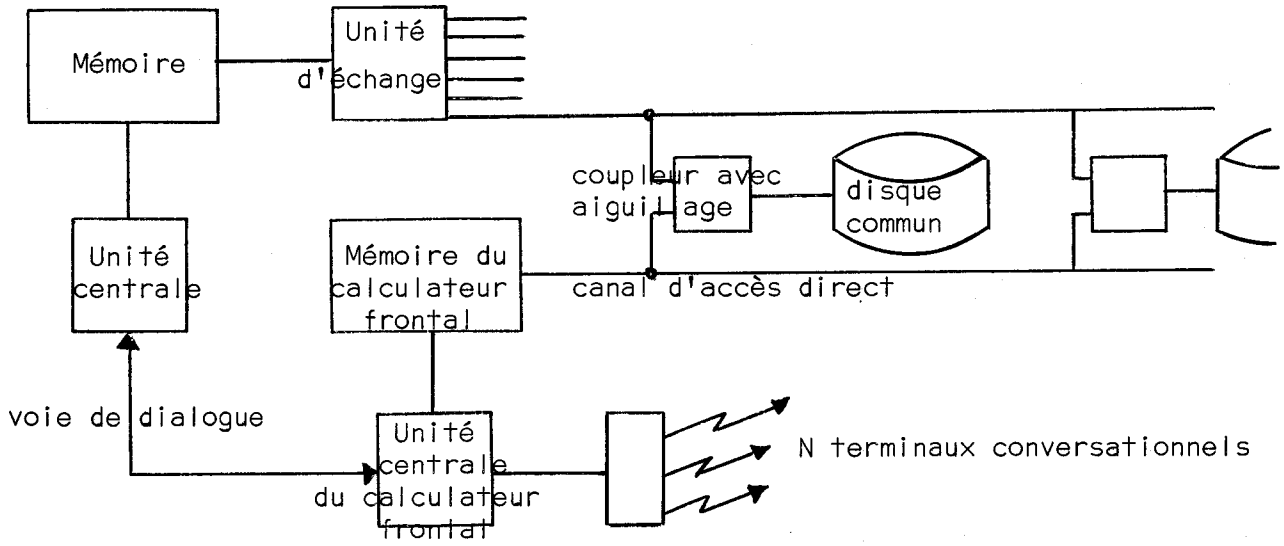


figure II_3 : Partage de l'ensemble des fichiers par les deux calculateurs

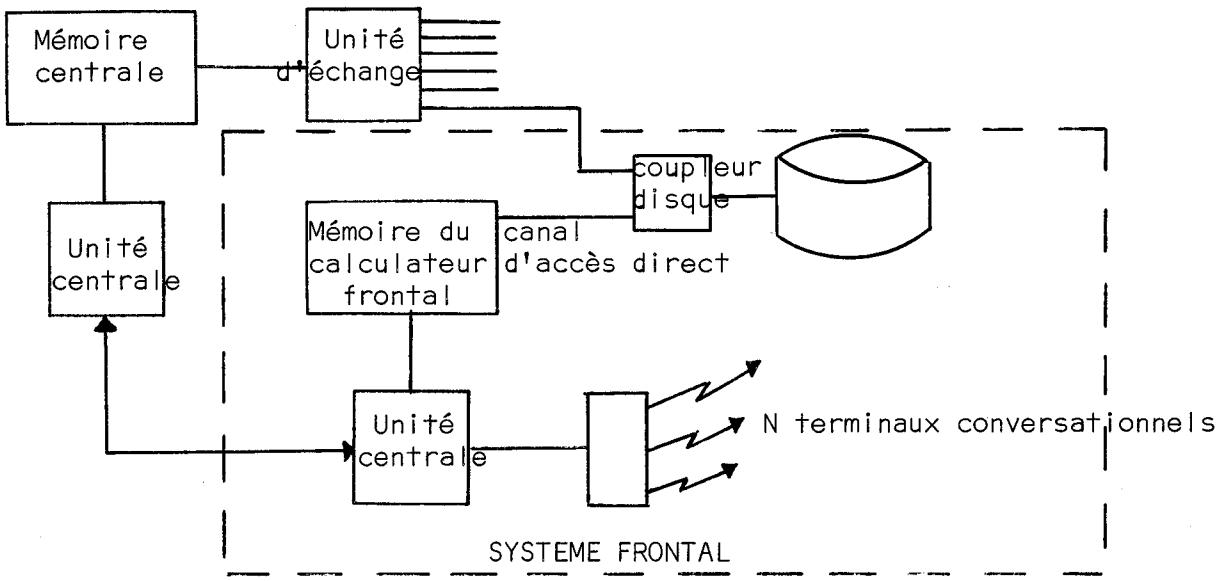


figure II_4 : Partage des seuls fichiers courants par les deux calculateurs

entre deux demandes d'accès à des secteurs i et i+k par le système central s'insère une demande d'accès à un secteur i-ℓ ou i+k+ℓ par le calculateur frontal, le temps d'accès pour le système central correspond au déroulement de N+k secteurs au lieu de k. (N représente le nombres de secteurs par piste).

Nous utiliserons l'architecture suivante qui réalise un compromis.

3.3.4 - Les deux calculateurs partagent un disque commun à têtes fixes et disposent d'une voie de synchronisation des échanges (figure II_4).

Le disque utilisé est adapté au rôle du calculateur frontal. Il est le support des fichiers sources courants. Remarquant que les programmes sources des utilisateurs sont de petite taille, on peut se contenter d'un disque de capacité moyenne. Compte tenu de sa taille, il est à têtes fixes et à faible temps d'accès. Comme il ne contient que les fichiers sources courants des usagers du time sharing, les accès par le système central sont relativement rares et il peut être partagé sans risque de rendre inopérante une optimisation des accès qui n'a pas de raisons d'être, compte tenu des temps d'accès. Pour la même raison, le transfert d'informations entre les calculateurs se fait par le disque. La seule liaison entre eux se fait par les unités centrales et est destinée à l'échange d'ordres courts utilisés à la synchronisation des transferts par le disque. Notons encore que de tels disques adaptés aux minisystèmes sont découpés en secteurs de petite taille. Ceci nous permettra de limiter les échanges réalisés par le calculateur frontal lors de la phase d'acquisition des lignes de programme émises par les usagers.

On peut faire la critique à cette architecture de nécessiter une recopie des fichiers lorsque les usagers demandent leur stockage ou inversement lorsque les usagers déclarent courants des fichiers précédemment stockés. En fait ces transferts sont peu nombreux en regard de ceux que nécessite une architecture classique lors de l'édition de texte, et une organisation particulière du système nous permettra de les limiter dans certains cas (chapitre IV).

i) avantages concernant la reconfiguration du système.

Une telle architecture est modulaire ; elle permet éventuellement l'utilisation de deux calculateurs frontaux si l'on désire améliorer les performances du système frontal lorsque le nombre de terminaux devient important.

De plus, le système central ayant accès au disque utilisé par le système frontal, on remarquera que les modifications du système d'exploitation des calculateurs frontaux sont simplifiées. En effet il n'aurait pas été possible de réaliser un générateur de système au niveau du calculateur frontal, celui-ci sera implanté au niveau du système central. Hors utilisation du système frontal, le système central aura ainsi la possibilité de recharger un nouveau système sur le disque du calculateur frontal.

ii) choix du calculateur frontal.

Les traitements qui seront réalisés par le calculateur frontal fixent déjà un certain nombre de contraintes de structure. La réalisation d'une analyse syntaxique même simple nous indique qu'il sera nécessaire de disposer d'instructions de recherche en tables, il en est de même pour le transcodage. D'autres contraintes apparaîtront au fur et à mesure de l'étude.

Les instructions, dont il serait utile de disposer, n'existent en général que sur de gros calculateurs, et encore sont-elles souvent le résultat de compromis. Nous étant imposé d'utiliser un calculateur de puissance réduite, son manque de performance intrinsèque doit être compensé par une plus grande spécialisation des instructions. Ceci nous impose d'utiliser un calculateur microprogrammable afin de pouvoir implanter ces instructions. Une organisation particulière peut permettre une interaction constante entre le microprogramme et le système dans le but de simplifier l'écriture du système d'exploitation. C'est ainsi que le microprogramme prendra en charge la réalisation des entrées-sorties multiplexées. La réalisation du système d'exploitation nécessite la mise en place de mécanismes de synchronisation qui existent rarement dans des calculateurs de puissance réduite. L'implantation de ces mécanismes par microprogramme permet de les adapter à l'utilisation du calculateur et augmente ainsi leur efficacité.

Nous avons choisi d'utiliser un calculateur de type Multi 8 ou Multi 20. La plupart des traitements réalisés portent sur des caractères ; la taille de leurs mots mémoire (1 octet) simplifiera l'écriture du microprogramme.

4 - DESCRIPTION SOMMAIRE DU SYSTEME FRONTAL

Le traitement réalisé par le système est découpé en phases indépendantes dont la réalisation utilise en interaction les tâches du système. Chaque phase est activée par l'émission d'une commande depuis le terminal de l'usager. Le calculateur frontal participe à l'exécution des phases création du fichier source courant et exécution mais réalise seul la phase édition de texte. Lorsqu'une phase est en cours d'exécution, l'usager dialogue avec celle-ci à l'aide de sous commandes s'il y a lieu. La synchronisation des tâches exécutées par le système frontal et le système central se fait par l'échange d'ordres entre les deux calculateurs.

4.1 - Le langage de commande du système frontal

Le système frontal ne reconnaît que deux types de commande :

- Appel de compilateur
- Appel de l'éditeur de texte.

4.1.1 - Appel du compilateur.

Ceci signifie pour le système frontal que chaque message reçu est une ligne de programme, un appel implicite à l'éditeur de texte ou une commande d'exécution. La phase de création du fichier source utilisera donc certaines tâches d'édition de texte. Seuls cinq types d'édition de texte pourront être réalisés au cours de cette phase :

- appel implicite d'insertion : si le numéro de clé de l'article reçu est inférieur à celui du dernier article chargé ;

- appel implicite d'effacement : si l'article reçu est vide et a un numéro de clé inférieur à celui du dernier article chargé ;

- sous commande explicite d'effacement : à la réception du message <ligne 1>, <ligne 2> EFFACER alors que l'éditeur de texte n'a pas été appelé. Le traitement consiste à effacer dans le fichier source courant tous les articles dont les numéros de clé sont compris entre ligne 1 et ligne 2 incluses;

- sous commande explicite d'impression : <ligne 1>, <ligne 2> IMPRIMER ;

- sous commande explicite d'insertion :

<ligne 3> INSERER <identificateur de fichier>, <ligne 1>, <ligne 2>.
Insertion derrière l'article du numéro de clé ligne 3 de tous les articles du fichier déclaré dont les clés sont comprises entre ligne 1 et ligne 2. Ces articles voient leur numéro de clé translaté pour qu'ils puissent s'insérer dans le fichier source courant.

S'il s'agissait d'un appel au compilateur "on line", après la réorganisation du fichier source courant, des informations sont transmises au système central de sorte que la compilation puisse reprendre. Ce n'est pas le cas s'il s'agissait d'un appel au compilateur "off line", de plus, cet appel est enregistré par le calculateur frontal mais n'est pas transmis au système central. Ceci n'est fait qu'à la réception de la sous commande d'exécution qui met fin à la phase de création du fichier source courant.

Le graphe de la figure II_5 schématise l'organisation de la phase création du fichier source courant.

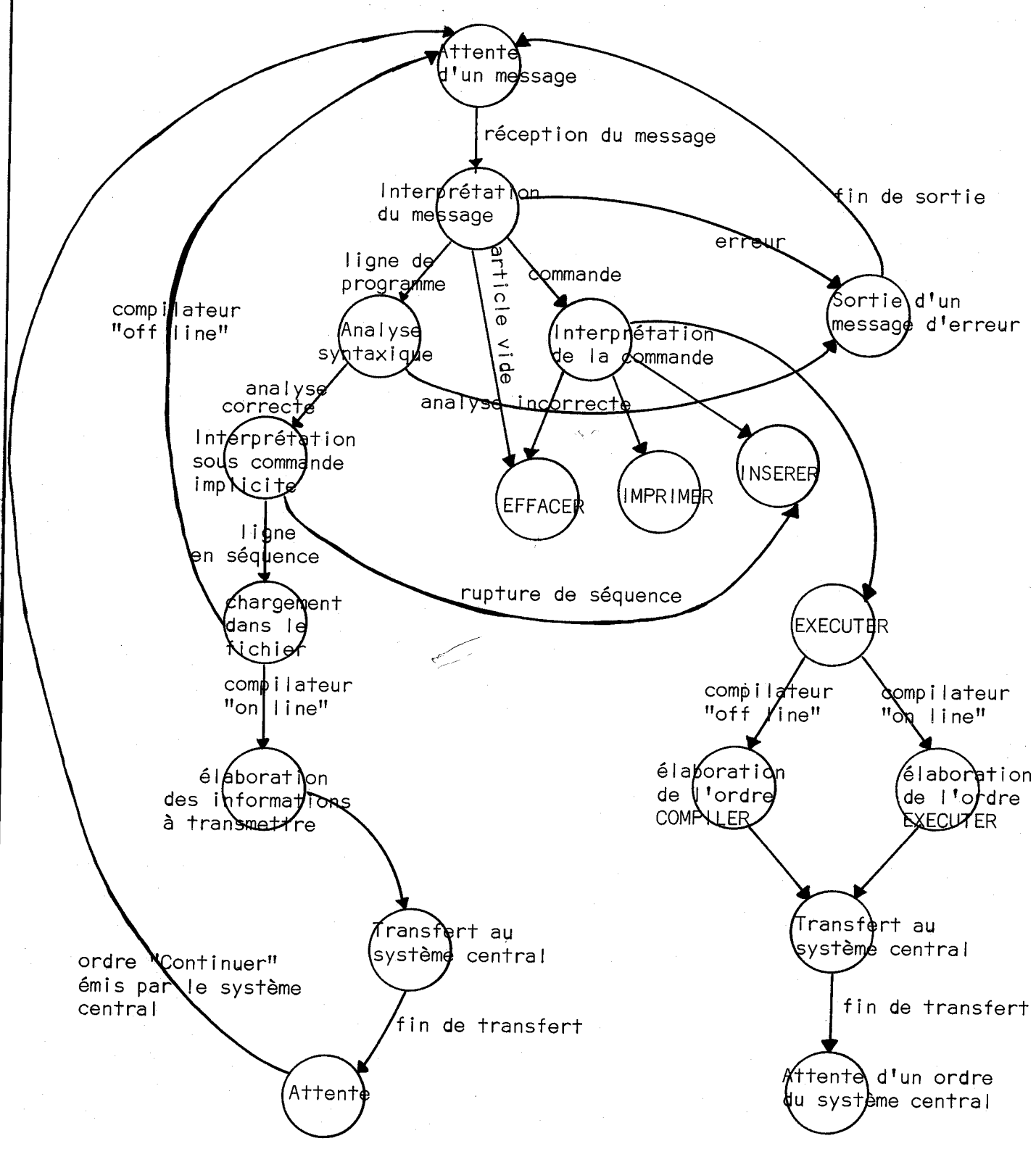
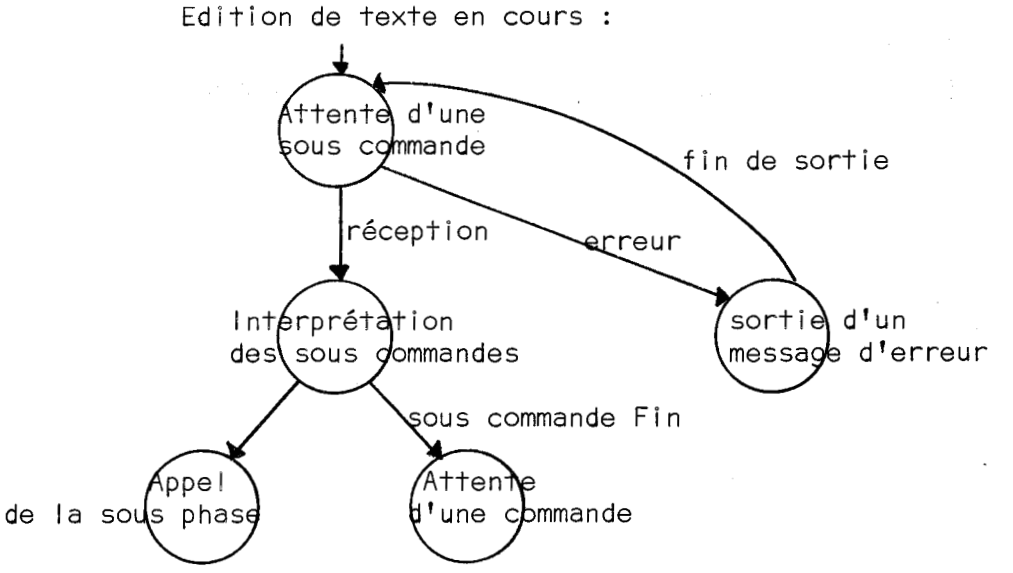


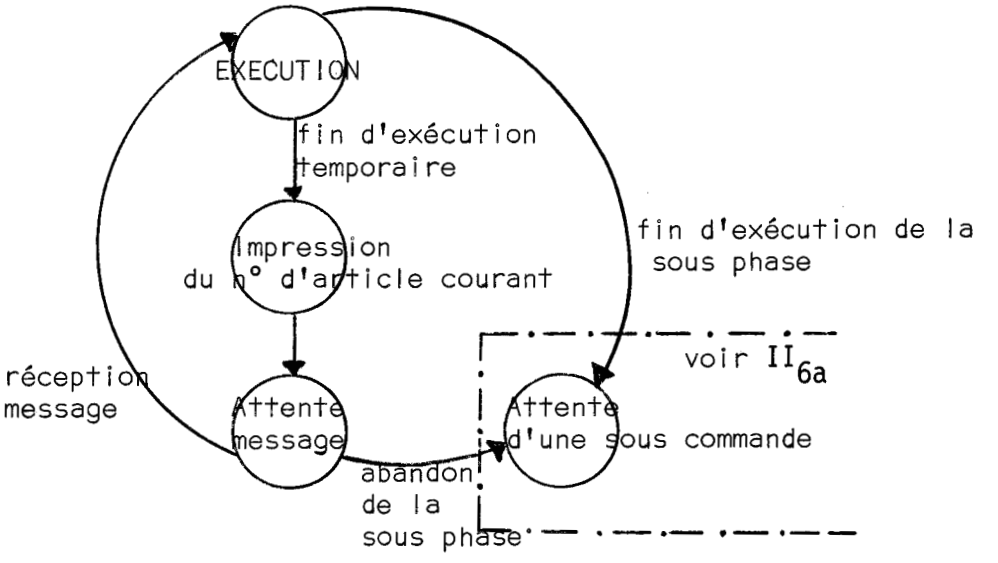
figure II_5 : Création du fichier source

4.1.2 - Appel de l'éditeur de texte.

L'éditeur de texte est conversationnel. A son appel, le système frontal passe en attente d'une sous commande qui correspondra à l'exécution d'une sous phase (exemple : sous phase IMPRIMER). Chaque sous phase peut être abandonnée par l'usager. A la fin de l'exécution d'une sous phase, le système frontal reste en attente d'une sous commande qui peut être la sous commande d'abandon de la phase édition de texte.



a) Synchronisation des sous phases



b) Graphe d'état d'une sous phase

figure II_6 : Organisation de principe de l'édition de texte.

4.2 - Synchronisation des phases et sous phases

Il est important de guider l'usager dans l'utilisation du langage de commande du système. A cet effet, le système frontal émet vers le terminal de l'usager un caractère que nous qualifions de caractère préfixe. L'émission de ce caractère peut être le résultat d'un ordre reçu depuis le système central, c'est le cas en ce qui concerne l'émission des commandes moniteur. D'autres caractères peuvent être émis, ils sont utilisés à la synchronisation des sous phases ; ceux-ci peuvent être le résultat d'un ordre du système central (utilisation du compilateur on line) ou provenir du système frontal (éditeur de texte). L'utilisation de ces caractères évite l'émission de messages explicites et par conséquent parfois longs.

L'usager garde à chaque instant la possibilité d'interrompre un traitement à l'aide de deux caractères spéciaux appelés "attention 1" et "attention 2". Leur rôle est différent suivant la phase en cours. S'il s'agit de l'édition de texte, la sous phase en cours est abandonnée et le système passe en attente d'une sous commande ; sinon la phase est abandonnée et le système attend une commande. Ainsi, lorsque l'usager est dans l'état "entrée de donnée", celle-ci est abandonnée.

ETUDE DU SYSTEME FRONTAL

Chapitre III

ADAPTATION DE L'ENVIRONNEMENT HARDWARE DU SYSTEME

Rappelons que cet environnement est formé :

- du système de concentration et de diffusion de caractères en provenance ou vers les terminaux des usagers conversationnels ;

- de la liaison permettant le partage d'un disque par le calculateur central et le calculateur frontal.

Avant de préciser ces adaptations, nous devons montrer comment se réalisent les entrées-sorties sur le Multi 20. En effet, celles-ci sont réalisées par microprogramme , c'est cette particularité qui permettra de simplifier la mise en place de ces liaisons.

I - PRINCIPE DE REALISATION DES ENTREES-SORTIES SUR MULTI 20 [4]

Le microprogramme de réalisation des instructions d'entrées-sorties utilise une microinstruction de génération de signaux. Cette microinstruction (microinstruction de type 7) permet de positionner les trois bits d'un registre de commande d'entrées-sorties. Le décodage du contenu de ce registre autorise ainsi l'émission de sept signaux différents utilisés à la synchronisation des différentes phases de l'instruction d'entrées-sorties (fig. III_1). Simultanément à l'émission des signaux, un transfert peut être réalisé entre deux registres.

Contenu du registre C.E.S.	Fonction du signal généré	Nom du signal
000	Remise à zéro du registre CES	-
001	Sortie commande	CØXX/
010	Sortie donnée	DOXX/
110	Entrée donnée	DIXX/

figure III_1 : Signaux d'entrée-sortie - Seuls sont indiqués les signaux utilisés.

Le principe de réalisation de la sortie d'un caractère serait alors le suivant :

① Générer CØXX - Simultanément afficher un octet sur le bus d'entrée-sortie. Commentaire : cet octet comporte deux champs dont l'un représente la fonction à réaliser (entrée, sortie, lecture de l'état etc) et l'autre, l'adresse du périphérique. Seul le coupleur qui a reçu cette adresse réalise le décodage du premier champ.

② Acquérir en mémoire le caractère à émettre. Le présenter sur le bus d'entrée-sortie.

③ Générer DØXX.

La microprogrammation permet ainsi avec une grande souplesse la réalisation d'entrées-sorties quelconques. On peut envisager par l'utilisation d'un signal d'interruption particulier d'exécuter des entrées-sorties multiplexées entre plusieurs périphériques, les caractères étant reçus ou émis au gré de ceux-ci. Cette souplesse de réalisation permet en outre une simplification notable des coupleurs de périphériques.

2 - REALISATION DES ENTREES-SORTIES MULTIPLEXEES

Il s'agit là des entrées-sorties sur les lignes de transmission. C'est le seul traitement que l'on puisse confier au microprogramme. Les traitements tels que le transcodage ou la détection des codes invalides que l'on aurait pu envisager de lui confier sont trop spécifiques du type de terminal utilisé. Leur réalisation par microprogramme figerait la structure du système frontal et ne permettrait pas l'adjonction éventuelle de terminaux dont les caractéristiques n'auraient pas été prévues lors de la définition du système.

Nous reprenons pour réaliser la gestion des entrées-sorties multiplexées un principe analogue à celui qui a été présenté au chapitre I.

2.1 - Principe de base

Il consiste à interroger périodiquement les multiplexeurs connectés au calculateur. Chaque multiplexeur comporte huit lignes d'entrée et huit lignes de sortie reliées au bus d'entrées-sorties du calculateur. Il est relié à huit terminaux

conversationnels qu'il connecte simultanément sur ordre du calculateur à chacune des lignes du bus d'entrées-sorties, permettant ainsi l'acquisition d'un bit d'information en provenance de chaque terminal relié.

Les transferts entre les terminaux et le calculateur sont asynchrones. Il est donc nécessaire pour être assuré de retrouver l'information d'interroger chaque multiplexeur à une fréquence supérieure à la fréquence d'émission des terminaux. En sortie, l'information est émise en synchronisme vers l'ensemble des lignes de transmission reliées au multiplexeur. L'échantillonnage en entrée et la synchronisation en sortie sont assurés par le microprogramme. Son exécution est déclenchée par la détection des interruptions que provoque une horloge externe.

Ces interruptions appartiennent à la classe des interruptions internes de même que les interruptions en provenance des canaux d'accès direct à la mémoire. Leur origine est recherchée par le microprogramme.

A la fin de la phase exécution de chaque instruction, le microprogramme correspondant renvoie à une séquence de recherche des interruptions. Celles-ci peuvent être détectées par examen du registre de conditions. Lorsqu'une interruption interne est présente, son origine est précisée dans le mot d'état interne. A chaque type d'interruption est associé un traitement particulier, d'une manière générale, celui-ci est programmé et le microprogramme assure un branchement vers la séquence d'instructions correspondante. Notons qu'alors une sauvegarde puis plus tard une restauration du contexte du programme interrompu doivent être exécutées ; sur le multi 20, chacune de ces opérations demande trente microsecondes d'unité centrale. C'est pourquoi la gestion des interruptions de l'horloge externe et la réalisation des entrées-sorties correspondantes sont assurées par microprogramme ; l'on peut en effet escompter un temps d'exécution de ce dernier de moins de cinquante microsecondes.

Le microprogramme développé dans ce but par le constructeur associe à chaque multiplexeur une horloge externe. L'élimination de certains codes non utilisés sur les terminaux "time sharing" permet de diminuer la fréquence d'échantillonnage, autorisant ainsi l'utilisation de la même horloge pour gérer plusieurs multiplexeurs. De plus aucun traitement des erreurs de transmission n'est réalisé par le microprogramme et la synchronisation entre celui-ci et le système d'exploitation n'est pas assurée. Il sera donc réécrit.

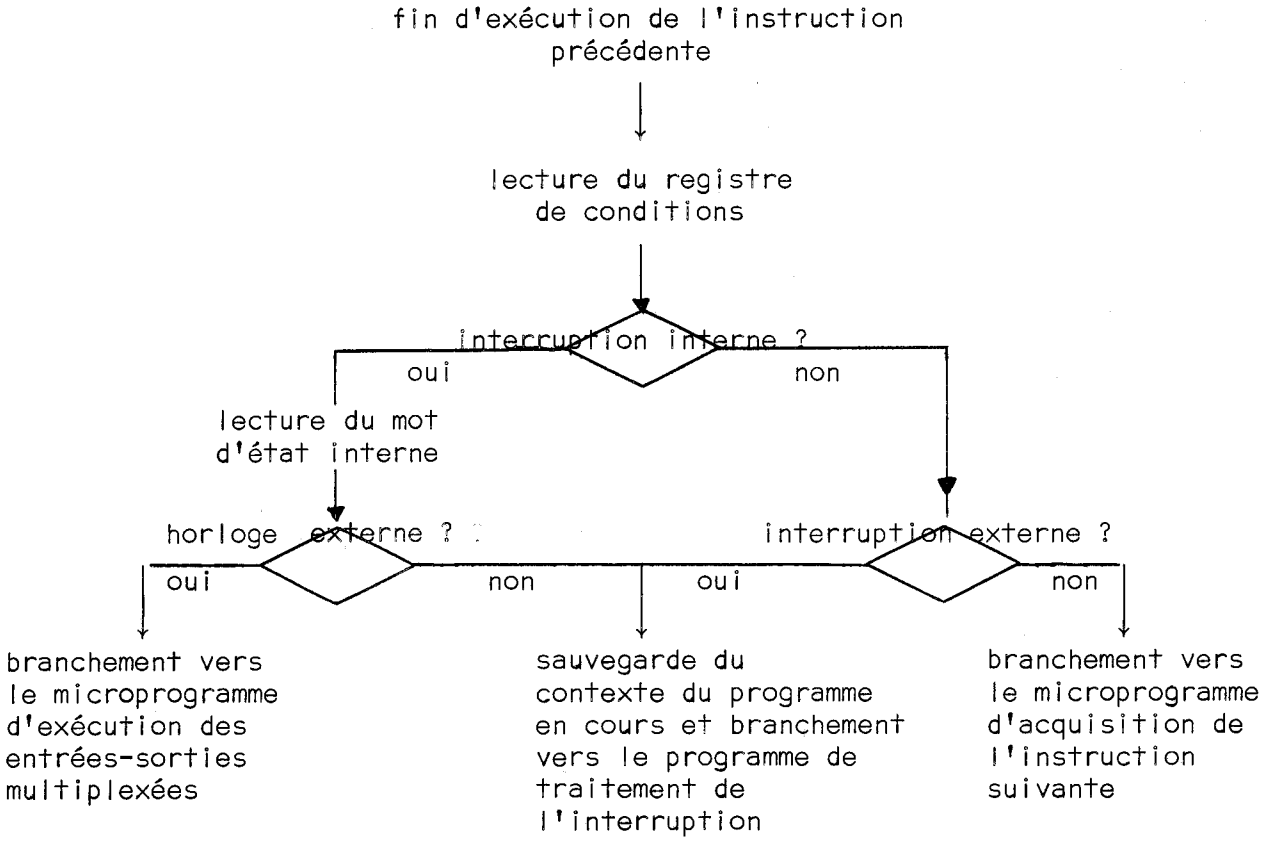


figure III_2 : Organisation générale du microprogramme

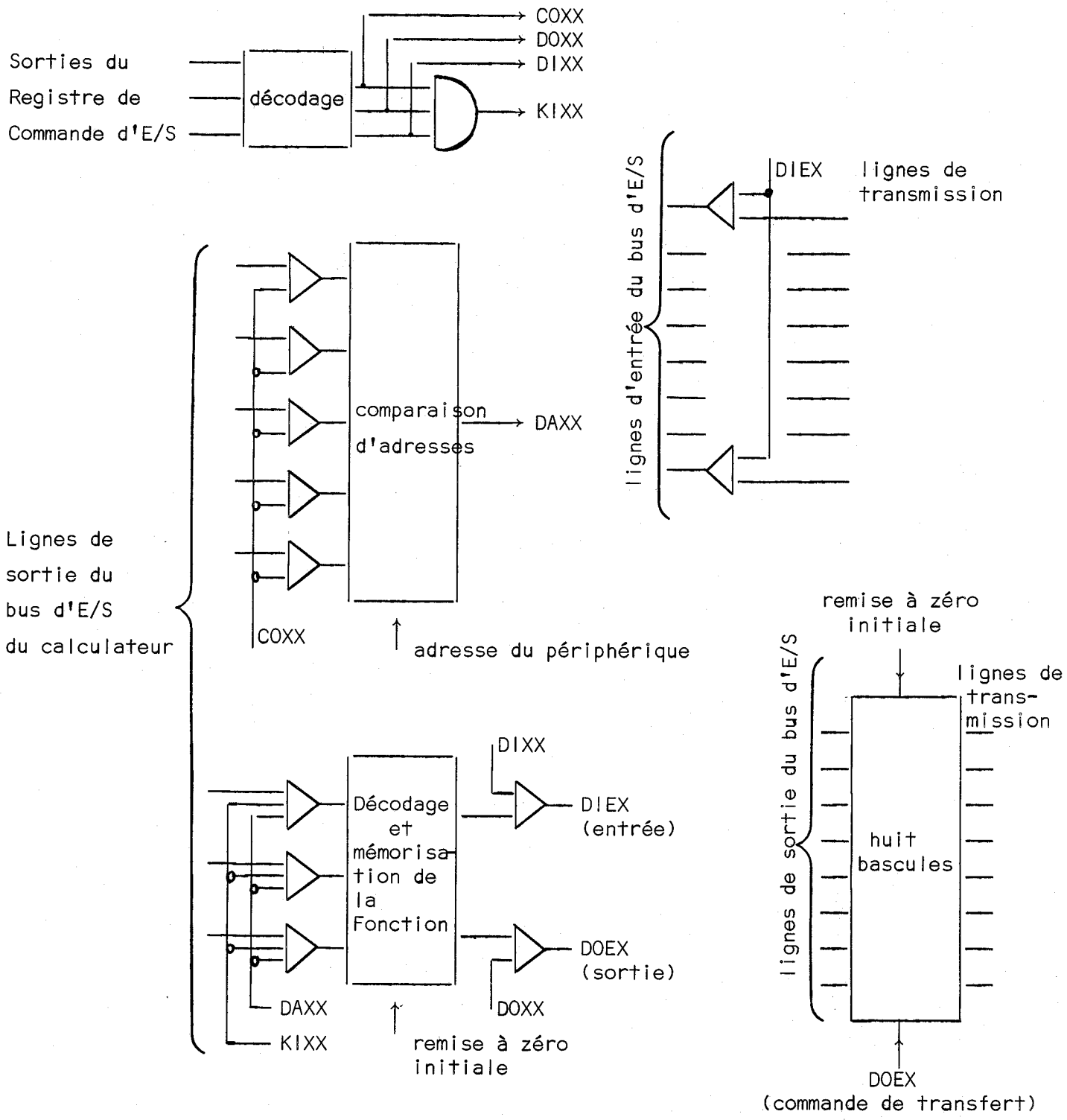
2.2 - Organisation d'un multiplexeur

Celui-ci est organisé très simplement. Le maximum du traitement est réalisé par le microprogramme de traitement des interruptions d'horloge. Le multiplexeur reprend le principe des entrées-sorties de caractères défini au paragraphe III_1. Vis-à-vis de la fonction de multiplexage ce n'est donc en réalité qu'une simple interface.

2.2.1 - Réalisation d'un multiplexeur [4,5]

Le multiplexeur n'exécute que deux fonctions : entrée d'information
sortie d'information.

Nous reprendrons le multiplexeur développé par le constructeur du multi 8. Sa simplicité se justifie par le fait que l'information est transmise sous forme de niveaux logiques. Ainsi qu'on peut le voir sur le schéma de la figure III_3, le multiplexeur n'assure aucun traitement de l'information qui transite dans ses circuits.



légende :

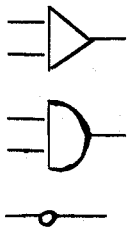


figure III_3 : Schéma synoptique d'un multiplexeur.

2.2.2 - Etat des lignes de transmission et du multiplexeur

La détection des erreurs est un problème important mais en général difficile à résoudre complétement. Cette difficulté est accrue dans le cadre d'une transmission en mode série.

Le problème qui nous importe est de détecter les codes erronés, c'est-à-dire ceux qui ne correspondent pas à l'émission réalisée par le terminal ou le calculateur. En ce qui concerne les caractères émis par le calculateur, aucune détection ne peut être faite par celui-ci. En effet, on peut constater d'après le schéma du multiplexeur que la liaison entre le calculateur et les lignes de transmission est presque directe. Compte tenu de la probabilité d'erreurs sur les lignes de transmission comparativement à celles qui peuvent apparaître sur cette liaison, il n'est d'aucun intérêt d'assurer une détection au niveau du multiplexeur. La seule détection d'erreur intéressante devrait être faite au niveau de l'information qui est reçue par le terminal, or son résultat ne peut être retransmis au calculateur. Ceci justifie l'affirmation faite précédemment, que cette détection ne peut être que laissée aux soins de l'usager.

Dans l'autre sens, les caractères étant reçus bit par bit, la seule détection possible se réalise par un codage particulier mettant en évidence les erreurs de transmission. Elle est donc fonction des terminaux utilisés. De plus, elle nécessite la connaissance du caractère complet et ne peut donc être assurée que par le microprogramme ou le programme. C'est ainsi que nous assurerons par microprogramme la détection des erreurs de parité lorsque la présence dans le code d'un bit particulier le permettra. Notons qu'une autre détection d'erreur peut être assurée, c'est celle que réalise l'usager à la réception sur son terminal des caractères émis en retour dans le cadre d'une liaison en mode full duplex.

Ces remarques concernent les erreurs dans la transmission des bits d'information composant le caractère émis ou reçu. L'erreur peut provenir également d'un décalage dans le temps du caractère. En effet si les bits d'un caractère sont transmis à fréquence fixe, l'instant de départ de la transmission est quant à lui totalement aléatoire. Afin de permettre la détection du premier bit du caractère , on utilise un code particulier qui consiste à encadrer le caractère à l'aide de bits de niveaux logiques contraires. En général un bit de départ et deux bits d'arrêt assurent cet encadrement. Le niveau logique des bits d'arrêt correspondant à une absence de transmission, on peut ainsi repérer le bit de départ et donc le début de la transmission.

Deux interprétations peuvent être données aux bits d'arrêt. En général on les considère comme représentant un délai minimal pendant lequel le caractère peut être traité et les circuits réinitialisés. Nous l'interpréterons quant à nous comme faisant partie du code. En effet, le bit de départ peut ne pas avoir été transmis, il se peut alors que l'on détecte un bit du caractère de même niveau logique que le bit de départ et qu'on l'interprète comme tel. Le caractère est alors décalé et donc erroné. Ceci sera détecté par l'absence des bits d'arrêt.

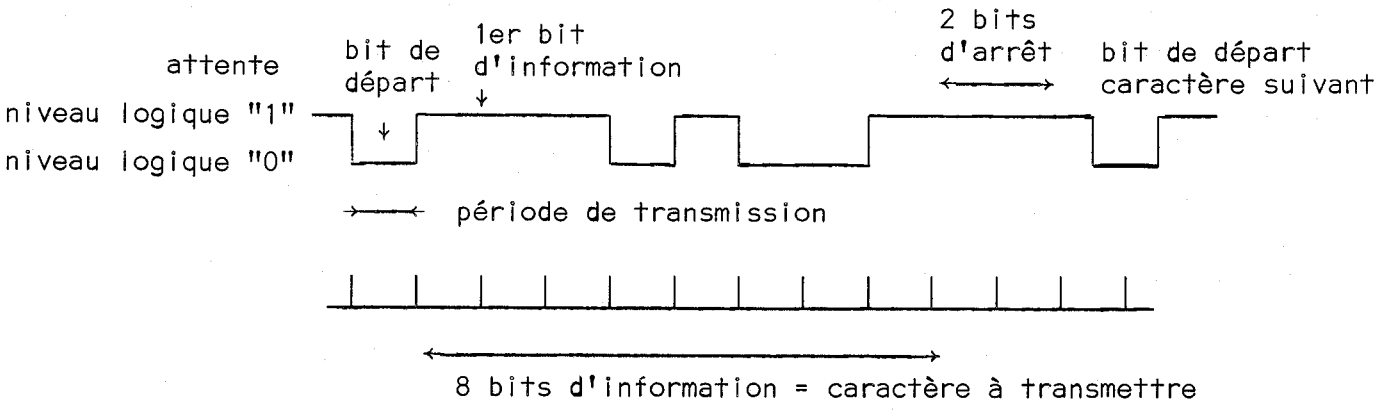


figure III_4 : mode de transmission de l'information.
Le caractère transmis ici est "11101001".

2.3 - Réalisation du multiplexage

2.3.1 - Principe

Le microprogramme gère N multiplexeurs auxquels sont connectées de une à huit lignes de transmission. Nous supposerons que chacun des terminaux travaille à la même fréquence de transmission. Ce sera en général le cas dans un système "time sharing" auquel ne sont pas connectés des terminaux conversationnels rapides (consoles de visualisation par exemple). Nous verrons par la suite comment le cas contraire peut être résolu.

Les multiplexeurs sont interrogés alternativement avec une période multiple de la période h de l'horloge temps réel. Cependant l'émission d'information par les différents terminaux reliés à un même multiplexeur ayant lieu en complet asynchronisme, il est nécessaire de réaliser un échantillonnage des lignes. La fréquence d'échantillonnage f est une fonction de la méthode d'échantillonnage utilisée et des caractéristiques des circuits d'interface. C'est donc une constante.

$$h = \frac{1}{Nf}$$

On conçoit que f doit être choisie telle que $f > \frac{1}{T}$ si T est la période de transmission. La réalisation de l'échantillonnage par microprogramme impose que f soit un multiple de la fréquence de transmission. Nous devrions donc choisir : $f \geq \frac{2}{T}$.

Le choix de la valeur de f doit être fait également en fonction de la méthode utilisée. Celle-ci consiste à déceler la présence du bit de départ, puis à calculer le numéro de l'échantillon qui est le plus proche du milieu de l'impulsion. Supposons que $f = \frac{5}{T}$ et que le bit de départ soit détecté lors du deuxième échantillon, la lecture des bits du caractère sera toujours exécutée sur le quatrième échantillon.

Par le principe même de la transmission (modulation puis démodulation) la largeur des signaux reçus peut fluctuer ; les fronts sont alors mal définis en regard de la largeur de l'impulsion d'échantillonnage. Afin de réaliser une lecture correcte, il faudrait donc la faire au centre de l'impulsion porteuse de l'information.

En choisissant $f = \frac{3}{T}$, nous réalisons le positionnement de l'échantillon de lecture par rapport au centre de l'impulsion avec une erreur inférieure à 17%. Cette fréquence nous assure un éloignement suffisant par rapport aux fronts des signaux pour que la lecture ne soit pas faussée par l'imprécision des niveaux logiques à cet endroit.

2.3.2 - Utilisation du microprogramme par le système d'exploitation

Le microprogramme se déroule indépendamment et le système d'exploitation n'a aucun moyen de contrôle sur lui. Il est donc nécessaire de définir un mécanisme qui permette au système d'exploitation de déposer des messages (caractères à émettre) pour le microprogramme et vice versa.

En ce qui concerne les entrées, on utilise deux piles du type "dernier entré, premier servi" (LIFO : Last in, First out). Lorsque le microprogramme a constaté l'acquisition d'un caractère complet, il range celui-ci dans la première pile accompagnée des numéros de ligne et de multiplexeur. En fin de cycle, si cette opération a eu lieu au moins une fois, le microprogramme exécute un déroutement vers une procédure de traitement des caractères reçus. Si un caractère est erroné (rappelons que les détections d'erreurs portent sur la parité et l'absence de bit d'arrêt), celui-ci est rangé dans la deuxième pile. En fin de cycle, le déroutement a lieu en priorité vers la procédure de traitement d'erreurs s'il y a lieu.

Pour réaliser les sorties de caractères, le microprogramme utilise en mémoire, pour chaque ligne connectée au multiplexeur, une zone de quatre octets :

octet 1 : nombre de bits à émettre
octet 2 : tampon de mémorisation du caractère suivant
octet 3 : indicateur de présence du caractère suivant
octet 4 : tampon contenant le caractère en cours de sérialisation.

L'octet 2 et l'octet 3 sont seuls utilisés par le système d'exploitation. Chaque fois que le microprogramme acquiert le caractère suivant pour le ranger dans l'octet 4, il stocke dans une pile LIFO le numéro du terminal concerné. A la fin du cycle correspondant, s'il y a lieu, un déroutement est exécuté vers une procédure chargée d'alimenter les tampons de sortie (octet 2). Cette procédure réalise le chargement directement à partir du tampon contenant le message ligne destiné à l'usager.

2.4 - Conclusion

2.4.1 - Nous n'avons traité que le cas de terminaux de même vitesse. Si des terminaux de différentes vitesses de transmission devaient être connectés, on résoudrait de manière simple ce problème en utilisant autant d'horloges externes que de vitesses utilisées. De même, il est possible éventuellement d'associer une horloge externe à chaque multiplexeur. On évite ainsi un réglage de l'horloge à chaque adjonction d'un multiplexeur.

Lorsque les terminaux sont très rapides, le microprogramme réalisant les entrées sorties multiplexées est souvent utilisé et la disponibilité du calculateur frontal pour d'autres tâches chute fortement. On peut pallier cet inconvénient en utilisant un multiplexeur cablé plus complexe qui réalise le travail laissé ici au microprogramme. En fait nous verrons que le calculateur frontal peut supporter de nombreuses lignes de transmission à cent bauds, aussi cette solution ne sera à envisager que si les terminaux rapides (600 bauds ou davantage) sont eux-mêmes nombreux.

2.4.2 - Ralentissement de l'unité centrale par les entrées sorties multiplexées

Nous conserverons dans le calcul qui suit l'hypothèse selon laquelle tous les terminaux travaillent à la même vitesse de transmission.

