Simulation d'un système conversationnel

Joëlle Coutaz

N° d'ordre

THESE

présentée à

LA FACULTE DES SCIENCES DE L'UNIVERSITE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

"Informatique"

par

Joëlle COUTAZ

Simulation d'un système conversationnel

Thèse soutenue le 4 Décembre 1970 devant la commission d'examen :

MM	N. GASTINEL	Président
	L. BOLLIET	Examinateur
	M. DREYFUS	Examinateur
	M. GRIFFITHS	Examinateur

L I S T E D E S P R O F E S S E U R S

Doyen honoraire : Monsieur M. MORET
Doyen : Monsieur E. BONNIER

PROFESSEURS TITULAIRES

MM.	NEEL Louis	Physique Expérimentale
	KRAVTCHENKO Julien	Mécanique Rationnelle
	CHABAUTY Claude	Calcul différentiel et intégral
	BENOIT Jean	Radioélectricité
	CHENE Marcel	Chimie Papetière
	FELICI Noël	Electrostatique
	KUNTZMANN Jean	Mathématiques Appliquées
	BARBIER Reynold	Géologie Appliquée
	SANTON Lucien	Mécanique des Fluides
	OZENDA Paul	Botanique
	FALLOT Maurice	Physique Industrielle
	KOSZUL Jean-Louis	Mathématiques
	GALVANI Octave	Mathématiques
	MOUSSA André	Chimie Nucléaire
	TRAYNARD Philippe	Chimie Générale
	SOUTIF Michel	Physique Générale
	CRAYA Antoine	Hydrodynamique
	REULOS René	Théorie des Champs
	BESSON Jean	Chimie Minérale
	AYANT Yves	Physique Approfondie
	GALLISSOT François	Mathématiques
Melle.	LUTZ Elisabeth	Mathématiques
MM.	BLAMBERT Maurice	Mathématiques
	BOUCHEZ Robert	Physique Nucléaire
	LLIBOUTRY Louis	Géophysique
	MICHEL Robert	Minéralogie et pétrographie
	BONNIER Etienne	Electrochimie et Electrométallurgie
	DESSAUX Georges	Physiologie animale
	PILLET Emile	Physique Industrielle-Electrotechnique
	YOCCOZ Jean	Physique Nucléaire théorique
	DEBELMAS Jacques	Géologie Générale
	GERBER Robert	Mathématiques
	PAUTHENET René	Electrotechnique
	MALGRANGE Bernard	Mathématiques Pures
	VAUQUOIS Bernard	Calcul Electronique
	BARJON Robert	Physique Nucléaire

MM.	BARBIER Jean-Claude	Physique
	SILBER Robert	Mécanique des Fluides
	BUYLE-BODIN Maurice	Electronique
	DREYFUS Bernard	Thermodynamique
	KLEIN Joseph	Mathématiques
	VAILLANT François	Zoologie et Hydrobiologie
	ARNAUD Paul	Chimie
	SENGEL Philippe	Zoologie
	BARNOUD Fernand	Biosynthèse de la Cellulose
	BRISSONNEAU Pierre	Physique
	GAGNAIRE Didier	Chimie Physique
Mme.	KOFLER Lucie	Botanique
MM.	DEGRANGE Charles	Zoologie
	PEBAY-PEROULA Jean-Claude	Physique
	RASSAT André	Chimie Systématique
	DUCROS Pierre	Cristallographie Physique
	DODU Jacques	Mécanique Appliquée I. U. T.
	ANGLES D'AURIAC Paul	Mécanique des Fluides
	LACAZE Albert	Thermodynamique
	GASTINEL Noël	Analyse numérique
	GIRAUD Pierre	Géologie
	PERRET René	Servo-mécanisme
	PAYAN Jean-Jacques	Mathématiques Pures

PROFESSEURS SANS CHAIRE

MM.	GIDON Paul	Géologie
Mme.	BARBIER Marie-Jeanne	Electrochimie
Mme.	SOUTIF Jeanne	Physique
	COHEN Joseph	Electrotechnique
	DEPASSEL R.	Mécanique des Fluides
	GLENAT René	Chimie
	BARRA Jean	Mathématiques Appliquées
	COUMES André	Electronique
	PERRIAUX Jacques	Géologie et Minéralogie
	ROBERT André	Chimie Papetière
	BIARREZ Jean	Mécanique Physique
	BONNET Georges	Electronique
	CAUQUIS Georges	Chimie Générale
	BONNETAIN Lucien	Chimie Minérale
	DEPOMIER Pierre	Physique Nucléaire-Génie Atomique
	HACQUES Gérard	Calcul numérique
	POLOUJADOFF Michel	Electrotechnique
Mme.	KAHANE Josette	Physique
Mme.	BONNIER Jane	Chimie
MM.	VALENTIN Jacques	Physique
	REBECQ Jacques	Biologie
	DEPORTES Charles	Chimie
	SARROT-REYNAULD Jean	Géologie
	BERTRANDIAS Jean-Paul	Mathématiques Appliquées
	AUBERT Guy	Physique

PROFESSEURS ASSOCIES

MM.	RODRIGUES Alexandre	Mathématiques Pures
	MORITA Susumu	Physique Nucléaire
	RADHAKRISHNA	Thermodynamique

MAITRES DE CONFERENCES

MM.	LANCIA Roland	Physique Atomique
Mme.	BOUCHE Liane	Mathématiques
MM.	KAHANE André	Physique Générale
	DOLIQUE Jean Michel	Electronique
	BRIERE Georges	Physique
	DESRE Georges	Chimie
	LAJZEHOWICZ Joseph	Physique
	LAURENT Pierre	Mathématiques Appliquées
Mme.	BERTRANDIAS Françoise	Mathématiques Pures
MM.	LONGEQUEUE Jean-Pierre	Physique
	SOHM Jean-Claude	Electrochimie
	ZADWORNY François	Electronique
	DURAND Francis	Chimie Physique
	CARLIER Georges	Biologie végétale
	PFISTER Jean-Claude	Physique
	CHIBON Pierre	Biologie animale
	IDELMAN Simon	Physiologie animale
	BLOCH Daniel	Electrotechnique I. P.
	MARTIN-BOUYER Michel	Chimie (C. S. U. Chambéry)
	SIBILLE Robert	Construction mécanique (I. U. T.)
	BRUGEL Lucien	Energétique I. U. T.
	BOUVARD Maurice	HYdrologie
	RICHARD Lucien	Botanique
	PELMONT Jean	Physiologie animale
	BOUSSARD Jean-Claude	Mathématiques Appliquées (I. P. G.)
	MOREAU René	Hydraulique I. P. G.
	ARMAND Yves	Chimie I. U. T.
	BOLLIET Louis	Informatique I. U. T.
	KUHN Gérard	Energétique I. U. T.
	PEFFEN René	Chimie I. U. T.
	GERMAIN Jean-Pierre	Mécanique
	JOLY Jean-René	Mathématiques Pures
Melle.	PIERY Yvette	Biologie animale
	BERNARD Alain	Mathématiques Pures
	MOHSEN Tahsin	Biologie (C. S. U. Chambéry)
	CONTE René	Mesures Physiques I. U. T.
	LE JUNTER Noël	Génie Electrique Electronique I. U. T.
	LE ROY Philippe	Génie Mécanique I. U. T.
	ROMIER Guy	Techniques Statistiques quantitatives I. U. T.
	VIALON Pierre	Géologie
	BENZAKEN Claude	Mathématiques Appliquées
	MAYNARD Roger	Physique

MM.	DUSSAUD René	Mathématiques (C. S. U. Chambéry)
	BELORIZKY Elie	Physique (C. S. U. Chambéry)
Mme.	LAJZEROWICZ Jeannine	Physique (C. S. U. Chambéry)
M.	JULLIEN Pierre	Mathématiques Pures
Mme.	RINAUDO Marguerite	Chimie
MM.	BLIMAN Samuel	E. I. E.
	BEGUIN Claude	Chimie Organique
	NEGRE Robert	I. U. T.

MAITRES DE CONFERENCES ASSOCIES

MM.	YAMADA Osamu	Physique du Solide
	NAGAO Makoto	Mathématiques Appliquées
	MAREZIO Massimo	Physique du Solide
	CHEECKE John	Thermodynamique
	BOUDOURIS Georges	Radioélectricité
	ROZMARIN Georges	Chimie Papetière

à mes parents, à Gerard,

Je tiens à remercier :

Monsieur N. GASTINEL, *Professeur à l'Université de GRENOBLE qui m'a fait l'honneur de présider le jury de Thèse,*

Monsieur L. BOLLIET, *Professeur à l'Institut Universitaire de Technologie de GRENOBLE qui m'a accordé sa confiance et son soutien tout au long de ma recherche,*

Monsieur M. DREYFUS, *Directeur technique du Centre de Calcul de l'Institut National d'Astronomie et de Géophysique de MEUDON, qui a aimablement accepté de participer au jury,*

Monsieur M. GRIFFITHS, *Maître de Conférence à la Faculté des Sciences de GRENOBLE pour l'intérêt sympathique qu'il a porté à mon travail, et les conseils éclairés qu'il n'a pas hésité à me donner.*

Je voudrais rendre hommage à Monsieur B. VAUQUOIS, *Directeur du Centre d'Etudes pour la Traduction Automatique, et* Monsieur J.C. BOUSSARD, *Maître de Conférence à la Faculté des Sciences de GRENOBLE, qui ont su me transmettre leur passion pour l'Informatique et leur goût de la recherche.*

Je souhaite également remercier Daniel DELPUECH et Jean-Claude CHUPIN *pour leur brillante et amicale collaboration ; et* Georges FAFIOTTE *qui a eu le patient courage d'une première lecture.*

Je tiens à remercier tous mes collègues du Laboratoire pour leur soutien chaleureux, et la gentillesse qu'ils m'ont témoignée.

Je dois enfin remercier les SERVICES DE DACTYLOGRAPHIE et de REPRODUCTION *à qui je dois la réalisation matérielle de cet ouvrage.*

TABLE DES MATIERES

C H A P I T R E I

INTRODUCTION A LA SIMULATION

1 -HISTORIQUE ET DEFINITION

1.1 Historique.

La simulation est devenue en quelques années le fait de toute démarche analytique ou décisionnelle; son origine remonte pourtant à l'Antiquité et s'explique par le désir croissant des hommes de se donner un moyen de prédire le futur. Avant le XVII^{e} siècle, cette recherche reposait sur les méthodes purement déductives des philosophes tels que PLATON, ARISTOTE, EUCLIDE. A cette méthodologie taxée de "philosophie spéculative" se substitue à partir du XVII^{e} siècle une philosophie scientifique introduite par Sir Francis BACON dans laquelle les méthodes de la logique inductive s'ajoutent à celles de la logique déductive. La méthode scientifique comporte quatre étapes :

1/ Observation du phénomène ou système physique

2/ Formulation d'une hypothèse,

3/ Prédiction du comportement du système par déduction logique à partir de cette hypothèse,

4/ Réalisation de diverses expériences pour tester la validité de cette hypothèse : En termes "modernes", ceci s'énonce de la manière suivante :

1) observation et analyse du système réel,
2) formulation d'un modèle qui traduit les observations faites sur le système,
3) prédiction du comportement du système en utilisant le modèle,
4) expériences de tests du modèle.

1.2 Définitions

Nous venons de présenter le modèle comme la représentation d'un phénomène : il contient tous les facteurs intéressant une recherche dans un

but déterminé.

Nous pouvons maintenant définir la simulation comme la représentation dynamique d'un modèle. De manière plus complète, le mot simulation recouvre un ensemble de traitements qui consistent à représenter à l'aide d'un modèle le phénomène réel à étudier.

La formulation que nous venons de donner des principes de la méthode scientifique propose aussi les quatre phases de la conduite d'une simulation.

La première phase est la décomposition du phénomène en un certain nombre de caractéristiques, telles que relations de cause à effet. La phase de construction demande le développement d'un ensemble d'opérateurs fonctionnels permettant de représenter tous les résultats de l'étude précédente. La phase d'expérimentation consiste à tester le modèle en suivant le déroulement d'une ou plusieurs expériences.

2 -NECESSITE DE LA SIMULATION

Lors de la conception d'un système, de quelque nature que ce soit (scientifique, technologique, économique...) on rencontre très tôt, le plus souvent dès le stade de l'avant projet et de toute façon bien avant que le prototype fonctionne, le besoin de prédire quels seront son comportement, ses performances. Cette fonction de prédiction peut viser plusieurs objectifs. Ou bien l'on désire une évaluation qui indique les avantages, les insuffisances, le degré d'utilité d'un système, ou bien l'on veut faire une comparaison entre plusieurs systèmes ou plusieurs versions d'un même projet, ou encore l'on souhaite optimiser un système d'après certains critères par un choix judicieux de paramètres ou par le choix d'une certaine configuration.