Soient
- t la période d'échantillonnage
- N le nombre de multiplexeurs
- h la période de l'horloge externe.

D'après un calcul précédent $h = \frac{t}{N}$.

Nous supposerons que chaque phase du microprogramme nécessite le même temps d'unité centrale. Soit u ce temps. Pendant chaque période h, un programme en cours d'exécution ne travaille donc en réalité que pendant un temps h-u.

Si l'on suppose que ce programme se déroule normalement en K périodes h, la fin de son exécution est retardée de $K.\frac{u}{h}$ périodes. Mais ce débordement est lui-même ralenti suivant le même processus.

Il vient donc $K' = K + K.\frac{u}{h} + (\text{entier de } K.\frac{u}{h}) \cdot \frac{u}{h} + \ldots$

On peut borner K' par $K'' = K \times \frac{1}{1-\frac{u}{h}}$

$\frac{u}{h} = \frac{u.N}{t}$.

Or nous avons calculé que $t = T/3$ où T est la période de transmission.

Alors $\frac{u}{h} = \frac{3.N}{T}.u.$

Tout traitement est donc ralenti dans une proportion :

$$\frac{K''-K}{K} = \frac{1}{1 - \frac{3.N}{T}.u} - 1.$$

La figure suivante, pour laquelle nous avons supposé que u=50 microsecondes, montre que ce ralentissement peut être sensible.

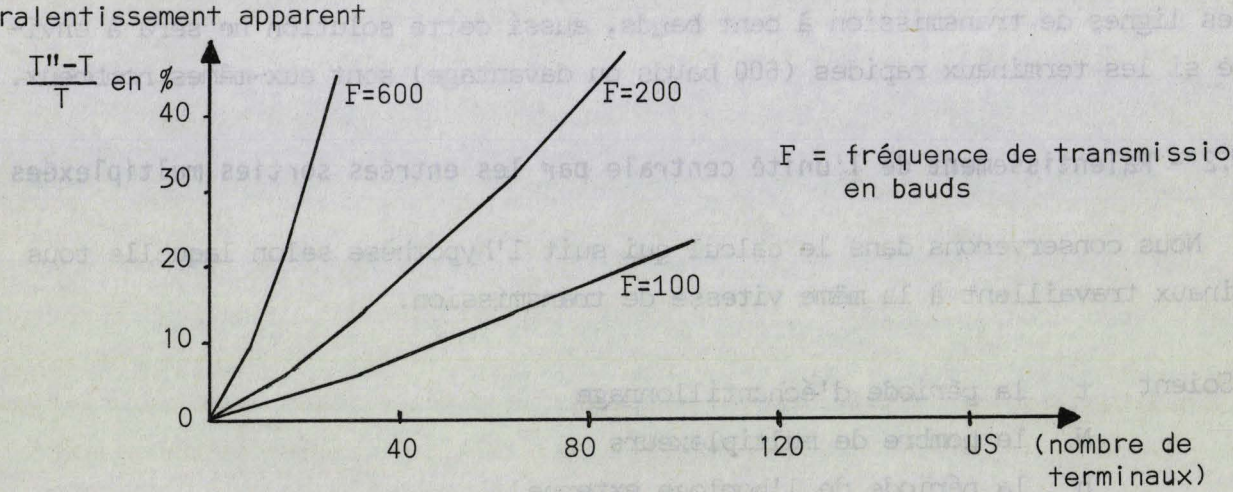


figure III_5 : Ralentissement apparent de l'unité centrale en fonction de la charge.

3 - REALISATION DE LA CONNEXION DISQUE - CALCULATEUR FRONTAL/CALCULATEUR CENTRAL [6,7]

Notre but n'est pas de décrire dans le détail cette connexion, mais d'indiquer les principes qu'elle doit respecter et comment.

Le disque partagé est relié au calculateur frontal par le biais d'un canal à accès direct mémoire (A.D.M.). C'est donc un dispositif simple mais qui se justifie par le fait que ce disque est le seul périphérique susceptible d'un accès direct à ce calculateur. Vers le calculateur central, il est connecté à une unité d'échanges, multiplexée s'il en existe une dans la configuration choisie. Ce choix se justifie par la faiblesse du débit du disque.

3.1 - Choix du disque

Pour les usagers, la mémoire secondaire du système frontal représente le support de leur fichier courant auquel ils accèdent article par article. La taille de ces articles est limitée par les contraintes physiques qu'imposent les terminaux. On doit donc prévoir une mémoire qui permette des accès en écriture et en lecture à des blocs d'information courts.

Mais, le même disque représente pour le système central le support de programmes sources. Il faut éviter, lors de la compilation, lorsque le système est de classe I, de gérer des accès nombreux au disque du calculateur frontal. Les blocs d'information vus du système central doivent donc être de taille importante. Ne réaliser que des accès à des blocs,dont la taille est du même ordre de grandeur que celle d'une page en mémoire centrale, serait une solution correcte.

Ces deux contraintes peuvent paraître contradictoires. Elles conduisent en fait à adopter un disque dont les secteurs soient de petite taille et dont le coupleur permette des lectures ou écritures en mode continu. C'est en général le cas.

La même mémoire secondaire sert également de support à la partie non résidente du système d'exploitation du calculateur frontal et de support des échanges d'informations entre les deux calculateurs. Elle doit donc autoriser des accès rapides. Ceci impose le choix d'un disque à têtes fixes. Son débit est limité par l'architecture du calculateur frontal et le coût. Mais les accès par le système central étant rares et concernant essentiellement des fichiers, le

temps d'acquisition de l'information par le système central sur ce disque a assez peu d'importance. Le débit du disque dans ce cas est d'ailleurs compensé par un temps d'accès réduit.

Ces remarques nous ont conduit à envisager l'utilisation du disque standard que propose le constructeur du calculateur frontal. Ses caractéristiques principales sont indiquées ci-après.

Capacité : 800 kilo octets
Taille secteur : 12 octets
Nombre de pistes : 256
Nombre de secteurs par piste : 256
Débit : environ 150 000 octets par seconde.

3.2 - Organisation du coupleur

D'une manière générale tout coupleur a une fonction de couplage logique et de couplage technologique, ces deux fonctions étant plus ou moins imbriquées suivant les réalisations.

Au niveau du calculateur frontal, l'utilisation d'un canal A.D.M. simple implique que l'initialisation du coupleur et du canal soient assurées par l'unité centrale. Les entrées-sorties exécutées dans ce but par l'unité centrale doivent être aussi proches que possible des entrées-sorties classiques telles que nous les avons décrites précédemment. Aussi le coupleur logique est-il assez simple et très différencié du coupleur technologique.

On utilisera cette nette différenciation entre couplage logique et couplage technologique pour réaliser la connexion du calculateur central. Celle-ci sera faite en bretelle sur le coupleur technologique de la liaison disque - calculateur frontal, un signal continu positionné lors de la sélection du périphérique validant l'une ou l'autre des connexions. Seul est donc à concevoir le coupleur logique de la liaison calculateur central - disque. Ainsi qu'il sera vu par la suite ceci est relativement simple à réaliser.

- Contrôle des périphériques

En ce qui concerne les périphériques travaillant en mode transfert par bloc et donc habituellement connectés à une unité d'échanges, on doit distinguer deux types d'instructions d'entrées-sorties :

. les instructions exécutives : ce sont celles qui nécessiteront pour leur exécution l'utilisation du périphérique , ce sont donc en particulier les instructions de transfert ;

. les instructions de contrôle : celles-ci ne concernent que le coupleur logique. Ce sont les instructions d'acquisition du mot d'état du périphérique, d'arrêt du périphérique, de traitement de l'interruption.

Les instructions exécutives se réalisent en trois phases :

i) Phase de sélection : au cours de laquelle l'unité centrale adresse le périphérique et analyse son état.

ii) Phase de transfert : au cours de laquelle un ordre est émis vers le périphérique et par la suite un transfert a lieu.

iii) Phase de test : au cours de laquelle le périphérique communique son mot d'état.

L'utilisation d'une unité d'échange permet d'accentuer la distinction entre chacune de ces phases. Seule la phase de sélection est alors exécutée sous contrôle de l'unité centrale. Il n'existe donc plus en fait qu'une seule instruction exécutive, l'instruction "lancement d'entrée-sortie" au cours de laquelle l'unité centrale initialise le coupleur et l'unité d'échange. Les phases de transfert et de test sont prises en charge par l'unité d'échange qui fonctionne alors comme un processeur indépendant utilisant la mémoire centrale du calculateur et allant y chercher ses ordres. Ces deux phases sont chacune initialisées par l'émission d'un ordre en provenance de l'unité d'échange, ce qui permet éventuellement de réaliser un chaînage de transferts. Le coupleur logique peut alors être simplifié dans son organisation.

Ceci est particulièrement le cas pour le CII 10070 [8] et permettra de réaliser le partage du disque du calculateur frontal.

3.3 - Protection des accès

Il est nécessaire d'éviter que les deux calculateurs tentent d'accèder simultanément au disque qu'ils partagent.

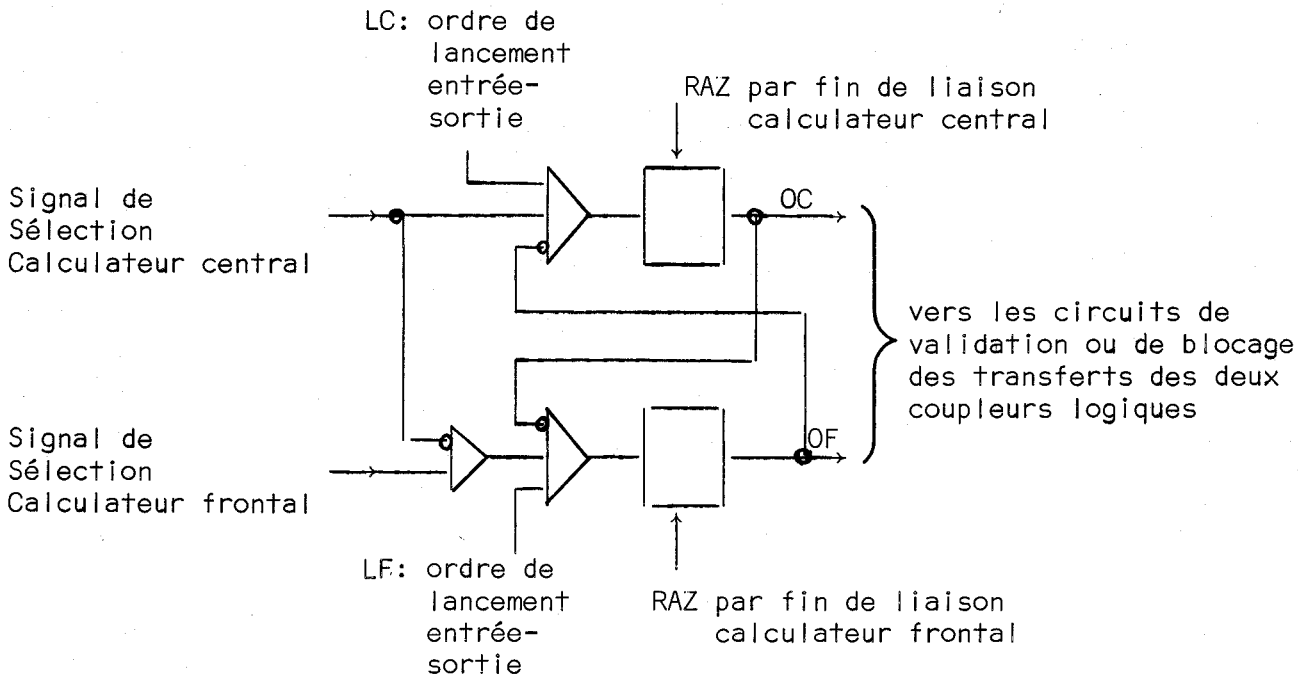
Le mécanisme de protection le plus simple consiste à tester et éventuellement positionner une bascule d'état au cours d'une opération qui doit elle-même être protégée. Cette opération élémentaire et protégée sera la phase de sélection. Le mécanisme est alors semblable à celui de l'instruction "Test and Set" qu'utilisent certains systèmes.

Chacun des deux coupleurs logiques élabore à la réception des signaux d'adresse un signal de sélection. Celui-ci est utilisé pour forcer à "un" une bascule si celle-ci était dans l'état "zéro"; si ce n'était pas le cas, la bascule reste dans l'état "un" et le signal de sélection reçu n'est pas accepté, on transmet alors au calculateur ayant tenté la sélection un signal d'occupation (transfert du mot d'état). La phase de sélection ne concernant que le coupleur logique, pendant celle-ci l'utilisation de la ressource par l'autre calculateur n'est pas perturbée.

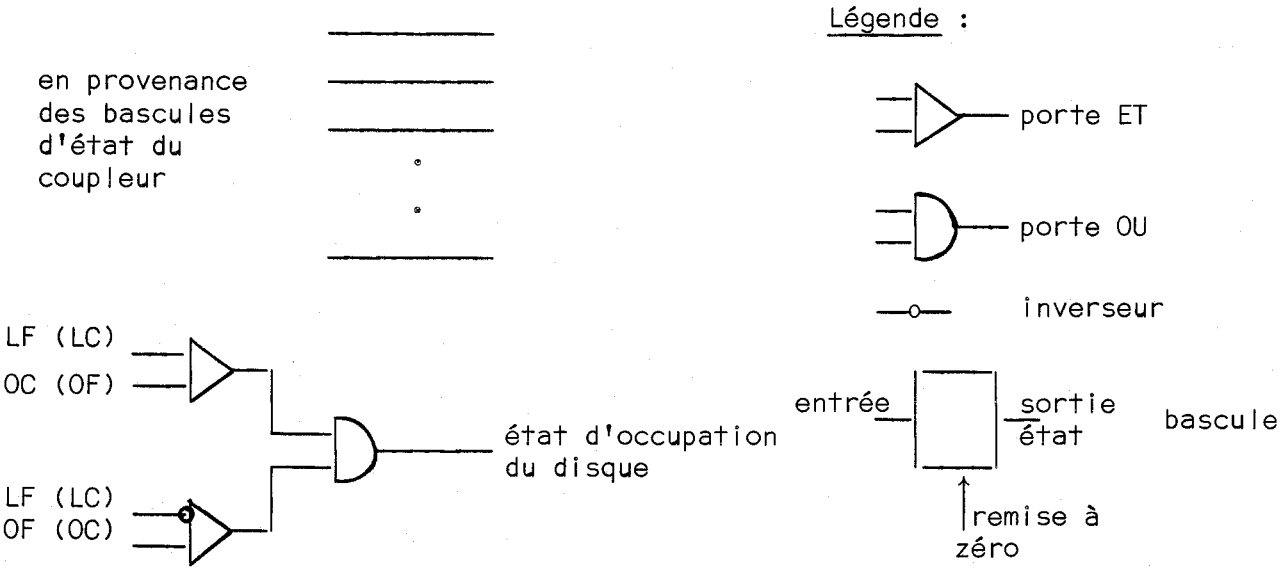
L'opération qui consiste à tester et positionner la bascule d'état doit alors être elle-même protégée. Ceci sera réalisé en imposant une priorité dans l'exécution de cette opération. La priorité sera donnée au calculateur central afin d'éviter toute perturbation dans son fonctionnement. La figure III_6 représente ce mécanisme.

Au mécanisme de verrouillage de la ressource doit être associé un mécanisme de déverrouillage. Cette opération sera réalisée à l'aide du signal de fin de liaison émis au coupleur par l'unité d'échange du système central ou le canal ADM du système frontal. Ce signal existe dans tout système et est utilisé en général pour effectuer la réinitialisation du coupleur.

Remarque : Le verrouillage de la ressource ne doit pas être réalisé s'il s'agit d'une instruction de contrôle. Le mécanisme utilisé ne doit donc être validé que si l'ordre reçu correspond à une instruction de lancement d'entrées-sorties (figure III_7).



a) Principe du mécanisme de protection des accès au disque



b) Principe de transmission du mot d'état du coupleur

figure III_6 : Principes de réalisation du double accès au disque.

3.4 - Remarques

. Implantation au niveau du calculateur frontal

Dans la version standard du microprogramme, le déroulement des entrées-sorties n'est pas géré comme nous l'avons indiqué en 3.1. Ceci est dû au fait que l'initialisation des transferts disque utilise les instructions d'entrées-sorties en mode caractère définies précédemment (§1) [9,10]. En particulier, il n'existe pas d'instruction de lancement d'entrées-sorties. Un microprogramme reprenant les principes décrits en 3.1 devra donc être écrit et le coupleur logique devra être légèrement modifié de sorte à respecter le diagramme de la figure III_7.

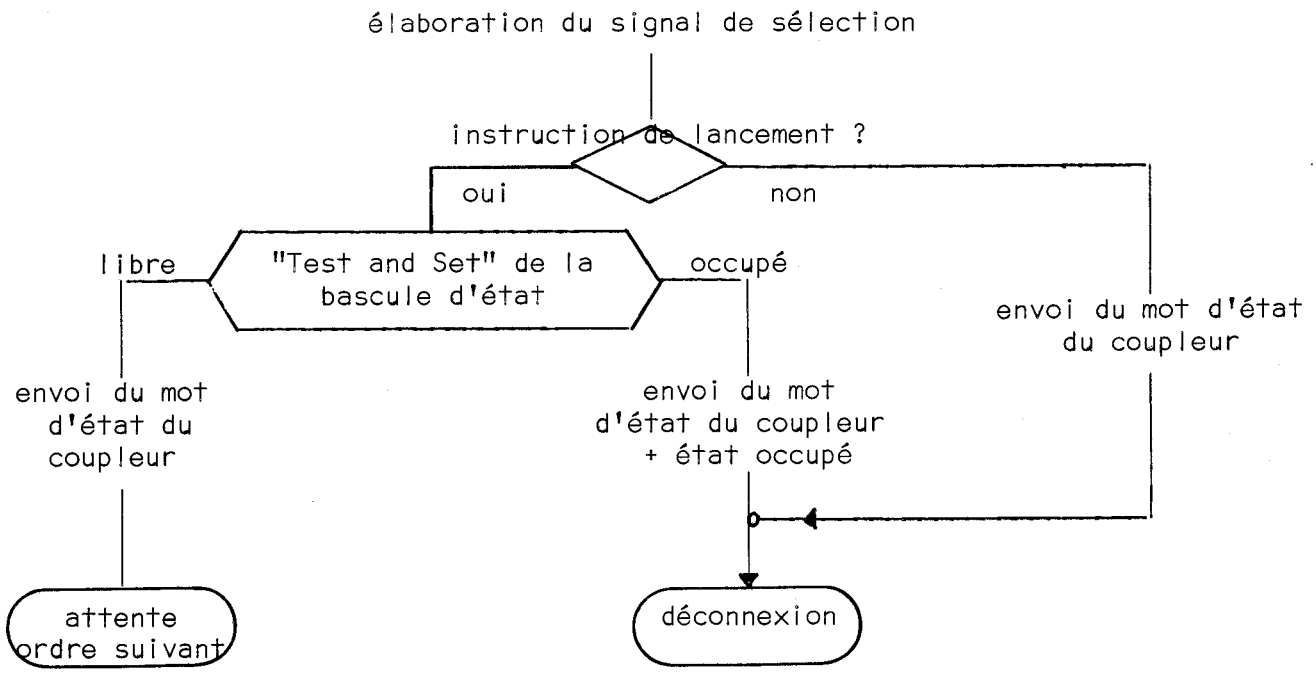


figure III_7 : Diagramme de la phase de sélection.

. Un autre aspect de la protection doit être envisagé, il concerne le contenu du disque. Ceci sera résolu en plaçant les accès réalisés par le calculateur central sous contrôle du système d'exploitation du calculateur frontal. En effet, tout accès au disque sera la conséquence d'une décision du calculateur frontal et se fera à une adresse définie par lui.

4 - REALISATION DE LA LIAISON INTERCALCULATEURS

4.1 - Rôle de liaison

Ainsi que nous l'avons indiqué rapidement lors de la présentation générale du système frontal (chapitre II 3.3.4), cette liaison a pour but de permettre la synchronisation de l'exécution par les deux calculateurs de tâches interactives. A cet effet, des messages courts sont échangés par les deux calculateurs sur cette voie directe. Eventuellement, ces messages sont l'indication pour le destinataire d'avoir à lire sur le disque une information plus précise.

- Définitions :

. Nous appellerons "ordres" les messages échangés sur la liaison inter-calculateurs.

. Nous appellerons "messages" les informations échangées par le biais du disque et initialisées par un ordre.

4.2 - Echanges de messages

Ces messages peuvent être :

- des messages en provenance du terminal de l'usager (commandes moniteur, messages d'entrées-sorties) ou destinés à ce dernier

- des commandes demandant l'exécution par le calculateur central d'une tâche dont le résultat est destiné au calculateur frontal, ou inversement

- des informations résultant d'une tâche exécutée antérieurement.

L'échange des messages se fait par un espace du disque appelé espace de dialogue scindé en deux fichiers occupant chacun trois pistes. L'un est réservé aux échanges de messages entre le calculateur central et le système frontal ou les usagers, l'autre aux échanges ayant lieu en sens inverse. Procéder de cette manière permet d'éviter la destruction d'informations lors des tentatives d'accès simultanés et simplifie la synchronisation des échanges car chaque dépot de message est ainsi sous le contrôle d'un rédacteur. Pour les mêmes raisons, dans chaque

fichier, il est attribué un article à chacun des rédacteurs potentiels, usagers ou système. Chaque fichier comporte donc un nombre d'articles égal au nombre d'usagers potentiels plus un. La taille maximale des articles a été fixée à 144 octets représentant 12 secteurs du disque. En fait les articles seront de longueur variable.

4.3 - Format des ordres

- Format général

Les ordres doivent fournir le type d'opération, le nom de l'usager concerné et, lorsque l'exécution doit être précédée d'une lecture de message, la longueur du message afin de minimiser les transferts. Les fichiers de dialogue doivent être implantés en permanence de sorte que les accès soient simplifiés. Chaque usager sera repéré par le numéro de son terminal qui jouera donc le rôle de clé d'article. L'article de clé zéro est réservé au système destinataire du message. La longueur du message sera donnée en nombre entier de secteurs (12 octets), une longueur nulle indiquant l'absence de message associé.

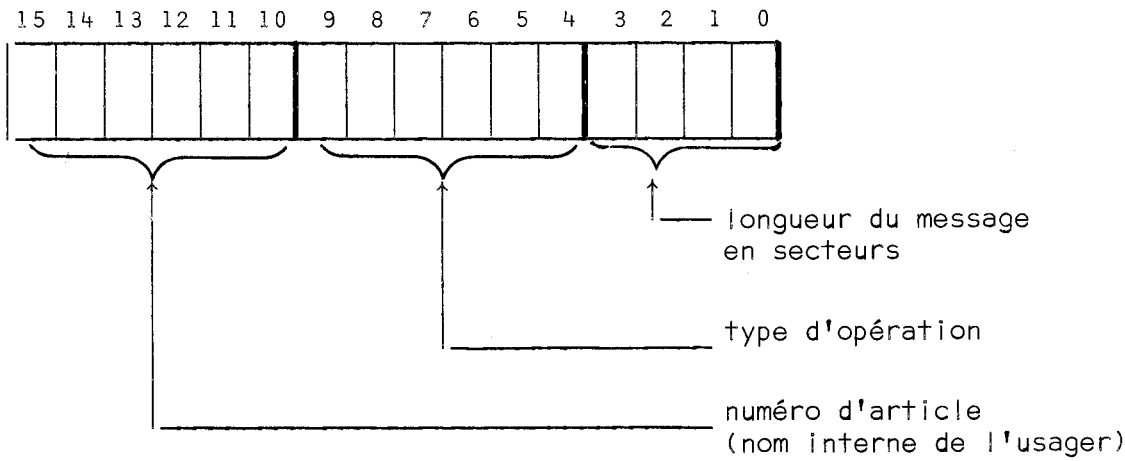


figure III_8 : Format des ordres.

- Type d'opération

La description complète de la tâche dont l'exécution est demandée nécessiterait un format trop important. De plus, nous devons faire en sorte que le système d'exploitation du calculateur central soit peu modifié, aussi, l'analyseur de commandes doit-il dans la mesure du possible pouvoir acquérir les commandes comme si elles provenaient des terminaux. Il ira donc chercher celles-ci dans le fichier de dialogue, tout comme dans le système précédent il y accédait sur le fichier périphérique. Le type d'opération est alors très simplifié et permettra la mise en place d'une interface.

entre le système frontal et l'analyseur de commandes.

Nous distinguerons :

1- ordre de création d'un usager : cet ordre ne peut être émis que par le calculateur frontal. L'utilisation d'un fichier de dialogue où l'on accède par le numéro interne de l'usager nécessite de distinguer cet ordre car il contient le nom interne d'un usager non connu du système central. Cet ordre demande la lecture par le système central d'un message formé des codes client de l'utilisateur.

2- ordre de refus d'exécution : il ne peut être émis que par le calculateur central. Il est une indication au calculateur frontal d'avoir à réinitialiser ses tâches. En général, ce refus sera accompagné de l'émission d'un message sur le terminal de l'usager. Ce message est lu par le calculateur frontal dans le fichier de dialogue.

3- ordre de demande d'exécution : nous devons distinguer deux cas selon le sens des transmissions :

. cet ordre provient du calculateur frontal : ce peut être un transfert de message ; le calculateur central ayant connaissance de son état, cet ordre ne nécessite aucune précision. Il peut également s'agir d'une demande d'accès du calculateur frontal au fichier de l'usager, l'ordre est alors précisé dans un message.

. cet ordre provient du calculateur central : ce peut être une demande d'aide au calculateur frontal, nous constaterons que c'est alors une demande d'espace sur le disque partagé (création de tampons de sortie). Dans tous les autres cas, ce sera une commande de synchronisation des états du calculateur frontal ; cet ordre étant court, il doit être transmis par la liaison intercalculateurs. On doit donc envisager les ordres suivants

sortir résultats
entrer données
attribuer espace
fermer fichier
sauver puis fermer fichier
fermer session
lancer phase : la phase doit alors être précisée (analyse, édition...), ce sera réalisé à l'aide des bits 0 à 3.

Toutes ces opérations sont codées à l'aide des bits 4 à 7 de l'ordre.

4- fin d'exécution : cet ordre est une indication de fin de phase ou une réponse à la suite d'une demande d'exécution particulière (par exemple lorsque le calculateur a fourni au calculateur central de l'espace sur le disque partagé), l'ordre est alors généralement accompagné d'un message.

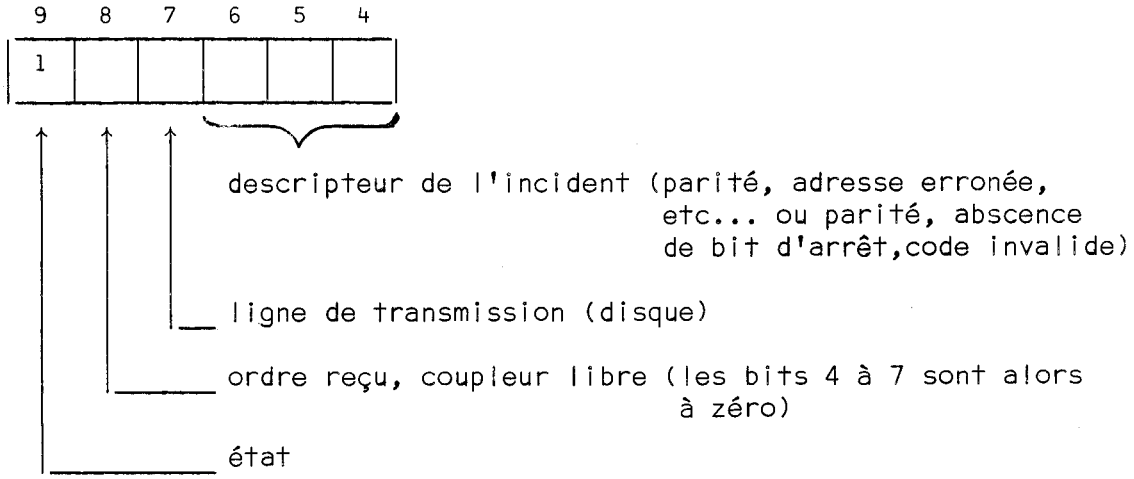
5- ordre d'arrêt de l'exécution : nous avons déjà signalé que l'usager devait conserver la possibilité d'arrêter l'exécution du traitement en cours. Lorsque ce traitement est réalisé par le calculateur central, le système frontal utilise cet ordre pour lui communiquer la décision de l'usager. Il n'y a pas de communication de message.

6- ordre de communication d'état : les incidents doivent être communiqués au système central pour transmission à l'opérateur. Cet ordre décrit le type d'incident, ce peut être :

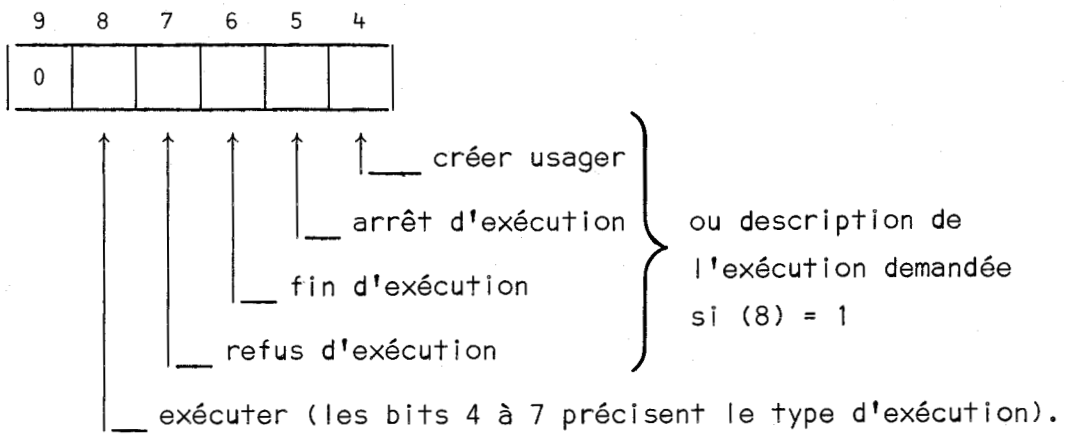
. un incident détecté lors d'un transfert sur le disque
. un incident sur la ligne de transmission de l'usager.

Une information plus précise peut être rangée dans le premier secteur de l'article associée à l'usager dans le fichier de dialogue. Cette information pourra être par exemple, l'adresse disque à laquelle cet incident s'est produit.

De même, les deux calculateurs travaillant en asynchronisme, lorsque l'un d'eux reçoit un ordre, il importe qu'il en indique la bonne réception à son interlocuteur afin que ce dernier ait connaissance de la libération du coupleur.



a) ordre de communication d'état



b) format général

figure III_9 : Format du type d'opération.

4.4 - Réalisation de la liaison

Celle-ci se fait par les unités centrales. Dans toute machine existent des entrées sorties directes du même type que celles qui ont été décrites en III.1. Elles permettent de dialoguer avec des périphériques particuliers, en général les périphériques temps réel. Le coupleur de ces périphériques est alors semblable à celui qui est décrit en III 2.2.

Notons que ces deux calculateurs sont des éléments fonctionnant en mode maître, alors qu'un périphérique est toujours un élément fonctionnant en mode esclave. Ceci implique pour la liaison intercalculateurs qu'il faille éviter les tentatives d'accès simultanés par les deux calculateurs. En effet, le dialogue met en jeu des interruptions ; l'utilisation du même coupleur pour réaliser les dialogues calculateur central à calculateur frontal et inversement nécessite un mécanisme de protection. On résoudra ce problème simplement en utilisant deux voies séparées l'une en lecture, l'autre en écriture. On assure ainsi que lorsque l'un des calculateurs initialise un dialogue, l'autre n'est pas gêné dans la même opération. La réalisation de la liaison exploite les possibilités du système d'interruptions externes de chacun des calculateurs. Chaque émission d'ordre est accompagnée d'un signal d'interruption pour le destinataire. Sa présence est interprétée comme une demande de lecture sur la voie correspondante. Cependant la prise en compte d'une interruption peut demander un temps non négligeable pendant lequel la voie doit être bloquée. Ceci implique donc une réponse du destinataire indiquant la bonne réception de l'ordre et le déblocage de la voie afin que l'émission des ordres en attente puisse être lancée à son tour. La réponse du récepteur se fait toujours sur la voie qu'a utilisée l'émetteur. On utilisera donc en fait deux

coupleurs, l'un d'émission pour le calculateur frontal et donc de réception pour le calculateur central et l'autre d'émission pour le calculateur central.

La séquence de transmission d'un ordre s'organise alors de la manière suivante :

coupleur 1 est le coupleur de la voie d'émission de calculateur 1

calculateur 1		calculateur 2
- adresser coupleur 1 et tester état		
- écrire l'ordre dans les registres de coupleur 1		
- alerter calculateur 2 par l'envoi d'une interruption	→	- réception de l'interruption et recherche du périphérique
		- adresser coupleur 1 et tester état
		- lire l'ordre présent sur les lignes
		- analyser l'ordre reçu et préparer l'ordre en retour
		- adresser coupleur 1
		- écrire l'ordre en retour dans les registres de coupleur 1
- réception de l'interruption et recherche du périphérique	←	- alerter calculateur 1 par l'envoi d'une interruption
- adresser coupleur 1 et tester son état		- préparer le lancement de la tâche
- lire l'ordre présent		- reprise de l'exécution
- analyser l'ordre reçu		
- débloquer le coupleur		
- lancement des émissions en attente ou reprise de la tâche interrompue		

Notons qu'il n'y a ainsi jamais de blocage de la transmission. La seule contrainte à respecter est que chaque calculateur organise ses émissions à l'aide d'une file d'attente et attende une réponse de son associé. Les émissions peuvent avoir lieu sur les deux voies simultanément.

Dans le cas particulier d'une liaison Multi 20 - 10070, deux remarques guideront la réalisation :

. les entrées sorties directes du 10070 portent sur trente deux bits alors que les ordres n'en comportent que seize. Seules seize lignes seront donc connectées.

. les entrées sorties réalisées par le Multi 20 portent sur un octet. Le coupleur sera organisé de ce côté de la voie comme un coupleur double et le microprogramme chargera successivement deux groupes de huit bascules. Ici encore le fait que le calculateur frontal soit microprogrammable permet de mener la réalisation avec un maximum de souplesse.

Nous avons dans ce chapitre uniquement défini l'environnement, c'est-à-dire les entrées-sorties ; la structure du Multi 20 n'est pas très adaptée à une utilisation en accès multiple et le microprogramme doit être modifié en conséquence. Ces adaptations portent sur l'architecture interne du calculateur, c'est-à-dire sur les instructions mises à la disposition de l'utilisateur. Des modifications sont souhaitables, d'autres sont essentielles ; c'est sur ces dernières que nous mettrons l'accent par la suite.

Chapitre IV

MECANISMES DE BASE DU SYSTEME FRONTAL

Il s'agit des mécanismes qui permettent de gérer les changements d'état de l'usager et d'assurer les échanges entre ses programmes.

Après avoir rappelé les notions qui sont à la base de l'organisation des systèmes d'exploitation, nous étudierons successivement chacun des mécanismes introduits.

1 - RAPPEL DE QUELQUES NOTIONS

Ces notions n'ont aucun caractère d'originalité, les travaux réalisés à l'IRIA ont largement contribué à leur diffusion si besoin était. Elles permettent de définir clairement les problèmes, et apportent une aide dans la conception et l'organisation de systèmes.

1.1 - Notion de processus

Un processus est la séquence dans le temps de toutes les actions élémentaires qui définissent un travail. La définition fluctue donc en fonction de la finesse de l'examen. En ce qui concerne cette étude , un processus pourrait aussi bien être une séquence microprogrammée qu'un programme. Lorsque nous parlerons de processus il s'agira toujours d'un programme.

La définition d'un processus implique qu'à celui-ci soient attachés :

- un algorithme définissant complètement la tâche qu'il accomplit
- des données d'entrée
- des données de sortie
- des ressources lui permettant de s'exécuter.

Les ressources nécessaires à l'exécution du processus peuvent être :

- passives (mémoire, disque ...)
- actives (les processeurs sur lesquels il s'exécute).

L'état d'un processus peut donc être défini par les ressources dont il dispose. Il pourra ainsi être actif ou bloqué suivant qu'il dispose ou non des ressources nécessaires à la réalisation de la tâche qu'il exécute.

Notons qu'on ne peut associer un processus au travail qu'accomplit la machine pour un usager. Un processus étant défini par la tâche particulière qu'il exécute, on conçoit que l'exécution d'un travail pour un usager peut nécessiter la coopération de plusieurs processus.

Nous pouvons donc donner une nouvelle définition de l'usager :

Un usager est l'ensemble des processus qui coopèrent à la réalisation complète d'une action qu'a demandée un utilisateur connecté.

1.2 - Notion de signal - Synchronisation des processus

La coopération des processus implique la possibilité pour certains d'entre eux d'être en attente de la fin de l'exécution d'un travail entrepris par d'autres. D'une manière générale un processus peut être en attente d'un évènement, il peut donc être actif ou bloqué.

On distinguera deux états de blocage :

- le blocage technologique (défini en IV 1.1, le processus ne dispose pas de la ressource lui permettant de s'exécuter)

- le blocage intrinsèque (le processus attend un évènement).

On a coutume pour séparer ces deux cas d'attribuer à chaque usager une machine virtuelle qui est l'image de la machine réelle et dans laquelle l'utilisateur possède toutes les ressources. Cette notion de machine virtuelle permet de synchroniser l'évolution des processus sans tenir compte des problèmes technologiques qui sont résolus séparément.

Nous ne considèrerons donc plus dans ce chapitre que le blocage intrinsèque.

A la notion d'évènement et de blocage doit être associée la notion de signal. L'apparition de l'évènement attendu par un processus doit provoquer son réveil. Celui-ci est assuré par l'émission d'un signal, assimilable aux signaux d'interruption des unités d'échange, par le processus ayant créé l'évènement. Notons qu'il n'est pas nécessaire que le processus qui émet ce signal sache à qui il est destiné ; de même que lorsqu'un périphérique émet une interruption de fin de transfert, il ne connaît pas le programme qui va être relancé par ce signal.

- **Signal mémorisé et signal impulsionnel**

D'une manière générale, les signaux entre processus seront mémorisés car il n'est pas d'une absolue nécessité qu'un processus se bloque immédiatement en attente de l'évènement demandé. Ce processus demandeur peut avoir la possibilité de continuer un certain traitement avant que le résultat que doit lui fournir le processus coopérant ne soit nécessaire. Le signal doit donc pouvoir être mémorisé de sorte à être encore présent au moment de son test par le processus demandeur.

Par contre, il peut être intéressant que certains signaux ne soient qu'impulsionnels, ceci correspondant au cas où n'étant pris en compte qu'avec retard ils ne sont plus d'actualité. Nous entendrons par signal impulsionnel tout signal qui n'est reçu que lorsqu'il est attendu.

- **Exclusion mutuelle**

L'exclusion mutuelle peut se définir de la manière suivante :

Deux processus peuvent au cours de leur évolution entrer dans une phase critique en ce sens qu'un seul d'entre eux peut être actif pendant cette phase. Il peut s'agir d'une phase d'accès à une ressource commune par exemple. Cette phase critique doit être en exclusion mutuelle, c'est-à-dire qu'il doit exister un mécanisme de règlement des accès tel que :

- lorsque l'un des processus entre dans sa phase critique, le second ne peut s'y engager avant que le premier ait terminé ;

- lorsque tous deux tentent simultanément d'entrer dans leur phase critique, un seul peut s'y engager, l'autre étant bloqué tant que le premier n'en est pas sorti.

1.3 - Sémaphores [11]

Différents mécanismes de synchronisation ou d'exclusion mutuelle ont été proposés. L'un d'entre eux semble particulièrement bien adapté à l'ensemble des problèmes. Il utilise des procédures ininterruptibles appelées primitives et notées *P* et *V*. Ces primitives ont été introduites par Dijkstra et agissent sur des sémaphores, variables entières associées chacune à un évènement. Ce sont les mécanismes associés aux sémaphores que nous utiliserons dans ce projet.

1.3.1 - Définitions :

On appelle sémaphore *S* l'association d'une variable entière, soit *E(S)* et d'une file d'attente, soit *F(S)*.

Le sémaphore doit être commun aux processus que l'on veut synchroniser et de plus il doit avoir été initialisé à une valeur positive ou nulle avant le lancement des processus coopérants.

Dijkstra introduit deux primitives de manipulation des sémaphores qui sont décrites en pseudo-algol :

```
P(S) : début E(S) := E(S) - 1 ;
       Si E(S) < 0 alors début
                          commentaire : Soit Pi le processus qui exécute cette
                                        primitive ;
                          état (Pi) := bloqué ;
                          Pi est rangé dans F(S)
                          fin
       fin ;

V(S) : début E(S) := E(S) + 1
       Si E(S) ⩽ 0 alors début commentaire : Soit Pi le processus qui exécute
                                             cette primitive ;
                          Sortir un processus de F(S) ;
                          commentaire : Soit Pj ce processus ;
                          état (Pj) := actif
                          fin
       fin ;
```

Tels qu'ils ont été définis, les sémaphores et les primitives *P* et *V* permettent d'assurer la synchronisation et l'exclusion mutuelle des processus.

1.3.2. - Utilisation des sémaphores

. exclusion mutuelle

Elle est résolue en introduisant un sémaphore, soit mutex initialisé à 1, et chaque processus exécute la séquence suivante :

```
P_i : début
          ----
      P(mutex) ;
      section critique ;
      V(mutex) ;
          ----
      fin ;
```

Notons que l'accès aux sémaphores est lui-même une section critique, les primitives *P* et *V* doivent donc être en exclusion mutuelle. Ce problème sera étudié au paragraphe suivant.

. synchronisation entre processus

Définition : Un sémaphore *S* est dit privé s'il est attaché à un processus particulier ;c'est-à-dire si seul ce processus peut exécuter la primitive *P(S)*, les autres processus n'agissant sur *S* que par la primitive *V*.

Un processus en attente d'un évènement se bloquera ainsi derrière son sémaphore privé par l'exécution de la primitive *P* et sera réveillé par l'exécution d'une primitive *V* sur le même sémaphore par le processus qui produit l'évènement attendu.

Ceci implique que : - un sémaphore privé ne possède pas nécessairement de file d'attente puisque seul le processus qui lui est attaché peut s'y trouver bloqué ;

- un sémaphore privé doit être initialisé à zéro de sorte que la première exécution de *P* provoque le blocage du processus.

Exemple : *E(S)* := 0 ;

```
P_i : début                P_j : début
        -----;                     -----;
      P(S) ;                     V(S) ;
        -----;                     -----;
      fin ;                      fin ;
```

Remarquons que la primitive *V* réalise la mémorisation d'un évènement. Si *V(S)* est exécuté plusieurs fois, le processus P_i a la possibilité de s'exécuter autant de fois. Il peut être intéressant parfois de simuler des signaux impulsionnels. L'utilisation de *V* pouvant avoir parfois des effets néfastes, on introduit la primitive *D* [12] :

```
D(S) : début tant que E(S) < 0 faire début E(S) := E(S) + 1 ;
                                           Sortir un processus Pi de F(S) ;
                                           état (Pi) := actif ;
                                    fin ;
                                    E(S) := 0
       fin ;
```

On voit alors que si le processus P_i était bloqué derrière son sémaphore privé, il est activé ; par contre s'il était actif, son sémaphore privé étant déjà à zéro, l'action de *D* n'a aucun effet.

. limitations d'emploi

Les mécanismes décrits n'assurent la synchronisation que dans la mesure où ils sont correctement employés. Si un processus devant entrer en section critique n'exécute pas *P*, il n'y a aucune garantie pour celui qui s'y trouve déjà. On peut ainsi imaginer des primitives plus complexes qui, à un nom de ressource critique, par exemple, associent un sémaphore et réalisent l'exclusion mutuelle, perdant ainsi leur souplesse d'emploi au profit d'une efficacité plus grande.

Les primitives *P* et *V* n'assurent pas qu'un processus bloqué sera activé au bout d'un temps fini; tout dépend de la stratégie de gestion de la file *F(S)*. Si celle-ci consiste à sortir les processus activés par ordre de priorité, plusieurs processus de priorité élevée peuvent se coaliser pour immobiliser indéfiniment un processus de priorité plus faible.