Tout ceci peut être réalisé, plus particulièrement lorsqu'on se trouve dans l'une des situations suivantes :

- Système vaste, présentant de grandes difficultés pour l'évaluation des interactions des différentes parties entre elles,

- modèle mathématique trop lourd ou trop compliqué pour être exploité analytiquement,
- impossibilité de tester le comportement du système réel, soit qu'il n'existe pas, soit qu'une telle méthode soit trop couteuse,
- système mettant en jeu un grand nombre de phénomènes aléatoires,
- essais trop longs sur le système réel.

La simulation sur calculateur s'impose alors par la variété des opérations arithmétiques et logiques, la capacité, la souplesse d'utilisation des modèles et la rapidité de traitement qu'offrent les ordinateurs.

On trouvera dans les chapitres qui suivent l'application de la simulation à l'étude d'un système conversationnel et dans le paragraphe 3 la présentation de cette étude.

3 -SIMULATION D'UN SYSTEME CONVERSATIONNEL : INTRODUCTION.

3.1 Déffnition d'un système conversationnel.

Les tendances actuelles dans le développement des systèmes d'exploitation s'orientent vers l'amélioration des services rendus à l'utilisateur. Après les divers systèmes "BATCH", se développent les systèmes conversationnels face auxquels les utilisateurs sont en liaison directe avec l'ordinateur par l'intermédiaire d'un téléimprimeur (console-terminal) : il s'établit un dialogue entre l'utilisateur et l'ordinateur, les ressources de ce dernier se trouvent partagées entre un certain nombre d'utilisateurs. La fonction du système est d'exécuter les programmes proposés simultanément par les utilisateurs et de donner à chacun d'entre eux une réponse dans un temps convenable.

Une technique largement employée pour réaliser le partage des ressources de l'ordinateur est le partage de temps(time-sharing) qui consiste à exécuter chaque programme utilisateur dans un certain ordre, pendant un certain temps, mais pas nécessairement jusqu'à l'achèvement complet du programme.

En l'occurrence la Partition conversationnelle à laquelle nous nous interessons * déroule un programme utilisateur jusqu'à ce que celui-ci lance un ordre d'entrée sortie. Son exécution est alors interrompue, puis reprendra à une date ultérieure à l'endroit où elle avait été arrêtée. La réalisation de ceci implique à un instant donné le rangement de l'état du programme interrompu et la restauration du programme quand l'exécution reprend. Le processus suppose donc tout un mécanisme relativement complexe de gestion des ressources du conversationnel.

3.2 Buts de la simulation de la partition conversationnelle.

Ce système conversationnel est encore à l'état de projet. Comme nous l'avons précédemment expliqué il est intéressant de prévoir son comportement, par exemple en fonction du nombre d'utilisateurs et de la configuration hardware. Il est effectivement utile que le constructeur connaisse les capacités du système qu'il propose à ses clients. Comment peut-il par exemple, sinon par intuition, affirmer que telle configuration accepte raisonnablement x téléimprimeurs ? La simulation est ici devant la complexité du système, la seule technique prédictive à pouvoir donner une réponse signifiante.

3.3. Comment réaliser cette simulation ?

Deux approches sont possibles pour la réalisation d'une telle simulation : construire un modèle de simulation microscopique ou encore utiliser une représentation analytique à l'aide d'un modèle de Markov.

3.3.1 Modèle de simulation

3.3.1.1. Les modèles de simulation déjà existants ont pour la plupart été construits pour mener à bien une étude particulière appliquée à un système conversationnel précis. Tel est le cas de la simulation proposée par A.SCHERR pour CTSS [Ref 17] ou par FINE et Mc ISAAC pour "SDC time Sharing system "[Ref 6] . Bien que ces modèles aient atteint leurs buts, ils

* l'une des 3 partitions (série, temps réel, parallèle) du système SIRIS II dans le cadre d'un contrat passé entre la C.I.I. et l'Institut de Mathématiques Appliquées de GRENOBLE.

ne conviennent plus à l'étude des performances des nouveaux systèmes à temps partagé en raison de leur complexité croissante. Seul, Nielsen -de l'Université de STANFORD- a construit un modèle d'application générale -Utilisable pour l'étude de la plupart des systèmes à temps partagé; Ce modèle peut être employé aussi pour la mise au point et le développement de nouveaux algorithmes ou de nouvelles techniques relatives aux exploitations à temps partagé. Ce modèle, écrit en FORTRAN, se veut efficace et simule entre 2 et 3 minutes d'opérations du TSS 360/67 en une minute de 360/50. [Ref 15] .

D'autres comme Jesse KATZ du centre scientifique d'IBM de LOS ANGELES s'est intéressé à la simulation du système 360 pris dans son ensemble avec tous ses concepts [Ref 8] . Il en donne une description intéressante, mais qui ne concerne pas notre problème particulier. J.KATZ a par ailleurs simulé un système multiprocessor également d'application générale mais qui a surtout servi à faciliter l'étude de la performance du système en fonction des travaux d'entrée. [Ref 9] .

3.3.1.2. Nous proposons à notre tour CASSCOU (Conception d'un Algorithme de Simulation des Systèmes Conversationnels sur OrdinateUr); modèle que nous voulons efficace, et auquel nous avons donné un caractère général, en sorte que son domaine d'application s'étende à la représentation des systèmes conversationnels simples. La partition conversationnelle a été découpée en un certain nombre de ressources, chacune étant représentée par un sous-programme PL/I et gérées par un programme principal d'ordonnancement. Les diverses requêtes et demandes d'exploitation des ressources par les utilisateurs sont rangées dans d s files d'attente organisées suivant une loi de priorité et suivant l'heure d'arrivée.

3.3.2. Représentation analytique : modèle de Markov.

Nous donnons ici l'exemple d'un modèle simple proposé par Alan SCHERR et qui reflète le système à temps partagé, CTSS [Ref. 17] .

Soit n, le nombre d'utilisateurs pouvant avoir accès simultanément au système conversationnel . L'état du processus qui représente CTSS est

déterminé par le nombre d'utilisateurs qui demandent un service du système : l'état j indique que j utilisateurs demandent un service. Ainsi, le processus de MARKOV possède n+1 états. De façon à utiliser un processus de MARKOV CONTINU, les distributions du temps d'exécution et du temps d'utilisation de l'unité centrale par commande-utilisateurs sont exponentielles. Le temps moyen d'utilisation de la console est T; le temps moyen d'utilisation de l'Unité centrale est P.

Donc, si j utilisateurs attendent un service, le taux de passage à l'état où j+1 utilisateurs attendent un service est (n-j)/T; en d'autres termes, n-j utilisateurs sont en train d'utiliser leur console (reçoivent des messages ou préparent la prochaine commande à envoyer). Le taux de passage entre l'état j et l'état j-1 est 1/p. (chacun des j utilisateurs reçoit 1/J des capacités de l'unité centrale).

La matrice réunissant les taux de transition est la suivante : (a_{ij} = taux de transition de l'état i à l'état j). [Ref 34] , [Ref 3] , [Ref 4] .

$$A = \begin{bmatrix} -\frac{n}{T} & \frac{n}{T} & 0 & 0 & 0\ldots\ldots & & \\ \frac{1}{p} & -(\frac{1}{p}+\frac{n-1}{T}) & \frac{n-1}{T} & 0 & 0 & & \\ 0 & \frac{1}{p} & -(\frac{1}{p}+\frac{n-2}{T}) & \frac{n-2}{T} & 0 & & \\ 0 & 0 & & & & & \\ & & & & \frac{1}{p} & -(\frac{1}{p}+\frac{3}{T}) & \frac{3}{T} & 0 & 0 \\ & & & & 0 & \frac{1}{p} & -(\frac{1}{p}+\frac{2}{T}) & \frac{2}{T} & 0 \\ & & & & 0 & 0 & \frac{1}{p} & -(\frac{1}{p}+\frac{1}{T}) & \frac{1}{T} \\ & & & & 0 & 0 & 0 & \frac{1}{p} & -\frac{1}{p} \end{bmatrix}$$

Soit $\pi = \{ \pi_o, \pi_1 \ldots \pi_n \}$

le vecteur de probabilité d'occupation du système :

Alors $\pi A = 0$

et

$$-\frac{n}{T}\pi_o + \frac{1}{p}\pi_1 = 0$$

$$\frac{n}{T}\pi_o - \frac{1}{p}\pi_1 - \frac{n-1}{T}\pi_1 + \frac{1}{p}\pi_2 = 0$$

qui conduit à la formule générale : $\pi_i = \frac{n!}{(n-i)!}\left(\frac{p}{T}\right)^i \pi_o$

Sachant que $\sum_{i=0}^{n} \pi_i = 1$ et en posant $p/T = r$ on obtient :

$$\pi_o = \frac{1}{\sum_{j=o}^{n} \frac{n!}{(n-j)} r^j} \quad \text{et} \quad \pi_i = \frac{\frac{n!}{(n-i)!} r^i}{\sum_{j=0}^{n} \frac{n!}{n-j\,!} r^j}$$

A partir de là déterminons t le temps de réponse moyen; Si Q est le nombre moyen d'utilisateurs attendant un service, $Q = n\frac{t}{t+T} = \sum_{i=0}^{n} i\,\pi_i$

alors :

$$n\frac{t}{t+T} = \frac{\sum_{i=0}^{n} \frac{in!}{(n-i)!} r^i}{\sum_{i=0}^{n} \frac{n!}{(n-i)!} r^i}$$

la résolution de cette équation en t conduit à la solution $\boxed{t = \frac{nP}{(1-\pi_o)} - T}$

π_o est la probabilité pour que le système soit inactif.

Ce modèle analytique permet donc d'obtenir un temps de réponse moyen de manière simple et efficace.

SMITH a utilisé lui aussi [Ref 19] un processus de Markov continu pour la représentation d'un système doté d'un mécanisme de pagination et son modèle a fait ses preuves, même dans des situations relativement complexes.

Néanmoins, ces modèles ont un domaine d'application relativement limité et ne sont pas suffisamment souples pour permettre des modifications à l'intérieur du système avec un effort minime.

Par ailleurs, cette approche suppose la connaissance d'un certain nombre de données (ici P et T) qui sont déterminées à l'aide des mesures réalisées sur le système réel. Notre partition conversationnelle, encore à l'état de projet, se prête assez mal à l'utilisation d'un modèle de Markov. Pour toutes ces raisons nous nous sommes délibérément engagés dans une voie plus séduisante, celle de la simulation pure.

Nous rendrons compte de cette approche de la manière suivante :

Après la présentation de l'élaboration d'un modèle de simulation faite dans le chapitre II, nous proposons dans le chapitre III une description détaillée de notre modèle, CASSCOU. Nous consacrons le chapitre IV à la représentation du flux du temps, élément fondamental de la dynamique d'un modèle de simulation. Nous analyserons ensuite au chapitre V les résultats des expériences réalisées à l'aide de CASSCOU. Nous avons utilisé PL/I, langage algorithmique général pour la mise en oeuvre de notre modèle; nous donnons les raisons de ce choix dans le chapitre III, mais il est utile de présenter puis de comparer quelques langages particuliers spécialement conçus pour la simulation dont l'existence est motivée par l'aide appréciable qu'ils apportent au programmeur (chapitre VI).

C H A P I T R E II

L'ELABORATION D'UN MODELE

L'expérience suggère d'articuler l'élaboration d'un modèle de simulation sur les sept points suivants (voir fig.1) : [Ref 14]

1- FORMULATION DU PROBLEME,

2- RECUPERATION DES DONNEES DU MONDE REEL,

3- FORMULATION DU MODELE MATHEMATIQUE,

4- EVALUATION DU MODELE,

5- FORMULATION DU PROGRAMME DE SIMULATION,

6- VALIDATION,

7- DESCRIPTION DES EXPERIENCES ET INTERPRETATION DES RESULTATS.

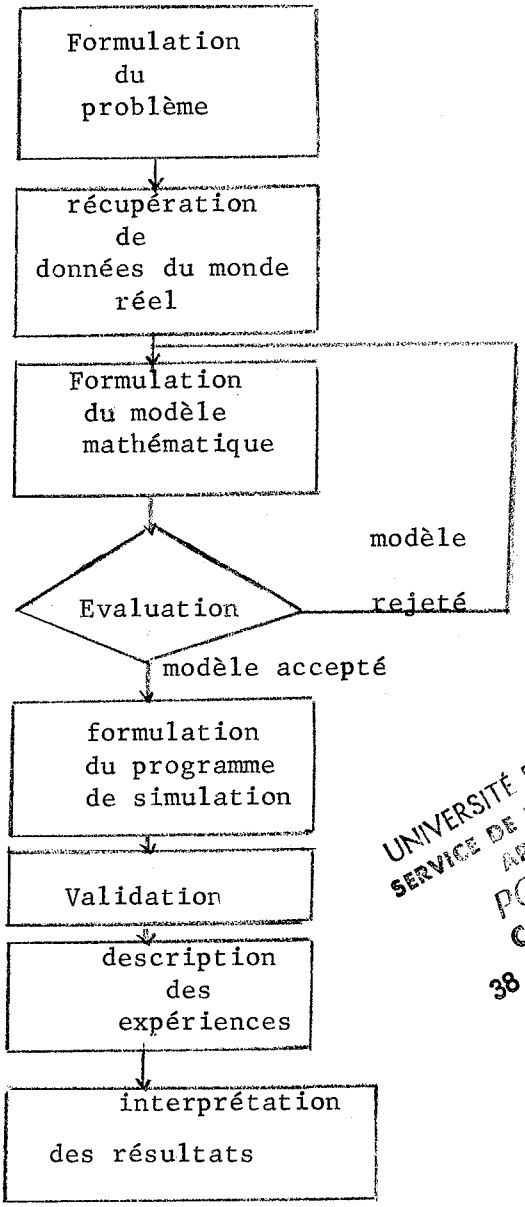


Fig.1 L'enchaînement des étapes pour l'élaboration d'un modèle de simulation.

Ce chapitre ne prétend pas faire autorité sur la manière de bien construire un modèle, mais présente la suite logique des étapes de toute recherche en simulation. Il doit être considéré à peu près comme une "CHECKLIST" servant à indiquer les difficultés essentielles et les facteurs importants à prendre en considération.

1 -FORMULATION DES OBJECTIFS

La simulation est une recherche coûteuse, ce qui nous incite à éviter d'en faire uniquement pour satisfaire un besoin intellectuel. Il faut donc veiller à ce que cette recherche aboutisse à un résultat concret, atteignant les objectifs que nous avions clairement définis dès le départ. Car là est le premier pas : fixer des objectifs pour répondre à un ensemble de questions précises, pour confirmer la validité de certaines hypothèses ou encore pour estimer les effets de certaines modifications sur le comportement du modèle.

Dans l'étude des systèmes digitaux, les questions qui reviennent le plus fréquemment portent sur :

- le temps de réponse que nous définissons comme le temps qui s'écoule entre l'instant où l'utilisateur lance une commande et l'instant où le système lui donne une réponse (impression de messages, de résultats ou déblocage de la console). Le temps de réponse est différent du temps de réaction du système qui est le temps qui s'écoule entre l'instant où l'utilisateur lance une commande et l'instant où le système prend cette demande en considération.

- le pourcentage d'activité ou d'inactivité des éléments de la configuration et plus particulièrement de l'Unité centrale et des canaux,

- le nombre maximal d'utilisateurs en ligne dans un système conversationnel, pour lequel les temps de réponses restent acceptables,

- le choix du système d'exploitation qui répond le mieux aux besoins courants des utilisateurs,

- etc....

Il n'est pas suffisant de se borner à formuler ces questions : il est nécessaire aussi de préciser les critères qui permettront d'évaluer les réponses possibles. Nous devons par exemple définir avec exactitude ce que nous entendons par temps de réponse "convenable" ou par besoins courants "mieux" satisfaits des utilisateurs, de façon à déterminer dans la troisième question

le nombre maximum d'utilisateurs, ou à décider lequel, entre plusieurs systèmes d'exploitation, satisfait les exigences des utilisateurs.

Le second aspect des objectifs que se propose la simulation est la vérification d'hypothèses inspirées parfois par les résultats d'une précédente simulation. On peut avoir conçu un algorithme de pagination à priori satisfaisant, et qui se révèle au jour de la simulation tout à fait inéfficace parce qu'il se veut sans doute trop rigoureux. Les mêmes remarques sont valables pour les algorithmes d'agencement des travaux dans lesquels s'imbrique tout un système de priorité qu'il est bon d'évaluer.

L'étude des effets de certaines modifications sur le comportement d'un système est souvent liée à la vérification d'hypothèses, comme le montre l'expérience que Nielsen a faite dans sa simulation de systèmes à temps partagé. [Ref 15] ; les résultats d'une précédente simulation dénonçaient l'existence d'un goulot d'étranglement au niveau des disques de pagination. Une attitude raisonnable consistait à mesurer les performances du système en affectant aux unités de disques de pagination des canaux distincts. Placé dans ces conditions le système se révéla efficace mais certains résultats ne répondirent pas aux espérances de Nielsen : le goulot d'étranglement avait disparu, mais l'inactivité de l'Unité Centrale avait cru de manière inacceptable. Nielsen fit donc évoluer la configuration de son système jusqu'à ce qu'il trouvât une solution satisfaisante. Nous mesurons là la souplesse d'utilisation d'un programme de simulation quand celui-ci est bien conçu. Il est certain que l'élaboration d'un programme de simulation est longue et couteuse ; mais l'exploitation astucieuse de ce programme permet de réaliser les plus grands rêves en matière de configurations idéales du système; configuration dont la réalisation serait rendue impossible par les contraintes qu'imposent des moyens financiers limités.

Ainsi, avant d'entreprendre tout travail de recherche en simulation nous devons en premier lieu définir les raisons et les objectifs de cette recherche, déterminer ensuite les critères permettant d'évaluer le degré auquel les objectifs seront remplis par l'expérimentation. Il se peut d'ailleurs qu'après avoir précisé ces éléments nous décidions d'abandonner la simulation devant l'impossibilité d'atteindre les objectifs désirés ou plus simplement

pour des raisons de coût prohibitif ou de complexité excessive.

2 -LA RECUPERATION DES DONNEES DU MONDE REEL

D'une manière générale, les données du monde réel servent à construire le modèle d'entrée du simulateur, à préciser les conditions initiales et à élaborer les fonctions de distribution.

La récupération de ces données, parfois simple et évidente, peut se montrer complexe ou fastidieuse. Certaines informations de base nécessaires à la simulation des systèmes digitaux, telles que les temps de positionnement des bras de disques sur les enregistrements désirés, la rapiditité des imprimantes et les lecteurs-perforateurs de cartes, la valeur du cycle de base du calculateur et d'une manière générale les performances des composants HARDWARE sont évalués par le constructeur.

Mais l'on se heurte à une difficulté essentielle dès que l'on désire estimer le SOFTWARE du système pour les besoins de la simulation en particulier. Dans ce cas, la seule façon sérieuse de procéder consiste à apporter certaines modifications au système d'exploitation réel que l'on désire simuler(à condition évidemment que celui-ci soit déjà opérationnel), de façon à récupérer des données réelles qui reflètent l'état du système et le profil des travaux qui ont été pris en compte. Mais il est bien évident que la récupération de ces données modifie l'efficacité du système. C'est pourquoi, l'essentiel est de s'assurer que les modifications que l'on entreprend n'auront pas ultérieurement d'incidence sensible sur le comportement du système. Une telle méthode demande donc beaucoup d'attention et n'est pas d'une justesse tout à fait absolue. Cependant, implémentée correctement, elle permet de récupérer des données qui donneront satisfaction, employées dans le contexte d'une simulation.

Si l'on ne dispose pas de système réel pour effectuer ces mesures, il reste l'évaluation intuitive(approximative), avec l'inconvénient majeur lié à sa nature, ou dans le cas le plus favorable l'utilisation de cette méthode sur un système réel proche du système que l'on désire simuler.

3 -FORMULATION DU MODELE MATHEMATIQUE

Cette formulation se fait en trois étapes :

1°/ Spécification des composants,

2°/ Spécification des variables et des paramètres,

3°/ Spécification des relations fonctionnelles.

Le processus qui consiste à observer le système réel, à formuler une ou plusieurs hypothèses sur la façon dont celui-ci fonctionne,et à réduire ces hypothèses à un degré d'abstraction qui permette la formulation du modèle mathématique correspondant, n'est en aucune façon un processus direct et immédiat.

Nous allons donc

(1) suggérer les caractéristiques désirables d'un modèle mathématique,

(2) souligner quelques unes des difficultés essentielles dans la construction d'un modèle et

(3) en spécifier les éléments de base.

L'une des premières considérations qui entre en jeu dans la formulation d'un modèle mathématique et la détermination du nombre des variables à y introduire. En général, on ne rencontre pas de difficultés à définir les variables de sortie puisqu'elles représentent les résultats de la simulation et que l'on a clairement défini dès le départ les objectifs de cette simulation. (Le nombre de ces variables est évidemment limité par la capacité mémoire de l'ordinateur). La difficulté réside réellement dans le choix des variables internes (dont certaines peuvent être stochastiques) qui affectent et déterminent les variables de sortie. La carence de variables internes engendre des modèles de simulation incomplets donc invalides et inexploitables. La surabondance de ces mêmes variables peut fournir des modèles complexes inéfficaces et coûteux, ou encore rendre la simulation impossible en raison de la capacité-mémoire limitée de l'ordinateur.

Une seconde considération important dans la formulation d'un modèle mathématique concerne la complexité du modèle. La structure des systèmes digitaux se révèle fort délicate à analyser; prétendre que les modèles mathématiques qui en décrivent le fonctionnement sont nécessairement complexes est une affirmation sensée, mais seulement dans une certaine mesure : si l'on pousse l'étude jusqu'à l'extrême on élabore des modèles qui 'collent' à la réalité mais qui nécessitent des temps d'exécution en mémoire centrale démentiels et tout à fait inadmissibles. De manière générale, nous cherchons à construire des modèles qui fournissent une description raisonnablement précise et qui prédisent le comportement du système réel donné tout en minimisant le coût d'exécution et de programmation. Les caractéristiques que nous venons de présenter ne sont pas indépendantes les unes des autres : le nombre de variables dans un modèle et sa complexité sont directement liées à sa validité, au temps nécessaire pour la mise au point, et au coût de l'exécution. En modifiant l'une des caractéristiques du modèle, nous modifions toutes les autres caractéristiques.

Une troisième point porte sur l'efficacité du modèle mathématique. Cette efficacité se mesure par le temps d'exécution nécessaire en mémoire centrale pour atteindre un objectif expérimental donné. En règle générale on cherche à minimiser la durée d'exécution des programmes qui simulent des phénomènes réels observés sur de longues périodes de temps (quelques journées, quelques mois ou quelques années selon le cas); on cherche concurremment à réduire le temps nécessaire à l'évaluation des paramètres statistiques en cherchant à atteindre un certain degré de précision.

Le quatrième élément qui intervient directement dans le "prix de revient" d'un modèle est le temps d'écriture et de mise au point du programme de simulation. Ce temps dépend essentiellement du degré de complexité du système à représenter, du nombre et de la nature des variables employées dans le modèle. Par ailleurs, la validité et la vitesse d'exécution d'un programme se paient souvent d'un effort supplémentaire de programmation. Dans le cas où les objectifs de la simulation acceptent quelques libertés vis à vis du fonctionnement du système réel, il est intéressant de formuler le modèle dans un langage spécialisé. Les gains ainsi réalisés en temps de programmation peuvent intégralement compenser la perte en efficacité résultant souvent de l'emploi d'un tel

langage spécialisé.

Un autre point d'intérêt porte surla validité du modèle , c'est-à-dire la "quantité de réalisme" que l'on a introduit : le modèle décrit-il de manière adéquate le comportement du système réel et les prédictions sont-elles raisonnables ? Si ces questions ne reçoivent pas de réponse positives, le modèle perd de son intérêt pratique; notre recherche devient alors exercice de logique déductive. Nous ne jugeons pas utile de revenir sur les rôles que jouent certaines caractéristiques du modèle (nombre de variables, complexité) dans l'appréciation de sa validité.

En dernière remarque, nous insisterons sur la nécessité de la comptabilité entre le modèle et les expériences que l'on désire réaliser. La simulation consiste à représenter un système réel, mais également répétons-le à analyser le comportement de ce système placé dans les conditions différentes. A l'élaboration du programme de simulation, il faudra par exemple prévoir le moyen de représenter q unités réparties sur p canaux où q et p seront des paramètres que l'on fera varier suivant les besoins de l'étude que l'on se proposera.