Sur la base de ces définitions et de ces notions, maintenant assez usuelles, nous allons construire un ensemble homogène de processus élémentaires en tenant compte :

- des impératifs du hardware
- des objectifs fixés à l'exploitation.

2 - CONTRAINTES PHYSIQUES-MICROPRIMITIVES

Les primitives décrites précédemment doivent être ininterruptibles. On peut penser les réaliser par microprogramme à l'exemple du T 1600 de la Téléméca-nique. Une telle réalisation impose que les primitives soient simplifiées dans leur définition afin que les microprogrammes correspondants n'encombrent pas la mémoire semi-morte. C'est ainsi que dans le calculateur précédemment cité, les sémaphores occupent un (ou plusieurs) mot mémoire comprenant le compteur et une file d'attente où chaque processus bloqué occupe un bit dont la position repère le numéro du processus. Cette solution simple limite les primitives à leur fonction essentielle décrite par Dijkstra. Il nous semble qu'au contraire, l'avantage de ces primitives est que leurs fonctions peuvent être développées afin d'y inclure la totalité de la gestion des processus au moins du point de vue de l'allocation d'unité centrale. De plus nous envisageons d'implanter de nombreuses primitives de manière à simplifier l'écriture du système et à contrôler les processus de l'usager. Les primitives doivent alors être simulées par programme.

Une contrainte physique due à l'architecture du Multi 20 apparaît à ce niveau :

- le microprogramme réalisant les entrées-sorties multiplexées peut provoquer à tout instant un déroutement sans qu'aucune protection puisse être envisagée

- les interruptions en provenance du canal d'accès direct ne peuvent être masquées, cette éventualité n'ayant pas été prévue par le constructeur.

Nous devons donc mettre en place un mécanisme simple qui permette de pallier ces difficultés. On remarquera à cet effet que rendre ininterruptibles les primitives, c'est placer l'unité centrale en section critique pendant leur exécution et donc réaliser une exclusion mutuelle. Nous pouvons penser pour cela à l'instruction *TAS N* (Test and Set) dont sont dotés les multiprocesseurs. Cependant l'utilisation de cette instruction suppose que le processus qui l'exécute, lorsqu'il constate par le test de la variable N le verrouillage de la ressource, entre en attente active, c'est-à-dire réexécute constamment l'instruction *TAS*. Son utilisation n'est donc pas envisageable selon cette définition dans le cas d'un monoprocesseur. Il faut associer au verrou *N* une file d'attente dans laquelle puissent être repérés les processus (les primitives) en attente de la ressource et donc compléter l'instruction de verrouillage par une instruction de déverrouillage de la ressource.

verrouiller (N) : début Si *N* = 0 alors *N* := 1
sinon mettre le processus dans la file d'attente *F(N)*
fin ;

déverrouiller (N) : début Si *F(N)* non vide alors sortir un processus de *F(N)*
sinon *N* := 0
fin ;

Nous préférons utiliser les primitives *P* et *V* après les avoir restreintes dans leur définition. Nous les appelons microprimitives et les notons *p* et *v*, elles sont réalisées par microprogrammes et sont ininterruptibles.

2.1 - Microprimitives

Nous appelons ici contexte d'un processus le contenu de l'ensemble des registres de la machine lorsque le processus est en exécution. Cet ensemble suffit à définir complètement une procédure si l'on est assuré qu'elle est résidente en mémoire. C'est le cas pour ce qui concerne les procédures simulant les primitives de synchronisation.

Nous associons à l'unité centrale un sémaphore *UC* d'exclusion mutuelle. Soit *F(UC)* la file d'attente associée à *UC*.

Puisque nous désirons simuler une machine à sémaphores, toute interruption provoquera par microprogramme un déroutement vers la procédure exécutant la primitive *V* ; aussi nous pouvons traiter le problème comme une demande simultanée d'exécution de primitives. Deux fonctions doivent être remplies à cet effet :

- la sauvegarde provisoire du contexte du processus interrompu. Ce peut être le contexte du processus (ou de la procédure simulant une primitive) qui était en cours d'exécution lorsque l'interruption s'est produite ou le contexte d'un processus réalisant l'appel d'une primitive.

- le test du sémaphore *UC* et éventuellement le rangement dans la file d'attente *F(UC)* du contexte de la primitive demandée. Le contexte des processus est sauvegardé par ailleurs dans les files d'attente associées à la gestion des ressources donc gérées par les primitives.

Afin d'associer ces deux fonctions, la file *F(UC)* est composée de deux piles LIFO placées tête bêche, la pile de sauvegarde provisoire des contextes et la

pile *F(UC)* proprement dite. Le choix de cette organisation se justifie par la nécessité de simplifier le plus possible la réalisation du microprogramme. Les différentes demandes d'exécution de primitives ne sont donc pas prises en compte dans l'ordre de leur arrivée. A ce niveau de la description, une remarque importante doit être faite.

Certaines interruptions doivent être traitées impérativement dans l'ordre de leur arrivée. L'analyse du système montre que le seul cas où cela se présente concerne les interruptions pour défaut de secteur et reprise d'alimentation secteur. C'est un cas bien précis qui sera totalement traité par microprogramme, le traitement qui lui est associé consistant simplement à ranger dans la pile de sauvegarde le contexte de la procédure en cours d'exécution et à arrêter le calculateur (micro-instruction d'arrêt) dans l'attente de l'interruption de reprise dont le traitement consiste à replacer dans les registres le contexte précédemment sauvegardé. La nécessité de traiter les interruptions systématiquement par ordre d'arrivée pourrait se présenter dans le cadre d'un système temps réel ; cependant on notera à ce sujet que les interruptions externes du Multi 20 peuvent être désarmées, permettant ainsi de les traiter selon une méthode classique.

2.1.1 - Appel d'une procédure système

Le traitement réalisé à l'appel d'une procédure consiste à ranger dans la pile de sauvegarde provisoire le contexte de la procédure interrompue par cet appel afin de libérer les registres pour que puisse y être élaboré le contexte de la procédure appelée. Cet appel est réalisé à l'aide de l'instruction *Brancher (AD,k)* où *AD* et *k* sont fournis dans les registres par l'appelant :

AD est l'adresse de la procédure appelée

k est l'ensemble des paramètres transmis par l'appelant dans les registres.

L'instruction *Brancher* s'écrit en pseudo-Algol :

Brancher (AD,k) : début ranger le contenu des registres en tête de *F(UC)*
ranger l'adresse *AD* dans le compteur ordinal ;
brancher à *INT* ;
commentaire : *INT* est le nom du microprogramme de test des interruptions puis d'acquisition de l'instruction suivante ;
fin ;

Cette instruction ne permet pas l'appel d'une primitive par un processus car elle est interruptible. De même, comme elle agit sur *F(UC)*, nous l'interdisons aux usagers. Elle n'est destinée qu'à permettre l'écriture des procédures simulant les primitives en autorisant le partage de sous-programmes. Pour réaliser l'appel d'une primitive il est nécessaire que le branchement réalise l'exclusion mutuelle de l'unité centrale, l'instruction utilisée doit donc inclure la microprimitive *p*.

A l'instruction **Brancher** doit être associée une instruction assurant le retour à la procédure appelante. Celle-ci s'écrit :

Retour : début ranger dans les registres le contexte qui se trouve en tête de *F(UC)* ;
retour à *INT*.
fin ;

2.1.2 - Microprimitives *p* et *v*

Ce sont des instructions sans paramètres et ininterruptibles. Elles s'écrivent en pseudo-Algol :

p : début *E(UC)* := *E(UC)* - 1 ;
si *E(UC)* < 0 alors début ranger le contenu des registres en queue de *F(UC)* ;
ranger dans les registres le contexte qui se trouve en tête de *F(UC)*
fin ;
Retour à *INT*
fin ;

v : début *E(UC)* := *E(UC)* + 1 ;
ranger dans les registres le contexte qui se trouve en queue de *F(UC)*
fin ;

On remarquera que la microprimitive *v* considère que la double pile *F(UC)* est une seule file d'attente. Celle-ci n'est jamais vide, elle contient au moins le contexte du processus que la primitive précédemment exécutée a rangé en tête de *F(UC)*. Si cette primitive est la dernière primitive demandée, on permet ainsi l'activation du processus qu'elle a sélectionné.

2.1.3 - Appel d'une primitive

L'instruction d'appel d'une primitive est un compromis entre l'instruction *Brancher* et la microprimitive *p*. En terminant l'instruction *Brancher* par retour à *RNI* (microprogramme d'acquisition de l'instruction suivante) et en imposant que chaque primitive commence par l'instruction *p*, nous pourrions assurer le branchement aux primitives. On remarquera cependant que cette méthode impose un mouvement de contexte non nécessaire, nous préférons donc créer l'instruction *Appel (k,a)* qui assure le branchement à la primitive de nom interne *k* et lui communique *a*.

Le changement d'un contexte étant une opération longue, nous n'utilisons pas la solution qui consisterait à utiliser la microprimitive *p* à l'intérieur de l'instruction *Brancher*;afin de minimiser le temps d'exécution, nous écrivons un nouveau microprogramme. Pour cela nous utilisons la structure du jeu de registres du multi 20. Un jeu de registres contient quinze registres dont neuf sont associés à l'unité centrale et les six autres sont les registres de travail du microprogramme. C'est dans ces six registres que le contexte de la procédure appelée est élaboré par le microprogramme, cette façon de faire permet d'éviter un changement de contexte. Le microprogramme s'écrit alors

Appel (k,a) : début déterminer l'adresse *K* de la procédure simulant la primitive à partir de *k* ;

commentaire : *k* est le déplacement dans une table qui permet de déterminer l'adresse du point d'entrée de la primitive ;

E(UC) := *E(UC)* - 1 ;

Si *E(UC)* < 0 alors Ranger *K* et *a* en queue de *F(UC)* (s'il s'agit d'un appel à la suite d'une interruption *a* se trouve dans les registres de travail) (*) ;

sinon début ranger le contenu des registres de l'unité centrale en tête de *F(UC)* ;

transférer *K* dans le compteur ordinal

fin ;

Retour à *INT*

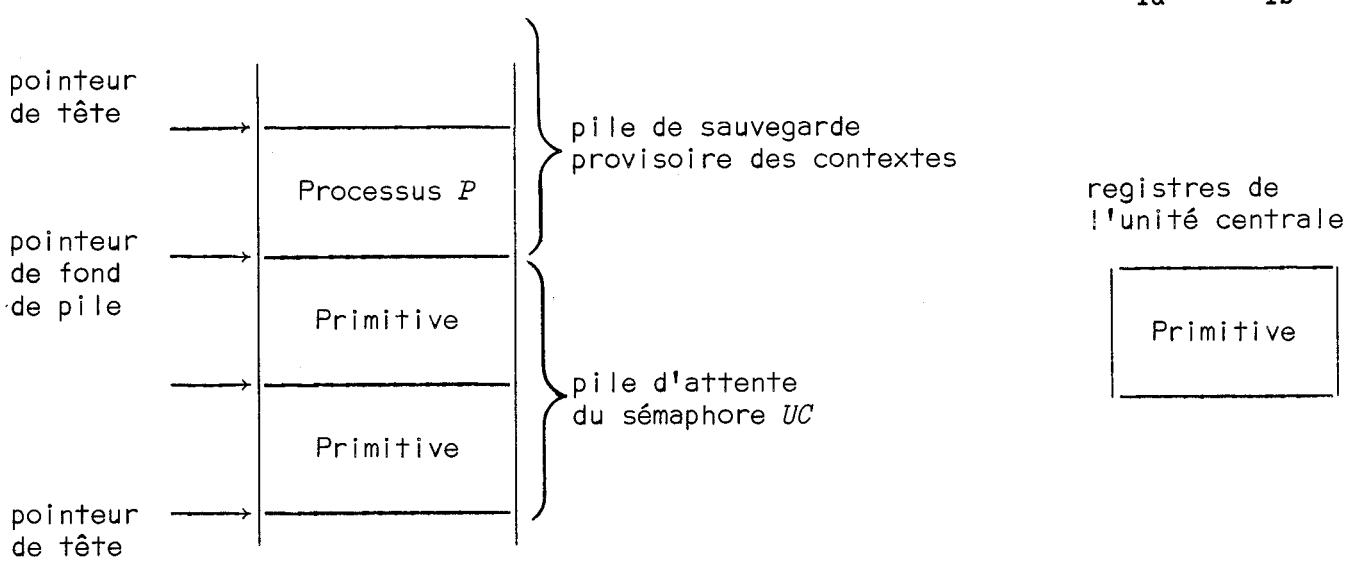
fin ;

(*) a est l'ensemble des paramètres transmis à la procédure appelée. Nous verrons (chapitre V) que les processus de l'usager utilisent l'adressage virtuel. Cette caractéristique limitera les utilisations de cette instruction car elle impose que les paramètres soient rangés dans les registres de l'appelant. L'opération "ranger a" se limite alors au transfert du contenu des registres de l'appelant dans les registres de l'appelée.

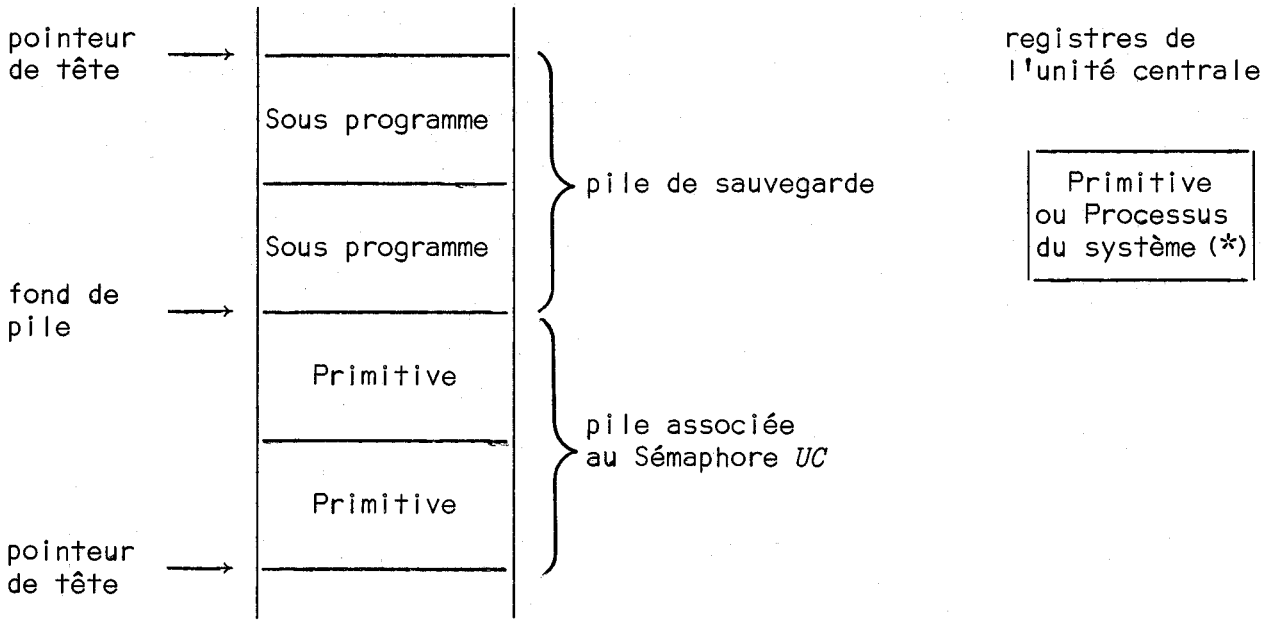
2.2 - Utilisation des microprimitives

Les instructions *Brancher, Retour, p, v* sont réservées à l'écriture du système et des primitives. Leur utilisation permet une utilisation optimale de la mémoire par le partage de sous-programmes. Cependant leur utilisation impose des contraintes. On remarquera que, lorsqu'un processus appelle une primitive dont l'exécution nécessite l'utilisation de sous-programmes, le contexte du processus interrompu cesse d'être accessible par ces sous-programmes. Aussi, la structure des primitives doit être telle qu'avant tout appel de sous-programme le contexte du processus interrompu ait été acquis. De même toute primitive se termine par le chargement en tête de *F(UC)* du contexte du processus à activer de sorte que l'exécution de la microprimitive qui termine la primitive permette d'acquérir ce contexte dans les registres de l'unité centrale.

F(UC) peut avoir deux aspects schématisés par les figures IV_{1a} et IV_{1b}.



a) lors de l'appel d'une primitive ou à la fin de l'exécution d'une primitive



b) au cours de l'exécution d'une primitive

figure IV_1 : Aspects de *F(UC)*.

(*) On remarquera que compte tenu de la structure de la microprimitive *v*, lorsque le contexte d'un processus d'un usager se trouve dans les registres, *F(UC)* est vide. Ce contexte étant chargé en tête de *F(UC)* et ne pouvant être rangé dans les registres que par la microprimitive *v*, lorsque ceci se produit, c'est que toutes les primitives en attente ont été exécutées.

3 - PRIMITIVES DE BASE

Nous avons défini jusqu'à présent des primitives de synchronisation donc des mécanismes "manipulant" des processus et des sémaphores. Différentes remarques s'imposent à ce sujet.

3.1 - Primitives travaillant sur des sémaphores

Le système décrit ici est de taille restreinte et est figé, en ce sens qu'un usager ne peut demander l'exécution d'un programme dont il serait l'auteur. On peut alors se poser le problème suivant :

Les sémaphores doivent-ils être créés à la génération du système ?

Compte tenu des travaux confiés au système par les usagers, les processus en cours d'utilisation seront certainement peu nombreux. Par contre, les usagers doivent disposer d'un potentiel d'exécution plus important. C'est-à-dire que les sémaphores utilisés à un instant donné seront sans doute peu nombreux mais qu'ils appartiennent à un ensemble de cardinalité plus importante.

Toute conception de système repose sur un compromis entre l'utilisation de la mémoire (encombrement minimal) et les performances souhaitées (rapidité d'exécution maximale). Pour respecter cette règle, il semble donc que les sémaphores doivent être déclarés au fur et à mesure des besoins.

En fait, nous devons distinguer deux types de sémaphores :

- les sémaphores manipulés par le système, ils sont peu nombreux et souvent utilisés. Ils doivent donc être d'accès rapide et seront tous déclarés à la génération du système.

- les sémaphores manipulés par les processus des usagers. Ils appartiennent à un ensemble dont la cardinalité peut être élevée et ne doivent être accessibles qu'aux processus d'un même usager. Ils seront déclarés au fur et à mesure des besoins et ne seront accessibles que par leur nom. Ce mode de représentation nous permettra par ailleurs d'assurer la protection des sémaphores d'un usager vis-à-vis de ses concurrents.

A ces deux types de sémaphores doivent donc être associés deux types de primitives :

- Des primitives travaillant sur des adresses de sémaphores (ou des indices dans la table des sémaphores). Ce sont les primitives *P(S)*, *V(S)*, *D(S)* où *S* est un indice dans la table des sémaphores.

- Des primitives travaillant sur des noms de sémaphores. Les concepteurs d'Esope les appelent : *Demander (S)*, *Libérer (S)*, *Délivrer (S)* où *S* est le nom d'un sémaphore de l'usager et dont les actions sont semblables respectivement à celles de *P*, *V*, *D*. Les procédures qui leur sont associées sont identiques aux procédures qui simulent *P*, *V*, *D* mais possèdent une entrée commune à une procédure de recherche du sémaphore dans la table des sémaphores. Tous les sémaphores d'un même usager sont chaînés entre eux. La tête de liste est donc définie complètement par la connaissance du numéro de l'usager et la procédure commune consiste alors à rechercher séquentiellement dans cette liste un sémaphore de nom *S*. Lorsque

celui-ci est trouvé, il suffit de remplacer le paramètre de la primitive par l'indice correspondant et de renvoyer à la procédure qui simule *P*, *V* ou *D*.

A ces primitives doivent être associées les deux primitives suivantes :

- *Déclarer (S,n)* : déclare un sémaphore de nom *S* dans l'espace de l'usager qui exécute cette primitive et lui affecte la valeur initiale *n*.

- *Effacer (S)* : supprime de l'espace de l'usager exécutant cette primitive le sémaphore de nom *S*.

3.2 - Primitives associées à des processus [12]

Les remarques faites précédemment peuvent être étendues aux processus. A chaque processus créé par l'usager doit être associé un espace mémoire que ce soit dans la mémoire physique ou la mémoire virtuelle de l'usager. Les processus coopérant à l'exécution d'une tâche seront sans doute peu nombreux, aussi devons-nous permettre une création dynamique de ceux-ci. A cet effet nous définissons les primitives :

créer (P,p) : crée dans l'espace de l'usager le processus de nom *P* avec la priorité *p*

détruire (P): détruit le processus *P*

disparaître : le processus qui exécute cette primitive disparaît.

Ces trois primitives sont interdites aux processus du système.

L'utilisation de ces primitives ne peut être laissée sans contrôle à l'ensemble des processus, en particulier lorsqu'il s'agit d'effacer un sémaphore ou de détruire un processus. Afin d'assurer ce contrôle, on affecte à chaque processus un pouvoir dont le rôle est de décrire ses droits d'accès aux primitives. Ce pouvoir est défini lors de la création du processus et est rangé dans son contexte.

4 - COMMUNICATIONS PAR MESSAGES

Lorsqu'un processus demande le concours d'un autre processus à l'aide de la primitive *V*, cette activation s'accompagne souvent d'un échange d'informations. Supposons que tous les processus d'un usager soient résidents en mémoire, cet échange peut se faire à des adresses définies par eux d'autant plus facilement que ces échanges sont en général peu nombreux. Par contre lorsqu'il s'agit d'un échange entre processus d'usagers et processus du système les échanges pouvant être nombreux, il importe qu'une structure de données soit mise en place afin qu'une utilisation incohérente ne perturbe pas les échanges (voir **figure IV_2**). En particulier, il faut éviter la destruction par un usager des messages déposés par ses concurrents. Il n'est donc pas souhaitable que l'usager puisse accéder directement à la structure de données utilisée . Ceci peut être réalisé en réservant à chaque usager une zone de dialogue en mémoire. Cependant cette zone de dialogue avec le système peut devenir rapidement encombrante car il faut tenir compte du nombre d'usagers potentiels et non du nombre d'usagers actifs. Aussi préfèrerons-nous utiliser un ensemble de tampons toujours présents en mémoire et organisés en listes fermées. Ces tampons étant banalisés, leur accès doit avoir lieu en exclusion mutuelle et les séquences exécutées par des processus coopérants de l'usager et du système s'écrivent :

Soit mutex le sémaphore d'exclusion mutuelle utilisé :

Processus émetteur *(PE)* :

1- *P* (mutex 1)
2- prendre un tampon dans la liste
3- *V* (mutex 1)
4- ranger le message dans le tampon
5- *P* (mutex 2)
6- ranger le message dans la liste des tampons destinés au processus *PR*
7- *V* (mutex 2)
8- *V(SR)* où *SR* est le sémaphore d'attente du processus destinataire
9- *P(SE)* où *SE* est le sémaphore derrière lequel le processus émetteur attendra éventuellement la fin de l'action demandée à *PR*.

Processus récepteur *(PR)* :

1'- *P* (mutex 2)
2'- prendre le tampon contenant le message de l'émetteur
3'- *V* (mutex 2)
4'- lire le message et définir le traitement demandé
5'- *P* (mutex 1)
6'- ranger le tampon dans la liste des tampons libres
7'- *V* (mutex 1)
8'- exécuter le traitement
9'- *V(SE)* après avoir éventuellement placé dans un tampon un message pour *PE*.

Ces opérations sont délicates car elles peuvent provoquer un blocage progressif de l'ensemble des usagers. En effet, supposons qu'un processus d'usager n'exécute pas *V* (mutex i) après avoir accédé aux tampons que protège le sémaphore mutex ; aucun processus ne pourra y accéder et de plus il se bloquera dès qu'il le tentera. Il n'est donc pas souhaitable d'utiliser ce mécanisme, même si nous décidons de n'autoriser ces accès qu'à des processus particuliers de l'usager (voir chapitre VI). De plus, un sémaphore étant repéré par le nom qu'il possède chez l'usager qui l'a déclaré, il n'est pas possible aux usagers d'accéder aux listes de tampons à l'aide des primitives définies précédemment. Plutôt que de créer des nouvelles primitives palliant cette difficulté nous préférons créer des primitives réalisant l'accès aux tampons.

On remarquera pour cela que ces tampons étant destinés aux échanges entre les usagers et le système dans le but de réaliser des entrées-sorties (voir chapitre VI), deux types de messages peuvent être transmis :

- des ordres pour les processus de gestion des périphériques
- des messages à émettre (ou à recevoir) vers le périphérique (ou du périphérique).

4.1 - Structures de données

De par l'orientation de l'organisation que nous allons mettre en place, nous utiliserons deux types de structures de données :

- un ensemble de tampons destinés à contenir des ordres
- un ensemble de tampons destinés à contenir des messages.

De plus nous devons associer à chacun des processus de gestion des périphériques une structure de données destinée à recevoir les ordres émis par les usagers.

Chacune des structures de données que nous allons définir est organisée en liste fermée. Nous appellerons réservoirs les listes de tampons non utilisés et directives les tampons contenant les ordres destinés aux processus du système.

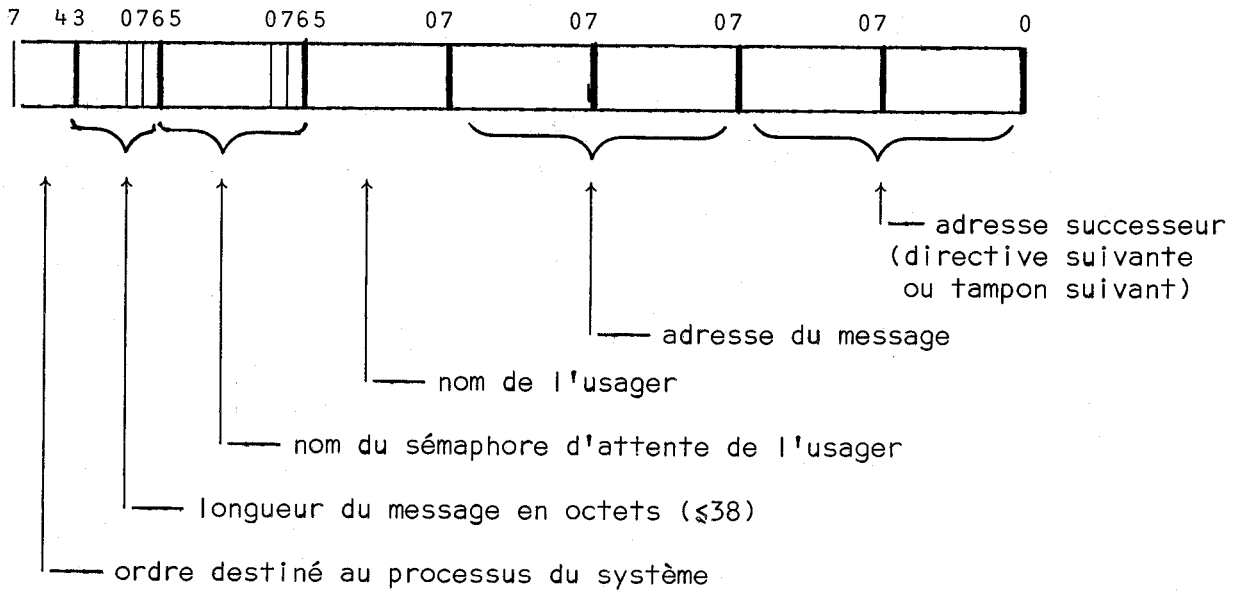
Logiquement quatre types de réservoirs devraient être prévus car chacun des tampons peut recevoir quatre destinations :

- directive pour un processus de gestion d'un périphérique
- message pour la zone de dialogue du disque
- message "ordre" pour le calculateur central (voir chapitre III)
- message pour un terminal conversationnel.

En fait nous n'utiliserons que deux formats de tampons (7 octets et 14 octets) et laisserons à l'usager la possibilité de chaîner des tampons dans la limite de trois. Les transferts d'ordres et de messages étant réalisés chez l'usager par un processus particulier que l'on peut interprèter comme étant la prolongation du système dans son espace,nous assurerons le contrôle de l'utilisation des réservoirs et en éviterons ainsi un emploi abusif.

Lorsque l'un des processus du système active un processus d'un usager, c'est qu'il a exécuté pour lui un traitement qui lui avait été demandé. En conséquence, il n'y a pas de directives associées aux processus des usagers. Par contre il se peut que l'activation du processus de l'usager demandeur doive être accompagnée d'un message en retour, ce message sera toujours rangé dans un tampon défini par l'usager lui-même. En procédant de cette manière, on est assuré que cet usager possède les informations nécessaires au retrait du message.

Ces remarques définissent le format des directives. Elles occupent des tampons de 7 octets.



Nous savons que le format des ordres échangés avec le calculateur central est de deux octets, il n'y aura donc pas dans ce cas de message.

A chaque processus destinataire est associé un descripteur (il n'y en a donc que trois) dans une table. Chacun comprend :

- l'adresse du dernier tampon directive reçu par le processus (2 octets)
- le nom de son sémaphore privé (1 octet).

Eventuellement l'adresse du premier tampon directive du processus peut être nulle, elle en indique alors l'absence. De même on associe à chaque type de tampon un descripteur de deux octets contenant l'adresse du premier tampon non utilisé.

L'ensemble de ces descripteurs est rangé à une adresse inaccessible par l'usager.

La figure IV_2 schématise cette organisation.

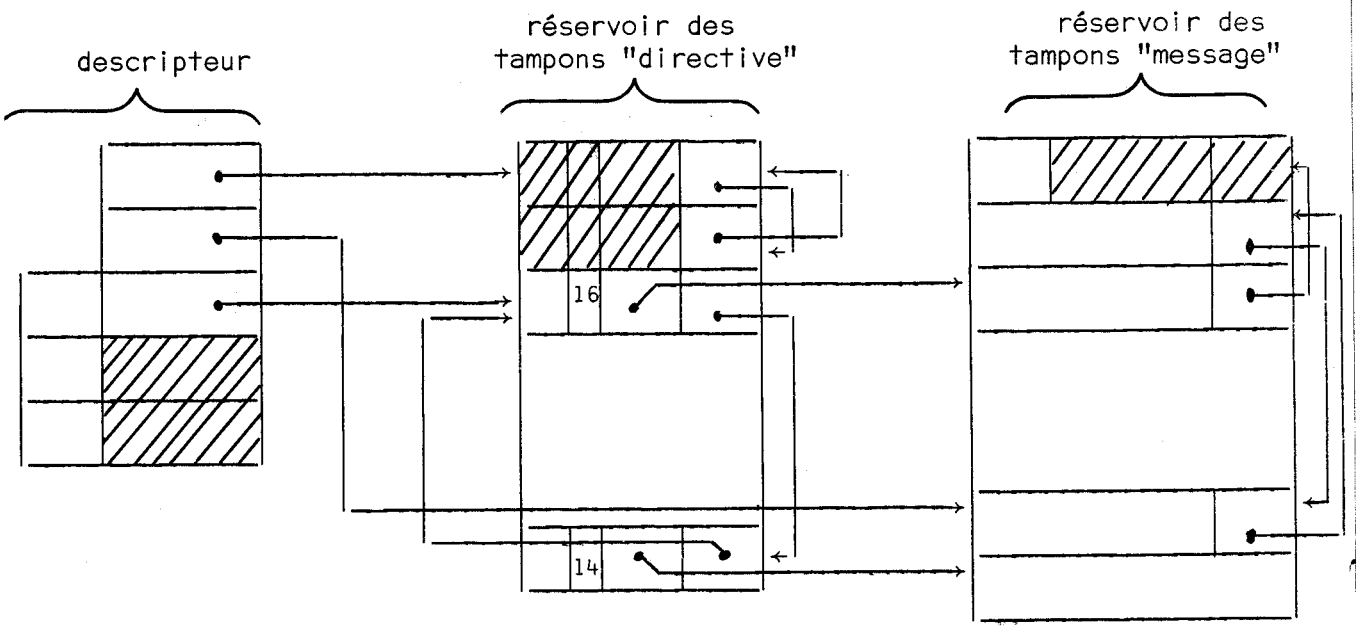


figure IV_2 : Structures de données utilisées lors des communications de messages.

4.2 - Primitives de communication

Compte tenu des structures de données utilisées, on doit distinguer trois types de primitives :

- primitive d'acquisition d'un tampon
- primitive de restitution d'un tampon
- primitive de transmission d'une directive à un processus.

4.2.1 - Primitive d'accès aux tampons

Nous nommons la primitive d'acquisition *Prendre*. Son rôle est de retirer un tampon de la liste des tampons libres et de fournir son adresse au processus demandeur. Elle utilise le paramètre k indiquant le type de tampon et fourni dans l'accumulateur par le processus demandeur et restitue dans celui-ci l'adresse du tampon.

Prendre (k) : début : lire *k* dans l'accumulateur ;
si *k*=0 alors adresser le pointeur de la liste des tampons "directive" libres
sinon adresser le pointeur de la liste des tampons "message" libres ;
si liste tampons libre = vide alors branchement vers "attendre" (voir IV 4.3)
sinon début sortir un tampon libre ;
réorganiser la liste ;
ranger l'adresse du tampon dans l'accumulateur
fin
fin ;

La primitive de restitution *Rendre (k,AD)* où *k* est le type de tampon et *AD* son adresse réalise l'opération inverse.

4.2.2 - Primitive de Transmission d'une directive

Son rôle est d'insérer la directive fournie par l'usager dans la liste des directives du processus destinataire et d'exécuter la primitive V sur son sémaphore privé.

Transmettre (k,AD) : début commentaire *k* désigne le destinataire et *AD* l'adresse du tampon utilisé ;
adresser le descripteur *k* soit *S* le sémaphore qui y est lu ;
chaîner le tampon d'adresse *AD* aux tampons déjà présents dans la liste des directives du processus ;
ranger le nom de l'usager dans la directive ;
V(S)
fin ;

Le processus destinataire étant un processus du système, il lui sera possible d'accéder à son descripteur et d'y trouver l'adresse de la directive reçue.

Nous avons choisi de ne pas bloquer le processus demandeur derrière un sémaphore qu'il aurait déclaré (voir IV 4 introduction) car il s'agira d'un

"pseudo processus du système" aussi il n'est pas nécessaire d'en assurer un contrôle strict. En agissant ainsi on permettra des traitements intermédiaires avant son blocage en attente de la fin du transfert.

4.3 - Remarque

Un problème n'a pas été abordé, il s'agit de l'action à entreprendre lorsqu'un tampon ne peut pas être alloué à un processus faute d'éléments dans la liste des tampons libres. Nous utiliserons ici encore une structure de liste dans laquelle sera introduit chacun des processus se trouvant dans ce cas (on simule ainsi la file d'attente d'un sémaphore). Il suffit alors de tester la présence de processus dans cette liste chaque fois que la primitive *Rendre* est utilisée et éventuellement de sortir le premier d'entre eux. Si lors de son rangement dans la liste, son compteur ordinal a été décrémenté de "un" il peut alors réitérer sa demande.

Cependant cette méthode n'est pas souhaitable car elle alourdit les primitives et ne garantit pas pour autant que les processus ainsi bloqués pourront se voir allouer un tampon. On doit plutôt considérer cette éventualité comme une "erreur système" (réservoirs insuffisants). De plus, il est possible que ce blocage intervienne après d'autres allocations de tampons au même processus, auquel cas ceux-ci sont perdus momentanément et le phénomène peut devenir cumulatif.

5 - CONCLUSION

Après avoir introduit le contexte d'une procédure, nous avons été amenés à y introduire des nouveaux vecteurs pour décrire le processus qui exécute cette procédure. La description qui a été faite au fur et à mesure que nous introduisions ces nouvelles définitions est encore insuffisante, en particulier on doit y inclure la description de la mémoire utilisée par le processus. Nous devrons y revenir au chapitre suivant.

Les primitives décrites ici peuvent sembler nombreuses, en fait, on remarquera qu'elles utilisent pour beaucoup d'entre elles des sous-programmes communs. On peut donc s'attendre à ce que la mémoire ne soit pas encombrée trop abusivement par ces procédures.

L'introduction de ces différentes primitives a pour but de permettre une écriture aisée du système d'exploitation et doit permettre une mise au point plus aisée de ce dernier. Nous serons amenés à en introduire d'autres par la suite afin de faciliter la gestion de la mémoire par le système.

Chapitre V

REALISATION DU SYSTEME D'EXPLOITATION DU CALCULATEUR FRONTAL

Nous étudierons dans ce chapitre :

- l'organisation de l'espace d'adresses du disque partagé
- le mécanisme d'adressage de la mémoire centrale
- l'organisation de l'allocateur de ressources.

1 - PRINCIPES D'ORGANISATION DU SYSTEME FRONTAL

1.1 - Organisation de l'espace d'adresses en mémoire secondaire

On considère habituellement que l'espace d'adresses disponibles en mémoire secondaire est infiniment grand, ce principe permet d'adopter une méthode de gestion de cet espace très simplifiée. Elle peut consister à ne considérer que les limites des zones libérées et à ne réallouer celles-ci que dans la mesure où elles permettent le rangement consécutif des informations d'un même fichier. L'ensemble des informations à gérer pour permettre l'utilisation de cette méthode peut être assez peu encombrant en mémoire, cependant cette méthode provoque une fragmentation de l'espace d'adresses en zones qui ne pourront jamais être réallouées à cause de leur taille insuffisante. Nous ne pouvons utiliser ici le même principe ; l'espace d'adresses dont on dispose étant restreint en regard du nombre d'usagers potentiels, il importe de limiter la fragmentation.

Notons que cette contrainte, déjà rencontrée dans les premiers systèmes multiprogrammés de par le partage de la mémoire centrale, a conduit à la conception de mémoires pagées. Nous adopterons ici le même principe pour le disque partagé, l'organisation des fichiers sous Siris 7 nous permettra d'adopter des pages de 1024 octets. Cependant cette méthode impose de posséder en mémoire centrale un descripteur pour chacune des pages de disque, et nous verrons que l'ensemble de ces descripteurs devrait être résident. Il est impossible d'admettre qu'il en soit ainsi car ceci provoquerait en mémoire centrale un encombrement important ; à titre d'exemple pour un disque de un million d'octets, le stockage des descripteurs

utiliserait 4096 octets en mémoire centrale. On remarquera que les usagers sont tous équivalents en ce sens qu'ils subissent des traitements mettant en jeu les mêmes procédures et travaillent sur des fichiers de taille voisine, aussi il est tout à fait vraisemblable qu'ils réclament tous un espace de taille identique sur le disque partagé. De plus, lors de la conception d'un système, il faut considérer que celui-ci sera utilisé à sa charge maximale ; nous attribuerons donc à chaque usager un espace de taille fixe sur le disque partagé, la pagination permettra d'utiliser celui-ci d'une manière optimale. Ainsi, il suffira de posséder en mémoire centrale les descripteurs des pages de disque de l'usager résident. A cet effet dans l'espace disque est allouée en permanence à chaque usager potentiel une page appelée "page de garde" et destinée à la mémorisation des descripteurs de son espace.

Il semble qu'il suffise d'attribuer à chaque usager un espace disque de 20 à 30 K octets, en fait, cette dimension devra être déterminée plus précisément par l'expérience. Dans une première période, la charge du calculateur frontal sera limitée à une trentaine d'usagers et le disque choisi sera suffisant. Eventuellement par la suite, un deuxième disque pourra être connecté.

Il peut sembler contradictoire d'utiliser un disque dont les secteurs sont d'une taille aussi réduite (12 octets) pour le réorganiser en pages de 1024 octets. En réalité, hormis le fait que le coupleur soit plus simple à réaliser puisqu'il existe déjà partiellement, l'adoption de cette solution nous permettra de réaliser le dialogue entre les deux systèmes.

L'organisation de l'espace d'adresses du disque en pages de 1024 octets doit être supportée par le "software" ; afin de limiter les temps de calcul des adresses disque, cette décomposition n'est que fictive, chacun des descripteurs de page comportera non un numéro de page mais l'adresse physique du premier secteur de la page. On remarquera que le nombre de secteurs sur une piste étant de 256, il y a une probabilité de 1/3 pour qu'une page se trouve sur deux pistes consécutives. L'incrémentation du numéro de piste n'étant pas réalisée automatiquement par le coupleur, tout transfert concernant une page répartie sur deux pistes nécessitera deux accès. On pourra simplifier notablement les accès en conservant toujours la description de l'espace disque de l'usager dans sa page de garde même lorsque celui-ci n'est pas en session.

1.2 - Réalisation du multitraitement

Une phase est initialisée par l'émission d'un message depuis le terminal de l'usager, on peut donc définir le temps d'interaction comme l'intervalle de temps qui sépare l'émission de deux messages consécutifs par l'usager. On admet couramment que le temps d'interaction est voisin de trente secondes, ce que nous avons pu vérifier par ailleurs (voir Troisième Partie).

Ce temps est long en regard des traitements courts que réalise le système frontal, on doit donc s'attendre à une fréquence de traitement très réduite et on peut donc penser qu'il suffirait d'adopter une méthode de partage de temps classique consistant à amener un à un en mémoire centrale les processus actifs. La mise au point d'un tel système est très simple, cependant on remarquera qu'il nécessite des chargements fréquents. Or compte tenu de l'organisation de l'espace d'adresses du disque partagé, le temps de chargement d'un processus est élevé. De plus cette organisation du système ne permettant pas une utilisation optimale de la mémoire, il est nécessaire de limiter sa taille, ceci se justifie par ailleurs par la taille réduite des procédures utilisées, et par conséquent d'organiser les données de chaque usager en fichiers provoquant ainsi une augmentation du nombre d'échanges avec la mémoire secondaire qui risque de devenir rapidement prohibitive lorsque la charge du système augmente. Nous nous proposons donc d'organiser la mémoire en pages de sorte que les organisations de la mémoire secondaire et de la mémoire centrale puissent se superposer, et d'adopter un mécanisme de chargement par page à la demande afin de permettre l'organisation en mémoire virtuelle de l'espace d'adresses des usagers. La mémoire centrale sera donc découpée en pages de 1024 octets ; ceci nous est imposé par la taille des pages de disque. Il est en effet nécessaire que la taille des pages en mémoire centrale soit un multiple de la taille des pages de disque afin que leur chargement puisse être assuré par un transfert simple.

En contrepartie, nous rendons plus complexe la mise en place du système frontal car nous allons devoir réaliser par microprogramme le mécanisme d'adressage. Mais nous verrons que le trafic entre mémoire secondaire et mémoire centrale peut être limité et que l'encombrement d'unité centrale ou de la mémoire par le système n'est pas nécessairement plus important que dans le cadre d'une organisation plus classique. Globalement le temps de réponse du système frontal organisé ainsi devrait donc être nettement meilleur.

On peut déterminer grossièrement le nombre d'usagers qui doivent être résidents pour qu'un système fonctionnant par chargement de page à la demande puisse être performant. Ce calcul consiste à considérer les états d'un usager. On peut définir à priori quatre états :

- en cours de chargement
- en cours de vidage
- bloqué
- actif.

Il semblerait donc qu'il faille quatre à cinq usagers en mémoire. Nous pensons pouvoir disposer d'au moins 32 K (1K = 1024 octets) de mémoire partageable entre usagers. Ceci signifie que s'ils sont cinq à être simultanément résidents, ils disposeraient en moyenne de 6 K de mémoire. Les procédures d'analyse et d'édition de texte étant courtes, si elles sont réentrantes nous pourrons certainement accepter ce nombre d'usagers.

1.3 - Réentrance des procédures

Nous avons déjà signalé que les usagers sont équivalents, en conséquence, la probabilité pour que les usagers présents en mémoire utilisent les mêmes procédures est élevée. Réaliser systématiquement la réentrance des procédures d'analyse et d'édition peut donc avoir une influence très favorable sur le comportement du système frontal :

- Si toutes les procédures sont réentrantes, on limite l'encombrement des usagers sur le disque partagé puisqu'il n'est pas nécessaire de recopier les procédures qu'ils utilisent dans l'espace qui leur est attribué. Cet espace peut donc être réduit ou organisé de façon à permettre une utilisation plus souple des données de l'usager.