Nous avons tenté dans les paragraphes précédents de dessiner le profil très général qu'il est souhaitable de donner aux modèles mathématiques d'une simulation. Malheureusement de telles propriétés sont des visées hautement idéalistes rarement atteintes en raison des problèmes que pose le monde réel. Parmi ces difficultés on trouve (1°) qu'il est impossible de mesurer ou de déterminer certains types de variables affectant le comportement du système, (2°) que le nombre de variables qui doivent être considérées pour la description du système n'est pas oompatible avec la capacité de l'ordinateur dont on dispose, (3°) qu'il peut exister des variables internes dont on ignore l'influence sur les résultats de la simulation, (4) que certaines relations entre variables internes et variables de sortie sont inconnues ou impossibles à obtenir, (5) que dans la plupart des cas, ces relations sont trop complexes pour être représentées par une ou plusieurs équations mathématiques.

Lors de l'élaboration des modèles mathématiques un dernier choix

s'impose sur le type de description, globale ou modulaire, que doit fournir le modèle. Les modèles à description globale simulent le comportement des systèmes réels pris dans leur ensemble. Bien que cette approche ait été souvent employée, notamment en économie, en général, elle ne peut prétendre à un succès total pour la représentation et la prédiction du comportement des systèmes réels (notamment économiques). La principale raison vient du fait que l'être humain a de grandes difficultés à circonscrire les vastes complexités de la plupart des systèmes quand ils sont oonsidérés dans leur ensemble et non en terme de composants essentiels . C'est pourquoi l'approche la plus sûre consiste à dépouiller le système réel de son "aura" de complexité jusqu'à ce que l'analyse, menée avec rigueur, fasse découvrir la charpente centrale qui elle, est essentielle puisque simple à représenter modulairement. Ainsi un système conversationnel apparait comme un enchevêtrement d'intéractions e tre les utilisateurs placés à leurs terminaux et le calculateur qui, à son tour lance des ordres d'entrée-sortie sur disques, tambours, bandes, imprimantes, etc....Si l'on veut représenter ce système conversationnel dans son ensemble, on est à peu près sûr d'aboutir à un échec. Par contre, si on le voit comme une Unité Centrale, un certain nombre d'unités périphériques, de canaux et de consoles, la structure du modèle dans laquelle chaque composant modulaire représente un élément hardware, parait simple et compréhensible.

4 -FORMULATION DU PROGRAMME DE SIMULATION

On distingue six étapes principales dans l'écriture du programme de simulation :

1/ Construction de l'organigramme,

2/ Choix du langage de programmation,

3/ Mise au point,

4/ Recherche des données d'entrée et des conditions initiales,

5/ Génération de certaines données,

6/ Sortie et présentation des résultats.

L'organigramme traduit comme il est d'usage la séquence logique des évènements et il n'est pas de notre propos d'insister sur son rôle que chacun connaît.

Vient ensuite le choix d'un langage de programmation. Celui-ci se fait suivant deux alternatives : écrire le programme dans un langage à vocation générale tel que FORTRAN, PL/I [Ref 29], ALGOL, ou utiliser un des langages spécialisés tels que SIMULA [Ref 16], GPSS[Ref 28], SIMSCRIPT[Ref 12], GASP [Ref 14] SIMPAC[Ref 25], DYNAMO[Ref 14] qui permettent de gagner du temps au niveau de la programmation pure. Les langages sont conçus pour faciliter la programmation de certains types de systèmes. En particulier, GPSS, SIMSCRIPT et GASP sont bien adaptés à la représentation des files d'attente. Bien que l'on puisse réduite le temps de programmation en employant un langage de simulation, nous payons généralement cet avantage d'une diminution de la flexibilité des modèles et d'un accroissement du temps d'exécution.

Un autre aspect de la phase programmation est l'introduction des conditions initiales et des données d'entrée pour l'expérimentation. Puisque les expériences menées par simulation sont de nature dynamique, la question se pose des valeurs à affecter aux paramètres et aux variables du modèle à l'heure où commence la simulation en sorte de ne pas engendrer des distorsions trop grandes sur les résultats . Pour certains systèmes, et en particulier pour les systèmes économiques, des méthodes d'essai et d'erreur (trial and error) aident à déterminer ces valeurs initiales.

Un problème directement lié à l'écriture du programme de simulation est celui du développement de techniques numériques pour la génération de données. Les données utilisées pour l'expérimentation seront lues sur des sources externes classiques (cartes perforées, bandes magnétiques) ou bien seront générées par des procédures internes (c'est le cas des variables de nature stochastique). Les langages de simulation facilitent la tâche du programmeur en fournissant ces fonctions de génération de données, pour certains, hélas, la définition de ces fonctions est inéfficace!

Un dernier point concernant la formulation du programme de simulation porte sur la présentation des résultats. En programmant dans un langage général, on peut se livrer à toutes les fantaisies que l'on jugera utiles, tandis qu'un langage spécialisé, s'il offre la possibilité de récupérer implicitement des résultats statistiques, impose au moins un format standard dans la présentation.

5 -VALIDATION

Il peut être aventureux d'affirmer qu'un modèle représente correctement le comportement d'un système réel; car il s'agit d'un problème dans lequel interviennent des éléments de nature théorique, pratique, statistique et même philosophique. En effet, la validation d'un modèle s'apparente au problème plus général de la vérification de n'importe quelle hypothèse :" que signifie vérifier une hypothèse ?" et " quels critères employer pour établir la validité d'une hypothèse ?".

Puisque les philosophes scientifiques ne sont pas d'accord sur les réponses à donner à ces questions, nous n'aurons pas la prétention de les poser à leur niveau.

Toutefois, il reste un moyen pour valider les modèles de simulation : vérifier que les variables de sortie intuitivement vraisemblables , sont conformes à des mesures réalisées sur le système réel (dans le cas où ce système est opérationnel), ou comparables à des mesures faites sur un système réel dont le fonctionnement se rapproche du système que l'on désire simuler.

Dans certains domaines, notamment en économie, les enseignements fournis par l'histoire interviennent directement dans l'appréciation de la validité d'un modèle. Mais, il ne reste hélas que l'intuition lorsqu'on ne dispose d'aucun moyen matériel pour vérifier qu'un modèle de système digital est valide, avec les risques d'erreur que cette démarche comporte.

6 -DESCRIPTION DES EXPERIENCES ET INTERPRETATION DES RESULTATS

La validité du modèle une fois reconnue, nous sommes en mesure de mener à bien les diverses expériences qui nous permettront d'atteindre les objectifs fixés au tout début de notre recherche.

L'analyse rigoureuse des résultats d'une expérience aide souvent à découvrir les points "noirs" du système que l'on simule (notamment les goulots d'étranglement), ou l'inefficacité d'un algorithme). Elle suggère les modifications à apporter au système avant l'expérience suivante, dont le rôle sera de confirmer l'utilité de ces modifications ou de proposer de nouveaux changements.

Il faut pouvoir aisément traduire dans le modèle de telles modifications : la conception modulaire du programme de simulation facilite donc le remplacement d'un algorithme par un autre et le paramètrage des composants du système permet le plus souvent de réaliser des modifications immédiates sans aucun effort de programmation.

C H A P I T R E III

DESCRIPTION DE CASSCOU

1 -INTRODUCTION ET OBJECTIFS

2 -CARACTERISTIQUES ET LIMITATIONS

3 -LA LOGIQUE DU MODELE

3.1 Définitions

- Ressource
- Intéraction

3.2 Représentation des

- Ressource
- Intéraction

3.3 Organisation générale

4 -LES ENTREES-SORTIES

5 -IMPLEMENTATION

5.1 Choix d'un langage de programmation

1 -INTRODUCTION ET OBJECTIFS

La courte histoire du calcul automatique a été marquée par une série de progrès Hardware suivis de brusques changements dans la manière d'utiliser les ordinateurs. L'extension des systèmes conversationnels est l'un des aspects de cette évolution et la complexité croissante de ces systèmes rend difficile pour l'acheteur (ou le locataire) le choix d'un équipement nouveau qui, dans tous les cas, nécessite l'engagement de capitaux importants.

Il est surprenant de constater qu'à l'apparition des premiers systèmes à temps partagé, des centaines de millions de dollars aient été investis dans l'acquisition du système dernier né sans qu'une étude sérieuse du rendement ait été faite au préalable. A l'époque, ni l'acheteur, ni le constructeur ne pouvaient affirmer da manière réaliste le nombre de terminaux que la configuration pouvait raisonnablement supporter. S'il était possible, pour les systèmes d'exploitation simples, de déterminer l'ordinateur et la configuration appropriées et de choisir ensuite le constructeur le plus offrant, il n'est plus concevable de nos jours de suivre cette approche pour plusieurs raisons.

Premièrement, bien que l'on puisse aligner les systèmes conversationnels dans un "catalogue", on ne peut les considérer comme des "packages" standard muni d'options particulières adaptables aux besoins croissants de l'exploitation. Les nouveaux systèmes sont faits sur mesure, taillés dans une série de composants standards. Ainsi les décisions doivent être prises sur la dimension et peut-être la rapidité d'accès de la mémoire centrale, sur le nombre et le type des canaux, sur le nombre et la capacité des disques et des tambours, etc.... De plus ce n'est un secret pour personne que les unités périphériques sont extrêmement couteuses (IBM a accompli dernièrement un effort pour la série 370) et que les décisions prises à ce niveau peuvent être lourdes de conséquences fâcheuses.

Deuxièmement l'intuition n'est pas un moyen sûre pour guider ces décisions car il n'est pas évident que la performance d'un système donné se trouve nettement améliorée par l'adjunction d'un bloc de mémoire centrale ou d'un tambour.

Enfin, les systèmes ne sont plus considérés de nos jours comme essentiellement Hardware. Le Software fait maintenant partie intégrante des nouveaux systèmes : il se peut qu'un ordinateur de la 3^{ie} génération ait des performances ridiculement pauvres en raison d'un système d'exploitation mal adapté et même qu'il soit moins efficace qu'une configuration de la deuxième génération à priori moins puissante.

Il est donc raisonnable de soumettre à un certain nombre de tests les algorithmes-clés comme le système de traitement des interruptions, le système d'allocation mémoire, l'algorithme d'ordonnancement des travaux, l'algorithme de pagination, etc... puisque ceux-ci ont un effet décisif sur l'efficacité générale du système d'exploitation.

Notre objectif est de fournir un outil qui facilite la résolution des problèmes concernant l'étude des systèmes conversationnels de moyenne importance. Cet outil est un modèle de simulation que nous appliquerons à la représentation d'une partition conversationnelle FORTRAN dont nous donnons la description générale dans l'Appendix A. (cette partition est l'un des 3 composants, série, parallèle, temps réel du système SIRIS II implémenté sur IRIS 50 par la CII [Ref 2]).

Dans les dernières années quelques chercheurs qui se penchaient sur les problèmes de la représentation microscopique des systèmes digitaux, ont conçu des modèles qui atteignaient les buts recherchés, à savoir l'étude des performances et l'analyse des modifications hardware et software. Certains se sont intéressés à la représentation des systèmes à temps partagé, mais leurs modèles restent soit trop simples et donc difficilement adaptables à la complexité des nouveaux systèmes à temps partagé, soit trop volumineux pour les besoins de notre simulation (7000 instructions FORTRAN), nécessitant alors un temps d'exécution en machine relativement important (1 mn d'exécution sur un IBM 360/50 simule 3 mn d'utilisation du 360/67 Time Sharing System !).

Nous avons donc choisi de créer un modèle de simulation adapté à nos propres objectifs : CASSCOU. En dehors de son domaine d'application générale, CASSCOU servira à prédire le comportement de la partition prochainement

opérationnelle (décembre 70). Nous imaginerons un certain nombre de configurations hardware et software et nous étudierons pour chaque cas les temps de réponse moyens c'est-à-dire le temps qui s'écoule en moyenne entre l'instant où un utilisateur lance une commande de son terminal et l'instant où il reçoit un signal du système; nous déterminerons ainsi pour chaque cas le nombre maximum de terminaux que le système peut raisonnablement supporter. L'analyse des résultats de la simulation et notamment des temps de réponse nous aidera à définir le critère de choix de ce nombre maximal.

Nous nous intéresserons maintenant aux caractéristiques de CASSCOU en faisant part de ses limitations (paragraphe 2) Le paragraphe 3 traitera de l'aspect logique du modèle en définissant les éléments de base ainsi que leur représentation, et en précisant l'organisation générale du modèle. Avant de parler de l'implémentation de CASSCOU dans le paragraphe 5 (choix du langage de programmation, représentation des files d'attente, comptabilité des pages), nous détaillerons les entrées-sorties (paraphe 4). Enfin le dernier point de ce chapitre aura trait au délicat problème de la validition du modèle (paragraphe 6) .