- La probabilité étant élevée pour que les usagers présents en mémoire utilisent les mêmes procédures, le partage de pages sera fréquent ; il autorisera par conséquent l'admission d'un nombre plus important d'usagers en mémoire ou limitera le trafic de pages entre mémoire secondaire et mémoire principale.

1.3.1 - Réalisation de la réentrance des procédures

Nous rappellerons ici le principe de la réentrance et l'étendrons afin de permettre une organisation plus souple du système et une utilisation rationnelle de l'espace d'adresses de l'usager.

Le principe de la réentrance implique que l'espace d'adresses partagé par les usagers, c'est-à-dire la procédure, ne puisse être modifié. On est donc conduit à distinguer l'espace contenant les données propres à l'usager et l'espace contenant la procédure utilisée. Certaines règles d'écriture doivent être respectées de sorte que l'espace alloué à la procédure ne puisse être modifié par la procédure elle-même, ainsi toutes les variables telles que les compteurs de boucle ou les variables d'aiguillage doivent être intégrées à l'espace des données de l'usager.

En fait, on doit aller plus loin dans l'organisation de la réentrance, une structure dans laquelle on n'associerait que ces deux types d'espaces à chaque processus ne pourrait être utilisée avec toute la souplesse souhaitable.

Appelons Segment un ensemble de mots d'adresses consécutives contenant des informations de caractéristique semblable.

On peut distinguer dans un processus quatre types de segments :

- Le segment procédure : il contient la pure procédure et les constantes qui lui sont associées.

- Le segment des objets locaux : ce segment est associé directement à la procédure. Il appartient en propre au processus, et contient donc des objets dont la durée de vie est celle du processus. On y trouvera en particulier toutes les variables nécessaires au contrôle de l'exécution de l'algorithme. Notons qu'un segment local ne doit pas nécessairement être chargé primitivement avec les valeurs initiales de ces variables, celles-ci peuvent être contenues dans le segment procédure.

- Le segment des objets rémanents : ce segment est associé à l'usager, il est partagé par l'ensemble des processus de ce dernier. Il se justifie par la notion de coopération entre processus et peut contenir en particulier les messages qu'échangent ceux-ci.

- Le segment des objets externes : ces segments sont associés aux fichiers. Leur existence permet un accès et un partage plus aisés des fichiers par l'ensemble des processus de l'usager.

1.3.2 - Conséquences pour l'organisation de l'espace d'adresses du calculateur frontal

L'organisation en pages de la mémoire centrale ne suffit pas à assurer la réentrance des procédures, il faut pouvoir segmenter la mémoire virtuelle des usagers. Lorsque la mémoire est seulement pagée cette segmentation peut être simulée à l'aide des registres d'index de l'unité centrale, cependant l'organisation de la segmentation doit alors être supportée par l'usager et l'écriture des procédures est rendue plus complexe.

Dans sa version standard le multi 20 ne possède qu'un seul registre d'index, aussi cette segmentation ne pourrait-elle pas être assurée. Puisqu'un microprogramme gérant en pages la mémoire doit être écrit, nous nous proposons d'utiliser les particularités du Multi 20 pour simuler à l'aide de celui-ci la segmentation de la mémoire.

2 - ORGANISATION DE L'ESPACE D'ADRESSES DU CALCULATEUR FRONTAL

2.1 - Architecture du Multi 20

Le Multi 20 possède deux jeux de quinze registres généraux de huit bits chacun dont un seul est utilisé en version standard. Le second bloc de registres est inutilisable par le software dans la configuration choisie puisque certaines interruptions d'entrées-sorties ne peuvent être masquées. C'est ce deuxième bloc de registres que nous utiliserons pour assurer la segmentation. La communication entre les deux blocs peut être réalisée par le biais d'un registre fonctionnel, on utilisera le registre T d'entrée-sortie de l'unité arithmétique et logique.

Le premier bloc de registres contient actuellement :

- le compteur ordinal (2 registres)
- l'accumulateur (2 registres)
- l'extension de l'accumulateur (2 registres)
- le registre d'index (2 registres)
- le registre de précision (2 bits du registre 9)
- le registre de débordement (1 bit du registre 9).

Ces deux derniers registres occupant trois bits permettent de pallier la faible taille des mots en autorisant des traitements en précision multiple. Les registres 10 à 15 sont des registres de travail du microprogramme. Nous conservons cette organisation mais utiliserons deux bits supplémentaires du registre 9 destinés à indiquer le mode d'accès du processus :

- 1 bit d'utilisation de l'adressage virtuel
- 1 bit d'indication de mode maître ou esclave, et nous

appelons ce registre, vecteur d'état du processus. Son accès est réservé au microprogramme, mais il est cependant intégré au contexte du processus.

2.2 - Recherche d'un mode d'adressage

Le deuxième bloc de registres ne comportant que quinze éléments de un octet chacun, il est impossible d'y décrire dans sa totalité la mémoire virtuelle de l'usager. Aussi, nous reportons cette description dans la page de garde de l'usager.

On distinguera trois niveaux d'organisation :

- Les fichiers externes du système frontal. Ce sont ceux que supportent les mémoires secondaires du calculateur central.

- Les fichiers internes. Ils sont supportés par le disque partagé et peuvent être formés d'un ensemble de pages de disque, non contigues.

- Les segments. Ils appartiennent à la mémoire virtuelle de l'usager. Ils ont pour support les pages de disque d'un fichier interne et éventuellement des pages de la mémoire centrale.

La page de garde est utilisée à la description de la mémoire virtuelle de l'usager et à la mémorisation des tables d'implantation des fichiers internes de l'usager.

2.2.1 - Organisation de la page de garde

Elle n'appartient pas à l'espace de l'usager mais à un fichier interne du système, elle est donc inaccessible à l'usager.

On limite à 32 le nombre de segments de l'usager et à 192 le nombre de pages de mémoire virtuelle de l'usager. De même le nombre de pages de disque des

fichiers internes déclarés par l'usager est limité à 85. Ce sont là des limites maximales susceptibles d'un ajustement en fonction de l'espace fourni à l'usager sur le disque partagé. La page de garde contient alors :

- La table d'accès aux descripteurs de segments : cette table contient trende deux éléments de un octet chacun. On y entre à l'aide du nom du segment, l'octet ainsi adressé fournit le déplacement nécessaire pour atteindre dans la page de garde le descripteur de segment. Cette méthode permet un accès rapide dans la table des descripteurs de segments et nous est imposée par la taille des mots mémoire.

- La table d'accès aux tables d'implantation des fichiers internes de l'usager : elle a la même structure que la table précédente.

- La table des descripteurs de segments (TDS) : elle contient trente deux éléments de quatre champs : ce sont

. PR : la protection attachée au segment (2 bits)

. ℓ : la longueur en pages du segment, elle est limitée à 15 pages (4 bits). Une longueur zéro indique que le segment n'a pas été créé.

. AIT : le déplacement nécessaire pour atteindre dans la page la table des pages virtuelles du segment (10 bits).

. AIF : l'adresse de la table d'implantation du fichier interne support du segment (2 octets). Le numéro de page dans cette adresse repère l'usager. Rappelons que le nom zéro est toujours réservé au système. Il renvoie alors aux fichiers internes supports des segments procédures, les tables d'implantation de ces fichiers sont résidentes.

- La table des descripteurs de fichiers internes (TDF) : chaque élément occupe deux octets et fournit le déplacement permettant d'atteindre la table d'implantation du fichier ainsi que la longueur du fichier et sa protection.

- Les tables de pages virtuelles attachées aux segments (TPV) : ces tables sont rangées séquentiellement dans la page de garde et contiennent chacune autant d'éléments qu'il est indiqué dans le descripteur de segment. Chaque descripteur de page dans la TPV occupe un octet comprenant trois champs :

. d : le bit de disponibilité de la page en mémoire centrale

. a : le bit d'altération du contenu de la page physique

. p : le numéro de la page physique associée à la page virtuelle.

Lors de la création du segment, chaque descripteur de page est mis à zéro. Le premier octet de la table contient le nom du segment.

- Les tables d'implantation des fichiers internes propres à l'usager (TIF) : chaque descripteur occupe trois octets déterminant l'adresse début d'une page de disque et un indicateur d'utilisation. Le numéro du disque est décrit par ailleurs. On suppose que chaque usager possède son fichier sur le même disque. Le premier descripteur contient le numéro du fichier et éventuellement le nom du segment qui lui est associé.

- Deux tables réservées à la gestion de la mémoire physique TPA et TPD que nous décrivons en (§ V 4.3.3)

. la liste des pages de disque allouées à l'usager (LPDA) et encore disponibles

. le nombre de pages de disque encore disponibles pour l'usager (NDISP)

. le numéro du disque contenant l'espace de l'usager.

2.2.2 - Accès à la mémoire virtuelle

La translation adresse virtuelle-adresse réelle doit être rapide afin que les performances du calculateur ne soient pas diminuées par l'utilisation de ce mécanisme. On pourrait donc penser utiliser le deuxième jeu de registres du Multi 20, en remarquant qu'un registre permet de décrire une page virtuelle ; le nombre de registres ne le permet pas. On remarquera qu'il n'est pas nécessaire en réalité de décrire la totalité de la mémoire virtuelle de l'usager en mémoire rapide, on peut se contenter de décrire les dernières pages utilisées. Ce principe se justifie pleinement par la propriété de localité énoncée par ailleurs (§ V 3.1.3).

La réalisation d'un mécanisme de ce type dans le cadre du Multi 20 associerait deux registres à chaque page. Cependant, ne disposant pas d'une mémoire associative, nous ne pouvons que réaliser des accès séquentiels aux registres descripteurs. Un rapide calcul montre que le temps moyen de recherche d'une page dans la mémoire rapide ainsi formée serait de près de trois micro-secondes. Nous préférons donc utiliser la page de garde et organiser les registres disponibles de façon à accélérer la translation d'adresse.

Ayant remarqué que la réalisation de la réentrance d'une manière pleinement satisfaisante nécessitait qu'à chaque processus soient associés quatre segments, nous utilisons quatre groupes de trois registres du deuxième jeu de registres du Multi 20 et nous associons chaque groupe à un segment.

. Protection

Il n'y a pas de protection associée à la page, il y a deux raisons à cette particularité :

- Le nombre des segments disponibles permet d'associer un segment à chaque type d'information, on peut donc attribuer la même protection à l'ensemble des informations appartenant à un segment. On traitera le cas des fichiers en imposant à l'usager de réaliser un couplage segment-fichier. Par ailleurs cette façon de faire simplifie l'accès aux fichiers par le système.

- Le format des mots mémoire étant l'octet, il est nécessaire que les descripteurs de page n'occupent que huit bits afin de limiter le temps de recherche de l'adresse réelle. Les indicateurs de service (bit de présence, bit d'altération, adresse réelle de la page) occupant huit bits, il est impossible d'affecter une protection au niveau de la page.

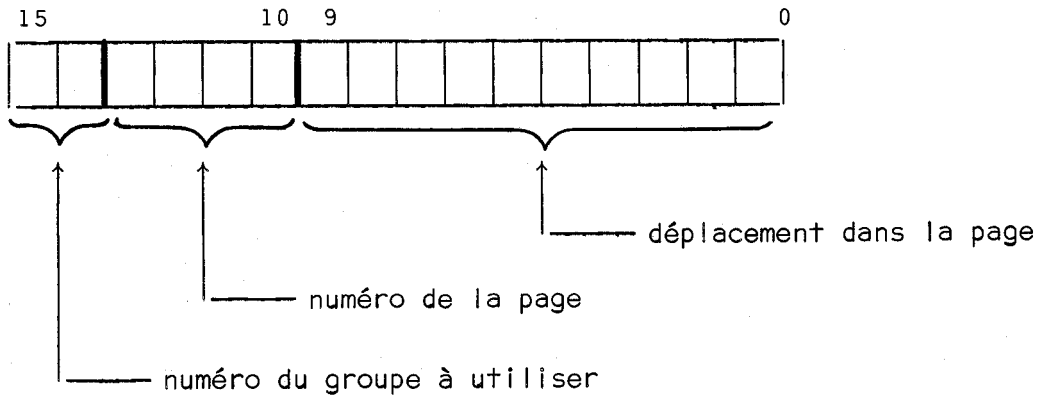
Les quatre modes d'accès au segment définis par les deux bits de protection sont :

- tout accès
- lecture et exécution
- écriture et lecture
- lecture.

. Recherche d'adresse

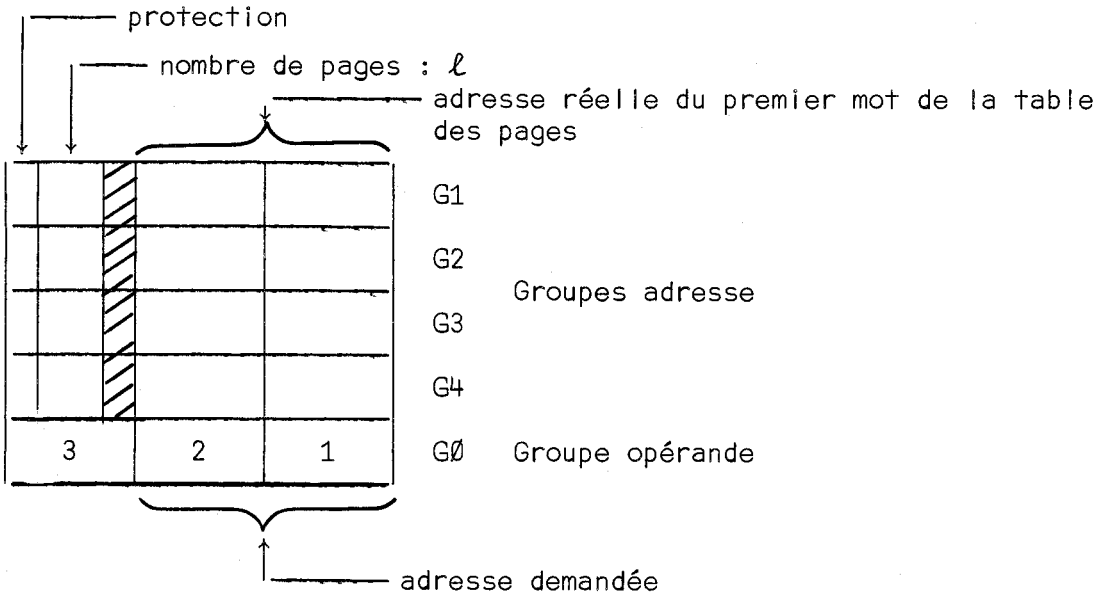
Seule l'affectation du groupe 1 est imposée, le groupe 1 repère nécessairement un segment procédure. Les autres groupes peuvent être affectés au gré du processus en cours d'exécution. D'une manière générale, les affectations se font à l'intérieur de primitives (§ V 2.2.3).

Au cours de l'exécution d'un processus, chaque groupe de registres contient l'adresse réelle de la table des pages du segment qui lui est associé, la protection du segment et sa longueur, une adresse dont les poids forts sont zéro indique l'absence de référence d'un segment. Tout adressage en mode virtuel se fait par référence à un groupe de registres. Le format d'une adresse est donc le suivant :



Le compteur ordinal fait implicitement référence au groupe 1, lors d'un adressage indirect dans une instruction de contrôle, l'adresse effective doit donc faire référence au groupe 1. Le vecteur d'état permet de déterminer le mode d'adressage à utiliser.

Le microprogramme de calcul d'adresse utilise cinq groupes de trois registres, ce sont les groupes adresse déjà cités et un ensemble de trois registres de travail appelé groupe opérande. On repèrera chaque registre par le nom du groupe auquel il appartient et par un numéro de registre. L'organisation du deuxième jeu de registres est donc la suivante :



Nous donnons ici l'algorithme général de calcul d'adresse, on supposera que les registres 1 et 2 de GØ contiennent l'adresse demandée.

Le calcul exige que les huit bits de poids forts de l'adresse demandée aient été recopiés à la fois dans GØ2 et dans GØ3. Cette opération réalisée systématiquement lors de la lecture de l'adresse demande 200 nanosecondes supplémentaires.

début commentaire : on commence par rechercher le groupe adresse utilisé, soit i son numéro, et à tester s'il y a violation de protection. Ceci nécessite qu'après avoir calculé le numéro de groupe, on le mémorise dans un registre du premier bloc. La violation de protection est testée par comparaison d'un code élaboré au cours du décodage de l'instruction avec le registre Gi3 ;

Faire apparaître le numéro de la page demandée ;

si numéro de page demandé > ℓ alors déroutement pour dépassement mémoire

sinon

début Additionner ce numéro de page à Gi et charger le registre adresse mémoire ;
Lire le descripteur de la page ;
si bit de disponibilité = 0 alors déroutement pour défaut de page

sinon

début commentaire : on supposera que lors du décodage de l'instruction un masque d'altération a été préparé et rangé dans un registre ;

Modifier le bit d'altération par une opération OU entre le masque et le descripteur de page ;

Elaborer l'adresse dans GØ ;

Recopier GØ1 et GØ2 dans les registres adresse mémoire

fin

fin

fin ;

L'adressage peut être considéré comme équivalent à une indirection suivie d'une indexation. On peut donc s'attendre à ce que le temps d'exécution du micro-programme soit élevé, d'autant que les microinstructions sont d'exécution assez lente et de puissance réduite. Le temps d'exécution d'une instruction classique (chargement de registre) est de 4,2 à 5,2 microsecondes auxquelles il faut ajouter le temps de calcul de l'adresse. Dans la version la plus rapide fournie par le constructeur, ce temps est de 2,2 microsecondes en moyenne. Le mécanisme que nous mettons en place devrait doubler approximativement ce temps. On peut donc compter qu'en adressage virtuel le temps total d'exécution (adressage compris) d'une instruction classique serait de l'ordre de dix microsecondes. Le calcul d'adresse en mode virtuel augmente ainsi le temps total d'exécution d'une instruction classique dans la proportion de vingt pour cent. Ceci peut paraître excessif, cependant on notera que :

- Dans le cas d'une instruction faisant référence à plusieurs octets successifs, cette proportion est sensiblement réduite puisque le calcul n'a lieu que lors du premier accès à la mémoire. Le temps d'exécution de ces instructions pourra donc rester acceptable.

- Les performances ne peuvent être jugées dans l'absolu, elles doivent être comparées à celles que permettrait un système ne possédant pas l'adressage virtuel. Nous avons déjà remarqué que dans un système de ce type les transferts entre mémoire secondaire et mémoire centrale sont très nombreux, or pendant tout ce temps le calculateur reste oisif.

Si on compare le Multi 20,dans la version que nous en proposons, au CII 10070, on constate que l'instruction LAD du 10070 demande 3,2 microsecondes de traitement en mode virtuel. Cette instruction peut être considérée comme équivalente à celle que nous avons citée à propos du Multi 20, car toutes deux nécessitent deux accès à la mémoire. Il y aurait donc un rapport trois en faveur du 10070 ; ce rapport montre que les performances du calculateur restent encore acceptables compte tenu de son utilisation.

. Traitement des déroutements

Chacun des déroutements introduits dans l'algorithme de calcul d'adresse correspond au branchement à une procédure du système (par exécution du microprogramme de l'instruction Appel). On peut assurer que si ce déroutement n'est pas traité immédiatement, les seules exécutions qui précèdent le traitement du déroutement concernent des primitives. En conséquence, on peut assurer que le contenu des registres qu'utilise le microprogramme dans le deuxième jeu de registres n'est pas détérioré. Ceci permet à la procédure de traitement du défaut de page de connaître,à l'aide d'une instruction d'accès à ce deuxième bloc, l'adresse virtuelle ayant provoqué le déroutement. Pour cela, on impose que le calcul d'adresse soit effectué dans le deuxième jeu de registres.

Il n'existe pas dans la version standard du microprogramme du Multi 20 d'instruction d'accès au deuxième jeu de registres. On introduira donc, en mode maître, deux types d'instructions de rangement et de chargement des registres de ce deuxième bloc. Ces instructions assurent le transfert de l'accumulateur A et des huit bits de poids faibles de l'extension accumulateur B dans un groupe de registres du deuxième bloc et inversement. Elles utilisent l'accumulateur car tout rangement dans un groupe sera la conséquence d'un calcul préalable, de même que tout chargement de l'accumulateur est destiné à permettre un calcul.

2.3 - Accès aux segments

Nous distinguons trois types de segments :

- Les segments de données dont le support est un fichier interne d'un usager. Ces segments ne sont pas partageables puisque la TIF qui leur est associée est contenue dans la page de garde d'un usager particulier.

- Les segments procédures dont le support est un fichier interne du système. Ces fichiers sont partageables.

- Les segments de données ayant pour support un fichier partageable. Ce sont par exemple les segments contenant la grammaire utilisée par les procédures d'analyse syntaxique. Ils ont pour support un fichier interne du système.

Un nom de segment est codé suivant cinq bits, le segment correspondant appartient à l'usager.

2.3.1 - Ouverture d'un fichier interne

Ouvrir un fichier interne c'est créer une table d'implantation pour ce fichier. Cette opération est réalisée à l'aide de la primitive *Ouvrir (f,p,ℓ)* où f est le nom du fichier, p la protection qui doit lui être affectée et ℓ sa longueur. Elle peut s'écrire

```
Ouvrir (f,p,ℓ) : début  commentaire f ≤ 32 et ℓ ≤ 15 ;
                 si descripteur = 0 alors
                    début lire l'adresse de fin de la TIF du dernier
                              fichier ouvert ;
                          créer le descripteur associé au fichier ;
                          pour i=1 pas 1 jusqua ℓ faire
                          début lire une adresse de page de disque ;
                                la ranger dans la TIF
                          fin ;
                          NDISP := NDISP - ℓ ;
                    fin
          fin ;
```

2.3.2 - Fermeture d'un fichier interne

La fermeture d'un fichier interne est assurée par la primitive *Fermer (f)* où f est le nom du fichier. Son action consiste à détruire le descripteur du fichier, restituer les pages de disque occupées à la LPDA de l'usager et décaler les TIF des fichiers créés ultérieurement à f. L'usager ne recevant en pratique qu'un nombre réduit de pages de disque, cette opération pourra être relativement rapide.

Fermer (f) : début lire le descripteur du fichier ;
descripteur := 0 ;

commentaire : soit ℓ la longueur du fichier ;
Pour i=1 pas 1 jusqua ℓ faire ranger le descripteur dans la LPDA ;
NDISP := NDISP + ℓ ;
Pour toutes les TIF qui suivent faire début
lire le descripteur suivant, il contient le numéro du fichier dont la TIF doit être déplacée ;
modifier son descripteur
fin

fin ;

2.3.3 - Création d'un segment

Un segment ne peut être créé qu'à la condition qu'un fichier interne ait été ouvert. L'opération consiste à créer un descripteur pour ce segment, puis une TPV. La protection donnée au segment est celle du fichier. La création de la TPV se limite à ranger le numéro du segment dans le premier descripteur de la TPV (il permettra d'assurer la destruction du segment) puis à mettre à zéro tous les descripteurs suivants dans la TPV. La longueur du segment est celle du fichier interne qui lui sera associé. L'opération est réalisée à l'aide de la primitive *Associer (s,f)* où *f* est le nom du fichier interne support du segment et *s* le nom du segment. *f* est codé sur huit bits dont le premier indique le type de segment (fichier partageable ou fichier de l'usager). On permet 64 fichiers partageables. Il est entendu que le nom d'un fichier d'usager ne peut dépasser 32, de même que le nom d'un segment.

2.3.4 - Destruction d'un segment

La primitive *Dissocier (s)* où *s* est le nom d'un segment assure, sur la TPV du segment *s* une opération semblable à celle que réalise la primitive *Fermer (f)* sur la TIF du fichier *f*. La destruction d'un segment précède la destruction du fichier et remet à jour le premier descripteur de la TIF du fichier interne qui lui était associé en effaçant le nom du segment.

2.3.5 - Appel d'un segment

Nous avons indiqué que les groupes 2 à 4 des registres d'accès à la mémoire virtuelle de l'usager référençaient des segments quelconques définis par le processus en cours d'exécution. Pour le permettre, nous créons la primi-

tive *Unir (n,S)* où *n* est le numéro du groupe de registres à utiliser et *S* le nom d'un segment. Par l'appel de cette primitive, le processus s'engage à référencer ce segment à l'aide du groupe *n*.

```
Unir (n,S) : début si n=1 alors déroutement accès interdit
                    sinon début
             Adresser le descripteur du segment S et le recopier dans
                  le groupe opérande ;
             commentaire : soit ℓ la longueur de la table des pages
             du segment, elle se trouve dans le groupe opérande ;
             si ℓ=0 alors déroutement segment inexistant
                    sinon début

                                ranger le numéro de page réel de la
                                     page de garde dans le groupe
                                     opérande ;
                                recopier dans le groupe n les registres
                                     du groupe opérande qui contiennent
                                     l'adresse de table élaborée précé-
                                     demment
                          fin
                    fin
        fin ;
```

Aucun appel à un segment procédure ne peut avoir lieu par cette primitive. Le chargement du groupe 1 est réalisé lors de la création du processus.

2.3.6 - Conséquences pour les processus travaillant en mode virtuel

La primitive *Créer (P,p)* doit être précisée. Nous avons indiqué que celle-ci assurait la création du processus *P*. En fait, elle lui associe le segment de nom *P*. On crée donc dans le contexte de chaque processus un champ de 4 octets appelé champ mémoire et destiné à mémoriser le nom des quatre segments manipulés par le processus. Le premier octet de ce champ est également interprété comme le nom du processus étant entendu que le nom d'un processus est le doublet (nom de l'usager, nom du segment procédure du processus). L'absence d'un segment est représentée par un nom 255. Ce sera le cas pour les octets 2 à 4 du champ mémoire au moment de la création d'un processus. C'est au processus de déclarer les segments de données qu'il utilise et au besoin de les créer. La primitive *Unir (n,S)* doit donc être complétée par la recopie du nom *S* dans l'octet *n* du champ

mémoire du contexte du processus qui l'exécute.

En conséquence, lorsqu'un processus perd l'unité centrale, le contenu du deuxième jeu de registres n'est pas sauvegardé puisqu'il en existe déjà l'image dans le contexte du processus. Ceci permet de simplifier l'action des microprimitives qui n'ayant pas connaissance du mode d'adressage utilisé par la procédure interrompue devraient systématiquement assurer cette sauvegarde. De plus on réduit ainsi l'encombrement en mémoire du contexte des processus. Inversement lorsqu'une primitive active un processus, elle doit assurer le chargement des registres utilisés par le microprogramme d'adressage, pour cela elle doit créer leur contenu par une recherche dans les tables associées aux segments nommés dans le champ mémoire.

Nous allons maintenant étudier l'allocation des ressources du système frontal. Nous serons conduits à choisir des mécanismes déjà implantés dans ESOPE [12,16]. Nous n'étudierons donc que leurs principes et nous les discuterons en fonction de leur adaptation au système frontal défini ici.

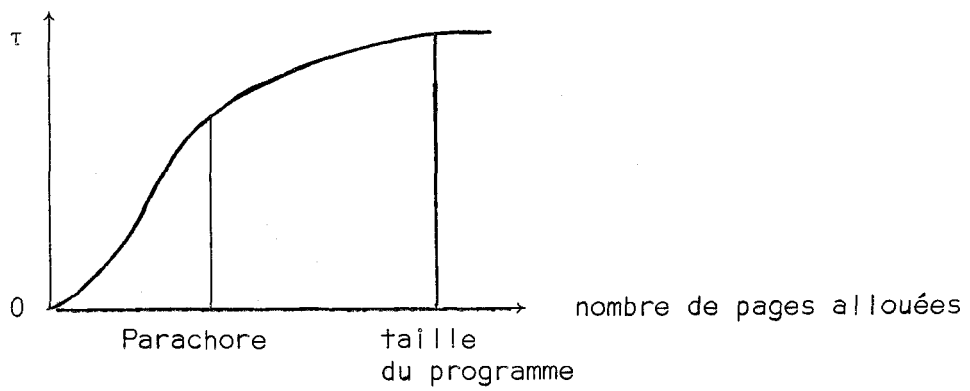
3 - PRINCIPES D'ALLOCATION DES RESSOURCES DU CALCULATEUR

3.1 - Comportement dynamique des programmes

3.1.1 - Parachore

Considérons un programme en exécution dans un espace mémoire inférieur à sa taille. Il est intéressant de connaître le comportement du programme en fonction de la taille de la mémoire qu'on lui alloue. Ce comportement peut être analysé à l'aide du temps τ qui s'écoule entre deux références à des pages non présentes.

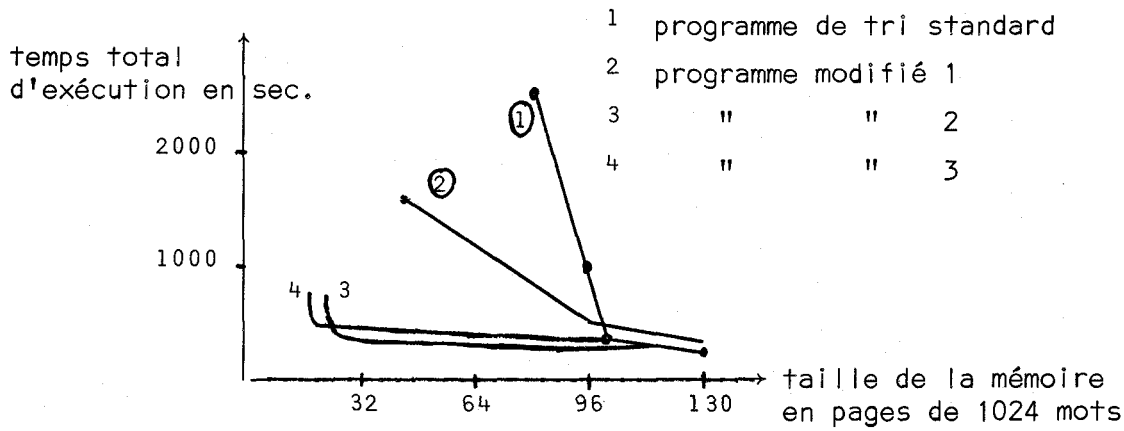
En fait, τ est également une fonction de l'algorithme de chargement des pages manquantes et aussi des caractéristiques du programme. On obtient une courbe ayant l'allure suivante :



Cette courbe met en évidence un point particulier appelé "parachore". Au-delà de ce point, τ tend à se stabiliser montrant ainsi qu'il n'est pas nécessaire d'allouer au programme une taille de mémoire égale à son encombrement. Le parachore est en général très inférieur à la taille du programme.

3.1.2 - Influence du style de programmation

Brawn et Gustavson [13] ont comparé quatre versions d'un programme de Tri. D'une version à l'autre, les changements n'affectaient que le mode de référence à l'information. La première version est le programme standard, les autres essayent de tenir compte de la pagination. Elles ont obtenu les courbes suivantes



Il semble donc que l'influence du style du programmeur soit très importante. C'est un atout important pour les performances du système frontal étudié ici car il n'exécutera que des programmes de service qui pourront être écrits en fonction de la pagination. La faiblesse intrinsèque des performances du calculateur (lenteur de l'unité centrale) pourra être compensée si nous tenons compte des caractéristiques de l'allocateur de ressources lors de l'écriture de ces programmes.

Brawn et Gustavson ont poussé plus loin leurs expériences et ont étudié l'influence des algorithmes de remplacement. Les courbes obtenues montrent que l'influence des algorithmes est nettement moindre que celle du style de programmation.

3.1.3 - Working Set [14]

D'une manière générale on peut admettre que le parachore provient de la propriété de localité qui caractérise les programmes. Dans un intervalle de temps donné, toutes les pages d'un programme n'ont pas la même probabilité d'être réfé-

rencées. Ceci provient de ce que :

- les programmes sont écrits séquentiellement
- ils sont souvent structurés en modules
- les données sont regroupées
- les programmes contiennent des boucles internes.

Partant de cette propriété, Denning définit le Working Set $W(t,\tau)$ d'un programme à l'instant t comme l'ensemble des pages auxquelles il a fait référence dans l'intervalle $(t-\tau,t)$. On montre que $W(t+\alpha,\tau)$ tend vers $W(t,\tau)$ lorsque τ est petit.

L'utilisation optimale de la mémoire dans un système utilisant le partage des ressources voudrait donc qu'un programme ne soit autorisé à utiliser l'unité centrale que s'il possède son Working Set. En utilisant le chargement de pages à la demande on peut en pratique réduire cette condition à ne donner l'unité centrale à un processus que si l'on peut lui réserver un nombre de pages égal à la cardinalité de son Working Set.

Les notions de parachore ou de Working Set nécessiteraient pour être utilisables qu'une mesure dynamique soit faite pour chaque processus. Cette méthode est trop complexe à mettre en oeuvre. On peut citer cependant un système où une adaptation particulière de l'architecture proche de celle que propose Denning permet la mesure dynamique du Working Set [15].

3.2 - Réalisation - Méthode des catégories

Cette méthode tend à déterminer par approches successives une valeur approximative du Working Set. Chaque programme appartient à un instant donné à une catégorie définie par la taille de la mémoire réservée au programme et un temps de résidence. Un programme ne peut être admis en mémoire que si le système peut lui allouer un nombre de pages correspondant à sa catégorie. Il n'y a jamais restitution d'espace dans un programme au profit d'un autre programme. Un programme perd la mémoire lorsque son temps de résidence est écoulé ou lorsqu'il tente de dépasser sa catégorie. Il changera alors de catégorie. On distingue deux cas de changement de catégorie :

- lorsque le programme tente d'accéder à une page non présente alors qu'il a atteint la limite de sa catégorie, il passe alors dans une catégorie où la taille est plus importante

- lorsque, après l'écoulement de son temps de résidence, on constate que l'usager a fait référence à un nombre de pages inférieur à un minimum défini par la catégorie à laquelle il appartient. On lui affecte alors une catégorie inférieure. Cette limite peut être par exemple la limite maximale d'une catégorie inférieure. Cette méthode est utilisée dans ESOPE et c'est elle que nous appliquerons.

En pratique il ne sera pas nécessaire de définir un nombre important de catégories, car les usagers sont équivalents et subissent dans le système frontal des traitements bien définis et peu nombreux. Elles devront être ajustées par l'expérience.

On remarquera que les pages de garde associées à chaque usager ne peuvent être toutes simultanément résidentes. Aussi on définit la page de garde comme appartenant à l'espace même de l'usager. En conséquence un processus d'un usager ne peut être admis en mémoire qu'à la condition que sa page de garde y soit déjà. On décide donc afin de limiter le trafic de pages que l'allocation de mémoire se fait par usager et non par processus. Il en sera de même pour l'unité centrale qui sera allouée sur la base d'un quantum à chaque usager (§ V 4.2.1). Cette allocation a lieu pour un temps fini. Cette politique permet de plus d'assurer le partage des segments de données entre processus d'un même usager.

On distingue deux niveaux de gestion de la mémoire sur le principe des catégories :

- niveau 1 : la mémoire est distribuée aux usagers dans la limite des pages disponibles. L'usager reçoit en fonction de sa catégorie un certain nombre de pages. C'est la réservation.

- niveau 2 : l'usager reçoit une page physique chaque fois qu'il tente d'accéder à une page non connue. Ceci se fait dans la limite de la réservation faite précédemment. C'est l'allocation.

Nécessairement la réservation précède l'allocation.

4 - REALISATION DE L'ALLOCATEUR

4.1 - Etats des usagers

L'allocation de ressources ne peut se faire dans n'importe quel ordre sous peine d'étreinte fatale.

Exemple : Soient deux processus parallèles A et B demandant chacun l'utilisation des ressources M et N. On supposera que leur exécution nécessite ces deux ressources. Initialement les deux ressources sont libres. Supposons que l'algorithme d'allocation consiste simplement à tester la disponibilité des ressources. La ressource M étant libre, elle est attribuée à A, N étant libre elle est attribuée à B et aucun des deux processus ne peut s'exécuter. Les deux ressources sont perdues car les processus A et B ne pouvant s'exécuter, ils ne peuvent les restituer. Supposons que M soit la mémoire et N l'unité centrale en allouant N à B le système perd toute possibilité d'intervenir et le système est définitivement bloqué.

Cette remarque conduit à définir une hiérarchie dans les états des usagers en fonction des ressources dont ils disposent.

On peut distinguer trois classes d'états :

- oisif : tous les processus de l'usager sont bloqués intrinsèquement
- prêt : il existe au moins un processus de l'usager qui est actif mais l'usager ne dispose d'aucune ressource réelle (Exécution des processus sur la machine virtuelle de l'usager). Pour distinguer le blocage technologique du blocage intrinsèque nous dirons que ces processus sont prêts.
- courant : l'usager participe à la distribution des ressources de la machine réelle.

Les seules ressources partagées dans le système frontal sont la mémoire centrale et l'unité centrale. En effet nous avons attribué d'une manière définitive à chaque usager potentiel un espace sur le disque du calculateur frontal et les terminaux sont nécessairement associés chacun à un usager déterminé.

On peut donc définir deux états principaux dans la classe des usagers courants :

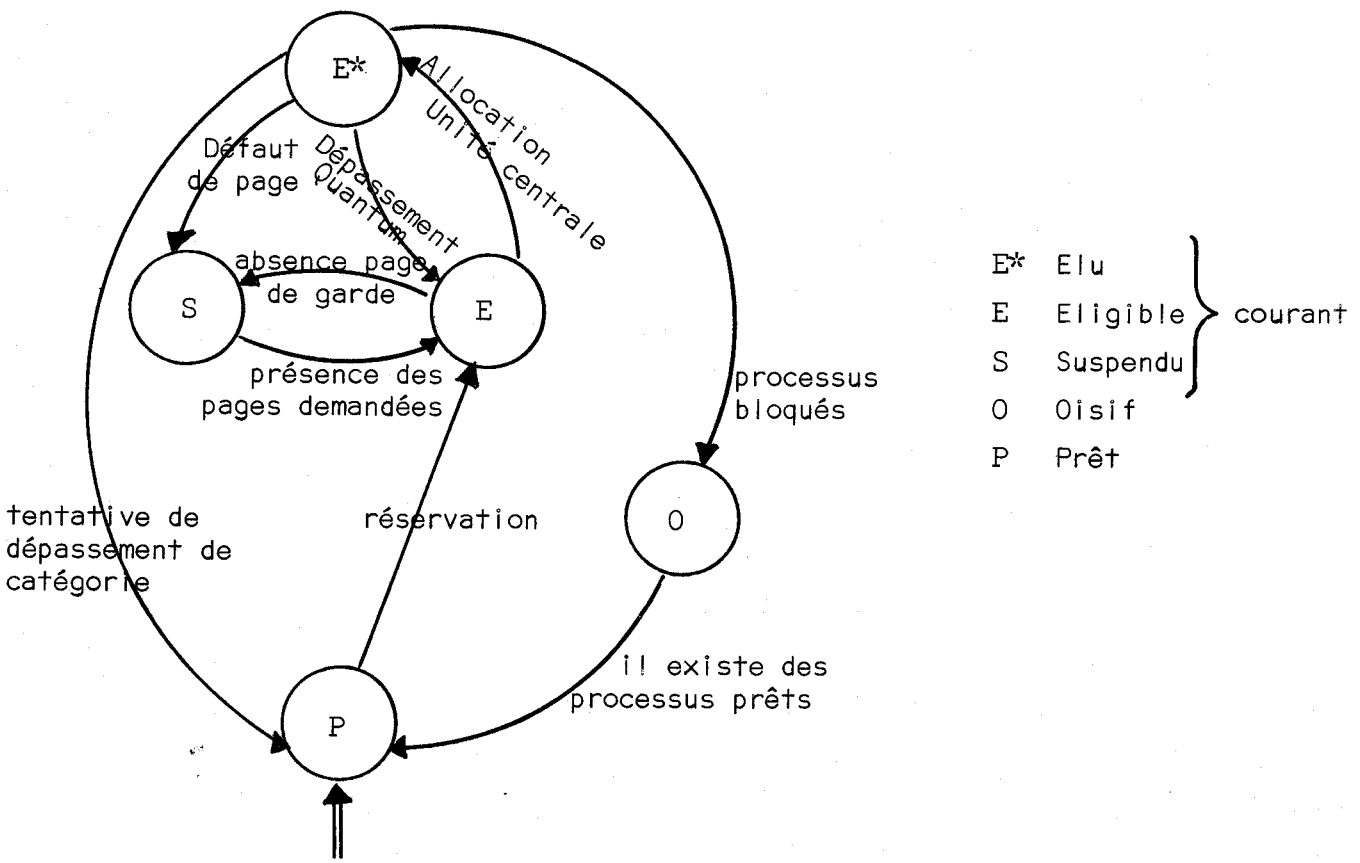
- éligibles : l'usager a reçu un espace mémoire et sa mémoire virtuelle est connue

- élu : l'usager possède la mémoire et l'unité centrale.

En fait, le principe même de l'allocation conduit à distinguer un état intermédiaire :

- suspendu : l'usager est en attente du chargement de pages dont il a demandé l'allocation.

On notera un cas particulier où l'usager est dans cet état : lorsque l'usager a participé à la réservation, on n'est pas assuré que sa page de garde se trouve en mémoire. Aussi dans la plupart des cas il passera dans l'état suspendu en attente de la présence de sa page de garde en mémoire. Ceci est nécessaire pour qu'il puisse être élu.



4.2 - Etude globale de l'allocateur

4.2.1 - Principes

Nous avons indiqué qu'il était nécessaire d'allouer la mémoire à l'usager et non à un processus de l'usager. Ceci provient de ce que les processus ne peuvent participer à l'allocation d'unité centrale que dans la mesure où la page de garde de l'usager est présente. L'unité centrale quant à elle pourrait être allouée par processus. Cependant, on notera que cette politique serait incompatible avec le principe d'allocation de la mémoire. Lorsqu'un usager possède la mémoire, on doit tenter de l'en faire sortir le plus vite possible afin de libérer de l'espace pour un autre usager. Cependant ceci doit être fait en tentant de minimiser le trafic de pages,aussi doit-on exécuter un maximum de processus de l'usager pendant qu'il réside en mémoire. En allouant l'unité centrale à l'usager on augmente la probabilité de trouver chez celui-ci des processus actifs, un processus en cours d'exécution pouvant réveiller un processus bloqué. Cependant, on ne doit pas laisser l'usager exécuter ses processus pendant toute la durée du temps de résidence que lui attribue sa catégorie. Ceci favoriserait les usagers dont les processus bouclent. Afin d'améliorer le temps de réponse apparent du système, on définit donc un temps maximal d'allocation d'unité centrale appelé quantum et choisi de sorte que les travaux courts puissent s'intercaler et provoquer rapidement une libération de la mémoire qu'occupe l'usager pour lequel ils s'exécutent. Le quantum devra être déterminé expérimentalement car il est fonction du type d'utilisation du système. Nous avons mesuré le temps d'exécution de l'analyse syntaxique d'une phrase Algol sur le Multi 20, en moyenne il est de 2 millisecondes par caractère. Les performances étant partiellement réduites par l'adressage virtuel et l'analyse syntaxique devant être complétée par d'autres traitements, nous pensons qu'un quantum de 100 millisecondes devrait convenir.

4.2.2 - Conséquences

. Pour les demandes de pages :

Les demandes de page de l'usager peuvent être traitées globalement, ou chaque fois qu'elles se produisent. On remarquera que si cette dernière solution était retenue, il pourrait arriver que le temps de résidence soit atteint alors qu'un transfert de page est en cours. En conséquence, les demandes ne sont que mémorisées. Chaque fois qu'un processus est dérouté pour un défaut de page, il est bloqué derrière un sémaphore technologique associé à ce type de traitement et propre à l'usager. Lorsque l'usager n'a plus de processus actif, le chargement

des pages est demandé dans la mesure où l'usager n'a pas dépassé son temps de résidence.

. Pour les processus des usagers :

Puisque l'on décide d'allouer l'unité centrale à l'usager, il faut définir une relation d'ordre entre ses processus prêts. Ceci est réalisé à l'aide de la priorité qu'ils reçoivent lors de leur création. A chaque usager est associée la file de ses processus prêts. Lorsqu'un usager quitte la file d'attente d'un sémaphore (y compris le sémaphore associé aux défauts de page), il est inséré dans la file des processus prêts du même usager en fonction de sa priorité. L'utilisation de la priorité permettra d'optimiser le partage des segments de données et donc de réduire le trafic de pages. On peut en effet considérer que chacun des processus,qui en crée un autre, partage avec lui les mêmes données, les références aux pages de l'espace virtuel de l'usager sont donc moins dispersées.