2 -CARACTERISTIQUES ET LIMITATIONS-

Nous avons choisi d'étendre le domaine d'application de notre modèle à la simulation de l'ensemble des systèmes conversationnels simples, moyennant certaines modifications et au prix, dans la plupart des cas, d'effort de programmation minime. Puisqu'il doit être capable de simuler plusieurs systèmes conversationnels, CASSCOU décrit au moins le plus petit dénominateur commun à ces systèmes. Du point de vue hardware, il permet la représentation d'un ensemble de consoles (ou terminaux), d'au moins une unité centrale, d'au moins un canal sélecteur et d'au moins une unité de disques. En outre, au niveau du soft ware, il est en mesure d'accueillir un système de pagination -contrainte que rend nécessaire, dans la plupart des systèmes conversationnels actuels, une gestion de la mémoire soumise au contrôle d'un système de pagination. (La pagination consiste à découper la mémoire centrale et les mémoires secondaires en

blocs, appelés pages, dont la taille varie suivant les constructeurs. On obtient ainsi un ensemble de mémoire physiquement hétérogène mais rendue homogène pour l'utilisateur que l'on appelle mémoire virtuelle du système. La mémoire virtuelle de l'utilisateur est le potentiel de mémoire dont il dispose; et l'intérêt de la pagination porte sur le fait que ce potentiel peut être plus grand que la seule mémoire rapide). Par ailleurs les systèmes conversationnels plus anciens, tel CTSS, qui utilisent du "SWAPPING" , pur sont, à la limite, des cas particuliers de systèmes orientés vers la pagination(étant donnée la taille du programme à activer, dans le "SWAPPING" le système d'allocation de mémoire détermine la partie de la mémoire centrale à vider sur la mémoire secondaire pour permettre le chargement intégral du programme à activer).

Cette généralité va de pair avec la souplesse du modèle :

La configuration hardware du système réel est ajustable par paramètres. Ceux-ci indiquent le nombre de consoles connectées, le nombre d'unités périphériques (disques, tambour, bandes magnétiques), le nombre de canaux, le nombre et la dimension des pages, voire même le nombre d'unités centrales! un tableau reflète avec exactitude l'organisation de la configuration en précisant dansla $i^{ième}$ ligne les caractéristiques de l'Unité périphérique i à savoir le numéro du canal auquel elle est reliée, ses performances c'est-à-dire les temps d'accès et de transmission des données. Enfin, chaque canal est caractérisé par un numéro et un indicateur de type (sélecteur ou multiplexeur).

Au niveau software, les algorithmes-clés du système réel seront relativement isolés du reste du modèle, de façon à ce que les modifications que l'on veut introduire pour les besoins de la simulation soient rapidement et facilement réalisables. Ces modifications s'étendent du changement de valeur de certains paramètres au remplacement intégral d'un algorithme par un autre.

La modularité du système ne se limite pas seulement à l'isolement des algorithmes essentiels mais concerne aussi la construction logique du modèle. L'analyse du système réel nous amène à le considérer comme un ensemble de ressources exploitables par les utilisateurs. Chacune de ces ressources (Unité

centrale, mémoires secondaires, canaux, consoles,etc...) est décrite dans le modèle par un sous-programme -un même sous-programme représentant l'ensemble des mémoires secondaires. Une telle option a plusieurs avantages :

1°/ L'organisation traditionnelle des calculateurs digitaux est conservée, ce qui facilite l'élaboration et la compréhension du programme de simulation,

2°/ L'introduction d'une modification dans une ressource du système réel n'entraine que la modification du module correspondant dans le modèle(par exemple au moment d'adapter CASSCOU à la représentation d'un système conversationnel différent)

3°/ Une quelconque extension du système réel est traduite par l'adjonction de ressources nouvelles dans le modèle.

La sensibilité de CASSCOU aux commandes qui lui sont "lancées" souligne encore la faculté qu'il a de s'adapter aux contraintes expérimentales qu'on lui impose. Un tel modèle sert en particulier à refléter les effets du flot de commandes sur la performance du système réel. Il doit donc pouvoir s'ajuster facilement aux demandes diverses des utilisateurs. A cet effet, un sous-programme génère les données nécessaires au déroulement de la simulation, à partir du code qui définit la commande. L'étude de l'influence du flot de commandes sur le comprtement du système se fait de manière simple en faisant varier la suite des commandes codées qui constitue l'entrée du modèle.

CASSCOU possède donc la souplesse nécessaire d'une part à la représentation de différents systèmes conversationnels simples et d'autre part à l'étude du comportement d'un système conversationnel donné,placé dans divers contextes.

Nous soulevons maintenant le problème de la détermination du degré de simplicité que nous devons donner au modèle. Il est certain qu'une description trop proche du monde réel nous conduirait à écrire un programme de simulation complexe donc inefficace et couteux. Nous devons aussi nous garder de trop

simplifier la réalité, ce qui donnerait un modèle incomplet donc inexploitable et inutile. La difficulté est donc de s'en tenir à un juste milieu et surtout de s'assurer qu'il s'agit effectivement d'un juste milieu. Nous nous référons ici au problème de la validité d'un modèle que nous traiterons dans le septième paragraphe.

Une première approximation consiste malheureusement à ignorer l'OVERHEAD, c'est-à-dire d'une manière globale, le temps d'activité des diverses fonctions du superviseur (agencement des travaux, gestion des interruptions, comptabilité des travaux, gestion de l'horloge, etc...) Le problème de l'OVERHEAD est souvent de la plus grande importance mais en ce qui nous concerne, son évaluation est difficile pour une raison essentielle : la partition conversationnelle que nous simulons n'est pas encore opérationnelle. Nous pouvons, à la limite, nous contenter d'une approximation purement intuitive. Si le système avait fonctionné, nous aurions pu nous livrer à une série de mesures pour l'adjonction au système réel d'un certain nombre de programmes destinées au relevé de ces mesures. Encore faudrait-il s'assurer que cette opération n'affecte pas de manière sensible les performances de la partition conversationnelle! Nous estimons de plus que le déroulement de la plupart des sous-programmes superviseur est d'une durée négligeable devant la lenteur des entrées-sorties. Nous avons cependant accepté une exception pour l'algorithme de pagination que nous avons intégralement introduit dans le modèle, et pour lequel nous avons approximativement évalué la durée d'exécution. Le DISPATCHER (ou sous-programme d'ajencement des travaux) est un second algorithme essentiel. Ce sous-programme fait intervenir un grand nombre de tables qu'il aurait été couteux d'introduire dans le modèle. C'est pourquoi, nous nous limitons à décrire les effets du DISPATCHER au moment de l'activation des programmes utilisateurs, sans introduire le DISPATCHER réellement utilisé par la partition conversationnelle, et sans faire intervenir le temps nécessaire à son déroulement.

Lorsqu'il y a parallélisme (OVERLAP) entre opérations d'entrée-sortie et travail de l'unité centrale, cette dernière est interrompue en fin d'entrée-sortie. Une deuxième simplification consiste à faire abstraction de la dégradation des performances de l'unité centrale qui entraine cette simultaneité .

En effet, le retard imposé à l'unité centrale est de l'ordre de plusieurs microsecondes, alors que l'unité de temps choisie pour la simulation est la milliseconde et qu'en fait les temps de réponse du système sont de l'ordre de la seconde.

Quand la partition conversationnelle sera opérationnelle, il sera intéressant d'introduire dans le modèle l'influence de l'OVERHEAD sur le comportement du système en général et celui de l'OVERLAP sur l'unité centrale.

Pour des raisons d'efficacité, nous n'avons pas jugé utile d'introduire des fonctions aléatoires de génération de données concernant les performances des unités périphériques (par exemple positionnement des bras des disques sur les enregistrements désirés) ou sur les heures d'arrivée des commandes lancées par les utilisateurs. L'abscence de génération de variables stochastiques nous donne la certitude d'avoir des modèles d'entrée identiques pour un certain nombre de tests portant sur le comportement du système muni chaque fois d'une configuration hardware différente ou pourvu d'algorithmes de décisions et de pagination distincts.

Nous aurions pu améliorer le modèle en imaginant une fonction de dégradation des performances du système réel correspondant à l'apparition des pannes Hardware et Software. Elle prendrait le contrôle aux instants statistiquement déterminés par une loi de distribution des évènements pannes. Mais il s'agit là de situations exceptionnelles que nous ne désirons pas prendre en compte dans une première étape.

Ceci constitue l'essentiel des simplifications que nous avons voulues introduire dans le modèle. Nous allons maintenant présenter la construction logique du programme de simulation.

3 -LA LOGIQUE DU MODELE-

Nous avons vu dans le chapitre portant sur l'élaboration d'un modèle de simulation, l'importance de l'effort qui doit être fait pour dégager de la complexité du système réel, les éléments fondamentaux qui servent de support à la construction du modèle.

L'analyse de la partition conversationnelle nous a mené à définir deux unités logiques sur lesquelles s'appuie la charpente du modèle. La première est liée à la notion d'espace : c'est la ressource, la seconde fait intervenir le temps : c'est l'intéraction.

3.1 Définitions.

3.1.1. La ressource.

Une ressource est un élément Hardware ou Software du système réel exploitable par les utilisateurs du système.

On distingue deux types de ressources : celles qui satisfont une seule demande à la fois et celles qui sont en mesure d'admettre plusieurs requêtes simultanément. Dans la première catégorie se rangent les ressources CPU (unité centrale), DISQUES, CONSOLES, CANAL selecteur, PAGINATION tandis que le CANAL multiplexeur est une ressource du second groupe. Par souci de clarté dans la lecture du programme de simulation, nous avons défini une ressource mi-software , mi-hardware : PAGINATION .L'utilisateur en demande l'exploitation quand le déroulement ultérieur de ses travaux nécessite une nouvelle page en mémoire centrale. La création de cette ressource n'apparait pas nécessaire puisqu'elle consiste à faire appel au CPU pour l'exécution de l'algorithme de Pagination, puis du canal et des disques. Cependant sa présence -en plus de faciliter la compréhension du modèle- évite l'introduction supplémentaire dans le programme de simulation de variables qui joueraient le rôle d'indicateurs sur le type d'entrée-sortie demandée . La présence de cette ressource est justifiée aussi par l'intérêt tout particulier que nous portons à la pagination dans la partition conversationnelle et dans notre modèle (Rappelons que nous utilisons

l'Algorithme de pagination réellement employé par la partition conversationnelle).

3.1.2 L'intéraction

L'intéraction peut se définir comme l'unité logique d'action dans le modèle; de manière plus simple, une intéraction consiste en la demande d'exploitation des ressources du système par l'utilisateur et en l'allocation de ces ressources par le système.

Les évènements qui constituent une intéraction sont habituellement : le temps de réflexion de l'utilisateur au terminal (qui inclut en plus du temps de réflexion proprement dit, le temps nécessaire à la frappe de la commande), l'attente d'une réponse en provenance du système et en dernier lieu l'impression des messages à la console.

Au point de vue du système, l'utilisateur se trouve soit dans l'état d'attente d'une réponse du système, soit absent du système, ce qui peut se formuler différemment : le système a ou n'a pas de programme à exécuter pour le compte d'une utilisateur donné.

Nous pouvons maintenant préciser notre définition : l'intéraction est la suite des évènements qui se produisent entre les deux instants successifs où le système interrompt sur une entrée-sortie console ou achève l'exécution d'un programme utilisateur.

La figure 1 décrit une intéraction classique en précisant les activités de l'utilisateur à la console et l'état du programme-utilisateur dans le système.

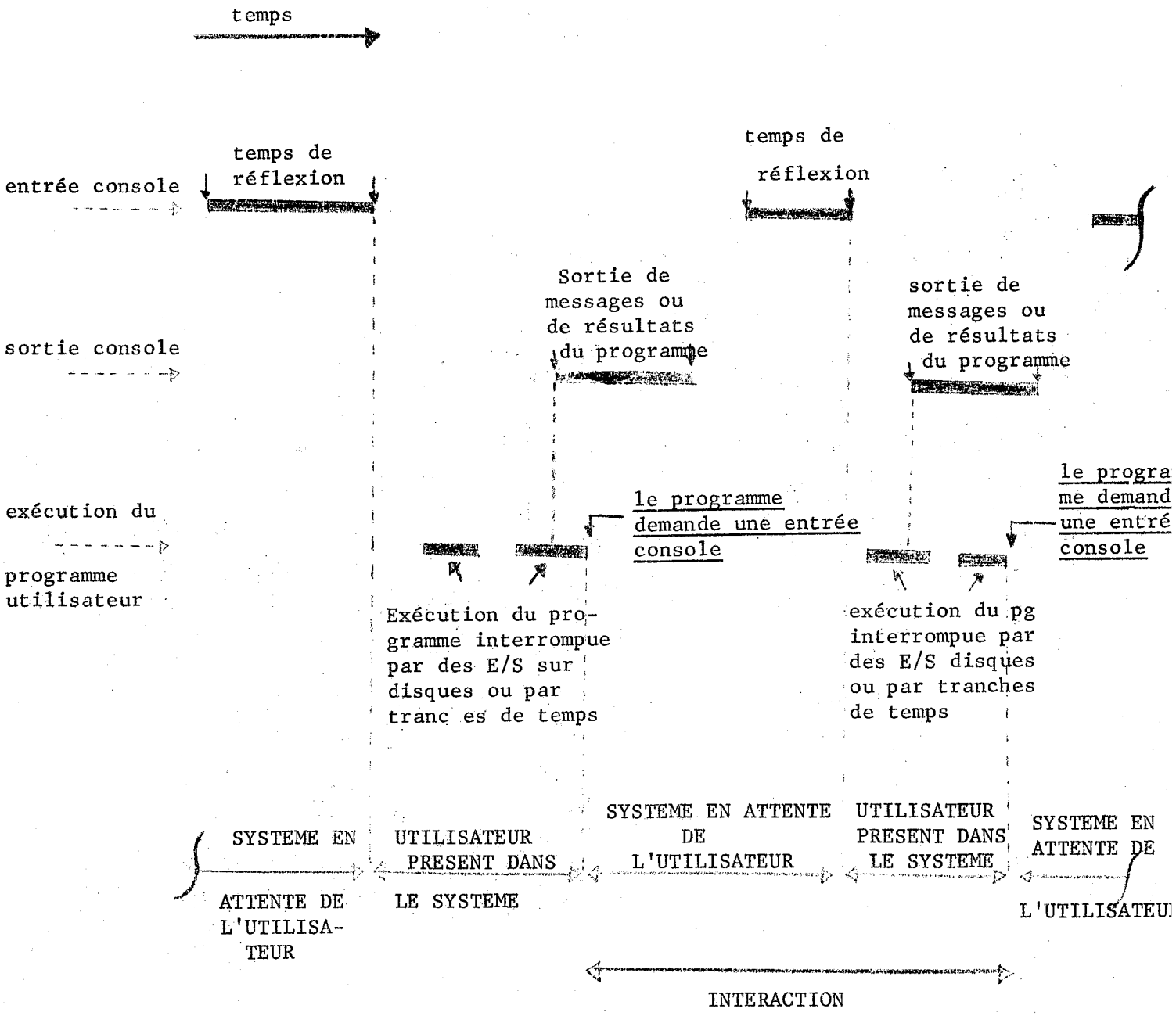


Figure 1 : exemple d'intéraction

Avec cette définition de l'intéraction, toute commande lancée par l'utilisateur à partir de son terminal est composée d'une ou plusieurs intéractions.

3.2 Représentation des éléments de base.

3.2.1 Les ressources.

Chaque ressource du système réel est représentée dans le modèle par un sous-programme distinct, et nous ne revenons pas sur les avantages qu'offre cette modularité. Nous remarquons qu'un périphérique ressemble à un autre; seules changent les caractéristiques du Hardware transmises dans le modèle par des paramètres. Un sous-programme peut alors les représenter, ce qui permet d'optimiser sensiblement le programme de simulation.

Les ressources n'ont pas toujours des activités indépendantes : il faut pouvoir décrire et prévoir au coeur même de ces sous-programmes les liens qui existent entre elles. La représentation de ces liens est la traduction de la structure logique de la partition conversationnelle qui fera l'objet d'un prochain paragraphe.

3.2.2 L'intéraction.

L'intéraction, nous l'avons vu, est directement liée à la notion de temps puisqu'elle fait intervenir dans sa définition une suite d'évènements. Sa nature suppose donc une représentation dynamique dans le temps. Cependant la suite des évènements qui la composent est partiellement définie par la commande à laquelle elle appartient. Nous verrons dans l'étude du modèle d'entrée que ces commandes sont décrites par un code et sont déterminées à l'avance. L'intéraction est donc en partie définie de manière statique. Reste alors à introduire la représentation du temps pour fixer la date d'apparition des évènements qui la constituent.

Pour une intéraction concernant un utilisateur donné, un évènement se produit à l'instant où cet utilisateur reçoit satisfaction du système, c'est à-dire à l'instant où il est en mesure d'exploiter la ressource dont il a besoin. La représentation du flux des évènements est décrite dans le chapitre IV et nous n'en présentons ici que le principe : chaque utilisateur dispose d'une horloge que le système de simulation met à l'heure aux instants opportuns. (par exemple en fin d'exploitation d'une ressource : si l'horloge de l'utilisateur I indique HEURE(I) avant l'exploitation d'une ressource, au moment de quitter cette ressource, le système de simulation la met à l'heure HEURE(I) + temps d'exploitation de la ressource par I); l'heure de simulation est la plus faible des heures indiquées par les horloges des utilisateurs. Celui qui possède une horloge "à l'heure" (par rapport à l'heure de simulation) est rendu activable et correspond à l'arrivée du prochain évènement.

En résumé, représentation statique des commandes et représentation dynamique du flux des évènements définissent de manière complète les intéractions.

3.3. Organisation logique générale.

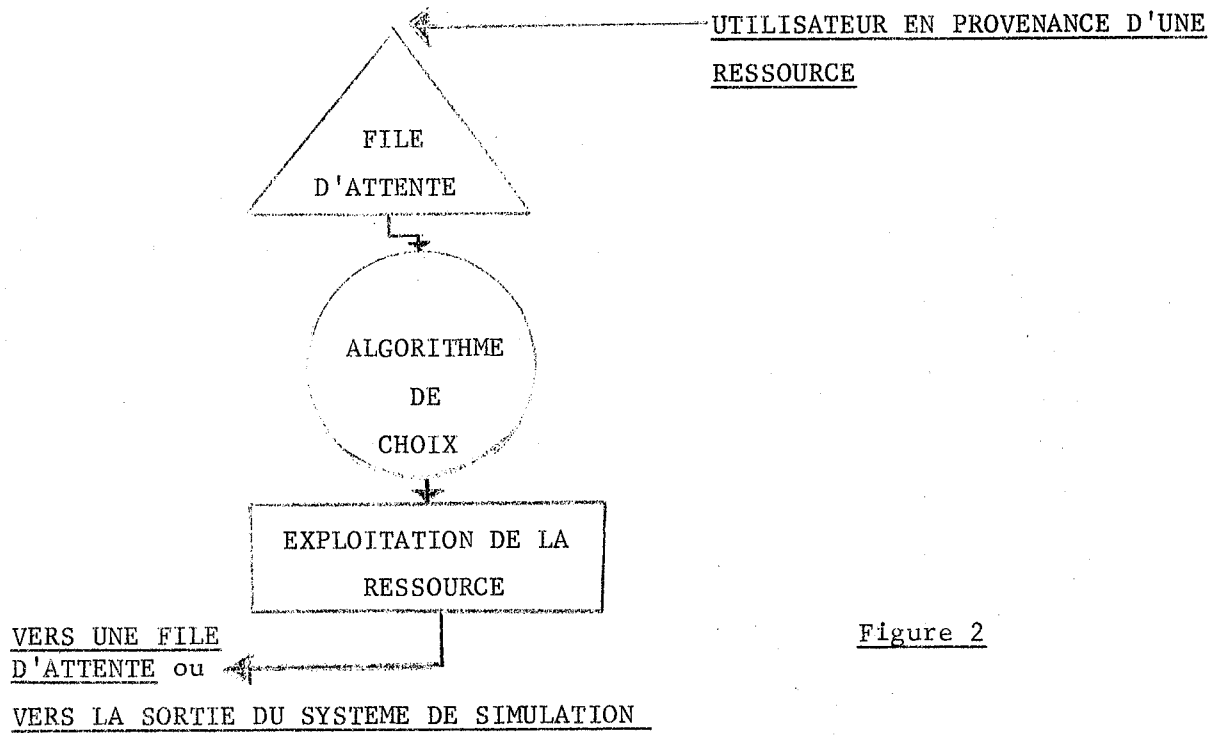


Figure 2

Le principe de l'organisation logique du modèle est simple et peut s'appliquer à tous les modèles de représentation des systèmes réels (digitaux ou non) dans lesquels apparaissent des ressources partageables entre plusieurs utilisateurs [Ref 1] .

Nous avons vu comment étaient représentées les ressources ainsi que les intéractions. Nous examinerons de quelle manière ces deux notions logiques s'imbriquent et se complètent dans l'organisation générale du modèle.

Devant chaque ressource s'aligne une file d'attente d'utilisateurs qui en demandent l'exploitation. Le système de simulation, par consultation des horloges de tous les utilisateurs présents dans le système, détermine où doit se produire le prochain évènement, c'est-à-dire dans quelle file d'attente se trouve le prochain utilisateur activable. Dans le cas où plusieurs utilisateurs d'une même file d'attente sont à l'heure de simulation, un algorithme de choix fondé sur des principes de priorité permet de décider quel utilisateur doit pouvoir exploiter la ressource. Après avoir reçu satisfaction l'utilisateur est placé dans une nouvelle file d'attente ou encore est éliminé du système de simulation notamment en fin de traitement d'une commande. Ce processus est réitéré jusqu'à la fin de la simulation.

Files d'attentes, Algorithmes de choix, exploitation de ressources forment un ensemble qui, conjugué au concept d'évènements constituent l'ossature (logique) du modèle.

4 -LES ENTREES-SORTIES

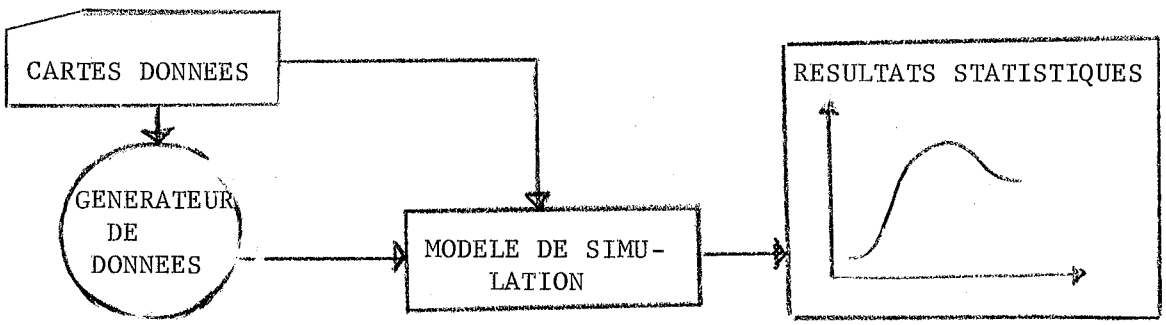


Figure 3 : ORGANISATION DU MODELE AU NIVEAU DES ENTREES-SORTIES.

Le modèle de simulation puise les renseignements nécessaires à son fonctionnement dans les cartes données définies manuellement par le programmeur et dans l'ensemble des informations déterminées par un programme de génération de données. Le déroulement du programme de simulation fournit un certain nombre de résultats exploitables pour la construction automatique de courbes. La figure 3 illustre cette organisation classique, employée par CASSCOU au niveau des entrées-sorties.

4.1 ENTREE du simulateur.

Les cartes données contiennent certains renseignements directement exploitables par le modèle de simulation, d'autres destinés au programme générateur ; les paramètres controlant la simulation, les paramètres hardware, software et environnement sont lus par le modèle de simulation; les codes correspondant aux commandes lancées par les utilisateurs concernent le programme générateur uniquement.

4.1.1. Les paramètres de contrôle de la simulation.

Une premier paramètre précise pour chaque utilisateur la longueur de sa session, c'est-à-dire le nombre de commandes qu'il a lancées pendant la période de simulation.

L'heure à laquelle la simulation prend fin est implicitement exprimée par la somme des longueurs des sessions. Elle s'achève au passage à zéro du compteur des commandes non traitées, compteur que l'on met à jour à chaque prise en compte d'une nouvelle commande.

Un second paramètre indique l'heure à laquelle on désire initialiser la récupération des données statistiques. En effet, il est souvent intéressant d'attendre que le modèle soit dans une situation d'équilibre avant de commencer l'étude du comportement du système réel.

On pourrait évidemment ajouter un troisème paramètre qui expliciterait l'heure à laquelle on désire que la simulation s'achève et qui serait prioritaire sur l'heure implicite déterminée par la somme des longueurs des

sessions. L'existence de ce paramètre impose un test supplémentaire au moment où le modèle essaie de déterminer l'arrivée du prochain évènement et nous jugeons ce paramètre superflu, dans le cadre de l'étude que nous voulons mener.