Le changement d'usager élu et éventuellement l'activation du processus de chargement des pages demandées supposent que l'on puisse reconnaître le dernier processus d'un usager. Cette détection devrait se faire chaque fois qu'on réalise la destruction ou le blocage d'un processus donc en particulier dans la primitive P. Celle-ci étant très utilisée, il faut réduire son temps d'exécution. Les concepteurs d'ESOPE [12,16] ont proposé d'attribuer à chaque usager un processus FINAL dont le seul rôle est d'activer le processus du moniteur qui réalise les changements d'usager. Ce processus a la plus faible priorité chez l'usager et est toujours prêt, de sorte qu'il soit toujours activé lorsque l'usager n'a plus de processus prêt. Nous adopterons cette solution. Ce processus est toujours résident, il pourrait s'écrire :

```
FINAL : V (Sémaphore du moniteur)
        Allera FINAL ;
```

. Allocation d'unité centrale

L'allocateur gère trois files circulaires correspondant aux trois états fondamentaux : Oisif, prêt, courant. La file des usagers prêts est formée des usagers en attente de mémoire, celle des usagers courants comprend ceux qui attendent l'unité centrale. Elle comprend les usagers éligibles et les usagers suspendus, un mot d'état les repère. L'usager élu est désigné par un pointeur courant. Un deuxième pointeur repère le point d'entrée dans la file. A chaque usager est associée sa file des processus prêts. On définit ainsi une chaîne d'activation.

L'unité centrale est allouée à chaque instant, soit au premier processus du moniteur (celui-ci est repéré par le pointeur du point d'entrée) si celui-ci possède des processus prêts, soit au premier processus de l'usager élu si ce n'est pas le cas. Lorsque la file des processus prêts de l'usager élu est épuisée, son processus FINAL appelle le processus du moniteur chargé de changer les états des usagers. Celui-ci élit un nouvel usager.

L'allocation d'unité centrale est incluse dans les primitives, qui agissent suivant le principe des priorités entre moniteur et usager élu et entre processus d'un même usager.

4.2.3 - Amélioration de l'allocateur

Kaiser cite encore trois méthodes permettant d'améliorer les performances du système. Nous allons discuter l'implantation de celles-ci sur le système frontal :

. Préchargement

Dans le système proposé, lorsqu'un usager devient courant (on supposera que sa page de garde est chargée), pour la première fois il n'a participé qu'à la réservation. Ses processus sont donc déroutés l'un après l'autre pour défaut de page et l'usager passe très vite à l'état suspendu.

On peut éviter cela en chargeant préalablement les pages dont on peut prévoir l'utilisation. Ce sont les pages référencées par les compteurs ordinaux des processus de l'usager et celles qui n'avaient pu être chargées lors d'un passage précédent en mémoire.

On remarquera que cette méthode ne permet que d'éviter les différents changements de contexte et le traitement des déroutements qui sont nécessaires (un par processus), avant d'atteindre le processus FINAL de l'usager lors du premier passage. Elle peut être plus intéressante et permettre de retarder un deuxième passage à l'état suspendu si on essaye également d'approximer le Working Set de l'usager à l'aide des n dernières références qu'il a faite. Diminuer le nombre de passages à l'état suspendu d'un usager, permet d'utiliser au maximum son temps de résidence en mémoire et donc de limiter son temps total de séjour entre l'ouverture et la fermeture de sa session.

Dans notre système, cette méthode nous permettrait de rentabiliser le chargement de la page de garde de l'usager en y associant d'autres pages. Cependant

ceci nécessitant l'acceptation d'un encombrement supplémentaire de la mémoire et de l'unité centrale, nous préférons ne pas l'utiliser dans une première version.

. Vidage préventif

Il consiste à utiliser le temps d'oisiveté du disque pour recopier sur disque les pages non allouées dont le contenu a été modifié lors d'une utilisation précédente.

Ceci permet de toujours posséder une liste de pages allouables dès qu'une demande est présente et tend à optimiser l'utilisation du disque en régularisant son taux d'occupation.

L'utilisation de cette méthode peut avoir une influence importante sur les performances du système frontal, en effet :

- Hormis la réalisation du chargement ou du vidage de pages du système frontal, le disque est utilisé par le système central ; de plus, il participe à l'exécution des entrées-sorties multiplexées et au dialogue entre calculateurs.

- De par la structure du canal d'accès direct et du coupleur, les transferts ne peuvent être optimisés que dans la mesure où ils concernent des secteurs consécutifs d'une même piste. Ceci nécessite que le premier secteur d'un transfert suive le dernier secteur du transfert précédent. Ces caractéristiques peuvent avoir pour conséquence des régimes transitoires pendant lesquels la charge du disque est importante. On réalisera le vidage préventif en utilisant deux files d'attente des transferts sur disque. L'une d'entre elles est réservée au vidage préventif, aux échanges de messages entre les deux calculateurs, c'est la moins prioritaire. Ce fait est peu important pour les usagers qu'elle concerne car ils sont oisifs ou ont une forte probabilité de le devenir, ou encore le transfert demandé ne nécessite pas une grande rapidité (voir chapitre VI). Chaque fois que le vidage d'une page a eu lieu, la liste des pages libres est mise à jour. La deuxième file d'attente est utilisée pour les chargements de page ou les vidages nécessaires à un chargement.

On notera que le vidage préventif ne doit se faire que pour un nombre réduit de pages de sorte que l'allocation puisse toujours avoir lieu.

. Regroupement des usagers

Le regroupement des usagers dans la file prêt, en fonction des segments qu'ils partagent, permet d'augmenter fortement la probabilité pour que les usagers présents en mémoire partagent des mêmes pages. On peut ainsi récupérer des pages et augmenter le nombre des usagers courants.

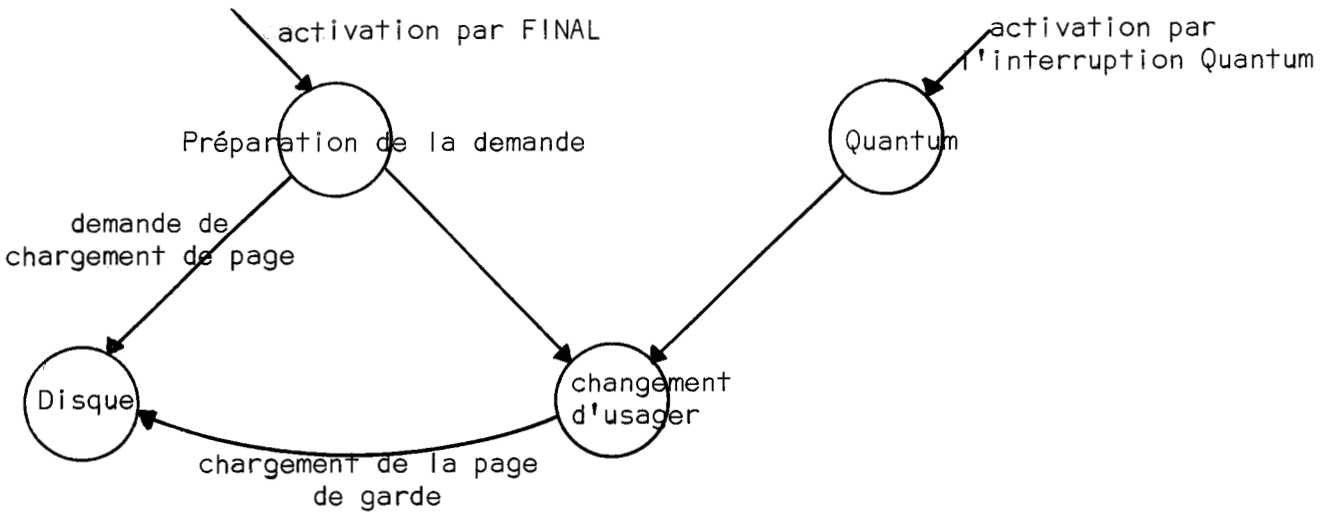
En pratique, il ne nous semble pas utile d'implanter cet algorithme car l'équivalence des usagers nous permet de compter sur un regroupement naturel de ceux-ci.

4.2.4 - Organisation de l'allocateur

L'allocateur peut être activé en deux occasions :

- le processus FINAL a lui-même été activé
- l'interruption de fin de quantum est arrivée.

Ceci conduit à organiser l'allocateur en trois processus, dont l'un est chargé de déterminer le changement d'usager. La figure suivante schématise cette organisation :




```
Processus Quantum : début X := usager élu ;
                    si dépassement du temps de résidence alors état (X) := prêt;
                                                           sinon état (X) := éli-
                                                                 gible ;
                    activation de "changement d'usager"
                    fin ;
```

```
Processus Préparer demande :

        début A : P(S)
                  X := usager élu ;
                  si pas de demandes de pages alors état (X) := oisif
                                                sinon
                     début si limite de catégorie atteinte alors état (X):=prêt
                                                          sinon préparation de
                                                                la demande
                     fin ;
                     activer "changement d'usager" ;
                     allera A
        fin ;

Processus "changement d'usager" :
        début A : P(S) ;
                  X := usager élu
                  si état (X) = oisif alors début libérer les pages allouées à X ;
                                                  supprimer sa réservation de pages;
                                                  mettre X dans la file des oisifs
                                           fin ;
                                  sinon si état (X)=prêt alors
                                                  début libérer les pages allouées
                                                                        à X ;
                                                        supprimer sa réservation ;
                                                        mettre X dans la file des
                                                                        prêts
                                                  fin ;
                  Assurer les changements usager oisif → prêt ;
                  Entrer un maximum d'usagers prêts dans la file des courants
                        et leur réserver de la place en mémoire ;

                  Commentaire : on recherche alors un usager à élire , soit Y
                  l'usager suivant dans la file ;

              B : si état (Y) ≠ éligible
                     alors début Y := successeur de Y ; allera B
                          fin ;
                     sinon début si page de garde présente alors
```

```
                    début état (Y) := élu ;
                          relancer horloge quantum ;
                          charger registres UC
                    fin ;
                                              sinon
                    début chercher adresse page de garde sur
                                                    disque ;
                          placer la file des processus prêts
                          de l'usager derrière son sémaphore
                          de défaut de page ;
                          lancer "Disque" ;
                          allera B
                    fin
              fin
        fin ;
```

4.3 - Allocation de pages en mémoire physique

4.3.1 - Réservation

La réservation doit précéder l'allocation. Elle est faite sur la base des catégories dès qu'un usager prêt est admis à l'état courant. En conséquence, si NDISP est le nombre de pages allouables en mémoire physique et NMAX le nombre de pages de la catégorie à laquelle appartient l'usager, il ne peut devenir courant qu'à la condition que NMAX ne soit pas supérieur à NDISP.

Kaiser propose que lorsqu'un usager ne peut passer courant parce que cette condition n'est pas respectée, l'on ne recherche pas un autre usager prêt. Ceci doit permettre de ne pas favoriser les usagers utilisant peu de mémoire au détriment des autres.

4.3.2 - Etat des pages physiques

Afin de faciliter la mise en place des mécanismes de traitement des accès aux pages de la mémoire virtuelle, nous avons implanté dans chaque descripteur de page virtuelle un bit de disponibilité et un bit d'altération. L'absence du bit de disponibilité n'indique pas nécessairement la non présence de la page virtuelle dans la mémoire physique, elle a pour but de provoquer un déroutement pour défaut de page qui permette au système de traiter l'accès à la page. Ainsi que nous le

verrons, on peut par ce procédé mettre systématiquement à jour l'état de la page. Le bit d'altération positionné à un, dès qu'une écriture a été faite dans la page, permet de définir dans quelles conditions la page peut être réallouée. On connaît donc ainsi l'état de chacune des pages virtuelles d'un usager. Cette connaissance permet de tenir à jour l'état des pages de la mémoire physique. On peut ainsi définir pour les pages physiques deux états selon que la page est intacte ou écrite. Une page intacte pourra être réutilisée immédiatement lorsque l'usager quittera la mémoire, au contraire une page écrite devra être vidée avant toute utilisation.

L'état d'une page physique peut encore être "en cours de transfert" ou non. Cet état doit être distingué car une page en transit ne peut être utilisée immédiatement.

Enfin une page peut être allouée ou non. Simultanément, elle peut être écrite ou intacte. On distingue donc parmi les pages non allouées les pages libres et les pages quasi-libres et on entretient deux listes. Par priorité lorsqu'une allocation doit être faite, on choisit une page dans la liste des pages libres, afin d'éviter un vidage. Ceci permet d'assurer une disponibilité plus rapide de la page demandée. Ces états peuvent être codés à l'aide de trois bits AL, A, T :

AL = 0 page non allouée A = 0 page intacte T = 0 page en non transit

AL = 1 page allouée A = 1 page écrite T = 1 page en transit.

En fait nous utiliserons non le bit AL mais un compteur de partage afin de mémoriser le nombre d'usagers utilisant la page.

4.3.3 - Tables utilisées pour la gestion des pages physiques

4.3.3.1 - table des pages physiques (TPP)

Elle contient un descripteur de quatre octets par page et on y accède par le numéro de la page physique. Le format des mots du Multi 20 impose pour que l'accès soit rapide de rechercher d'abord dans une table d'accès le déplacement par rapport à l'origine de la table. Chaque descripteur comprend quatre champs :

- les bits d'état de la page: A, T
- l'adresse d'implantation du contenu actuel de la page physique. Cette adresse occupe 17 bits et comprend :

. le numéro du disque (1 bit)

. le numéro de piste (1 octet)

. le numéro du premier secteur de la page de disque (1 octet). Elle permet de retrouver à qui est allouée cette page.

- un compteur de partage C (6 bits) destiné à permettre le partage des fichiers sans recopie dans la mémoire physique et le blocage des pages de garde des usagers

- un pointeur (6 bits) contenant un numéro de page physique et dont le but est de permettre l'organisation de listes fermées. On définit à l'aide de ce pointeur la liste des pages non allouées et utilisables directement (liste des pages libres : LPL) et la liste des pages utilisables après vidage sur disque (liste des pages quasi-libres : LPQL).

4.3.3.2 - tables d'accès aux fichiers partagés

Elles sont résidentes et sont organisées suivant le principe des tables de la page de garde des usagers :

- table d'accès : elle occupe 64 octets dont chacun indique un déplacement permettant d'atteindre un descripteur de fichier

- table des descripteurs de fichier : chaque descripteur a pour but de permettre l'accès à la table d'implantation et occupe deux octets

- tables d'implantation (TIFP) : il y a une table de ce type par fichier partagé ; chacun des descripteurs occupe trois octets :

. la dernière adresse d'implantation en mémoire physique (6 bits), c'est un numéro de page

. le numéro du disque contenant le fichier (1 bit)

. un bit d'utilisation

. l'adresse d'implantation sur disque (2 octets).

L'une des TIFP repère le fichier mémorisant le contenu des pages de garde des usagers.

4.3.3.3 - table des pages allouées (TPA)

La gestion des pages physiques nécessite que l'on puisse retrouver rapidement le numéro des pages qui ont été allouées à l'usager, on entretient dans ce but une TPA dans la page de garde de chacun des usagers résidents en mémoire. Le numéro de la page physique contenant la page de garde de l'usager est mémorisé par ailleurs. Cette table contient autant de descripteurs que le permet la catégorie la plus grande dans le système. Chacun d'eux occupe un octet contenant un numéro de page physique. Un compteur est associé à la table et fournit le nombre de pages allouées.

4.3.3.4 - table des pages demandées (TPD)

Cette table,implantée elle aussi dans la page de garde de l'usager, mémorise l'ensemble des demandes de chargement de pages de l'usager. Ayant décidé de ne pas utiliser le préchargement, cette table n'est utilisée qu'à la condition que l'usager soit en état courant ; c'est pourquoi elle est implantée dans sa page de garde.

La TPD possède autant de points d'entrée que de demandes, sa taille maximale est définie par la catégorie maximale. On y associe un compteur. Chaque descripteur occupe cinq octets découpés en trois champs :

- l'adresse d'implantation sur disque de la page qui doit être chargée (17 bits)
- l'adresse d'implantation sur disque du contenu actuel de la page physique (17 bits)
- le numéro de la page physique allouée.

Une adresse nulle indique que l'opération correspondante ne doit pas être effectuée.

4.3.4 - Accès aux fichiers

Deux types d'actions peuvent être demandés par un usager, la recopie d'une page de fichier interne dans un fichier externe ou le transfert du contenu d'un fichier externe dans un fichier interne.

. **Recopie d'une page de fichier interne dans un fichier externe :**

Pendant toute la durée de réalisation de cette action, il est nécessaire d'interdire toute écriture dans la page concernée car il s'agit d'une remise à jour du fichier. Faute de place, nous n'avons pu affecter une protection particulière à chaque page de segment, il est donc nécessaire d'en bloquer tous les accès. Tout processus tentant d'y accéder sera bloqué en attente derrière un sémaphore de l'usager. Cette situation devrait se produire rarement et ne pas perturber le fonctionnement du système.

. **Transfert d'un fichier externe** : Pendant toute la durée de cette action, tout accès à la page doit être interdit car son contenu n'est plus à jour. De plus, lorsque la page du segment associé au fichier est présente en mémoire, elle doit être restituée de sorte qu'une récupération ne puisse être tentée lors du déblocage. Dans ce cas, elle est rendue à la liste des pages libres. Afin d'éviter toute tentative de récupération, on doit donc détruire l'adresse physique présente dans la TPV. Cette opération est possible car les transferts entre fichier interne et fichier externe ne concernent que les fichiers de l'usager aussi il n'y a pas de partage de ces fichiers.

Le principe du blocage consiste à provoquer un déroutement de tous les processus qui tentent d'y accéder afin de pouvoir les bloquer derrière le sémaphore associé à la page. On distinguera ce déroutement à l'aide du bit d'altération. On imposera que toute libération de page (la page cesse d'être allouée à l'usager), lors du passage d'un usager de l'état courant à l'état oisif ou à l'état prêt, soit accompagnée d'une remise à zéro des bits de disponibilité et d'altération. Ceci n'aura donc pas lieu si la page est bloquée puisque la page n'est plus allouée.

Le blocage est laissé à l'initiative d'un processus particulier de l'usager, son processus d'échanges. Il est réalisé à l'aide d'une primitive. Afin de déterminer le sémaphore derrière lequel doivent être bloqués les processus tentant d'accéder à la page, on entretient à chaque blocage ou déblocage une liste implantée dans la page de garde de l'usager. Le blocage et le déblocage étant à la charge de l'usager, on est assuré qu'à chacune de ces opérations cette liste est présente en mémoire. La liste des pages bloquées (LPB) utilisée a une structure de liste fermée dont chaque élément occupe quatre champs :

nom du segment (5 bits)
numéro de la page (4 bits)
nom d'un sémaphore de l'usager (1 octet)
successeur (en fonction de la taille maximale de la liste)

Deux primitives sont utilisées :

Bloquer (s,p,S,k) : début commentaire : *s* = nom du segment
p = numéro de la page
S = nom d'un sémaphore
k = paramètre indiquant le type d'accès
k=1 pour un transfert de fichier externe à fichier interne ;

si *k*=1 alors début si d = présence alors restituer la page à la LPL ; descripteur dans la TPV = 0
fin ;
d:=0 ;
a:=1 ;
indicateur d'utilisation dans la TIF := 1 ;
ranger *s,p,S* dans la LPB de l'usager
fin ;

Débloquer (s,p) : début chercher l'élément (*s,p*) dans LPB ;
lire *S* ;
supprimer l'élément (*s,p*) de la liste ;
D(*S*)
fin ;

Remarque : On pourrait éviter le blocage pour les accès en lecture lors d'une recopie, si l'on utilisait pour chaque page une protection codée suivant deux bits au lieu d'un bit de disponibilité et un bit d'altération. C'est le cas dans le 10070. Cependant, cette méthode impose de distinguer deux types de protection pour chaque page, la protection effective et la protection mise en place pour provoquer des déroutements [16]. Cette protection doit être chargée pour détecter les défauts de page puis la première écriture. Elle impose une augmentation des formats des descripteurs et de plus provoque une augmentation des temps de traitement. Pour cette raison nous préférons utiliser le blocage en remarquant que ce genre d'accès est rare et correspondra la plupart du temps à un passage de l'usager à l'état oisif.

4.3.5 - Traitement des déroutements

Un déroutement est provoqué par l'absence du bit de disponibilité. Deux cas peuvent donc se présenter :

- il s'agit d'un blocage
- il s'agit d'un défaut de page

4.3.4.1 - Cas d'une page bloquée

Ainsi qu'il a été indiqué le bit d'altération doit être présent. S'il en est ainsi, le processus est rangé dans la file d'attente du sémaphore associé à la page.

4.3.4.2 - Cas d'un défaut de page

Le défaut de page ne signifie pas nécessairement que la page demandée n'est pas présente en mémoire, il doit être interprêté comme l'indication que la page demandée n'a pas fait l'objet d'une allocation. On pourrait penser allouer systématiquement une page libre à l'usager, ce serait augmenter l'occupation de la mémoire et le trafic de pages. On doit tenter de récupérer la page physique qui avait été allouée à l'usager lors d'un passage précédent de celui-ci à l'état courant ou, s'il s'agit d'un segment ayant pour support un fichier partagé, tenter de récupérer une page allouée à un autre usager.

a) La page demandée n'a jamais été l'objet d'une allocation et le segment a pour support un fichier interne de l'usager. Ces fichiers n'étant pas partageables, il est inutile de tenter une récupération.

b) La page demandée n'a jamais été l'objet d'une allocation et le segment a pour support un fichier partagé. La récupération peut être tentée, car le fichier peut être associé à un segment d'un autre usager et la page avoir été chargée. Pour le déterminer on se reporte à la TIF du fichier. Le descripteur adressé fournit le numéro d'une page réelle. On se reporte donc à la TPP où l'on trouve une adresse d'implantation sur disque. La comparaison de cette adresse avec celle que fournit la TIF indique si la page physique est récupérable.

c) La page demandée a déjà été l'objet d'une allocation et a pour support un fichier interne privé. On se reporte à la TPP et l'on compare les adresses d'implantation sur disque.

d) La page demandée a déjà été l'objet d'une allocation et a pour support un fichier partagé. On procède alors comme en b) car le numéro de la page réelle contenant la page du fichier est susceptible d'avoir été modifié à la suite d'un vidage et la TIF est remise à jour chaque fois qu'un usager y fait référence.

L'algorithme serait donc le suivant :

```
début commentaire:le microprogramme fournit (s,p) où s est le nom du segment et
                  p le numéro de la page virtuelle demandée ;
    Se reporter à la table des descripteurs de segments, soit f le nom du
    fichier qu'on y trouve ;
    Si f = fichier interne privé de l'usager alors
            début se reporter à la TPV, soit P le numéro de page physique qu'on
                                        y trouve ;
                 si P≠0 alors début se reporter à la TPP que l'on adresse avec
                                    P, soit A' l'adresse d'implantation qu'on
                                    y trouve ;
                                    si A=A' alors allera Récupération
                               fin ;
                               allera Allocation
            fin ;
                               sinon
            début se reporter à la TIF, on y trouve P et A ;
                 si P≠0 alors début se reporter à la TPP, on y trouve A' ;
                                    si A'=A alors allera Récupération
                               fin ;
                               allera Allocation
            fin ;
    Récupération : On y teste C, A, T ;
    Allocation : On alloue une page et on prépare la demande
fin ;
```

Une page ne peut pas toujours être récupérée. On doit le vérifier à l'aide des bits A et T et du compteur de partage. Nous appellerons A' l'indicateur d'altération fourni par le microprogramme lors du déroutement.

a) C>0, A=1, T=0 ou C>0, A=0, T=0. La page est présente en mémoire et est utilisée par un autre usager, elle est déjà récupérée. On incrèmente le compteur d'utilisation C de la page. Elle est allouée à l'usager demandeur et le nombre de pages qu'il peut encore recevoir est diminué de un. Le numéro de la page est rangé dans la TPA de l'usager. On modifie A' par l'opération A:=A' ou A.

b) C=0, A=1, T=0 ou C=0, A=0, T=0 la page appartient à la LPQL ou à la LPL. On la sort de la liste et on l'alloue à l'usager en incrémentant C de 1 et en rangeant le numéro de la page dans sa TPA. De même que précédemment A:=A' ou A.

c) C=0, A=1, T=1. Il s'agit d'une page de la LPQL en cours de vidage préventif, elle peut être récupérée. On laisse se terminer le transfert mais on alloue la page à l'usager. D'où : C:=1 ; A:=1 ; T=0.

d) C=1, A=1, T=1 ou C=1, A=0, T=1. Il s'agit d'une page allouée dont le chargement a été demandé. Deux cas peuvent se présenter :

- La demande a été faite par un autre usager. Elle concerne alors un fichier partagé. La récupération ne peut avoir lieu.

- La demande a été faite par un processus de l'usager. La récupération peut avoir lieu. On ne peut le savoir directement que si elle concerne un fichier privé de l'usager. Si elle concerne un fichier partagé, il faut rechercher la demande dans la TPD. Les déroutements pour défaut de page étant fréquents et cette recherche étant assez longue, nous ne la ferons pas.

Lorsque la récupération a échoué, une page de la LPQL ou de la LPL doit être allouée. Cette allocation se fait en priorité dans la LPL. Alors dans la TPD, le descripteur de la demande a son bit de demande de vidage préalable en zéro et l'adresse d'implantation sur disque du contenu actuel de la page physique n'est pas rangée. La page est sortie de la LPL. Les opérations sont les mêmes lorsqu'il s'agit d'une page de la LPQL, cependant une adresse de vidage est indiquée dans la TPD. Lorsque le contenu du fichier est sans signification (indicateur d'utilisation à zéro) l'adresse d'implantation sur disque du contenu de la page n'est pas rangée dans la TPD on indique ainsi qu'il n'y a pas de chargement à réaliser. Lorsqu'il s'agit d'une page de la LPL le processus peut être relancé. Dans tous les cas le descripteur de la page dans la TPP est mis à jour.

Utilisation du compteur de partage.

Celui-ci est systématiquement incrémenté lorsque la page est allouée et décrémenté lorsqu'elle est retirée à un usager. On évite ainsi de restituer une page à la LPL ou la LPQL alors qu'un usager l'utilise . En l'absence de ce compteur, le bit de disponibilité de la TPV utilisée par les usagers partageant la page ne pouvant pas être remis à jour sans une recherche compliquée, le retour de ces usagers à l'état élu risquerait de provoquer un accès à une page ne contenant plus l'information désirée. En particulier cet accès pourrait provoquer la destruction du code d'une procédure chargée pour un autre usager.

Chapitre VI

ORGANISATION DU SYSTEME FRONTAL

Nous n'en examinerons que les aspects généraux.

1- ORGANISATION GENERALE DES PROCESSUS D'UN USAGER

1.1 - Entrée d'un usager dans le système

1.1.1 - Ouverture d'une session

Nous avons vu comment étaient gérés les terminaux, ceci suppose que le module du superviseur d'entrées-sorties qui participe à cette gestion soit toujours en attente d'un caractère. Tout caractère reçu par ce module est accompagné d'un numéro, celui du terminal. C'est ce numéro qui représente le nom interne de l'usager dans le système. En conséquence, toute ouverture de session peut être détectée par la réception d'un caractère spécial appelé ici "attention" en provenance d'un terminal non actif. A cet effet, le processus de gestion des terminaux conversationnels gère une table possédant autant de points d'entrée qu'il y a de terminaux connectables. Cette table, appelée table des usagers (TU), permet d'en décrire les activités. Chacun de ses éléments comprend quatre champs occupant trois octets :

- état de l'usager (2 bits). Il s'agit là de l'état de l'usager vis-à-vis du module de gestion des téléimprimeurs. Les états ainsi représentés sont :

  . non connecté / actif
  . entrée / sortie

- état de référence a_1 a_2 (2 bits). Nous reviendrons sur cet état en (§ VI.1.4)

- adresse du tampon utilisé (2 octets)

- nombre de caractères dans le tampon (4 bits)

L'état "non connecté" du terminal ayant été détecté, le processus de gestion des terminaux active un processus appelé processus d'ouverture dont le rôle est d'admettre l'usager en session.

L'admission d'un usager en session suppose son acceptation par le système central. En effet, il est fréquent que l'on tente de réguler la charge d'un centre de calcul en agissant sur les conditions d'utilisation du système. Ceci est en général réalisé en définissant la part de la charge "travail de fond" par rapport à la charge "time sharing" et peut être défini en particulier par le nombre d'usagers conversationnels admissibles dans le système. On peut décider de porter connaissance de cet état au système frontal, cependant un dialogue entre les deux systèmes doit subsister lors de l'admission d'un usager par le système frontal. Ce dialogue a pour but d'éviter le conflit dans lequel le système frontal accepterait l'ouverture d'une session pendant que le système central entrerait dans une situation où il réduirait sa charge "time sharing". Un dialogue devant toujours avoir lieu, nous avons décidé que l'ouverture d'une session resterait toujours à l'initiative du système central. Ainsi à la réception du caractère "attention", le système central est alerté à l'aide de l'ordre "créer-usager n". Le retour de l'ordre "exécuter-usager zéro" active le processus d'ouverture. Ce processus est résident et n'a qu'un rôle réduit :

- admettre l'usager dans TU en positionnant le bit "actif"
- créer une page de garde pour l'usager. Cette opération consiste à lui allouer une page de LPL et à initialiser dans celle-ci les principales variables nécessaires à son utilisation
- créer le premier processus de l'usager appelé "analyseur de commandes" à l'aide des primitives *Ouvrir (f,n)* et *Associer (s,f)*.

1.1.2 - Fermeture d'une session

Le processus de Fermeture est résident. Afin de limiter son encombrement on limite son action. Il est activé par le processus analyseur de commandes et son rôle est de libérer toutes les pages qui sont encore allouées à l'usager. Elles peuvent donc être restituées à LPL à moins d'être partagées, ce sera éventuellement le cas pour les pages du processus analyseur de commandes.

Ces deux processus sont cycliques de façon à toujours être activables. Cependant, ils ne peuvent prendre en compte qu'un seul usager à la fois, ceci permet d'en simplifier le code.

1.2 - Le processus analyseur de commandes de l'usager (processus AC)

Lorsque l'usager est "entré" dans le système, le rôle du processus analyseur de commandes est analogue à celui du processus analyseur de cartes de commandes dans un système de type "batch", il sert d'interface entre l'usager et sa machine virtuelle.

On peut définir quatre états de référence de l'analyseur de commandes :

- état initial
- état courant
- état oisif
- état final.

1.2.1 - Etat final

C'est l'état dans lequel se trouve l'analyseur de commandes lors de son réveil par le processus d'ouverture. Lorsqu'il est dans cet état, son rôle consiste à gérer le dialogue terminal-système central au cours duquel l'usager s'identifie. L'identification de l'usager n'est pas assurée par le système frontal car elle ferait double emploi avec celle que doit réaliser le système central pour lui permettre l'accès à ses fichiers. L'usager n'est donc connu du système frontal que sous le numéro de son terminal.

1.2.2 - Etat courant

Le processus analyseur de commandes passe dans cet état dès la réception d'un message en provenance du terminal de l'usager. La commande contenue dans ce message détermine une phase de traitement et le langage de commandes qu'il manipule est nécessairement celui du système central ou un surensemble. On remarquera que Siris 7 permet à l'usager de cataloguer des commandes dans un fichier, ceci permet lors de la conception du système de définir un langage de commandes ponctuelles et d'autoriser l'utilisateur à créer lui-même un langage de commandes plus adaptées à ses travaux. Dès lors certaines commandes émises par l'usager ne peuvent être reconnues par le système frontal. On pourrait donc penser qu'il est nécessaire que toutes les commandes reçues soient systématiquement transmises au système central, celui-ci étant chargé de les analyser et éventuellement d'activer le système frontal. Cependant procéder ainsi doublerait dans la plupart des cas les transferts par le disque de système à système. Nous avons donc décidé de réaliser l'analyse des commandes au niveau du calculateur frontal. Les commandes d'appel d'un catalogue non connu seront envoyées au calculateur central qui définiera l'action à entreprendre. De cette manière,

nous pouvons assurer l'émission de messages corrects et de plus l'utilisation des fichiers partagés du système frontal permettra le catalogage de commandes communes à l'ensemble des usagers et destinées à faciliter l'utilisation du système. Dès lors à la réception d'une commande en provenance d'un terminal les actions entreprises par le processus analyseur de commandes sont les suivantes :

```
début A : Analyser syntaxiquement la commande ;
          si Erreur alors sortir un message d'erreur sur le terminal
                     sinon début si commande frontal
                                    alors début
                                  si commande cataloguée sur frontal
                                        alors début acquérir la première commande
                                                    du catalogue ;
                                                    allera A
                                                    fin ;
                                        sinon activer le processus demandé
                                        fin ;
                                    sinon début
                                  demander le transfert au système central ;
                                  attendre un message du système central
                                        fin
                          fin
fin ;
```

Lorsqu'une commande cataloguée est transmise au système central, celui-ci peut éventuellement renvoyer une commande au système frontal.

1.2.3 - Etat d'attente

Chaque fois qu'une phase a été lancée le processus analyseur de commandes est en attente d'une nouvelle activation. L'état d'attente dans lequel il se trouve est défini par le type de phase en cours. Trois états peuvent ainsi être définis :

- attente de fin d'entrée d'un message en provenance du terminal de l'usager
- attente d'un message en provenance du calculateur central
- attente de la fin d'exécution d'une phase d'édition de texte ou d'analyse.

1.2.4 - Etat final

Dans cet état, le processus analyseur de commandes détruit les processus et les sémaphores qui avaient été créés par lui ainsi que les segments et fichiers internes. Il active ensuite le processus de fermeture.

1.3- Organisation générale des processus d'un usager

Le descripteur d'un processus occupe 21 octets, alors que le descripteur associé à un sémaphore n'occupe que 6 octets. Supposons que les performances du système nous permettent d'envisager la connexion de cinquante terminaux et que chaque usager possède cinq processus simultanément alors l'occupation en mémoire par la table des processus serait de près de 6000octets. Nous devons donc limiter le nombre des processus créés chez chaque usager, de plus cette limitation doit être faite en tenant compte du maximum de terminaux que l'on désire atteindre. La taille de la zone mémoire allouée aux tables de processus et de sémaphores sera fonction du nombre d'usagers potentiels et de l'encombrement du système. Dix pages permettent cinq processus et douze sémaphores par usager lorsque les usagers potentiels sont au nombre de cinquante.

On peut pallier cette contrainte en définissant un segment par processus nécessaire et en créant et détruisant ces processus au fur et à mesure des besoins. La structure de l'ensemble des processus d'un usager peut alors être la suivante :

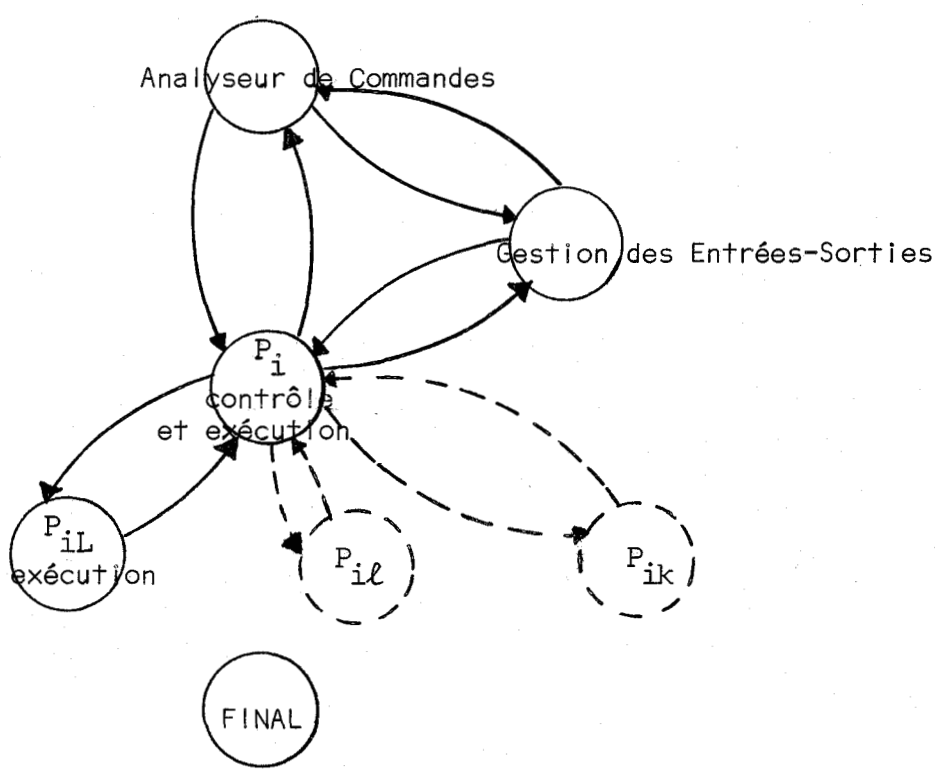


figure VI_1 : Organisation générale des processus de l'usager

où P_i pourrait être l'analyseur syntaxique et P_{iL} un processus de réorganisation du programme source de l'usager.

1.4 - Traitement des signaux "attention"

Chaque usager doit conserver la possibilité de revenir à l'un de ses états conversationnels soit au niveau du processus AC pour lui transmettre une nouvelle commande, soit au niveau de l'éditeur de texte par exemple pour lui transmettre une nouvelle sous-commande. C'est le rôle du caractère "attention" de permettre cette intervention en provoquant l'arrêt du traitement en cours. On définit deux types d'action du caractère "attention", sa réception par le système frontal pouvant avoir lieu dans deux conditions :

a) Une phase est en cours d'exécution sur le calculateur frontal. Le processus AC est donc bloqué derrière son sémaphore privé. Le processus de contrôle peut ne pas être actif, une sous-phase étant en cours d'exécution (édition de texte par exemple). Le traitement en cours pouvant ne pas être interruptible sans risque pour les fichiers internes de l'usager, la décision d'arrêter le traitement doit appartenir au processus en cours d'exécution. Pour le permettre on positionne un indicateur que les processus actifs peuvent tester en des points interruptibles du traitement qu'ils exécutent. Si l'arrêt du traitement est décidé, le processus de contrôle de phase repasse dans son état initial et est donc prêt à recevoir une nouvelle sous-commande qui peut éventuellement être une commande d'abandon.

b) Une phase est en cours d'exécution sur le calculateur central. Deux cas peuvent se présenter :

- Le processus AC a transmis une commande au système central, auquel cas il est en attente derrière son sémaphore privé.

- Le processus AC a activé un processus qui a émis un message vers le calculateur central dans le but d'activer un processus s'exécutant sur ce dernier. Le processus AC est donc bloqué derrière son sémaphore privé et le processus qu'il avait activé est en attente d'un message en provenance du calculateur central.

Il est a priori impossible dans ce dernier cas de tester l'indicateur. De plus il est impossible de déterminer lequel des processus est concerné par l'appel "attention". Nous avons donc décidé que le processus de gestion des ter-

minaux devait dans ce cas assurer le réveil du processus AC. Un mécanisme doit alors être mis en place pour permettre au processus de gestion des terminaux de distinguer les conditions a et b. Une fois réveillé, le processus AC doit déterminer l'origine de son activation, s'il s'agit de l'appel "attention" il en assure le transfert au calculateur central. Nous utilisons pour réaliser ce mécanisme les bits a_1 a_2 de l'état de référence de l'usager.

- $a_1 = 0$ le caractère "attention" n'a pas été reçu. Dès qu'il est émis par l'usager, le processus de gestion des terminaux le positionne en "un".

- $a_1 = 1$ le caractère "attention" a été reçu. Les conditions a et b sont distinguées à l'aide du bit a_2.

- $a_2 = 0$ le processus de gestion des terminaux n'entreprend aucune action.

- $a_2 = 1$ lorsque le caractère "attention" est reçu, le processus de gestion des terminaux positionne a_1 et active le processus AC.

L'activation du processus AC doit être réalisée à l'aide de la primitive *D*. En effet, un nouveau caractère "attention" peut être émis alors que le processus n'a pas encore réinitialisé les bits a_1 a_2, l'utilisation de la primitive *V* l'autoriserait à s'exécuter une nouvelle fois alors qu'il aurait entre-temps réinitialisé a_1 a_2. Comme le test de a_1 a_2 lui permet de déterminer qu'il s'agit d'un appel de l'usager, en leur absence le processus AC passerait en attente d'un message occasionnant ainsi un blocage de la machine de l'usager. D'autre part, lorsque les bits a_1 a_2 sont déjà positionnés, le réveil du processus AC doit néanmoins être tenté car il peut être en cours d'activité.

Enfin, lorsqu'une opération d'entrée-sortie est en cours sur le terminal, à la réception du caractère "attention", celle-ci est abandonnée.

L'accès à l'état a_1 a_2 de l'usager est réalisé à l'aide des primitives suivantes :

- *Lire* : permet à un processus de l'usager de lire a_1 a_2
- *Annuler* : remet à zéro les bits a_1 a_2
- *Autoriser* : remet à zéro le bit a_1 et positionne a_2 en "un".

2- REALISATION DES ENTREES-SORTIES

Celles-ci peuvent être complexes, un exemple suffit à le montrer. Supposons qu'un usager lance un dialogue avec le calculateur central, les états qu'il prend sont les suivants :

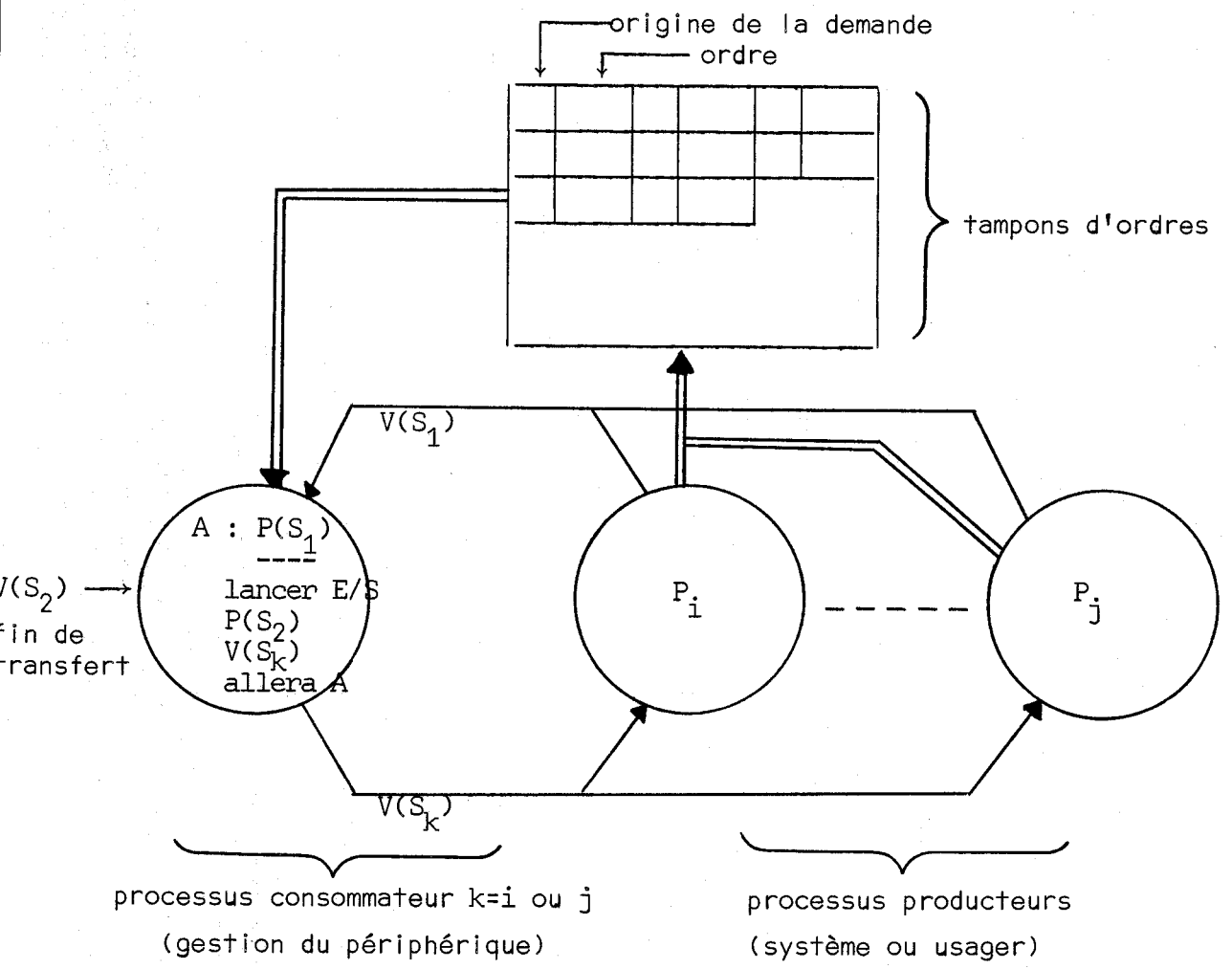
1 Demande de dépôt d'un message dans la zone de dialogue du disque
2 Attente de la fin du transfert
3 Demande d'envoi d'un ordre au calculateur central
4 Attente d'un ordre en provenance du calculateur central
5 Demande de lecture du message déposé par le calculateur central dans le fichier de dialogue
6 Attente de la fin du transfert.