4.1.2 Les paramètres hardware.

Leur introduction permettra de décrire avec précision les composants hardware du système réel . On pourra en particulier, au moyen de Matrices et de vecteurs, formuler les liens qui existent entre ces divers composants et préciser leurs performances :

- Le $I^{ième}$ élément du vecteur CANAL indique le type du canal I (multiplexeur, sélecteur).
- Une matrice EQUIPEMENT résume l'organisation des différentes unités ainsi que leurs propriétés : le I^{ieme} vecteur de la matrice EQUIPEMENT précise les performances (temps d'accès, débit) de l'unité I et le numéro du canal auquel elle est reliée. Nous n'avons pas explicité la capacité de ces unités (elle se traduit dans le cas précis de la partition conversationnelle par le nombre maximum de lignes FORTRAN que chaque utulisateur a la possibilité d'enregistrer). Mais en règle générale la capacité est un renseignement qui peut être nécessaire au déroulement d'un modèle de simulation. Nous aurions pu également préciser l'état des unités afin de savoir si une unité est accessible ou déconnectée. La partition conversationnelle que nous simulons est en fait un système simple dans lequel tous les composants sont prêts à l'utilisation. (Nous n'avons donc pas à faire intervenir non plus, dans les paramètres environnement, les temps de réponse de l'opérateur relatifs au montage des bandes ou des disques).

4.1.3. Les paramètres software.

Pour un système d'exploitation du type OS/360, il serait utile de préciser le nombre maximum de "JOB" exécutables concurremment, la limite supérieure de la durée d'exécution des steps, la plus haute priorité qui puisse être affectée à un "job", un indicateur spécifiant que l'ordonnancement des

travaux est séquentiel ou soumis à un système de priorité, etc....

Dans le cadre de notre partition conversationnelle, ces paramètres sont encore superflus. Toutefois, il est intéressant dans le cas où plusieurs algorithmes de choix sont disponibles, de préciser au moyen d'un code lequel de ces algorithmes est effectif dans telle ou telle expérience. Le second paramètre qui nous importe ici concerne la dimension des pages, ceci en vue de réaliser certains tests-fixés au tout début de notre étude-sur le comportement du système réel.

4.1.4. Les paramètres environnement.

Les paramètres environnement sont tout à fait classiques et permettent d'adapter facilement le modèle aux exigences de configurations nouvelles :

- nombre d'unités centrales,
- nombre de canaux,
- nombre d'unités périphériques,
- nombre de consoles,
- capacité de la mémoire centrale.

4.1.5. Représentation des commandes FORTRAN

Au cours d'un session, l'utilisateur peut faire des compilations ou des exécutions de programmes par l'intermédiaire d'un langage de commandes. Ces commandes sont groupées par type :

1°) Administration (connexion et déconnexion d'un utilisateur, messages adressés à l'opérateur, création et intérrogation d'horloge...),

2°) Compilation (compilation, modification, écriture d'un fichier source),

3°) Exécution (exécution d'un fichier objet, affectation et impression de variables, trace et exécution pas à pas),

4°) Fichier (duplication, changement de nom, suppression de fichiers sur disques).

A chacun de ces types correspond un code qui peut être : 0 pour le type Administration, 1 pour la compilation etc.. Dans le cas particulier de la partition conversationnelle, ce code coïncide avec la priorité de l'utilisateur : à chaque type de commande est associée une priorité à laquelle se refère le critère d'activation d'un utilisateur. Un second code combiné au premier détermine sans ambiguité la commande effectivement lancée par l'utilisateur.

Pour certaines commandes, il est nécessaire de préciser la longueur du fichier auquel elles se rapportent. Il faudra notamment indiquer que l'utilisateur demande la compilation, l'exécution ou la lecture-écriture de N lignes appartenant à tel ou tel fichier.

4.1.6. L'heure de lancement des commandes.

Le programmeur introduit, sur cartes perforées, le temps qui s'écoule entre l'instant où le système donne une réponse à l'utilisateur et l'instant où l'utilisateur lance une nouvelle commande. (Retour chariot classique). C'est le temps pendant lequel le système n'a aucun programme à exécuter pour le compte de cet utilisateur. Ce temps T comprend la durée d'impression des messages ou des résultats de la commande précédente, le temps de réflexion de l'utilisateur, et le temps nécessaire à la frappe de la nouvelle commande.

Connaissant la commande précédente, nous pouvons raisonnablement évaluer le temps d'impression des messages qui en résultent.

La traduction du comportement de l'utilisateur devant son terminal est plus délicate. Alan SCHEER s'est intéressé au problème. Des mesures expérimentales lui ont permis de déterminer une loi de distribution affichant une moyenne de 35 secondes de temps de réflexion et de frappe! A l'aide de ce résultat et connaissant la longueur de la nouvelle commande, nous avons pu estimer approximativement le temps de réflexion et de frappe.

Sachant que HEURE(I) est l'heure indiquée par l'horloge de l'utilisateur I, la prochaine commande sera lancée dans le système à l'heure HEURE(I) + T (T, valeur lue sur carte).

Nous aurions pu générer T par une fonction aléatoire. Nous avons précédemment exposé la raison qui nous a retenu.

4.1.7. Le programme générateur de données.

Ce programme reçoit en entrée la suite des commandes codées. A partir de ces renseignements il donne une représentation des utilisateurs , et décrit leur comportement, en termes compréhensibles et exploitables par le modèle de simulation. Le travail que l'utilisateur demande au système se traduit, dans le modèle de simulation par :

- le nombre d'instructions machine à exécuter,
- le nombre de pages à valider,
- le nombre de chargements de segments à réaliser,
- le nombre d'enregistrements de fichiers à manipuler,
- le nombre d'ouvertures et de fermetures de fichiers,
- enfin les heures d'apparition de ces diverses opérations.

La seule difficulté de cette génération des données concerne la détermination du nombre de pages à charger en mémoire centrale : pour chaque commande, le programme générateur calcule le nombre de pages nécessaires au déroulement du travail demandé. Par différence avec le contenu de la table des pages de l'utilisateur, le programme générateur détermine le nombre de pages à valider. (On pourra se reporter utilement au point particulier qui traite de la pagination dans le paragraphe "implémentation").

Le programme générateur doit aussi préciser les heures d'apparition des différentes opérations. Ces heures sont relatives par rapport à l'instant où le programme utilisateur prend le contrôle. Pour l'ensemble des commandes, nous avons considéré que les programmes utilisateurs pouvaient être interrompus toutes les millisecondes par une entrée-sortie sur disques et que ces opérations ne pouvaient être:entrée-sortie de pages, chargement d'un segment, ouverture de fichiers, manipulation d'enregistrements de fichiers, fermeture de

fichiers. Pour l'ensemble des commandes, la première opération est un chargement de page en mémoire centrale; car le modèle travaille au niveau de la milliseconde, et la probabilité pour qu'un programme utilisateur s'exécutant pour la première fois se déroule pendant une milliseconde sans demander de pages, est négligeable. La prochaine entrée-sortie a lieu une milliseconde plus tard si la durée prévue pour l'exécution du programme utilisateur est strictement supérieure à une milliseconde; sinon elle a lieu immédiatement après l'entrée-sortie courante. La réalisation d'une entrée-sortie provoque la mise à jour des heures relatives d'apparition des prochaines entrées-sorties.

Le programme générateur et les cartes données fournissent donc au simulateur les renseignements utiles à son bon fonctionnement. Le rôle du simulateur est de rendre compte des travaux de l'utilisateur par l'intermédiaire des variables de sortie.

4.2. SORTIE du simulateur.

Les sorties sont communes à la plupart des modèles de simulation des systèmes conversationnels puisqu'elles ont le même but : permettre l'étude des performances.

Le temps de réponse est évidemment essentiel : il intéresse de près autant le constructeur qui cherche à évaluer les possibilités de son système que l'utilisateur, lui quotidiennement concerné.

Nous pouvons imposer au modèle de simulation la récupération de nombreux résultats. Nous ne citerons ici que ceux qui nous intéressent et qui nous permettront d'atteindre les objectifs fixés lors de la première étape de notre recherche :

- longueur maximum des files d'attente,
- temps d'attente sur les canaux, les unités de disques et l'unité centrale,
- utilisation des différents organes Hardware et de l'Algorithme de pagination,

- Inactivité de l'Unité Centrale, du Canal et des Unités de Disques.

Ces résultats sont ensuite directement exploitables pour la construction de courbes qui facilitent l'analyse des performances de la partition conversationnelle.

5 - IMPLEMENTATION -

5.1. Choix d'un langage de programmation

Ce choix s'est fait suivant trois critères :

- Nous avons recherché en premier lieu un langage qui soit efficace à l'exécution puisque le modèle doit manipuler un nombre relativement important d'éléments.

- Nous avons pris en seconde considération l'encombrement mémoire puisque le modèle doit pouvoir tracer un grand nombre de variables, notamment tout ce qui se rapporte à la représentation du flux des évènements, des files d'attente, des Algorithmes de choix, des indicateurs, des variables de sortie, des paramètres d'entrée.

- Le troisième facteur est la disponibilité du langage dans les systèmes d'exploitation; ceci en vue de l'utilisation générale du modèle.

En ce qui concerne le premier point, il est bien connu que les langages spécialisés de simulation sont en général moins efficaces que les langages de programmation à vocation générale. Ceci est particulièrement vrai pour les langages qui sont exécutés de manière interprétable. L'inconvénient des langages généraux reste une difficulté supplémentaire de programmation, mais ce n'est pas là un critère de choix.

La plupart des langages de simulation sont encombrants en mémoire centrale . Ils ont été conçus pour résoudre un large éventail de problèmes de simulation; de plus ils utilisent des fonctions standards résidentes qui provo-

quent un encombrement souvent inadmissible de la mémoire centrale. Certains de ces langages (SIMSCRIPT)offrent quelques moyens d'améliorer l'utilisation de la mémoire centrale, mais cet avantage se paie d'une diminution sensible de la vitesse d'exécution.

Relativement au troisième critère, PL/I, ALGOL et FORTRAN sont les seuls langages qui soient largement utilisés. Bien qu'ils ne soient pas des langages universels, ALGOL et surtout FORTRAN sont implémentés pour la plupart des constructeurs. Leur efficacité est certaine et leur encombrement en mémoire centrale est raisonnable. PL/I reste, hélàs, exigeant en mémoire.

Notre choix aurait donc pu se fixer sur FORTRAN ou ALGOL. Pour des raisons sentimentales (FORTRAN n'est pas l'objet d'attentions particulières à l'Institut de Mathématiques Appliquées de Grenoble, et ALGOL commence à être marqué par le temps) notre intérêt s'est tourné vers PL/I : c'est un langage relativement récent et puissant dont l'utilisation tend à se généraliser. En quelque sorte PL/I prend la "relève" de FORTRAN. Nous pensions par ailleurs au début de notre recherche que l'option MULTITASKING de PL/I serait une aide efficace pour la représentation des évènements parallèles (notamment pour la représentation de la simultanéité réelle entre le travail de l'unité centrale et les opérations d'entrée-sortie. Le MULTITASKING permet la création de tâches pouvant s'exécuter en parallèle. Avec cette possibilité, nous avions réalisé une correspondance biunivoque entre les tâches et les ressources; la tâche représentant l'Unité centrale se déroulait en parallèle avec les tâches représentant les canaux, les unités de disques et les consoles. L'expérience a montré que cette façon de procéder ne facilitait en rien la gestion des évènements simultanés, que le programme de simulation (voir Appendix C) devenait volumineux sans qu'on puisse y remédier par l'OVERLAY, enfin que le système "monoprocessor" dont nous disposons à Grenoble n'ajoutait rien à l'efficacité de l'ensemble (un système "multiprocessor" aurait permis l'exécution simultanée effective et non pas apparente, de plusieurs tâches et aurait donné un programme de simulation extrèmement efficace).

Un second langage sur lequel s'est porté notre intérêt : APL/360 Bien sûr, il ne satisfait pas le troisième critère puisqu'il fait intervenir un

équipement particulier (terminal) mais c'est un langage de programmation à la fois simple et puissant, élégant même par la concision de ses notations. Il ne serait sans doute pas vain de traduire CASSCOU en APL car il se pourrait que l'efficacité du programme en encombrement mémoire s'en trouve améliorée.

En règle générale il faut reconnaitre que le choix d'un outil de programmation est délicat, car chaque fois, le programmeur est placé devant un cas particulier nouveau où interviennent des contraintes et des objectifs différents. Et une fois ce choix fixé, on ne peut pas affirmer que le langage de programmation retenu soit le seul à donner satisfaction où même qu'il soit celui qui remplisse le mieux sa tâche. Il est notamment difficile d'évaluer à priori, par pure intuition, la vitesse d'exécution de deux programmes logiquement identiques mais écrits dans des langages différents et de les comparer.