Tout échange avec le calculateur central doit impérativement se faire selon cette séquence. On conçoit que la mise en place d'un superviseur d'entrées-sorties capable de gérer des dialogues de ce type serait difficile. Nous préférons reporter ce superviseur dans l'espace de l'usager en remarquant que chaque usager dispose virtuellement d'une liaison avec le calculateur central puisqu'il dispose d'un article dans le fichier de dialogue et que les échanges d'ordres sur le coupleur de la liaison intercalculateurs sont repérés par le numéro de l'usager. Cette méthode nous permettra de limiter l'encombrement de la mémoire en ne rendant résidents que les processus système de gestion des périphériques. On notera que la liaison intercalculateurs nécessite deux processus car la voie d'entrée d'ordres est distincte de la voie de sortie, ceux-ci pourront éventuellement utiliser les mêmes tampons d'ordre. Le dialogue entre processus de l'usager et processus du système se réalise simplement par l'utilisation des primitives de communication définies au chapitre IV.

A cet effet, les pages "zéro" et "un" d'un segment de l'usager créé lors de son admission sont rendues résidentes. Seul le pouvoir du processus superviseur permet d'y accéder ; pour le contrôler il suffit d'inclure dans la primitive *Unir (n,S)* une comparaison entre le pouvoir du processus et la protection attribuée au segment. Les pages physiques ainsi allouées sont partagées par tous les usagers et le système. Elles contiennent tous les tampons de communication entre usagers et système et n'appartiennent jamais à LPL.

L'organisation des processus du système coopérant à la gestion des périphériques peut être alors très simple. Elle utilise le schéma classique du producteur-consommateur. En pratique, l'utilisation des primitives de communication

permet de pallier la difficulté qui consiste pour un processus de l'usager à activer un processus du système. De plus elle résout le problème de l'exclusion mutuelle lors de l'utilisation des tampons.



En pratique, si la gestion du périphérique peut être simple pour les périphériques uniques (disque par exemple), elle devient assez vite complexe lorsque le nombre de périphériques d'un même type augmente. Nous en donnerons le principe à propos des téléimprimeurs.

Réalisation des entrées-sorties conversationnelles

Elles sont gérées à l'aide de tampons de messages alloués aux usagers au fur et à mesure de leurs besoins à l'aide des primitives d'utilisation des tampons. Du point de vue de l'usager, l'opération est simple à gérer et consiste à déposer un ordre et un message pour le processus de gestion des téléimprimeurs puis à l'activer. Elle ne sera pas réalisée par le superviseur de l'usager mais par son processus AC de sorte qu'il conserve le contrôle des messages reçus. Tout ordre devant

pouvoir être reçu, nous devrons utiliser trois processus du système :

- Le processus de gestion des téléimprimeurs dont le rôle est de recevoir l'ordre puis de lancer l'opération. Le lancement de l'entrée-sortie consiste à placer le premier caractère dans le tampon caractère suivant utilisé en sortie par le microprogramme s'il s'agit d'une sortie, ou de placer le numéro de l'usager dans la pile LIFO utilisée par le microprogramme lorsqu'il s'agit d'une entrée.

- Le processus d'acquisition et de décodage des caractères reçus.

- Le processus d'envoi et de codage des caractères à émettre.

Ceci suppose l'utilisation d'une table des usagers dans laquelle sont rangés, par le processus de gestion, l'adresse du tampon utilisé et le nombre de caractères qu'il contient. En fin de transfert, l'usager est réactivé par le processus de réception ou le processus d'envoi.

Les transferts sur téléimprimeurs étant lents, on peut s'attendre à ce que l'usager soit oisif pendant toute la durée du transfert. Ceci signifie que lors de l'activation de son processus AC un chargement de la mémoire sera nécessaire. Celui-ci portera au moins sur trois pages :

- la page de garde
- la page destinée à recevoir le message
- une page contenant le code du processus AC.

Il importe donc a priori que les tampons utilisés soient de taille suffisamment importante pour que l'on n'influence pas ainsi le trafic de pages. Ceci est d'autant plus vrai si l'on envisage l'utilisation de terminaux de sortie rapides. Cependant la taille des tampons doit également être telle que l'on puisse en prévoir un grand nombre de sorte qu'un usager n'ait jamais à attendre la libération problématique de l'un d'eux. En effet, supposons qu'un usager US n'ait pas pu obtenir l'allocation d'un tampon. Dès la fin d'un transfert l'usager destinataire voit son processus AC réveillé, et celui-ci provoque la recopie du tampon dans sa mémoire virtuelle. Supposons que l'usager utilise à nouveau ce tampon pour réaliser une entrée-sortie, l'usager US précédent reste bloqué. Supposons qu'il libère le tampon, l'usager US en attente est libéré mais ne disposant pas de l'unité centrale il ne peut participer à l'allocation d'un tampon. Comme, vraisemblablement il n'est pas présent en mémoire mais qu'il vient de passer à l'état "prêt", un chargement va être

lancé dès l'activation du processus de changement. Si l'usager qui a libéré le tampon, ou tout autre usager résident, exécute une entrée-sortie, l'usager US sera à nouveau bloqué dès sa demande d'allocation. Trois pages au minimum auront été chargées en mémoire inutilement et d'autres usagers auront peut être attendu en vain.

Ceci conduirait à réserver un tampon par usager et à choisir pour celui-ci une taille voisine de celle d'un message de taille maximale. En supposant que nous envisagions la connexion de cinquante terminaux conversationnels, si nous utilisons des tampons de 72 octets, leur occupation de la mémoire serait de 3600 octets. En fait ce type d'échanges étant l'une des occupations essentielles du système frontal et les transferts de longueur importante étant assez peu fréquents, nous pensons que quarante tampons de 38 octets suffiront (en fait 120 tampons de 14 octets - voir chapitre IV 4.1).

Nous avons essayé d'imaginer un procédé qui profiterait de la faible taille des secteurs du disque pour réaliser le chargement (ou le rangement) direct des tampons depuis (dans) celui-ci. Cette méthode conduit à mettre en place dans l'espace de l'usager un processus spécialisé réalisant le blocage sur disque de la page utilisée et nécessite quatre processus de gestion des téléimprimeurs :

- un processus de contrôle chargé d'initialiser les processus de réception ou d'envoi
- un processus d'alimentation des tampons depuis le disque
- un processus de recopie du contenu des tampons sur le disque
- un processus d'acquisition des ordres de l'usager et de recherche de l'adresse d'implantation sur disque. Ce processus est nécessaire car la recherche d'adresse impliquant la présence de la page de garde de l'usager en mémoire, le processus qui en a la charge doit être prêt à la réaliser dès la réception de l'ordre de l'usager. De plus, ce processus doit être plus prioritaire que le processus de changement d'usager.

Cette méthode n'est donc pas souhaitable car les entrées-sorties conversationnelles sont alors délicates à gérer et elle provoque un encombrement supplémentaire de la mémoire sans qu'il soit possible de déterminer si le trafic de pages est ainsi diminué. Cependant, à moins de limiter leur nombre, nous serions obligés de l'utiliser si nous envisagions la connexion de terminaux rapides.

3- ACCES AUX FICHIERS DE L'USAGER

3.1 - Principe

La mémoire virtuelle du calculateur frontal ne permet à l'usager que d'accéder à ses fichiers internes. L'espace qui leur est réservé sur disque ne permet pas d'y stocker les fichiers externes auxquels l'usager veut accéder, aussi une recopie totale de ceux-ci, basée sur le fait que le disque étant partagé les accès seraient facilités, ne peut être envisagée. Nous devons donc amener par partie les fichiers externes de l'usager au fur et à mesure des besoins. Dès lors la propriété de localité étant applicable aux traitements de fichiers, il n'est pas nécessaire de disposer d'un espace important sur le disque partagé. C'est dans cet esprit que nous avons limité à seize pages la taille des segments et des fichiers internes.

On peut de plus remarquer que l'orientation des applications du système frontal étant essentiellement le "time sharing", la taille des fichiers traités sera très souvent limitée et certainement très inférieure à celle que permet la segmentation. Ainsi dans la plupart des cas, le système central pourra accéder sur le disque partagé à la totalité du fichier traité. Nous pourrions donc en fait limiter la taille des fichiers traités. Cependant, il nous a paru intéressant d'étendre les possibilités de traitement du système frontal à l'édition de texte en général ; aussi, lorsqu'il s'agira d'un appel à l'éditeur, nous utiliserons une méthode de chargement dynamique du disque partagé dont la gestion sera à sa charge. Eventuellement, on pourra à cet effet définir un processus spécialisé jouant le rôle du SGF d'un système plus élaboré. Cette organisation est d'autant plus aisée à mettre en place que dans le cas d'une édition de texte le calculateur central n'a pas à accéder à l'information que contiennent les fichiers internes.

Nous distinguerons deux cas :

- Edition d'un fichier destiné à la compilation : la taille du fichier est limitée, cette limite étant inférieure à seize pages et fixée en fonction des possibilités offertes par la taille du disque partagé comparativement au nombre d'usagers potentiels. La gestion du fichier courant de l'usager est alors simple si sa structure est imposée.

- Edition d'un fichier quelconque : la taille du fichier ne peut être limitée, celui-ci est donc amené par parties égales à une page dans un fichier interne de l'usager. Ce problème n'a pas été abordé, il nécessitera l'étude détaillée

des méthodes d'accès de Siris 7. Dans une première version, nous nous limiterons à l'édition des fichiers destinés à la compilation. Tout appel de l'éditeur de texte ou d'une fonction non réalisée par le système frontal provoquera son utilisation en mode message. Ceci se traduira par la création d'un processus chargé de ne réaliser que des transferts entre terminal et système central ou inversement, le calculateur frontal se comportant alors comme un concentrateur.

Dans les deux cas d'utilisation il se pose le problème d'assurer au système central la connaissance de l'implantation sur le disque partagé du fichier interne utilisé.

3.2 - Accès aux fichiers internes

L'accès par le système central à la page de garde d'un usager est aisé car elle appartient à un fichier interne du système et peut y être adressée par le numéro du terminal de l'usager. Si l'usager a nommé le fichier interne utilisé, le système central peut alors y trouver son implantation. Deux contraintes doivent être respectées pour le permettre :

- Il importe que l'image sur le disque de la page de garde de l'usager reflète l'état qu'elle possède en mémoire physique, c'est-à-dire que cette dernière n'ait pas été altérée. Il suffirait que la TIF du fichier n'ait pas été modifiée mais ceci ne peut être déterminé aisément. Il doit en être de même pour les pages de fichier qu'utilise le système central.

- L'organisation des fichiers externes de l'usager doit pouvoir se superposer à celle des fichiers internes définis par le système frontal. On utilisera à cet effet la caractéristique des fichiers externes gérés par Siris 7 dans la configuration choisie pour la machine 10070.

3.2.1 - Image d'un fichier externe sur un fichier interne

Sous Siris 7, les fichiers manipulés par le SGF sont de trois types :

- Fichiers d'organisation consécutive
- Fichiers d'organisation séquentielle indexée
- Fichiers d'organisation partitionnée.

Ce dernier type de fichiers étant réservé au stockage des programmes, nous n'aurons pas à l'envisager. Dans la première version du système, nous imposerons que les fichiers manipulés par les usagers du "time sharing" soient de type séquentiel

indexé. Cette organisation permet une réalisation plus souple de l'édition de texte et est compatible avec la structure imposée par les compilateurs "time sharing" sous Siris 7. Afin d'assurer la compatibilité entre fichiers externes et fichiers internes nous imposerons aux fichiers utilisés ainsi d'être structurés en blocs de 1024 octets. Nous assurerons ainsi une superposition triple : blocs de fichiers externes - pages de fichiers internes - pages de segments qui autorisera une grande souplesse d'utilisation et des performances élevées. La première page de ces fichiers internes sera réservée à la mémorisation de leur bloc d'indexs. L'utilisation de cette méthode limitera les modifications nécessaires au niveau du système central et simplifiera la réalisation du système frontal en permettant des dialogues directs entre les usagers et le système central sans qu'il soit nécessaire d'assurer une transformation des couples (nom de segment ou de fichier, numéro de page) en une adresse d'implantation sur disque.

3.2.2 - Réalisation des échanges de pages de fichier entre systèmes

Nous avons déjà remarqué que le blocage d'une page était nécessaire lors de la transmission d'une page de fichier interne vers un fichier externe ou inversément, il en est de même lorsque l'ordre est transmis au système central d'utiliser le fichier interne de l'usager. Si l'on ne procédait pas ainsi il se pourrait qu'un processus encore actif vienne altérer son contenu et qu'à la faveur du passage de l'usager à l'état oisif ou prêt, un vidage soit réalisé, rendant non significatif le contenu de la page adressée par le système central. Une mise à jour des pages de fichier doit de plus être réalisée, cependant cette éventualité ne peut être déterminée que par l'usager. Laisser cette possibilité à l'usager implique qu'un contrôle strict soit réalisé, aussi cette opération n'est-elle possible qu'à travers une primitive. Celle-ci s'écrit :

Mettre à jour (s,p,S) : début commentaire : *s* est le nom d'un segment, *p* le numéro d'une page de ce segment. *S* est le nom d'un sémaphore derrière lequel le processus *P* qui exécute cette primitive attendra la fin de la mise à jour. On suppose que la page *(s,p)* a été bloquée ;

si page de garde altérée alors début simuler *P(S)* ;
A:=0 ;
lancer le transfert sur disque
fin ;

si *(s,p)* présente et écrite alors début simuler *P(S)* ;
A:=0 ;
lancer le
transfert
fin ;
commentaire : le processus *P* sera réveillé à la fin des deux transferts par le processus de gestion du disque
fin ;

Une séquence de transfert vers le disque partagé s'écrit alors :

Bloquer (s,p,S_1) ;
Mettre à jour (s,p,S_2) ;
dialogue avec le calculateur central ;
attente d'un réveil par le calculateur central ;
Débloquer (s,p) ;

On remarquera que cette opération ne peut être réalisée par un processus quelconque de l'usager bien qu'il s'agisse de processus de service car une mauvaise programmation peut avoir des effets désastreux :

exemple A . Le processus *P* exécutant la séquence décrite précédemment déclare le même sémaphore *S* dans les primitives *Bloquer (s,p,S)* et *Mettre à jour (s,p,S)*. Alors, lorsque le processus de gestion du disque exécute *V(S)* il se peut qu'il réveille non pas *P* mais un processus *P'* de priorité supérieure bloqué à la suite d'une tentative d'accès à la page *p*. Non seulement le contenu de *P* peut être altéré par *P'* mais aussi, le processus *P* restant bloqué, il ne sera plus possible de débloquer la page *p*.

exemple B . Le processus *P* après avoir bloqué la page *p* tente d'y accéder : la page *p* n'est plus accessible à aucun processus et *P* ne pourra plus être activé.

L'accès aux fichiers internes de l'usager devra donc être géré par un processus particulier d'écriture soignée (le superviseur d'entrées-sorties de l'usager).

SIMULATION

Chapitre VII

EVALUATION DES PERFORMANCES DU SYSTEME FRONTAL

Nous avons déjà abordé le problème de l'environnement "hardware" du système, les problèmes qu'il pose sont classiques et la réalisation sera très simplifiée du côté du calculateur frontal par le fait qu'il est microprogrammable. Nous nous attacherons ici à justifier plus généralement l'utilisation du système frontal.

Seule une simulation complétée par une étude statistique du comportement des usagers pouvait nous y aider. Celle-ci devrait normalement tenir compte de la présence du calculateur central, cependant l'interactivité très forte qui peut exister entre les deux systèmes et la nécessité d'en simuler des points précis nous en ont empêché. Nous avons donc considéré le système frontal comme un élément indépendant réalisant des fonctions précises et le modèle a été élaboré dans ce sens.

1 - PRINCIPES

Tout système "time sharing" doit être considéré comme un système fermé en ce sens qu'il y a recyclage des demandes de traitements. En fait, deux cycles doivent être considérés : d'une part le cycle des travaux dû au fait que l'on attribue à chaque usager un quantum d'unité centrale et un temps maximum de résidence, d'autre part le cycle des commandes émises par les usagers (figure VII_2). Plus la réponse du système est lente, plus la réaction de l'usager est retardée ; le taux d'entrée des demandes s'adapte donc aux performances du système. Aussi, à la limite un système "time sharing" est autorégulateur. En fait, le temps qui s'écoule entre la fin du traitement d'une demande de l'utilisateur par le système et l'introduction d'une nouvelle commande par le même utilisateur (temps de réflexion) est élevé par rapport au temps de traitement. La figure VII_1 [17] montre que le temps de calcul est en moyenne de une seconde alors que le temps de réflexion moyen d'un usager est généralement considéré comme étant voisin de trente secondes, ce que nous avons pu vérifier.

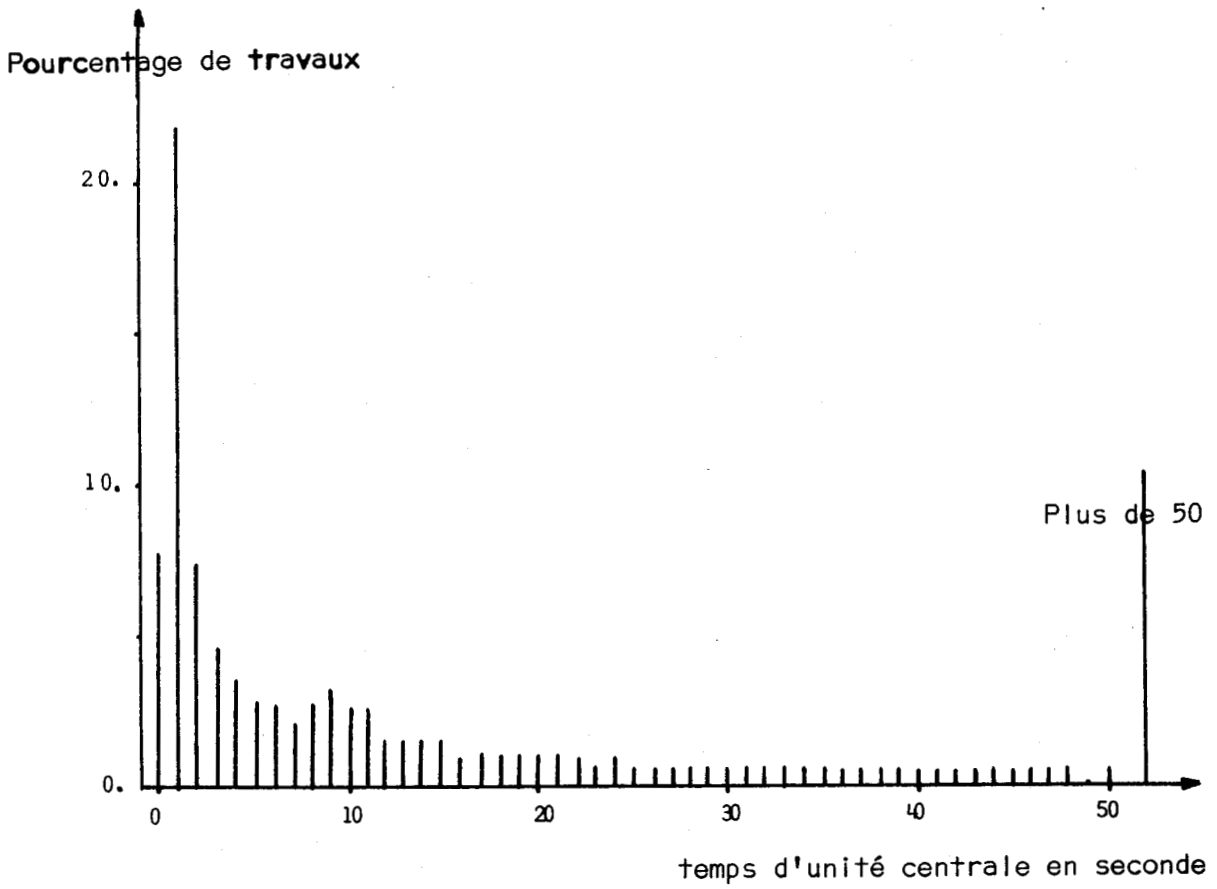


figure VII_1 : Utilisation de l'unité centrale dans le cas de travaux interactifs.

Ceci nous montre que la simulation doit porter sur un système en boucle ouverte dans lequel le taux d'entrée des demandes est celui d'un système non saturé. Ce n'est qu'ainsi que l'on peut déterminer les limites d'utilisation du système simulé. Nous avons donc adopté le schéma de la figure VII_3 où la boucle A de la figure VII_2 a été "oubliée" et nous avons étudié globalement le système, c'est-à-dire que nous avons considéré le taux d'entrée des demandes comme une fonction linéaire du nombre de terminaux.

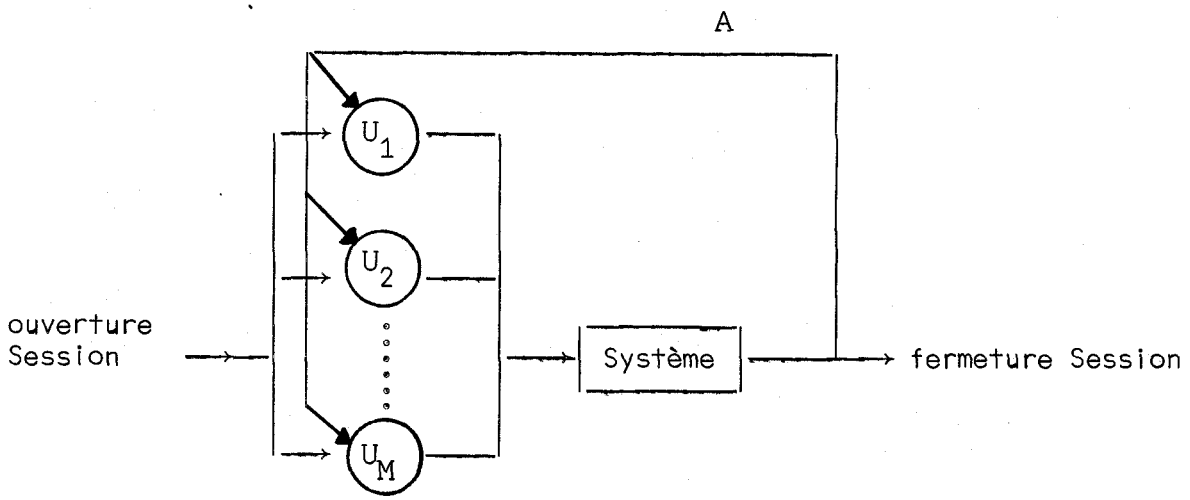


figure VII_2 : Modèle de principe des systèmes time sharing.

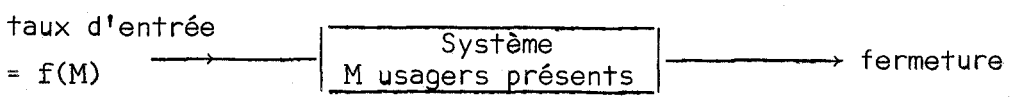


figure VII_3 : Modèle simplifié.

1.1 - Choix du modèle

Puisqu'il ne s'agissait que de justifier le rôle du calculateur frontal, nous avons choisi de simuler un système très classique de fonctions précises et d'organisation simple, réalisant :

- les entrées-sorties conversationnelles
- l'analyse syntaxique des lignes de programme au fur et à mesure de leur émission par les usagers
- la remise en ordre des fichiers sources des usagers.

A chacune de ces fonctions ont été associés dans le simulateur une file d'attente et un générateur de demandes. Deux contraintes nous ont limité dans la définition des fonctions déterminant les temps de traitement correspondants :

- Il ne nous était pas possible d'accéder à un système "time sharing" pour y mettre en place un programme chargé d'effectuer des mesures en des points précis du système.

- Les points qui nous intéressaient étaient trop précis pour que nous puissions trouver dans la littérature des résultats utilisables.

Nous avons donc décidé de simuler chacune des fonctions réalisées par le système frontal à partir de mesures faites indépendamment d'un environnement "time sharing". Cependant cette décision nous a conduit à limiter nos objectifs.

Nous nous sommes placé dans le cadre d'un système permettant l'utilisation d'un seul langage source. De plus, nous avons considéré qu'il était de classe I afin d'éliminer les interactions avec le système central. Enfin, puisqu'il ne s'agissait que de justifier le rôle du calculateur frontal et non son organisation, nous avons simplifié le système simulé. Ainsi le modèle peut être schématisé par la figure VII_3.

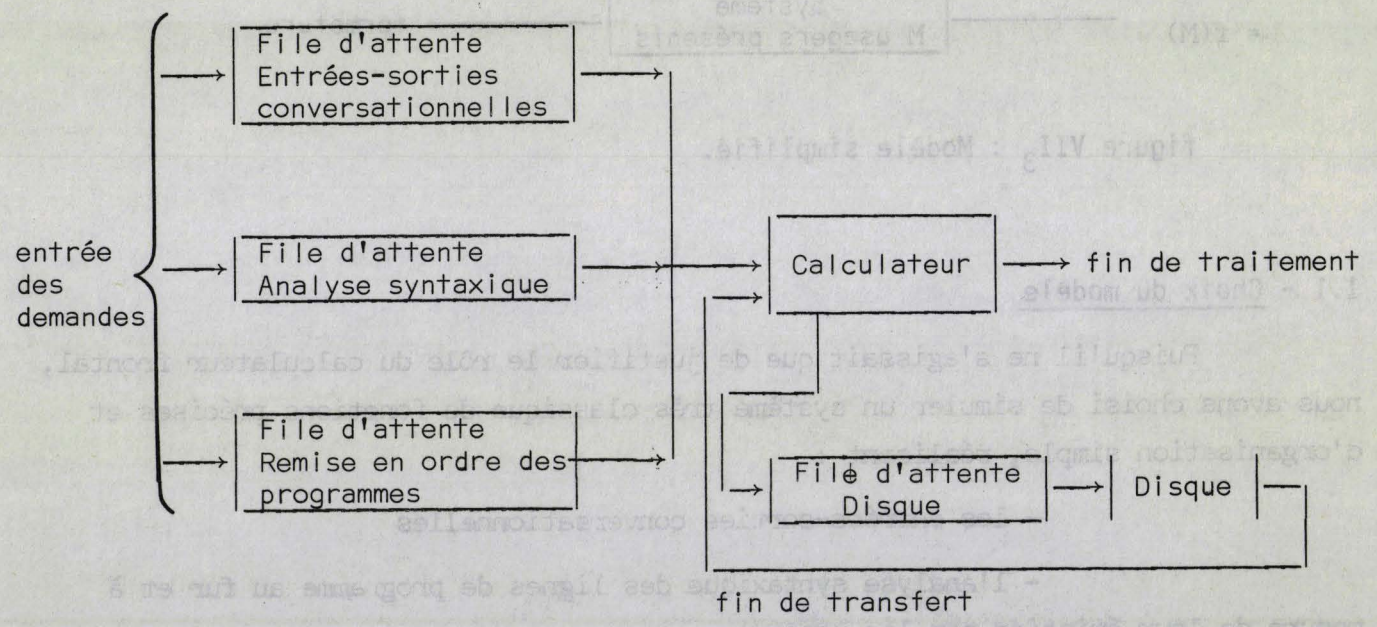


figure VII_4 : Organisation globale du modèle utilisé.

En particulier, nous n'avons pas introduit un recyclage des travaux car les mesures que nous avons faites pour déterminer les fonctions nous ont montré que les temps de traitement étaient réduits et ne justifiaient pas l'introduction d'un quantum d'unité centrale et d'un temps maximum de résidence en mémoire.

Ainsi, les traitements demandés sont effectués chacun à leur tour et les informations nécessaires à la réalisation d'un traitement demandé par un usager sont considérées comme se trouvant sur le disque. Dès qu'une demande est prise en compte, on provoque le chargement en mémoire des données et éventuellement du programme utilisé.

1.2 - Mesures préalables

Dans la mesure où l'on s'intéresse au comportement global du système, le temps de traitement nécessaire à la réalisation de l'analyse syntaxique peut être considéré comme une fonction linéaire de la longueur des lignes de programme introduites par l'usager. Pour déterminer cette fonction, nous avons utilisé l'analyseur syntaxique Algol qu'un groupe d'étudiants avait réalisé pour le Multi 8 du laboratoire. Le choix de cet analyseur fournit une borne supérieure des temps de traitements que l'on obtiendra dans la réalité. Les mesures nous ont montré que grossièrement le temps de traitement est de l'ordre de deux millisecondes par caractère. Par ailleurs, nous avons déterminé la fonction densité de probabilité des longueurs d'une instruction. Cette mesure a été faite sur un ensemble de programmes Fortran de tous types et portant sur plus de dix mille échantillons. Le choix de ce langage se justifie par le fait qu'il est l'un des plus couramment utilisé dans les systèmes conversationnels et qu'il correspond sous Siris 7 à un compilateur off line. La figure VII_5 donne la courbe obtenue à la suite de ces mesures.

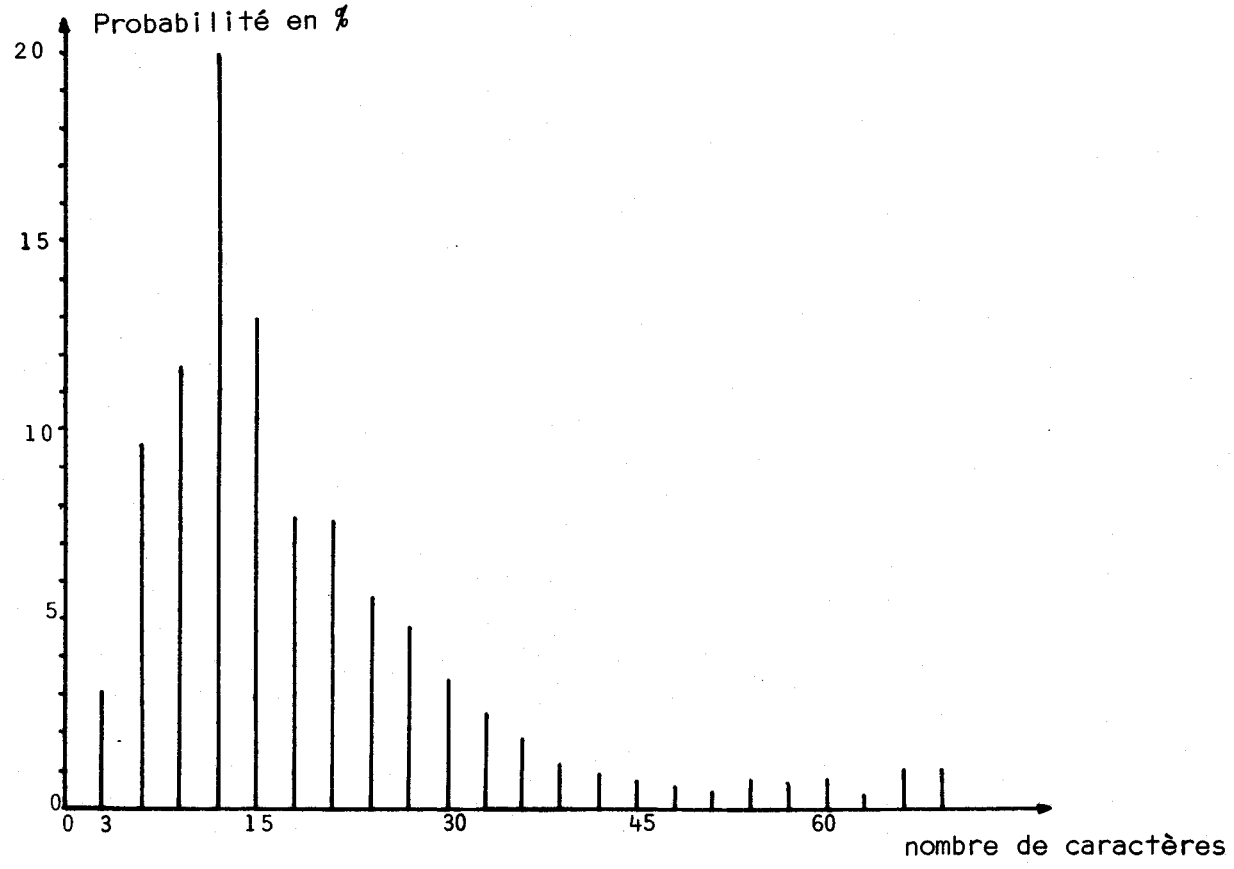
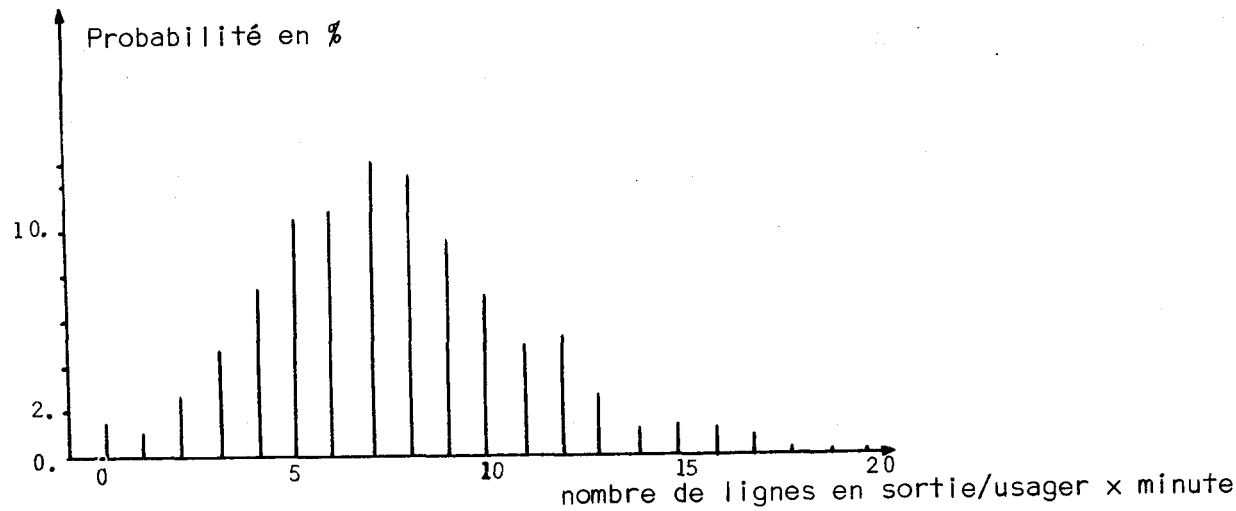
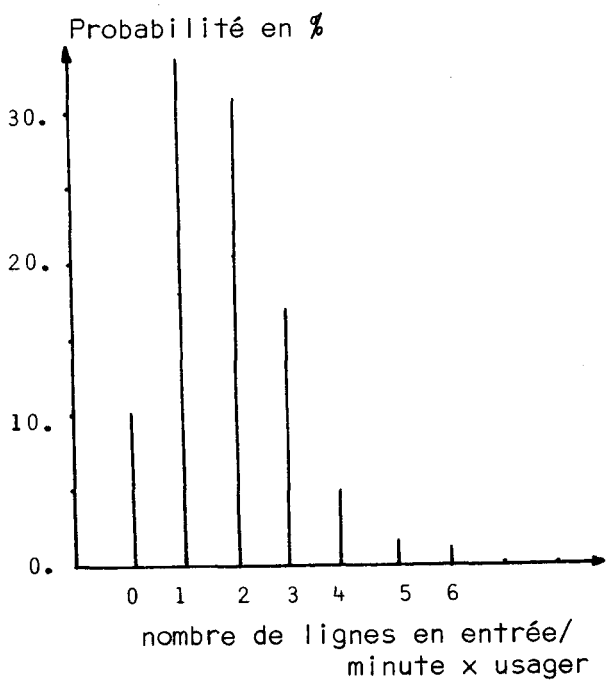


figure VII_5 : Longueur d'une instruction Fortran.

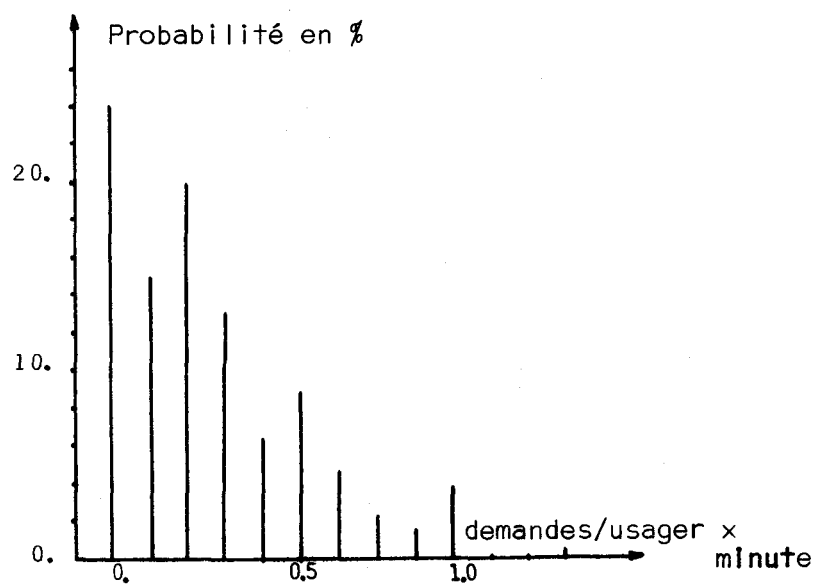
Afin de simuler les demandes des usagers, nous avons déterminé les fonctions densité de probabilité du nombre de demandes. Ces fonctions ont été définies à l'aide de mesures faites par son constructeur sur un système Time Sharing de petite taille. Les mesures que nous avons utilisées proviennent des sorties journalières faites au Centre de Calcul et fournissent le nombre d'entrées par minute en différents points du système ainsi que le nombre de terminaux en activité pendant cette période. Les données étant difficiles à exploiter nous nous sommes limité à l'utilisation des mesures faites sur une journée que nous avons choisie moyennement chargée (11 terminaux connectés au maximum, pour un maximum théorique de 32). De plus, ces mesures étant globales nous avons considéré que le nombre de demandes de traitement était proportionnel au nombre d'usagers courants. Les courbes obtenues sont représentées en VII_6.



a) demandes d'émission vers le terminal



b) demandes d'analyse syntaxique



c) demandes de Tri

figure VII_6 : Demandes de traitement des usagers.

2 - SIMULATION DES FONCTIONS REALISEES PAR LE SYSTEME FRONTAL

Nous avons simulé les demandes de traitement par tirage au hasard à l'aide de tableaux mémorisant les fonctions densité de probabilité décrites précédemment. Un premier tirage est effectué toutes les minutes, cette période nous étant imposée par les mesures en notre possession ; il fournit le nombre de demandes. Un second tirage aléatoire permet de déterminer la date d'arrivée de la demande ; nous nous sommes limité à un pas de une seconde, en considérant que l'arrivée de la commande coïncide avec la frappe du dernier caractère du message et que l'erreur faite sur la détermination de la date de frappe de celui-ci est de une seconde. La limitation de ce pas mettant d'avantage en évidence l'encombrement des files d'attente, nous avons considéré qu'il n'y avait pas d'inconvénient à procéder de cette manière.

2.1 - Analyseur syntaxique

Les demandes sont simulées à l'aide de la fonction densité de probabilité du nombre de lignes présentes en entrée. Nous avons considéré que le traitement nécessaire au transcodage des caractères est négligeable et qu'à chaque usager était attribué, en mémoire, un tampon pouvant contenir une ligne. Aussi toute présence de ligne en entrée provoque directement le rangement d'une demande dans la file d'attente de l'analyseur syntaxique. Chaque ligne a été assimilée à une instruction dont la longueur est déterminée dès le rangement de la demande dans la file d'attente. Cette longueur permet de déterminer le temps nécessaire au traitement.

2.2 - Entrées-sorties conversationnelles

Seules les sorties ont été simulées. De même que pour les entrées, nous avons considéré que celles-ci ne font l'objet d'aucun traitement. Aussi sont-elles directement chargées dans un tampon après lecture sur le disque. Après quoi, seul le microprogramme est concerné par l'émission sur les lignes de transmission.

2.3 - Remise en ordre des programmes

La simulation de cette fonction est délicate car elle dépend directement du comportement de l'usager. La solution qui consisterait à déterminer à l'aide d'un générateur de nombres aléatoires le numéro de la ligne à replacer et son emplacement dans le programme ne peut pas être retenue. De par la propriété de localité énoncée précédemment cette méthode fausserait les résultats. De plus, la réalisation de la procédure de simulation correspondante serait relativement

compliquée. Nous avons donc décidé de procéder plus simplement. Le système simulé réalise le tri du fichier source à la demande des usagers. Cette méthode s'adapte assez bien avec les données dont nous disposions car le système sur lequel ont été faites les mesures réalise normalement lui-même la numérotation des lignes de programme mais laisse à l'usager la possibilité d'intervenir par un ordre spécial. Nous avons assimilé à des demandes de tri les commandes émises par les usagers à cette occasion.

3 - ALGORITHME DE TRI

3.1 - Organisation des fichiers sources des usagers

Ceux-ci sont découpés en blocs de 85 secteurs (1020 octets) et les articles ont une longueur multiple de 12 octets afin d'en faciliter l'accès. Nous avons distingué deux types de blocs : d'une part les blocs de données supports des articles du fichier, d'autre part les blocs de clés contenant le numéro de l'article et son implantation dans un bloc de données. Nous avons supposé que le numéro d'un article comporte quatre chiffres décimaux codés sur deux octets, quatre octets suffisent donc à repérer un article du fichier et un bloc de clés permet d'accéder à un programme de 255 instructions.

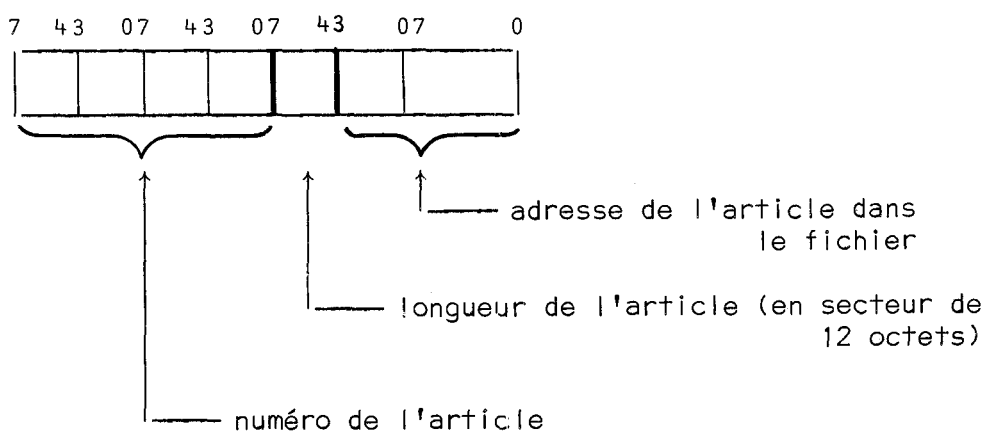


figure VII_7 : Format de la clé d'accès d'un article.

3.2 - Algorithme de Tri

Nous avons recherché un algorithme rapide qui soit assez proche de celui d'un éditeur de texte, afin de ne pas fausser les résultats de la simulation. Dans ce but nous avons supposé que les usagers étaient tous dans la phase de création de leur fichier source. On remarque alors que l'ordre des articles des fichiers peut être contrôlé si l'on mémorise pour chaque usager la clé du dernier article courant. La comparaison entre le numéro de cet article et celui de l'article qui

vient d'être reçu permet de détecter les ruptures de séquence au cours de la création de son fichier. L'article peut dont être chargé séquentiellement dans le fichier, il suffit d'élaborer sa clé et de mémoriser dans une table l'adresse de la clé du premier article de chacune des monotonies ainsi créées par l'usager. A la réception d'un ordre de tri, la remise en ordre du fichier consiste en une fusion des monotonies existant dans les blocs de clés suivie d'une remise en ordre des blocs de données. Celle-ci a lieu directement depuis le disque vers la mémoire car les clés rangées séquentiellement donnent les implantations sur disque des articles correspondants. Les ruptures de séquence étant peu nombreuses, les transferts de mémoire secondaire à mémoire centrale sont en nombre restreint et cette méthode permet de limiter la taille utile de la mémoire. Nous avons appelé respectivement Tri 1 et Tri 2 les deux procédures utilisées. Un programme exécutant Tri 1 a été écrit en Fortran et ses performances ont été mesurées sur IBM 1130. Son temps d'exécution est proportionnel au nombre de clés et est indépendant (aux erreurs de mesures près) du nombre de monotonies. La figure VII_7 donne la courbe que nous avons obtenue.

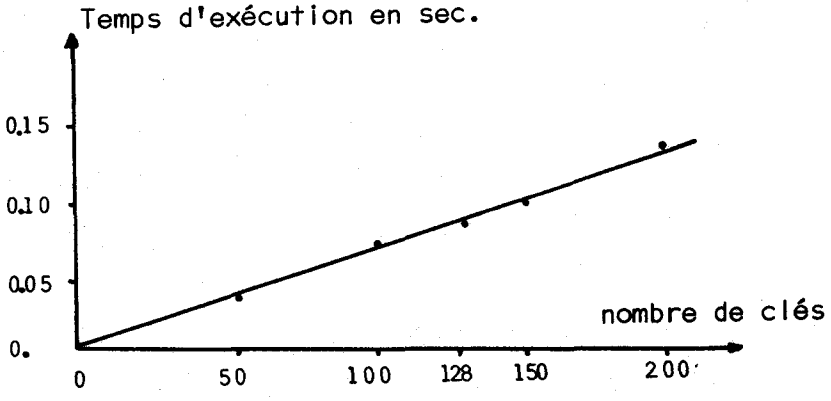


figure VII_8 : Performances de l'algorithme de Tri.

Pour chacune des demandes de tri présentes, on définit le nombre N de lignes du programme. Celui-ci est déterminé suivant une loi de poisson centrée sur un nombre de lignes égal à 200. Cette longueur permet le calcul du temps de traitement nécessaire à l'exécution de Tri 1 ($T = (0,68 \times N)$ millisecondes). Dès la fin du traitement correspondant une demande est déposée dans la file d'attente de Tri 2. Tri 2 n'exécute pas de traitement mais seulement des chargements de mémoire à mémoire pour élaborer chacun des blocs de données. La simulation de cette procédure consiste donc à déterminer le nombre de transferts à réaliser

puis à les simuler. Celui-ci est limité à dix dans chaque bloc de données et est fourni par un générateur de nombres aléatoires. La figure VII_8 schématise l'organisation du système simulé.

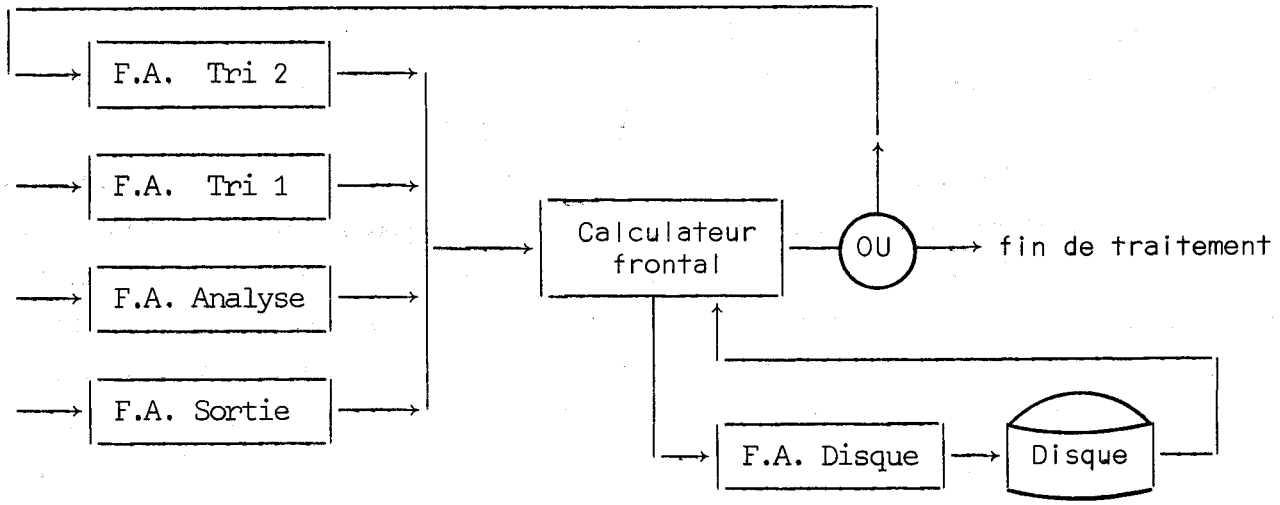


figure VII_9 : Organisation générale du système simulé.

4 - ORGANISATION DU SYSTEME SIMULE

A chaque type de traitement est affectée une priorité d'exécution. L'algorithme du moniteur consiste donc à tester le contenu des files d'attente dans l'ordre de ces priorités et à attribuer la mémoire et l'unité centrale à la première procédure rencontrée dont la file d'attente n'est pas vide. Chaque fois qu'une demande a été traitée, la procédure correspondante rend le contrôle au moniteur ; les files d'attente ne sont donc pas systématiquement vidées, ceci permet d'assurer un temps de réponse optimal des procédures les plus couramment utilisées. Aucune procédure de traitement n'est résidente en mémoire aussi utilise-t-on une procédure nommée Défaut dont le rôle est d'effectuer le chargement de la procédure de traitement qui reçoit le contrôle, lorsque le moniteur constate qu'elle n'est pas présente en mémoire. Le chargement des données de l'usager est laissé à l'initiative de la procédure de traitement utilisée. Les organigrammes de la simulation sont décrits en Annexe C.

Nous avons choisi les priorités suivantes classées dans l'ordre décroissant :

priorité 1 : Sorties - car elles ne demandent pas de traitement. Elles sont exécutées en parallèle avec les autres traitements

priorité 2 : Analyse syntaxique - car c'est le traitement le plus fréquent

priorité 3 : Tri 1

priorité 4 : Tri 2.

5 - RESULTATS DE LA SIMULATION

Nous avons étudié les réactions du système en fonction du nombre de terminaux actifs. Afin d'obtenir des résultats significatifs chaque situation de charge a été étudiée pendant 120 minutes de simulation (cette limite nous a été imposée par la taille des tableaux et le temps d'unité centrale utilisé). Les mesures portent sur le temps de réponse de chacune des procédures de traitement, temps mesuré entre l'entrée dans la file d'attente et la fin du traitement.

Nous avons calculé les temps de réponse moyens obtenus au cours de la simulation pour chacune des procédures ainsi que le temps moyen d'exécution. Nous n'avons pas pu étudier les densités de probabilité des temps de réponse, ceci nous étant imposé par la taille de la mémoire utilisable et la nécessité de limiter la durée de chacune des simulations que nous avons faites. Afin d'en réaliser une approche, nous avons mesuré le temps de réponse maximal obtenu dans chaque minute de simulation. Ce sont les densités de probabilité de ces temps de réponse que nous avons étudiées.

Quatre caractéristiques peuvent être dégagées des courbes que nous avons obtenues (celles-ci ont été reportées en fin de chapitre) :

- Les courbes de la figure VII_{10} montrent que le système est "viable" en ce sens que les temps de réponse obtenus sont corrects. De plus le nombre d'usagers peut être important. Ceci nous conduit à conclure que le système frontal pourra réaliser l'édition de texte et l'analyse syntaxique des fichiers sources des usagers, sans que nous puissions pourtant espérer une charge aussi élevée. Seule les conditions de simulation ont permis d'obtenir une limite de charge aussi importante. Dans la réalité les appels à l'éditeur de texte sont plus nombreux , cependant on remarquera qu'en simulant un tri l'on introduit des régimes transitoires qui seront moins apparents dans le système réel.

- On constate sur ces mêmes courbes que les temps d'attente sont très élevés. Pour trente deux usagers, le temps de réponse de l'analyseur syntaxique est plus que doublé. Ces temps d'attente élevés concernent à la fois le séjour de l'usager dans la file d'attente de la procédure et le chargement de ses données en mémoire. On ne peut pas éliminer l'attente de la procédure, mais on constate que le temps de réponse du disque est de cinquante millisecondes pour un temps de transfert de quatorze millisecondes. On peut donc limiter l'attente globale des usagers en admettant plusieurs d'entre eux en mémoire. Cette remarque nous a conduit à appliquer la notion de mémoire virtuelle et le chargement à la demande.

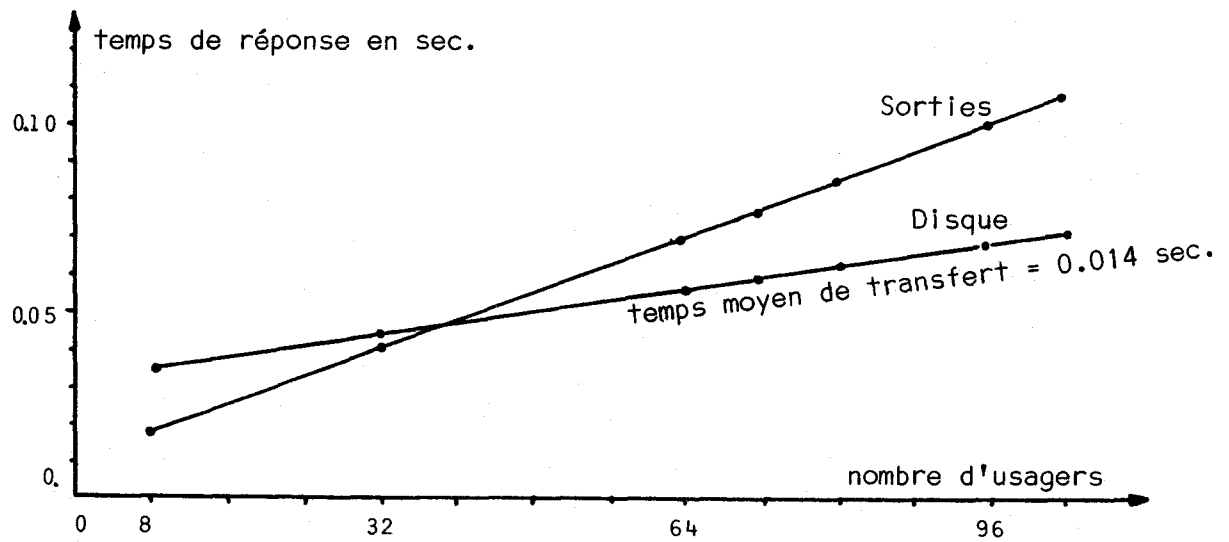
- L'utilisation du disque augmente fortement avec le nombre d'usagers (figure VII_{11}). Le taux d'occupation de onze pour cent obtenu ici pour 32 usagers serait dans la réalité plus élevé d'autant qu'un quantum devrait être affecté à chaque usager afin d'améliorer le temps de réponse apparent du système. Aussi , si l'on conservait une organisation semblable à celle du système simulé,devrait-on s'attendre à une augmentation du nombre de chargements et du taux d'occupation du disque. Celle-ci n'est pas souhaitable car elle implique une probabilité non négligeable pour que le système central ne puisse pas accéder au disque partagé. En conséquence lors de la réalisation du système réel, nous devrons tenter d'approcher au plus près le working set des usagers afin de limiter les transferts.

- Les courbes de la figure VII_{12} montrent que lorsque la charge du système augmente les courbes de densité de probabilité des temps de réponse maximaux mesurés s'aplatissent. Des temps de réponse maximaux élevés peuvent donc apparaître avec une probabilité non négligeable. La courbe VII_{12a} montre que ceci est dû à la fois au temps de réponse du disque et au temps passé dans les files d'attente des procédures. Aucun remède ne peut y être apporté, aussi ceci montre-t-il que le nombre d'usagers potentiel du système ne peut être fixé uniquement à l'aide des temps de réponse moyens souhaités. En pratique, celui-ci ne pourra être déterminé que par l'expérience en tentant d'évaluer la gène ressentie par les usagers.

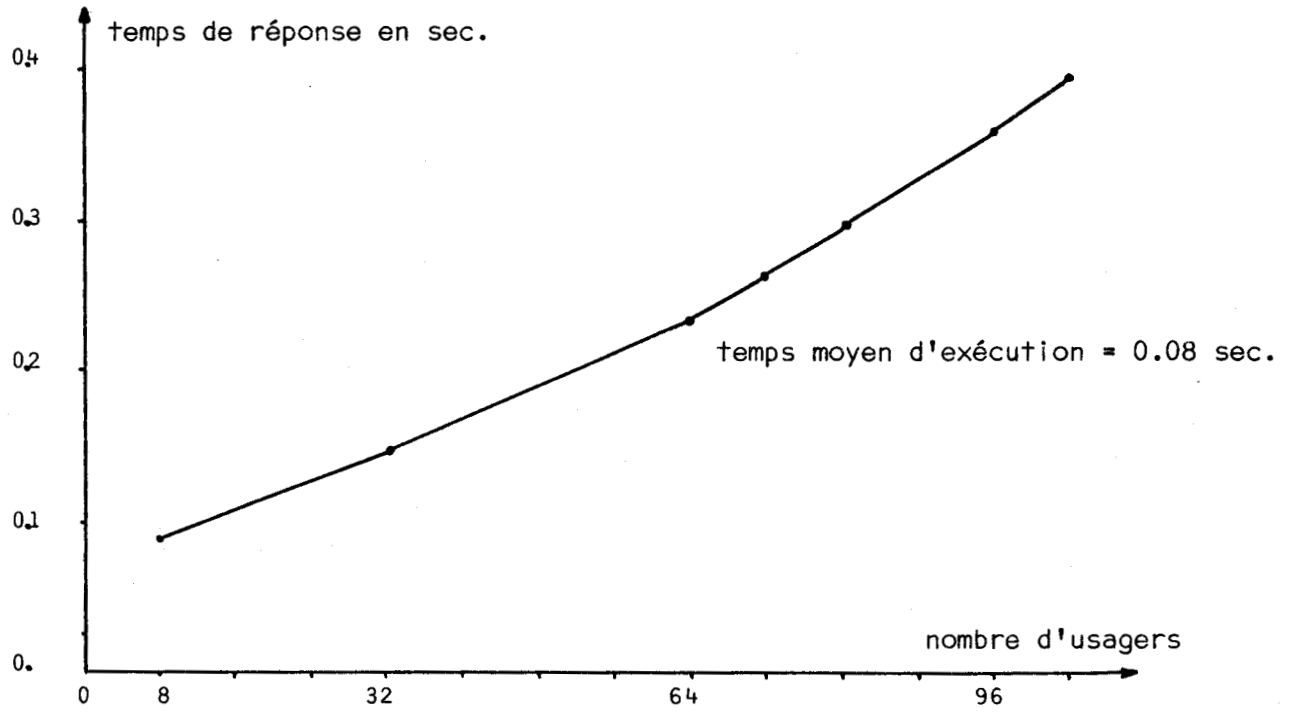
6 - CONCLUSION

On remarquera que tous les temps de traitement ont été bornés supérieurement, aussi cette simulation est-elle valable. Dans la réalité nous ne pouvons pourtant pas nous attendre à ces résultats car certaines fonctions considérées comme négligeables n'ont pas été introduites dans le simulateur. Cependant, le système central peut être considéré comme "viable" en ce sens que les temps de réponse que l'on peut espérer permettent des performances correctes. En simulant un tri, nous avons introduit des régimes transitoires importants qui seront moins nets dans la réalité d'autant que nous avons supposé un nombre de ruptures de séquences élevé. L'examen des courbes relatives aux temps de réponse maximaux peut donc permettre de tenter une évaluation du nombre maximum d'usagers potentiels que pourrait supporter le système frontal. On notera à cet effet que l'aplatissement des courbes commence à être sensible au-delà de 64 usagers. Il semblerait donc que la limite de charge du système réel corresponde à cette charge maximale du système simulé. Dans la mesure où nous arriverons à contrôler efficacement le trafic de pages du système réel, nous pouvons espérer admettre un nombre d'usagers voisin de 64.

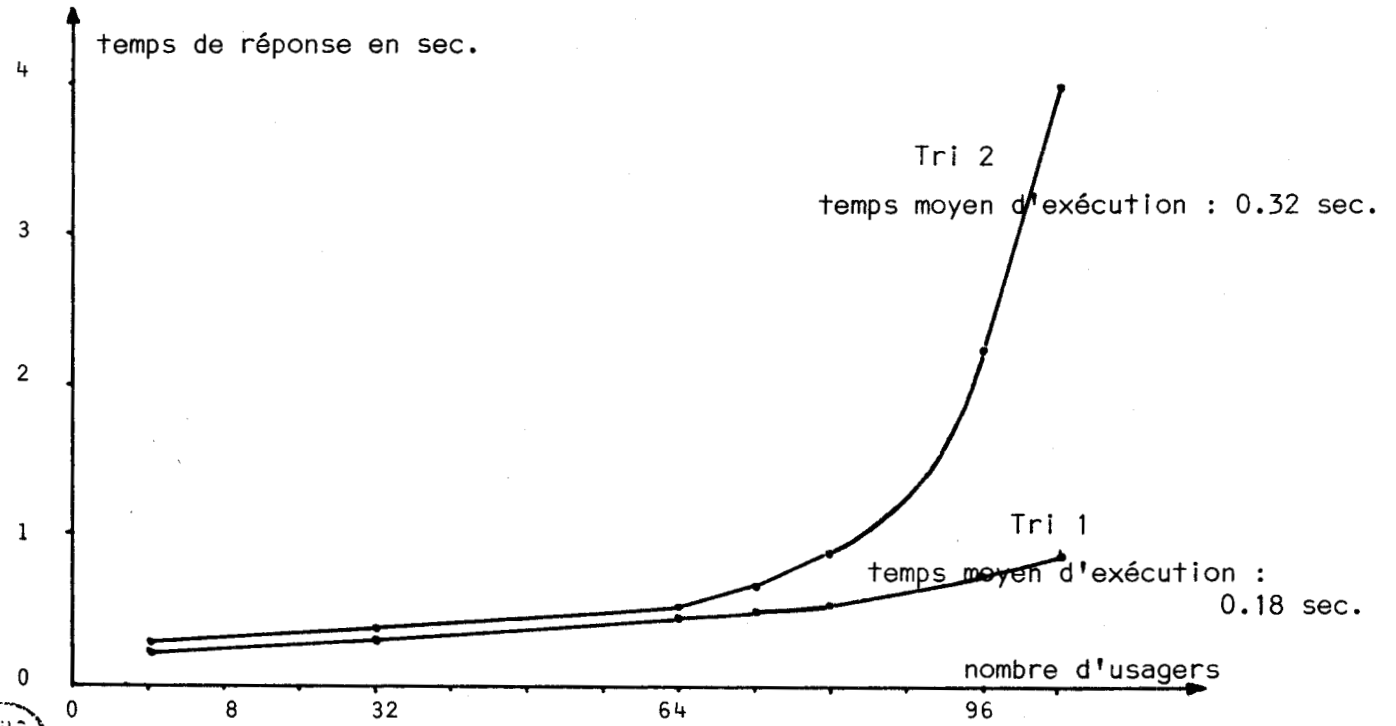
L'analyse des résultats de la simulation a mis en évidence des phénomènes qui nous ont conduit à adopter le principe des machines virtuelles. Celui-ci se justifie encore par la constatation que nous avons faite que le système d'exploitation du calculateur frontal est ainsi plus facile à écrire.



a) Cas des transferts

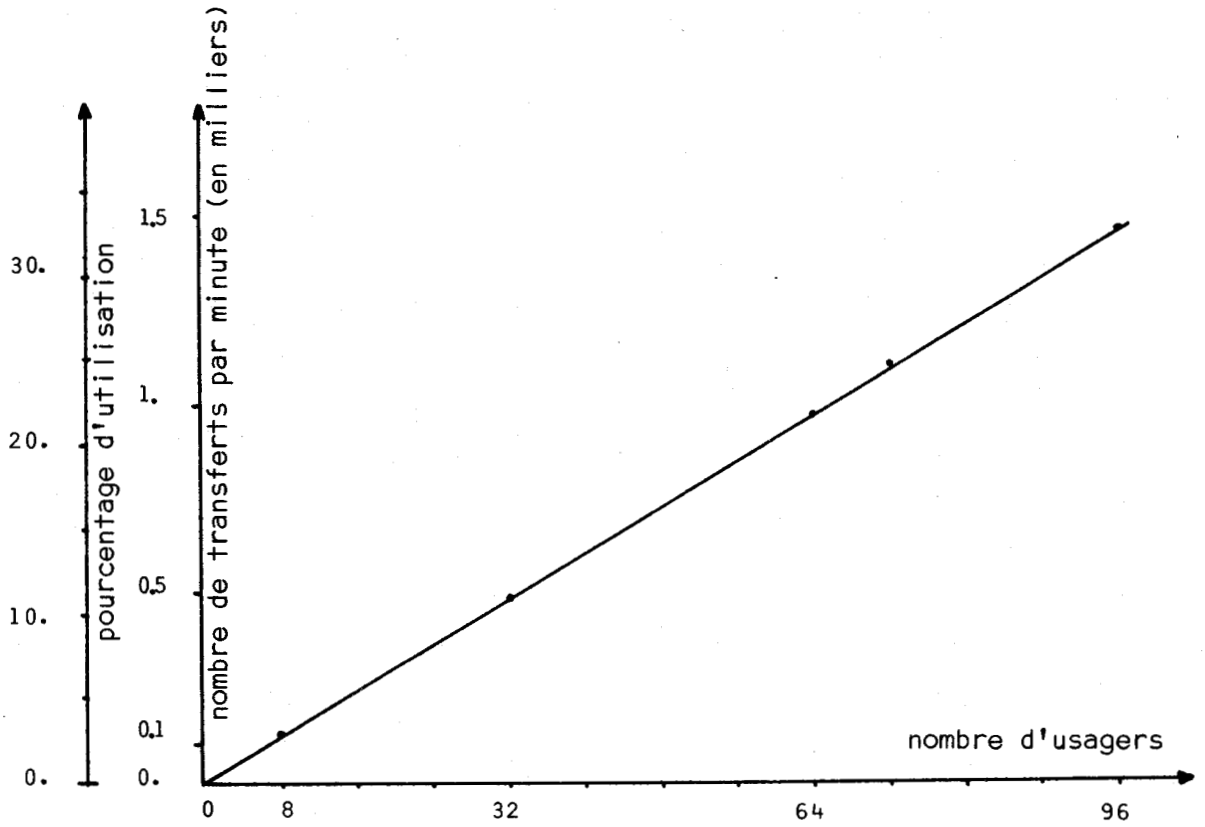


b) Réponse de l'analyseur syntaxique.

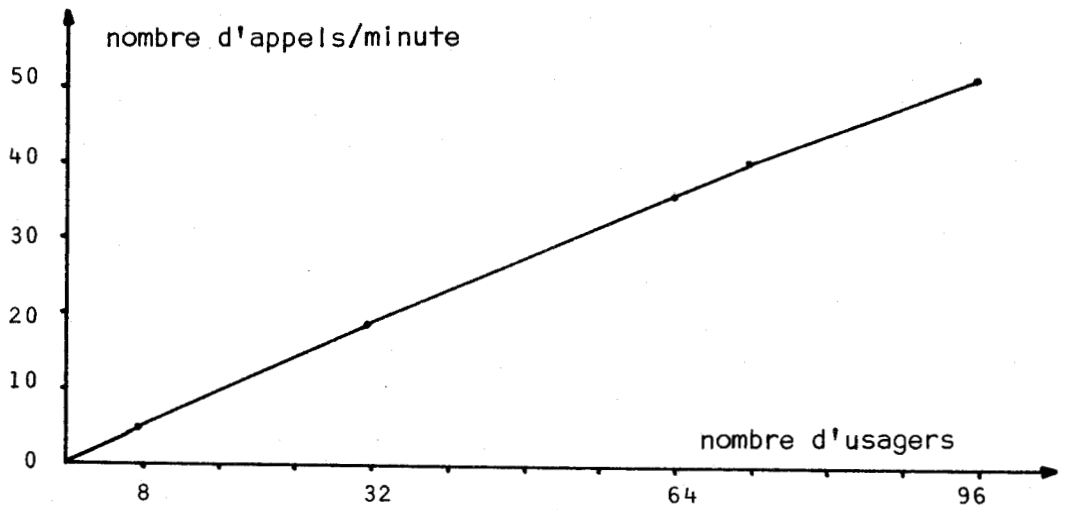


c) Réponse des procédures de Tri.

figure VII_{10} : Temps de réponse moyens des procédures de service du système simulé.

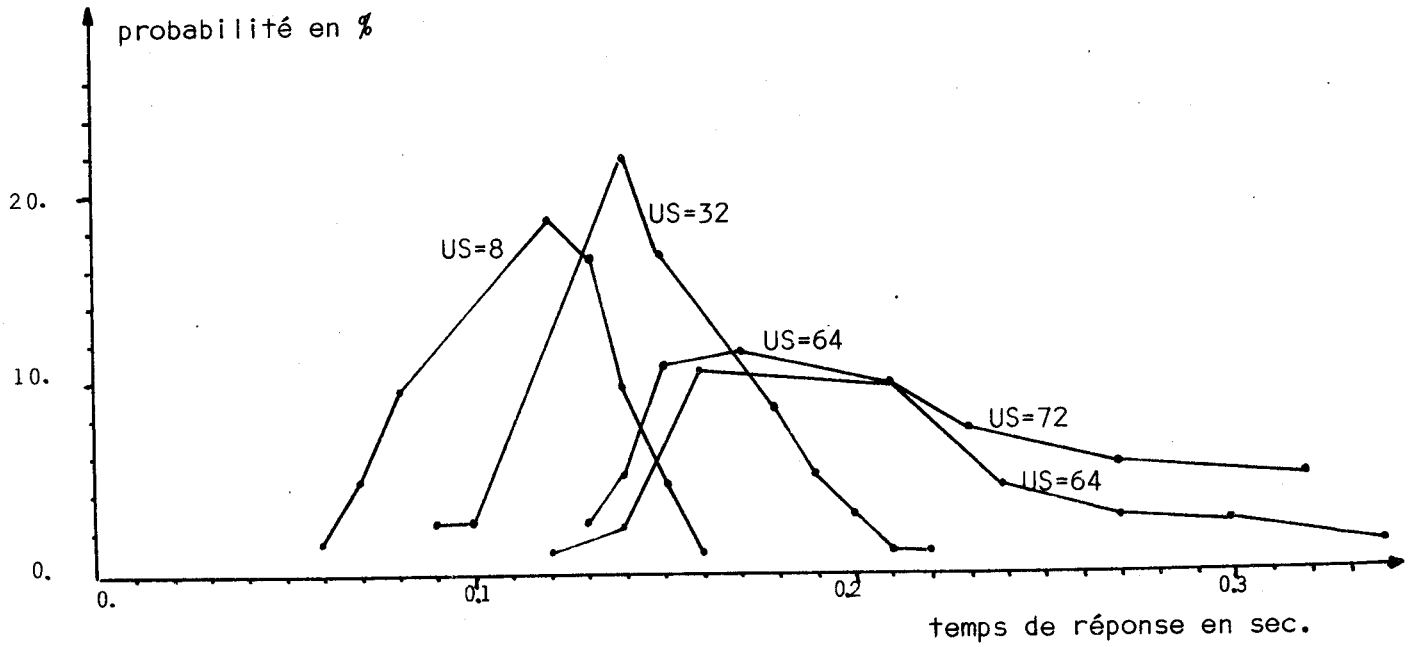


a) Evolution de l'occupation du disque en fonction de la charge

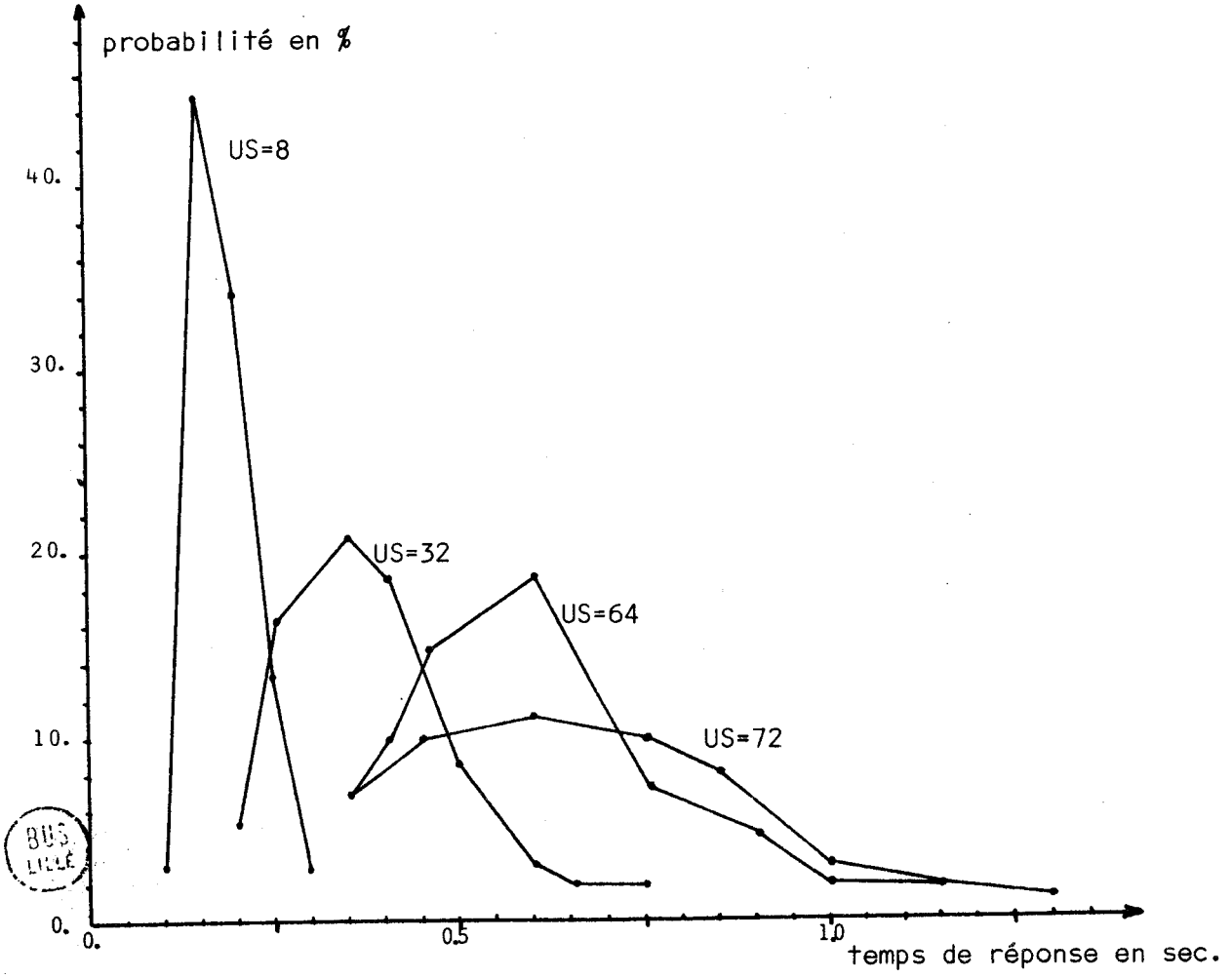


b) Evolution du nombre d'appels à Défaut en fonction de la charge

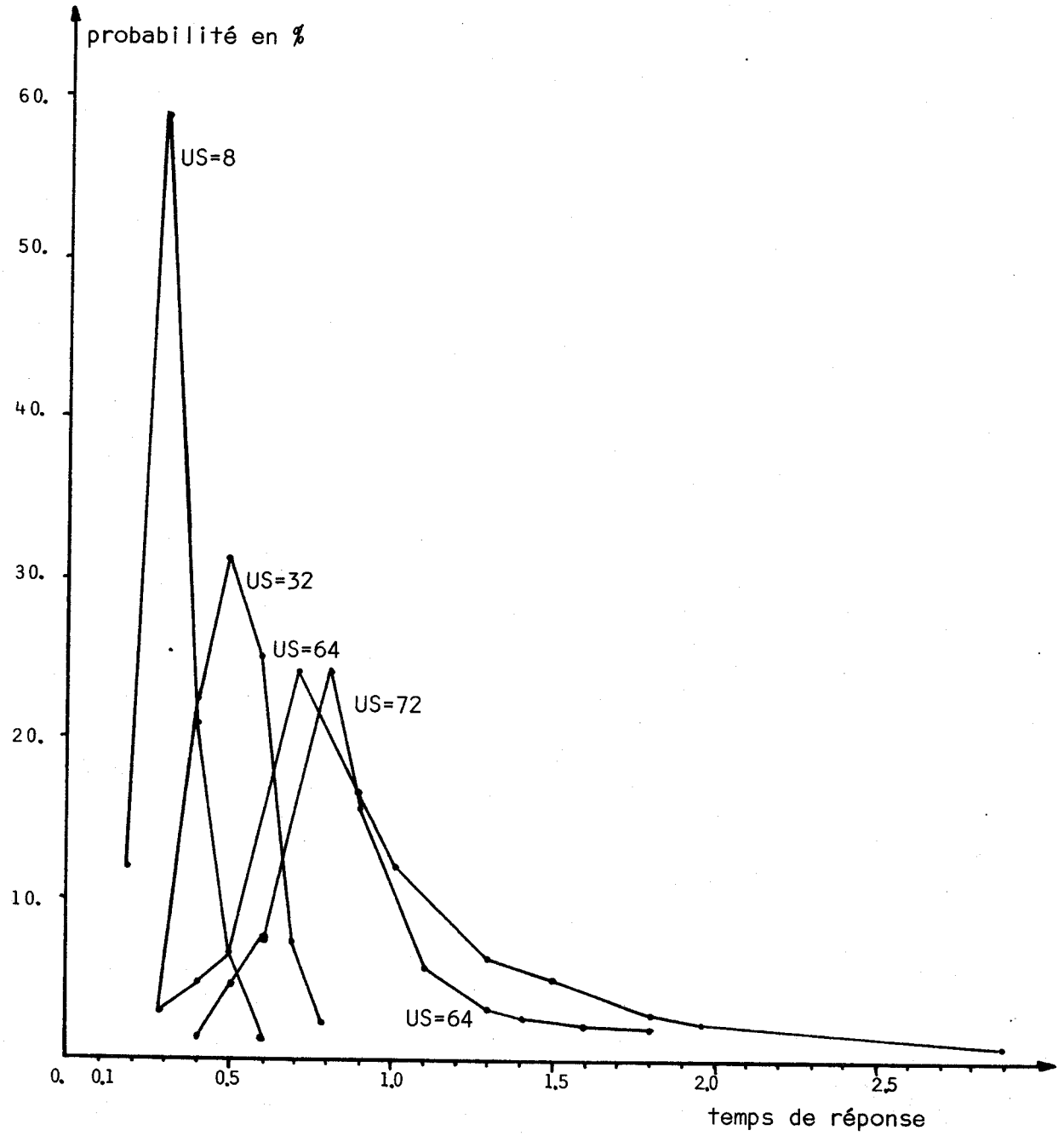
figure VII_{11} : Utilisation du système



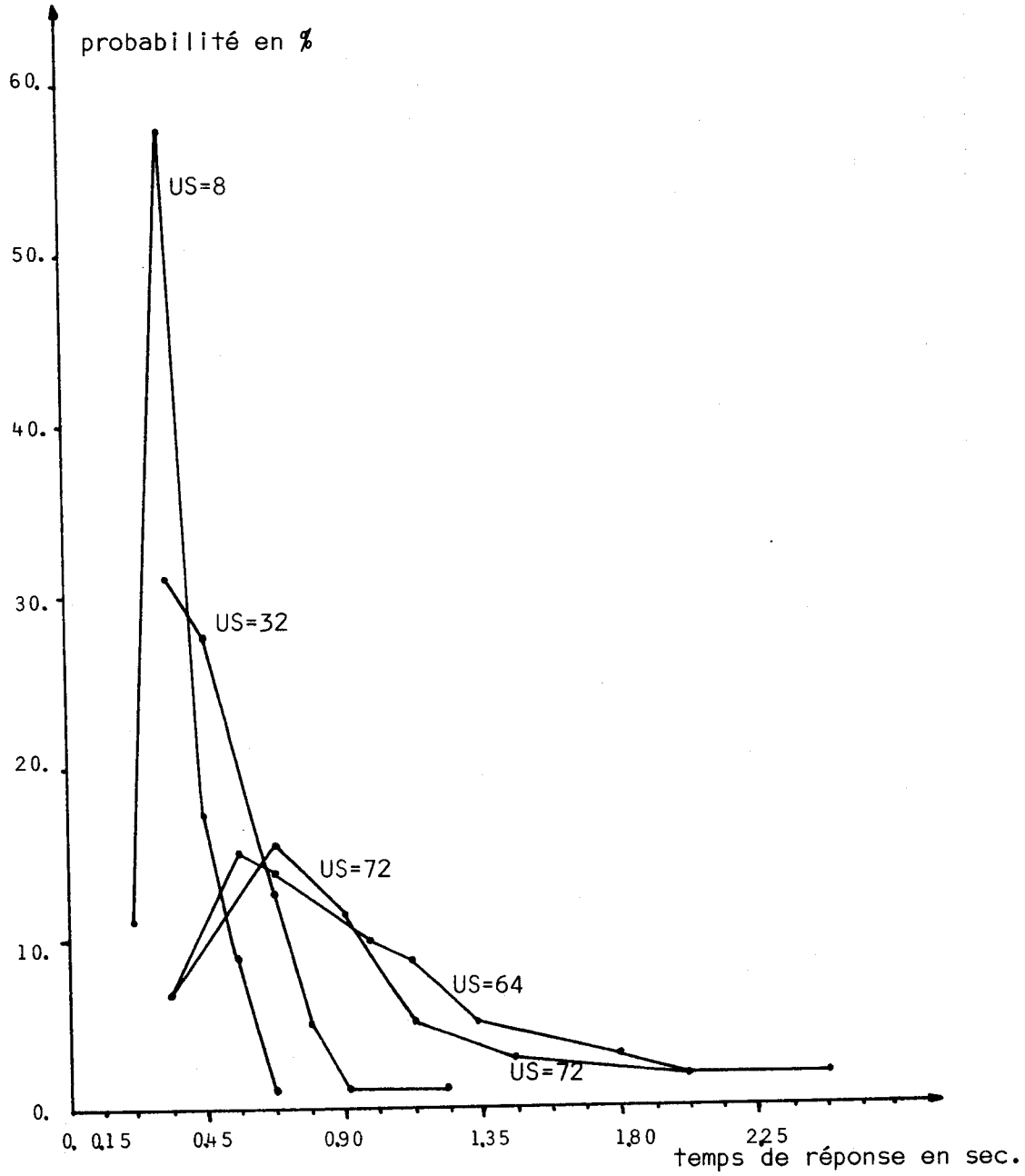
VII_{12} a) Cas du disque



VII_{12} b) Analyse syntaxique



VII_{12} c) Tri 1



VII_{12} d) Tri 2

figure VII_{12} : Evolution des temps de réponse maximaux mesurés en fonction de la charge.

- Ces courbes sont les enveloppes des résultats des mesures
- US est le nombre d'usagers.

CONCLUSION

Cette thèse ne pouvait être une étude exhaustive des problèmes que posera la réalisation effective du système frontal, certains points n'ont pas été précisés. Il s'agit essentiellement de l'adaptation du système central à ce nouveau mode de fonctionnement. Elle s'inscrit dans le cadre d'une direction d'études nouvelle pour le laboratoire, dans un domaine où la diversité des problèmes nous a imposé un choix :

- soit nous limiter dans leur approche
- soit n'aborder que les points essentiels.

C'est cette deuxième solution que nous avons retenue.

Nous avons voulu dès le lancement de l'étude nous affranchir de tout modèle existant et nous libérer de toute contrainte de réalisation. Ceci peut sembler être en contradiction avec la nécessité de concevoir un système frontal capable de supporter de nombreux terminaux. Ainsi, dans le système proposé, la mémoire devra avoir une taille importante (en regard de la classe du calculateur utilisé) ; ceci n'aurait pas été nécessaire si nous avions adopté un modèle plus classique. Cependant, on remarquera que la notion de machine virtuelle dont l'application nous impose cette taille de la mémoire rend le système d'exploitation plus efficace que ce que nous laisserait espérer un modèle plus classique. De plus, son écriture et celle des procédures de service sont ainsi plus aisées.

Deux aspects peuvent sembler gênants :

. La première concerne la taille des tables qui occupent la mémoire du calculateur frontal. Elle limite la taille de la mémoire disponible pour les usagers, mais ne peut être réduite. En fait, nous pensons compenser l'encombrement important des tables en réduisant fortement celui des primitives et du système d'exploitation. Pour cela nous utiliserons une instruction d'appel à des sous-programmes microprogrammés telsque : gestion de liste, recherche de sémaphore etc... Le choix judicieux de ces microprogrammes permettra de plus d'améliorer les performances des mécanismes de base du système.

. Le second concerne le temps de transfert du disque partagé. Ce temps peut sembler élevé pour un système à chargement de page à la demande. Cependant, on notera

que les caractéristiques des traitements que réalise le système frontal font que ces chargements ne seront nombreux que dans la mesure où les usagers sont eux-mêmes nombreux. D'ailleurs cet effet sera compensé par l'adaptation des catégories en fonction de cette caractéristique.

Le projet que nous avons présenté peut paraître d'une taille disproportionnée par rapport au rôle qui est attribué au système frontal. En réalité, l'influence que l'on peut attendre du calculateur frontal sur le comportement du système est prédominante : le projet s'inscrit dans le cadre d'une utilisation effective dans un centre de calcul et le système frontal peut avoir un impact important en permettant une rentabilisation accrue du matériel. Il est délicat d'évaluer le coût de cette réalisation ; sans que l'on puisse considérer ce jugement comme exact et définitif, il nous semble qu'il devrait être de quatre homme/année dans sa première phase. Celle-ci consistera à mener en parallèle la réalisation des coupleurs et des microprogrammes (donc du système d'exploitation). Ce n'est qu'ensuite que seront mises en place les procédures de service, leur étude pourra donc avoir été lancée dans la phase précédente.

Le système proposé n'est pas spécifique des caractéristiques du calculateur central utilisé et les applications que l'on peut envisager sont variées. Lors de l'étude préliminaire, nous n'avons retenu que l'aspect "time sharing" mais ce peut être également le moyen de développer le système de documentation automatique actuel du laboratoire en lui donnant un aspect conversationnel sans pour autant surcharger le calculateur central. D'une manière générale, le système frontal permettra toute extension se rapportant aux travaux sur fichiers.

*

* *

ANNEXES

Annexe A

Primitives utilisées

1- Microprimitives et instructions associées

nom	mode d'accès	paramètres	action
Brancher	maître	*k, AD*	range *AD* dans le compteur ordinal et retourne à *INT*
Retour	maître	aucun	range le contexte de tête de *F(UC)* et retourne à *INT*
p	maître	aucun	réalisent l'exclusion mutuelle lors des accès à l'UC - utilisent *F(UC)*
v	maître	aucun	
Appel	tous	*k, a*	appelle la procédure *k* et lui fournit les paramètres *a* dans les registres - exécute *p*

2- Primitives agissant sur les Sémaphores

P	maître	*S*	travaillent sur des indices de sémaphores
V	maître	*S*	
D	maître	*S*	
Demander	esclave	*S*	même action que *P, V, D*
Libérer	esclave	*S*	travaillent sur des noms de sémaphores
Délivrer	esclave	*S*	
Effacer	esclave	*S*	supprime le sémaphore *S*
Déclarer	esclave	*S*	crée le sémaphore *S*

3- Primitives agissant sur les processus

Créer	esclave	*P, p*	crée le processus *P* avec la priorité *p*
Détruire	esclave	*P*	détruit le processus *P*
Disparaître	esclave	aucun	auto-destruction du processus qui l'exécute

4- Primitives de communication

Prendre	tous	*k*	prend un tampon de type *k* - fournit son adresse dans l'accumulateur du processus qui l'exécute
Rendre	tous	*k, AD*	rend le tampon de type *k* et d'adresse *AD* à la liste des tampons libres
Transmettre	esclave	*k, AD*	range le tampon directive d'adresse *AD* dans la liste des directives du processus repéré par *k* - exécute *V* sur le sémaphore de ce processus

5- Primitives travaillant sur des fichiers internesou des segments

Ouvrir	esclave	*f, p, l*	crée dans la page de garde de l'usager le fichier interne *f* de longueur *l* et lui attribue la protection *p*
Fermer	esclave	*f*	détruit la TIF du fichier *f*
Associer	esclave	*s, f*	crée une TPV pour le segment *s* et lui associe la TIF de *f*
Dissocier	esclave	*s*	détruit la TPV du segment *s*
Unir	esclave	*n, s*	range l'adresse début de la TPV du segment *s* dans G_n et mémorise *s* dans le contexte du processus
Bloquer	esclave (contrôlée)	*s,p,S,k*	initialise le blocage de tous les processus tentant d'accèder à la page *(s,p)* derrière le sémaphore *S* de l'usager
Débloquer	esclave (contrôlée)	*s, p*	libère tous les processus bloqués derrière le sémaphore *S* déclaré dans la primitive *Bloquer*
Mettre à jour	esclave (contrôlée)	*s,p,S*	lance le transfert sur disque de la page *(s,p)* et le de la page de garde de l'usager si elles sont écrites et bloque le processus qui l'exécute derrière *S*. Il sera réveillé en fin de transfert par le processus de gestion du disque.

Annexe B

Tables associées à la gestion des processus

Elles doivent permettre un accès rapide aux sémaphores et aux processus des usagers et du système. De plus, la création dynamique des processus et des sémaphores des usagers impose que leur implantation soit telle que la destruction d'un processus ou d'un sémaphore n'en nécessite pas une réorganisation complète. On utilise donc des structures en listes fermées et l'on associe à chaque table un pointeur repérant le dernier élément créé. Cette structure permet une utilisation optimale de la zone mémoire réservée à ces tables. Elle est complétée par des listes décrivant les tampons non utilisés.

Trois types de tables sont utilisés, ce sont :

1- Table des processus

Pour chaque usager, il existe une liste dont chaque élément occupe 21 octets :

- contenu des registres (9 octets)
- priorité du processus (1 octet)
- pouvoir (1 octet)
- nom du processus (1 octet)
- nom de l'usager (1 octet)
- pointeur vers le processus suivant de l'usager (2 octets)
- pointeur vers le processus suivant dans la liste des processus prêts de l'usager ou dans la liste des processus bloqués derrière le même sémaphore (2 octets)
- champ mémoire (4 octets, un par segment possible).

2- Table des sémaphores

Il existe une liste par usager. Pour chaque élément on trouve un descripteur de six octets :

- valeur de l'entier associé au sémaphore (1 octet)
- nom du sémaphore (1 octet)
- pointeur vers le sémaphore suivant chez l'usager (2 octets)
- pointeur vers le dernier processus bloqué dans la file d'attente du sémaphore (2 octets).

3- Table des usagers

Chaque élément occupe cinq octets :

- nombre de processus créés par l'usager (4 bits)
- nombre de sémaphores créés par l'usager (4 bits)
- pointeur de queue de la file des processus prêts de l'usager (2 octets)
- pointeur de queue de la file des sémaphores de l'usager (2 octets).

Implantation des tables

Nous avons déjà signalé que dix pages devront être utilisées en mémoire pour implanter ces tables. Cet encombrement est très important mais ne peut être réduit. Deux solutions pourraient être envisagées dans ce but mais ne conviennent pas :

- utiliser l'adressage virtuel à l'intérieur des primitives pour adresser les sémaphores, les tables de chacun des usagers pourraient alors être reportées dans l'espace virtuel du système par exemple. Outre que l'accès serait ralenti, ceci ne peut être envisagé car le déroutement pour défaut de page provoque l'exécution de la microprimitive *p* qui a déjà été exécutée par la primitive ayant tenté l'accès au sémaphore ou au contexte du processus. Il y aurait donc un blocage définitif du système.

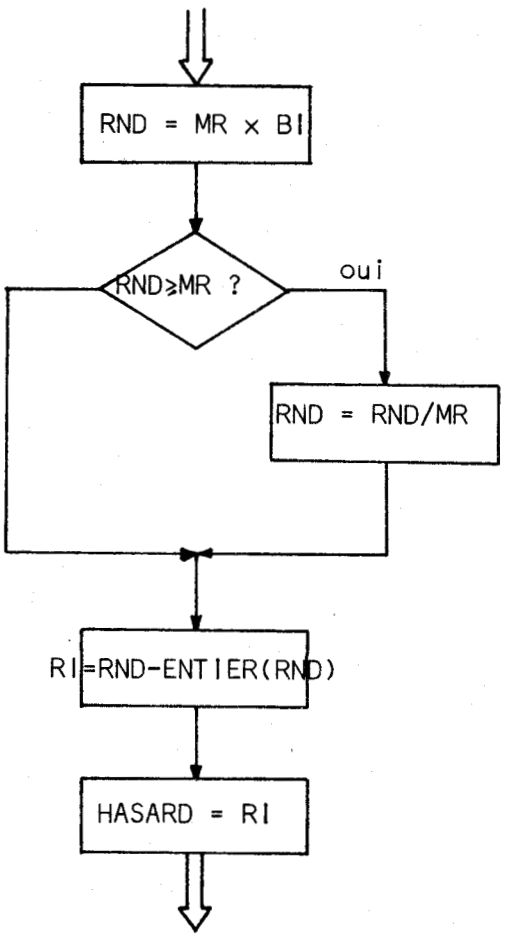
- attribuer sur disque à chaque usager une zone dans laquelle seraient rangées ses tables et simuler ainsi un adressage virtuel en ne provoquant le chargement de ces tables que lorsque leur utilisation est nécessaire. Cette solution ne peut être retenue car des actions peuvent être exécutées sur les sémaphores d'un usager alors qu'il n'est pas dans l'état courant. C'est le cas lors de l'activation de processus d'un usager par des processus du système. Procéder ainsi provoquerait une augmentation du trafic sur le canal disque et rendrait délicate la gestion des processus. Il est donc vraisemblable que les performances du système chuteraient fortement.

Il semble donc qu'il n'y ait pas d'autre solution que de limiter le nombre d'usagers ou le nombre de leur processus.

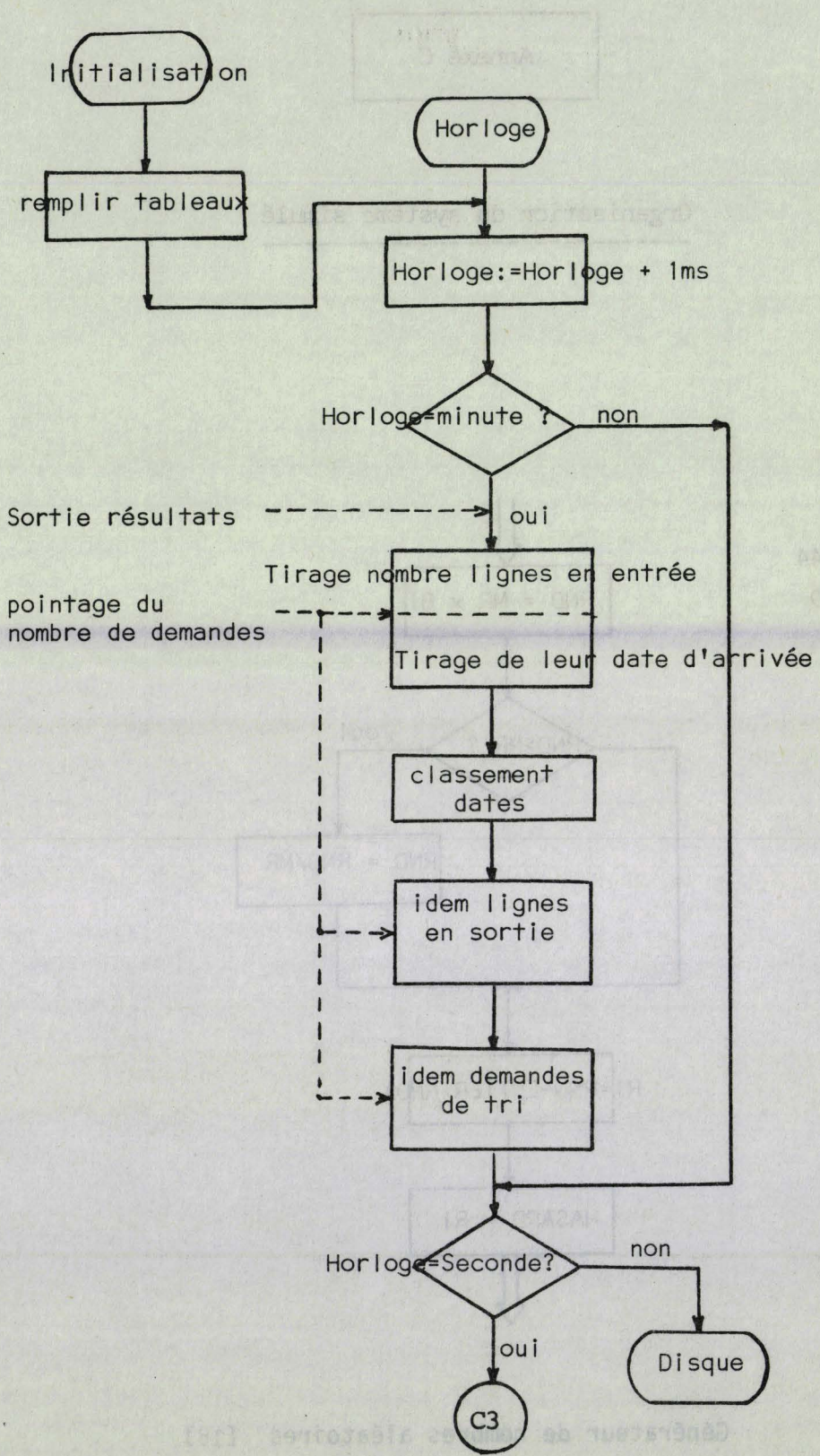
Annexe C

Organisation du système simulé

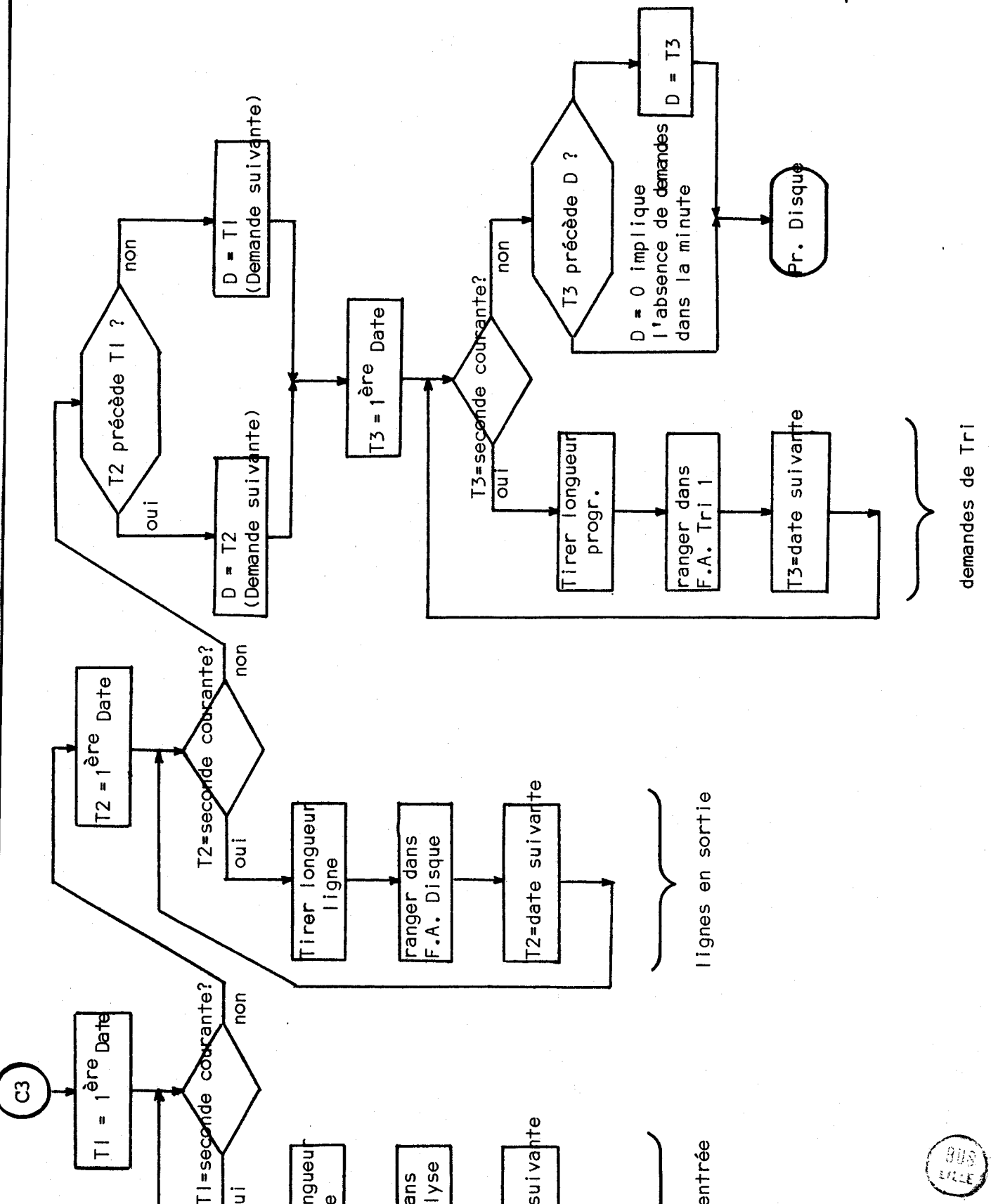
BI = 3215
MR = 262144
RND = 0.99

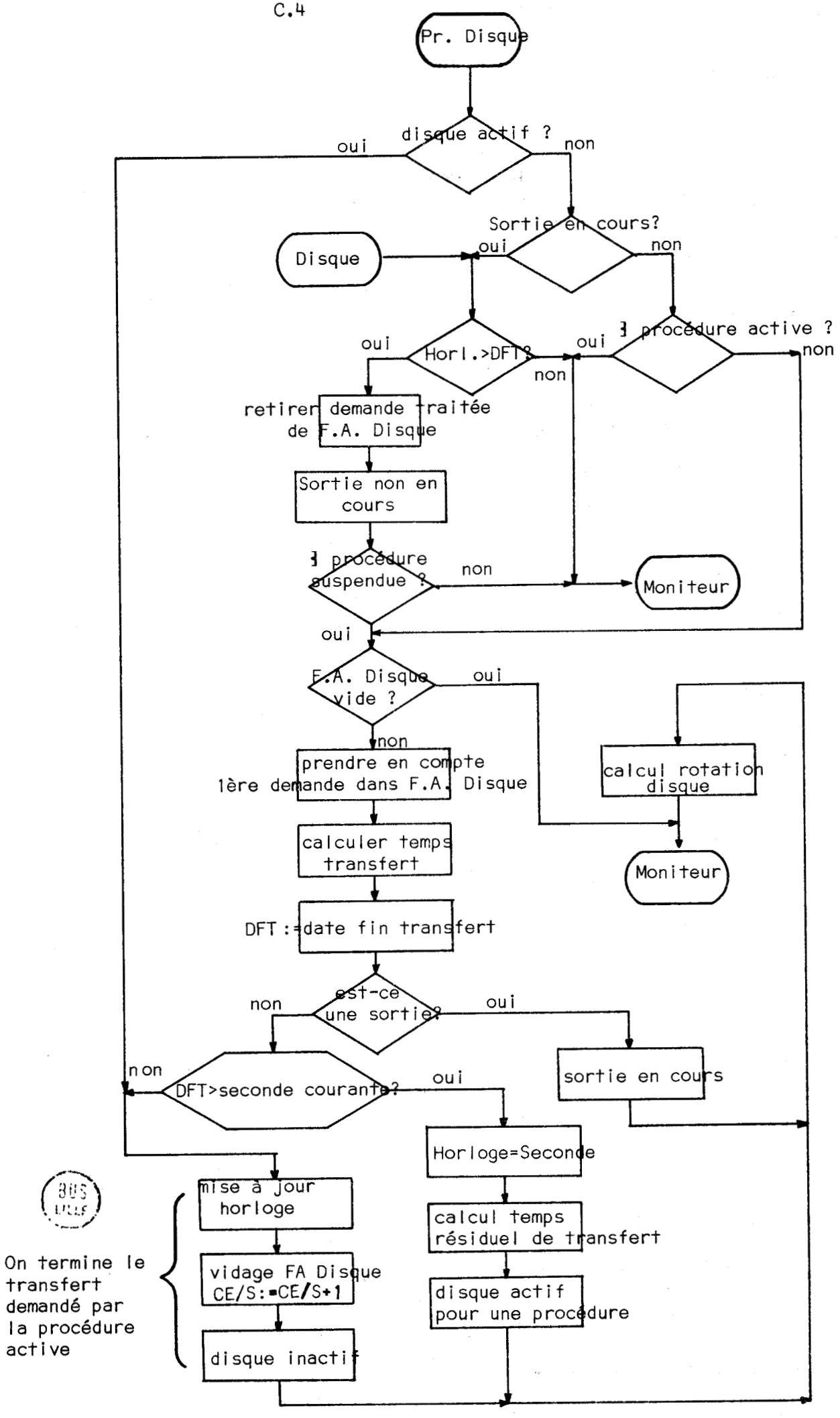


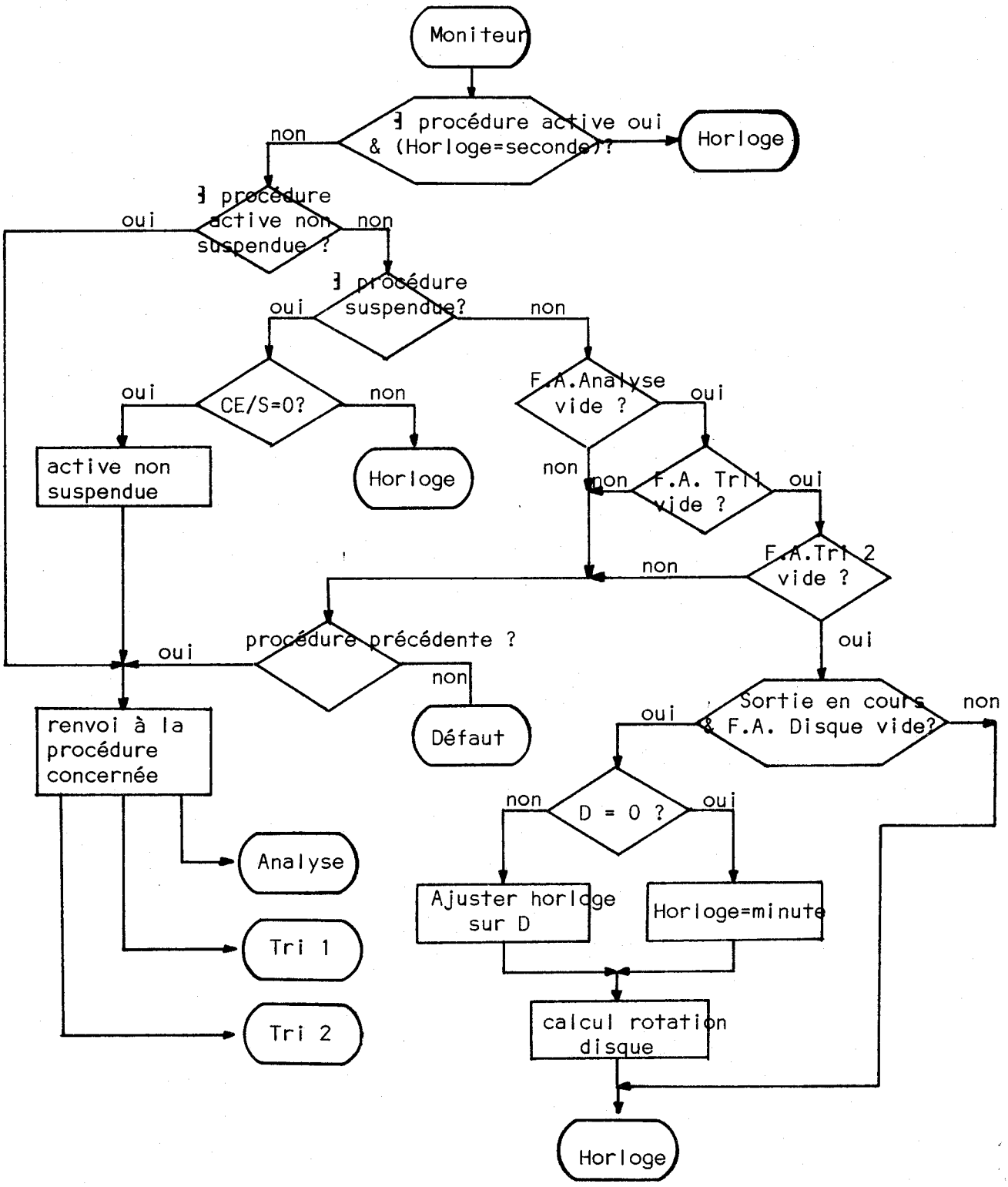
Générateur de nombres aléatoires [18]

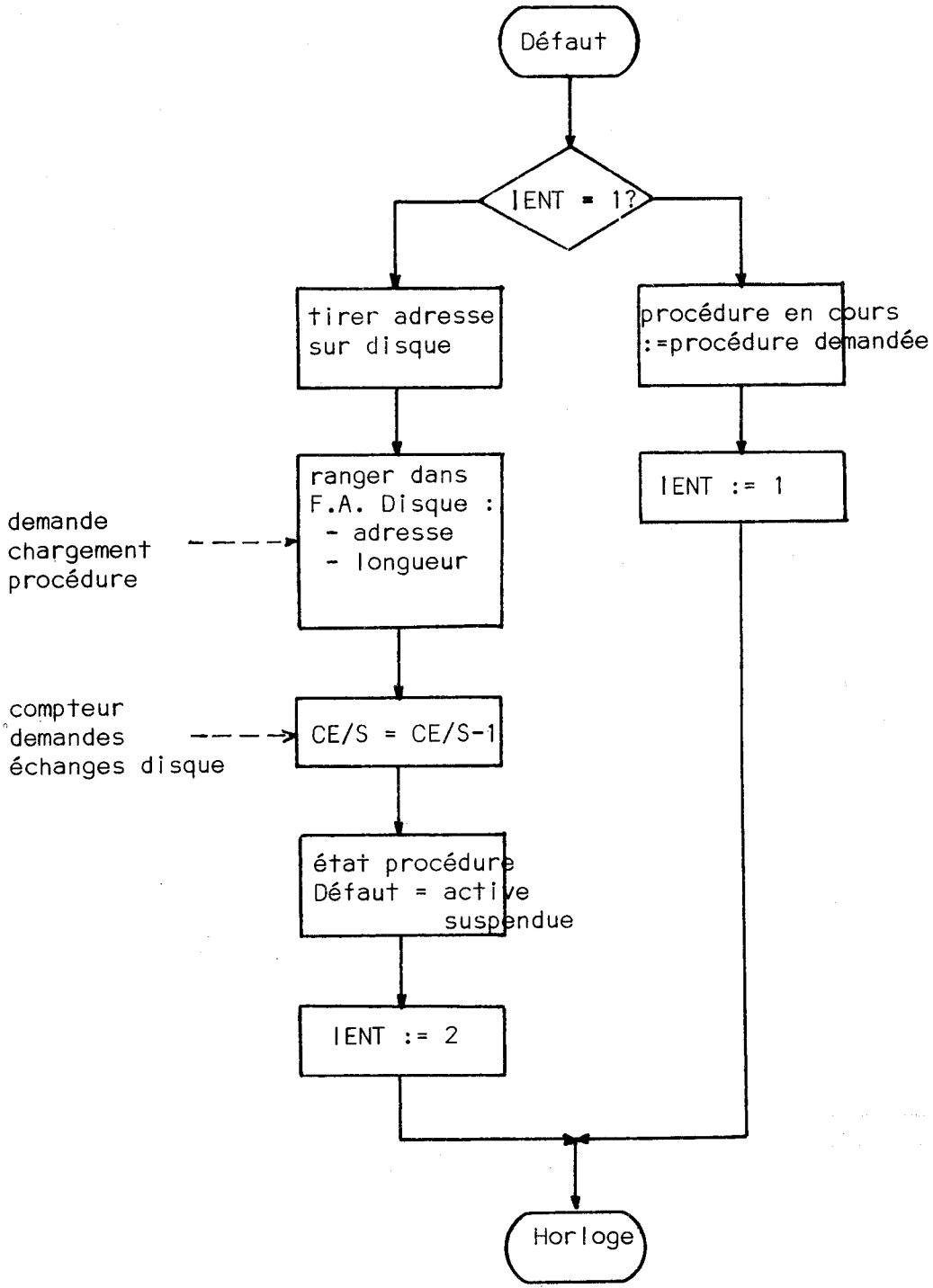

Initialisation
Horloge
remplir tableaux
Horloge:=Horloge + 1ms
Horloge=minute ?
non
Sortie résultats
oui
Tirage nombre lignes en entrée
pointage du
nombre de demandes
Tirage de leur date d'arrivée
classement
dates
idem lignes
en sortie
idem demandes
de tri
Horloge=Seconde?
non
oui
Disque
C3

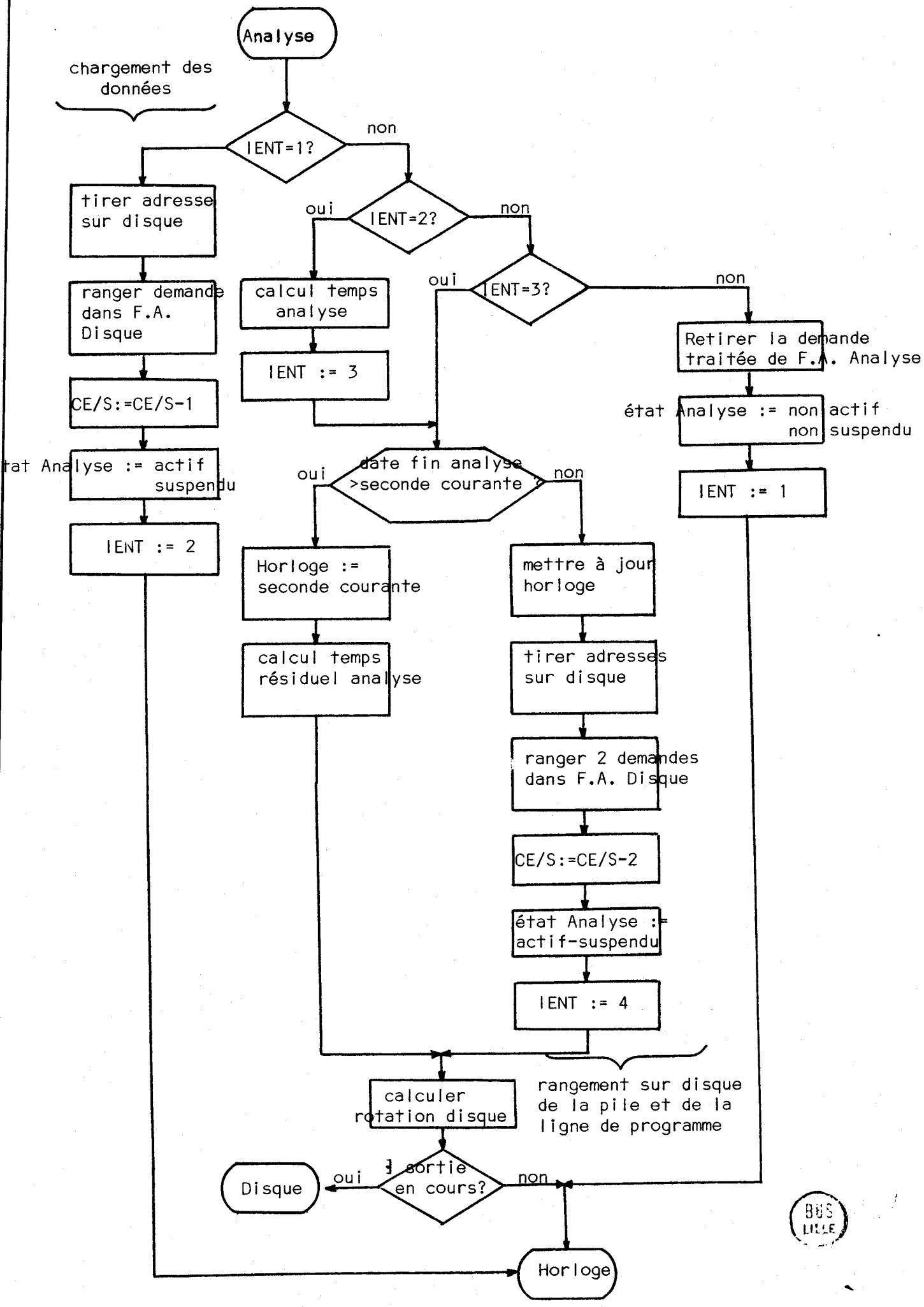
Recherche de la date de la première demande à traiter dans l'ordre des arrivées et des priorités.

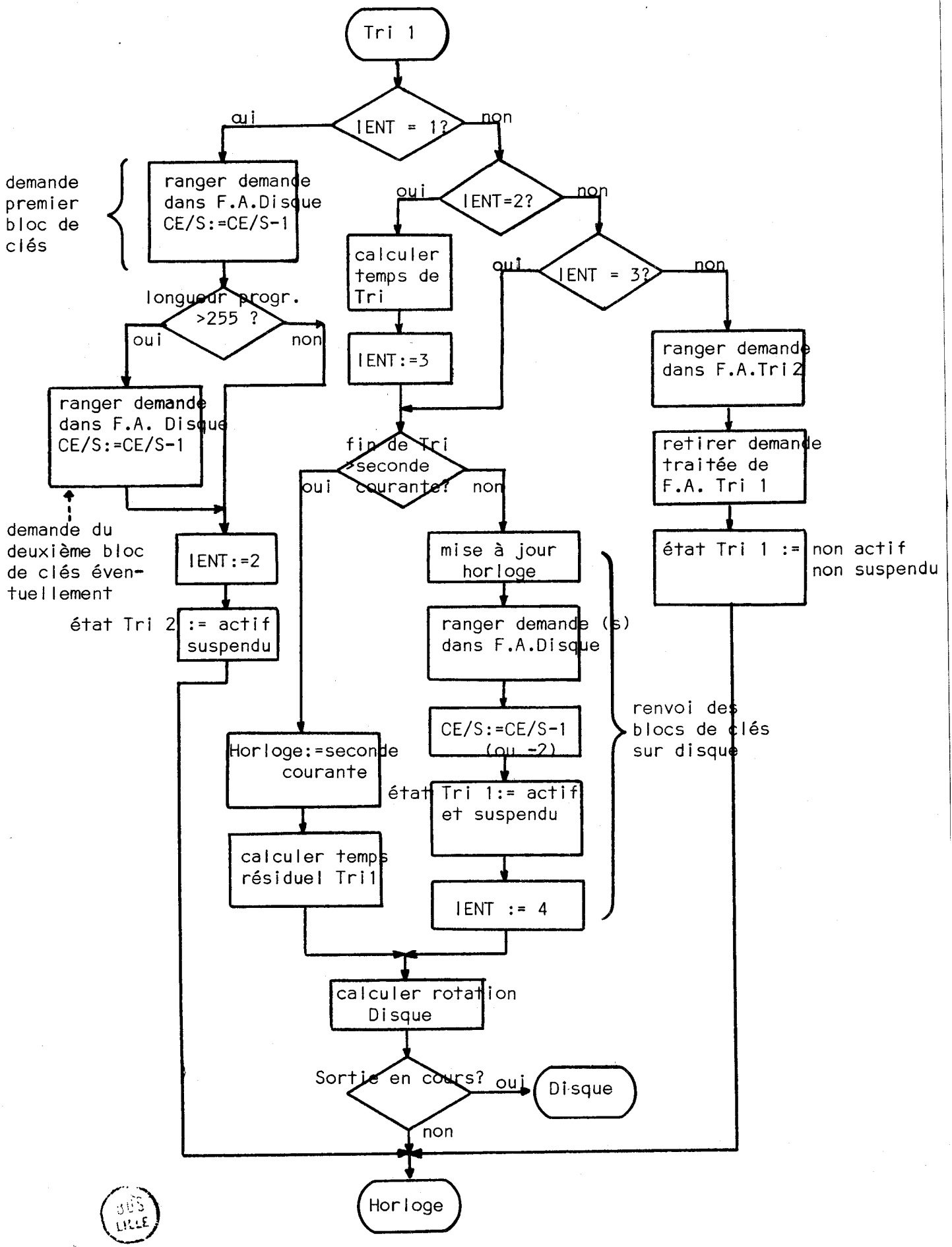


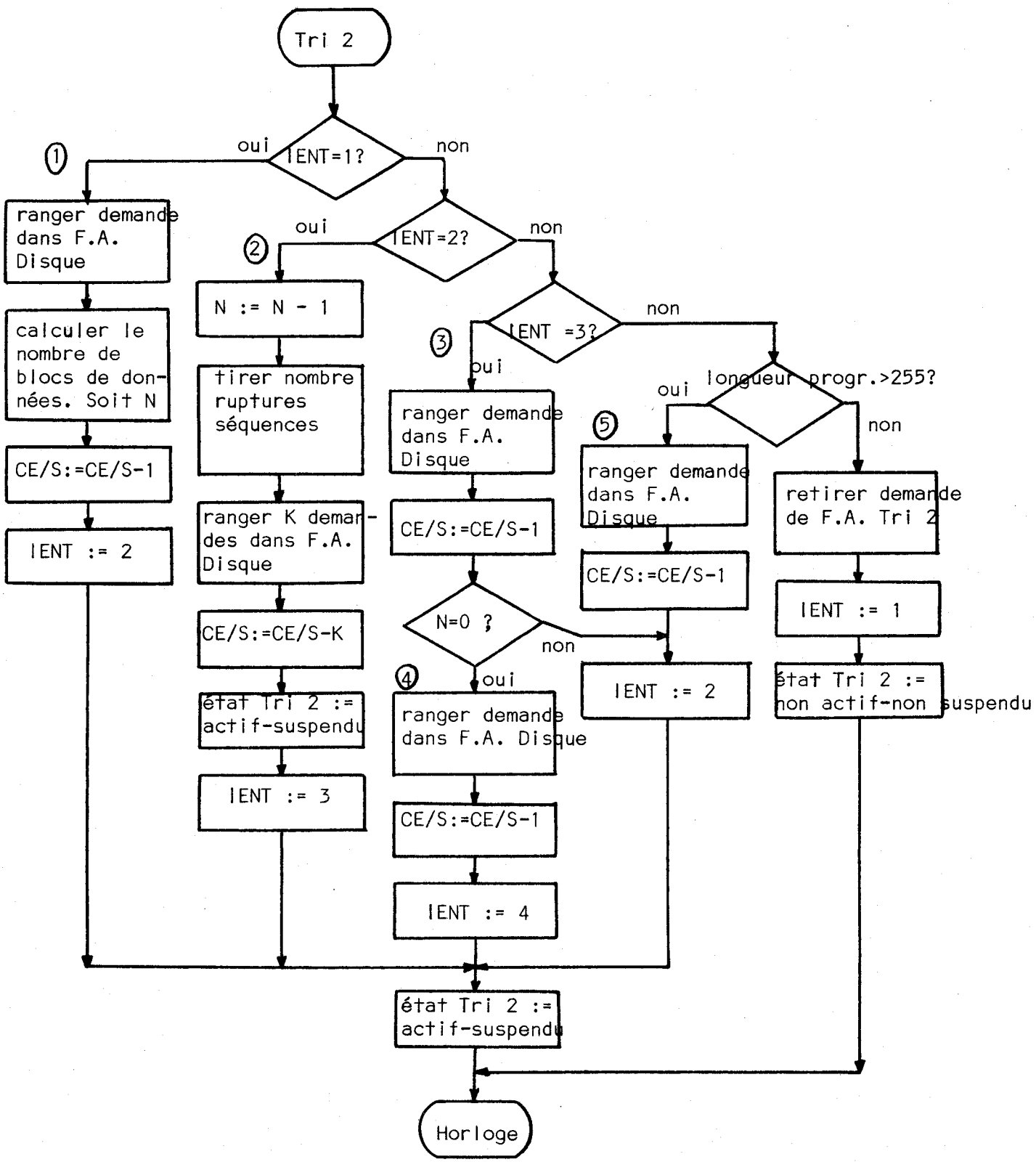

Pr. Disque
disque actif ?
oui
non
Sortie en cours?
oui
non
Disque
Horl.>DFT?
oui
non
∃ procédure active ?
oui
non
retirer demande traitée de F.A. Disque
Sortie non en cours
∃ procédure suspendue ?
non
Moniteur
oui
F.A. Disque vide ?
oui
non
prendre en compte 1ère demande dans F.A. Disque
calcul rotation disque
calculer temps transfert
Moniteur
DFT := date fin transfert
est-ce une sortie?
non
oui
non
DFT>seconde courante?
oui
sortie en cours
Horloge=Seconde
mise à jour horloge
calcul temps résiduel de transfert
vidage FA Disque CE/S:=CE/S+1
disque actif pour une procédure
disque inactif
On termine le transfert demandé par la procédure active


Moniteur
∃ procédure active oui & (Horloge=seconde)?
Horloge
non
∃ procédure active non suspendue ?
oui
non
∃ procédure suspendue?
oui
non
oui
CE/S=0?
non
active non suspendue
Horloge
F.A.Analyse vide ?
oui
non
non
F.A. Tri 1 vide ?
oui
non
F.A.Tri 2 vide ?
oui
procédure précédente ?
oui
non
Défaut
renvoi à la procédure concernée
Analyse
Tri 1
Tri 2
Sortie en cours & F.A. Disque vide?
oui
non
non
D = 0 ?
oui
Ajuster horloge sur D
Horloge=minute
calcul rotation disque
Horloge


Défaut
IENT = 1?
tirer adresse sur disque
procédure en cours :=procédure demandée
ranger dans F.A. Disque :
- adresse
- longueur
IENT := 1
demande chargement procédure
compteur demandes échanges disque
CE/S = CE/S-1
état procédure Défaut = active suspendue
IENT := 2
Horloge


Analyse
chargement des données
IENT=1?
non
tirer adresse sur disque
oui
IENT=2?
non
ranger demande dans F.A. Disque
calcul temps analyse
oui
IENT=3?
non
Retirer la demande traitée de F.A. Analyse
CE/S:=CE/S-1
IENT := 3
état Analyse := non actif non suspendu
tat Analyse := actif suspendu
oui
date fin analyse >seconde courante ?
non
IENT := 1
IENT := 2
Horloge := seconde courante
mettre à jour horloge
calcul temps résiduel analyse
tirer adresses sur disque
ranger 2 demandes dans F.A. Disque
CE/S:=CE/S-2
état Analyse := actif-suspendu
IENT := 4
calculer rotation disque
rangement sur disque de la pile et de la ligne de programme
Disque
oui
sortie en cours?
non
Horloge


Tri 1
IENT = 1?
oui
non
demande premier bloc de clés
ranger demande dans F.A.Disque CE/S:=CE/S-1
IENT=2?
oui
non
calculer temps de Tri
IENT:=3
IENT = 3?
oui
non
ranger demande dans F.A.Tri 2
longueur progr. >255 ?
oui
non
ranger demande dans F.A. Disque CE/S:=CE/S-1
demande du deuxième bloc de clés éventuellement
IENT:=2
état Tri 2 := actif suspendu
fin de Tri >seconde courante?
oui
non
retirer demande traitée de F.A. Tri 1
état Tri 1 := non actif non suspendu
mise à jour horloge
ranger demande (s) dans F.A.Disque
CE/S:=CE/S-1 (ou -2)
état Tri 1:= actif et suspendu
IENT := 4
renvoi des blocs de clés sur disque
Horloge:=seconde courante
calculer temps résiduel Tri1
calculer rotation Disque
Sortie en cours?
oui
Disque
non
Horloge



① Lecture du premier bloc de clés

② Simulation de la fusion des monotonies

③ Renvoi sur disque du bloc de données rangé

④ Renvoi sur disque du bloc de clés (les adresses des articles ont été modifiées à la suite de 2)

⑤ Lecture éventuelle du deuxième bloc de clés.

BIBLIOGRAPHIE

[1] L. SIRET, M. BELLOT et J.P. VERJUS
DIAMAG 2. R.I.R.O. n°12, 1968, p. 3-44.

[2] Herbert B. BARKIN et autres
A modular Computer Sharing System. C.A.C.M. vol.12, octobre 1969, p. 551-559.

[3] L. KLEINROCK Certain analytic results for time-shared processors.
IFIP 1968, Booklet D, Hardware 1.

[4] INTERTECHNIQUE Entrées/Sorties Multi 8.

[5] INTERTECHNIQUE Multi 8 / M 304 Coupleur pour lignes asynchrones basse vitesse - M 606. Réf. M8/021 F.

[6] D. LANCIAUX Etude préliminaire à la liaison DS 20 - M 40.
Laboratoire de Calcul LILLE - 1970 - Mémoire de DEA.

[7] D. LANCIAUX - NGUYEN
Liaison DS 20 - M 40. Laboratoire de Calcul LILLE - 1972 - Rapports internes.

[8] C.I.I. Ordinateur 10070 - Manuel d'exploitation. NT 2444 - 2/Fr.

[9] INTERTECHNIQUE Multi 8. Canal d'accès direct à la mémoire. M 601.
réf. M8/006F.

[10] INTERTECHNIQUE Multi 8. Coupleur de disque. M 614. réf. M8/019F.

[11] E.W. DIJKSTRA Co-operating sequential processes.
Dans Programming Languages. Academic Press (1968).

E.W. DIJKSTRA The structure of the T.H.E multiprogramming system.
C.A.C.M. vol. 11, Mai 1968, p. 341-346.

[12] J. FERRIE ESOPE - Gestion des processus.
Thèse de docteur-ingénieur. Faculté des Sciences de Toulouse. 1971.

J. MOSSIERE **ESOPE - Allocation de ressources.**
Thèse de troisième cycle. Faculté des Sciences de Paris. 1971.

[13] Barbara S. BRAWN, Frances G. GUSTAVSON, Effrem S. MANKIN
Sorting in a paging environment. C.A.C.M. vol. 13, Août 1970, p. 483-494.

[14] P.J. DENNING **The Working Set model for program behavior.** C.A.C.M. vol.11, Mai 1968, p. 323-333.

[15] J.B. MORRIS **Demand paging through utilization of Working Sets on the MANIAC II.** C.A.C.M. vol.15, octobre 1972, p.867-872.

[16] C. KAISER **Conception et réalisation de système à accès multiple : Gestion du parallélisme.** Note de travail ESOPE A23 Février 1973.

[17] B. ARDEN, D. BOETTNER
Measurement and performance on a multiprogramming system.
Second Symposium on Operating Systems Principles - Princeton 1969.

[18] J. DENEL, M. COQUET
Approximation de polyèdres dans R^n. Laboratoire de Calcul - 1969.
Mémoire de DEA.