Dans notre cas, APL est moins encombrant en mémoire centrale que PL/I et cependant moins efficace à l'exécution (APL est interprété). FORTRAN, AEL/360 et PL/I sont compétitifs et tous les trois sont censés donner satisfaction dans une large mesure. (On peut toujours réduire l'encombrement de PL/I en utilisant les possibilités de l'OVERLAY).

5.2. Comptabilité des pages.

Une première approche consisterait à "coller" à la réalité en définissant une table des pages par utilisateur dans laquelle on indiquerait leur validité, c'est-à-dire leur présence en mémoire centrale, leur réentrance c'est-à-dire le fait que leur utilisation les ait laissés inchangés.

Dans le cadre d'une simulation, cette méthode est difficilement envisageable : elle est couteuse en encombrement-mémoire, mais surtout il est complexe de tracer l'évolution de chaque page de chacun des utilisateurs. Il faudrait entre autres trouver un moyen de déterminer les pages auxquelles l'utilisateur aurait accès au cours du traitement d'une commande. Il n'est pas réaliste de prétendre générer stochastiquement cette suite de références en fonction de la commande lancée. Toutes les conditions sont réunies pour que nous nous heurtions à un certain moment au problème de l'inefficacité.

Nous nous sommes donc placés à un niveau plus global. Certes, la gestion de la mémoire est soumise à un système de pagination mais on peut réaliser (virtuellement) sur elle un découpage en PARTIES (découpage qui n'a rien de commun avec la segmentation au sens où on l'entend habituellement mais qui réunit dans une partie l'ensemble des pages utilisées de manière similaire).

Ces parties sont au nombre de 9 dans la partition conversationnelle

1/ partie contenant les diverses pages de contrôle,

2/ partie comprenant les pages de contrôle propre à la compilation,

3/ partie table des enchainements,

4/ partie fichier courant ,

5/ partie chaine de pseudo code,

6/ partie répertoire de fichiers et table de définition de fichiers,

7/ partie symboles de compilation,

8/ partie pile d'exécution ,

9/ partie réservations de tableaux et mémoire.

Dans ces conditions, connaissant la commande que l'utilisateur a lancée, il est aisé de déterminer dans le programme générateur les "parties" auxquelles il désire accéder (voir fig.3) C'est pourquoi chaque utilisateur dispose d'une table des parties dans laquelle on indique le nombre des pages valides ainsi que le nombre des pages réentrantes pour chacune des parties : TABLEPARTIE (I, J,1) est le nombre de pages valides appartenant à l'utilisateur I dans la partie J. TABLE PARTIE (I,J,2) est le nombre de pages réentrantes de l'utilisateur I dans la partie J(enseignement utile également au déroulement de l'Algorithme de pagination).

Connaissant les parties référencées et la commande lancée par l'utilisateur, le programme générateur détermine pour chaque partie référencée, le nombre de pages nécessaires au bon déroulement du travail de l'utilisateur; par consultation de la table des parties, il calcule le nombre de pages que le système aura à amener en mémoire centrale. Le chargement d'une page en Mémoire

centrale ou inversement sa réécriture sur mémoire secondaire demande seulement de connaitre la partie à laquelle elle appartient.

Les références aux différentes parties ne se font pas dans un ordre quelconque. Il faut pouvoir traduire (ici au moyen d'un tableau) la suite de ces références. L'analyse des commandes révèle que les références aux parties se font toujours suivant le même ordre. Il suffit donc de les ranger dans un ordre convenable qui soit efficacement exploitable par le modèle de simulation. Dès qu'une commande est introduite dans le système, le programme de génération met à jour le tableau REFPARTIE tel que :

REFPARTIE(I, J) est nul si l'utilisateur I ne fait pas de référence à la partie J et ,

REFPARTIE(I, J) indique le nombre de pages à charger en mémoire centrale dans la partie J pour le compte de l'utilisateur I.

Dans ces conditions, au cours de l'évolution de la simulation, le prochain chargement concernant l'utilisateur I s'effectuera dans la première partie J(J évaluant de la valeur 1 à 9) telle que REFPARTIE(I, J) est différent de zéro, et la valeur de J permet en même temps de repérer la partie dans laquelle on introduit une page.

PARTIE / COMMANDE	1	2	3	4	5	6	7	8	9
A (administration)	X								
C (compilation)	X	X	X	X	X		X		
E (exécution)	X		X		X			X	X
F (manipulation de fichiers)	X			X		X			
(*)CE (compilation et exécution)	X	X	X	X	X		X	X	X

figure 3 : X indiqu'il y a référence à la partie. Cette figure montre combien il est aisé de définir la suite des parties référencées.

* Il existe des commandes STORE et DISPLAY (affectation d'une valeur à une variable et impression de la valeur d'une variable)qui provoquent une compilatio suivie d'une exécution.

5.3. Les files d'attente.

L'efficacité des modèles de simulation dépend aussi de l'organisation et de l'exploitation des files d'attente.

Il existe à ce sujet de nombreuses théories proposant diverses représentations, utilisables dans certains cas, exploitables dans d'autres. (Le cours donné en 1968 par M.DUBY-IBM est un excellent résumé de toutes ces théories).

Dans un modèle de simulation, on rencontre deux types de files d'attente : celles des utilisateurs réclamant une ressource du système, celles se rapportant au chainage des évènements.

D'une manière générale, les files d'attente de la première catégorie sont organisées suivant les règles bien connues du "premier arrivé-premier servi" ou du "dernier arrivé-dernier servi". Dans ce cas, un chainage monodirectionnel suffit à la représentation de la file d'attente : chaque élément de la chaine pointe vers l'élément suivant comme le montre la figure 4. Les pointeurs DEBUT et FIN délimitent la chaine et facilitent l'introduction et le départ de nouveaux utilisateurs.

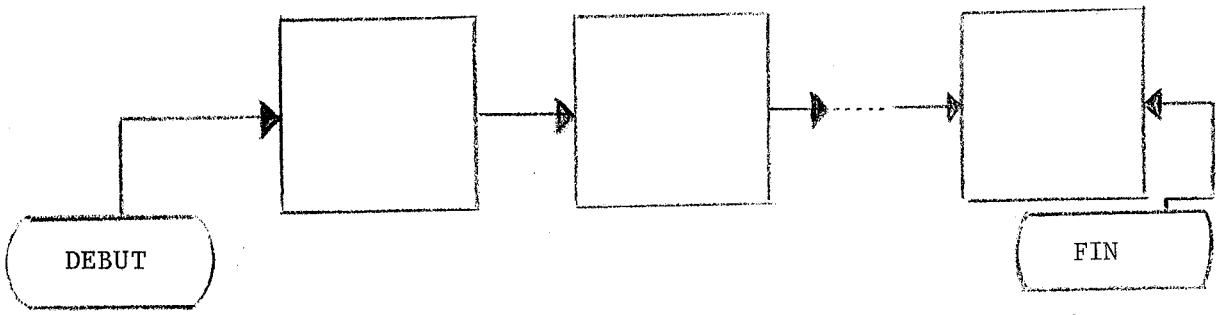


Figure 4 Exemple de chainage monodirectionnel

Relativement au chainage des évènements, deux attitudes sont possibles : ou bien le rangement des évènements se fait suivant l'ordre d'arrivée dans la chaine (qui n'est pas forcément l'ordre de sortie, c'est-à-dire d'apparition dans le modèle de simulation) ou bien une exploration de la chaine est

réalisée systématiquement à chaque arrivée de façon à placer l'évènement suivant l'ordre chronologique de sortie. Dans le premier cas, si l'on gagne du temps à l'arrivée de l'évènement dans la chaîne, au moment de déterminer l'apparition du prochain évènement, le système de simulation doit explorer la chaîne des évènements toute entière. Dans le second cas, la détermination du prochain évènement est immédiate puisqu'il se trouve en tête de la file d'attente; et au moment du rangement de l'évènement, le système de simulation n'a pas forcément eu à explorer toute la chaine avant de trouver la place qui convenait à cet évènement. En conséquence, la seconde méthode semble la plus efficace bien que la plus onéreuse du point de vue de l'encombrement mémoire.

Si dans le premier cas un chainage monodirectionnel suffit, un chainage bidirectionnel est appréciable dans le second cas : chaque élément de la chaine dispose d'un pointeur vers l'élément précédent et d'un pointeur vers l'élément suivant (cf figure 5)

Chainage monodirectionnel et chainage bidirectionnel sont les deux principes fondamentaux relatifs à l'organisation des files d'attente. A partir de là, il est possible d'introduire des 'subtilités' qui améliorent le temps d'accès aux éléments du chainage. Par exemple, dans un système conversationnel, nous distinguons les évènements internes à la machine qui se produisent à des intervalles de temps de l'ordre de la milliseconde, et les évènements en provenance des terminaux dont la période d'occurrence est de l'ordre de la seconde . On peut imaginer un point d'entrée intermédiaire dans la chaine qui jouerait le rôle de séparateur entre les évènements propres à l'ordinateur et ceux se rapportant aux terminaux. Ainsi suivant le type d'évènement arrivant dans la file d'attente, on remonte ou l'on déscend la chaîne à partir de ce pointeur (cf.fig.6).

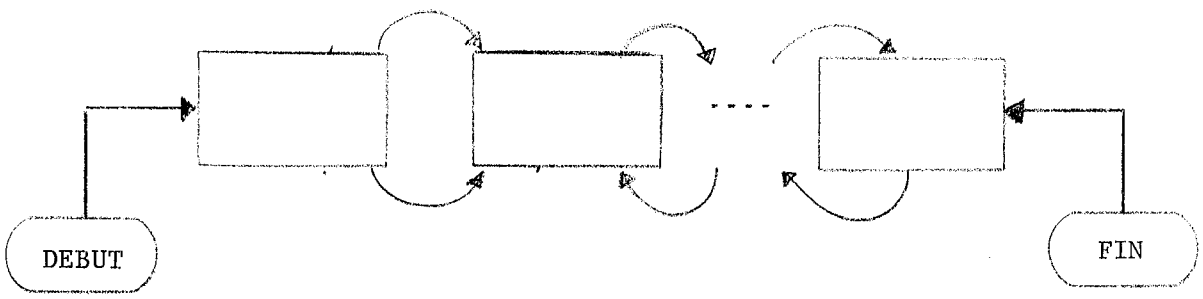


Figure 5 : Exemple de chainage bidirectionnel

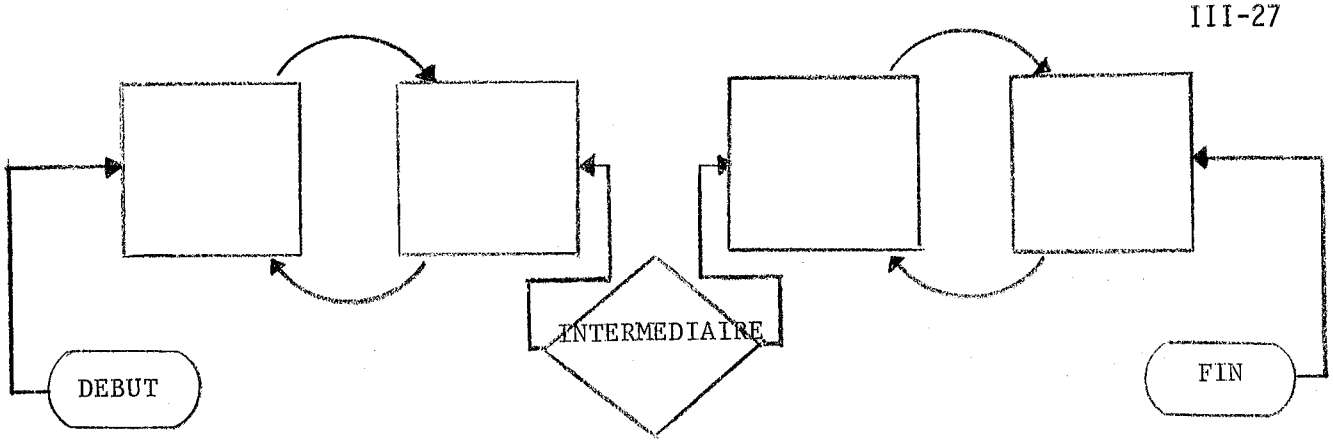


Figure 6. Exemple d'optimisation des temps d'accès aux éléments d'une chaîne.

Après ces considérations générales, nous sommes en mesure de présenter l'organisation retenue pour les files d'attente de CASSCOU.

A chaque ressource correspond une file d'attente de candidats. Elle est constituée en fait de quatre sous-chaines affectées de priorités différentes correspondant, on le rappelle, aux quatre types de commandes disponibles. Le prochain utilisateur à satisfaire est choisi dans la sous-chaine de plus haute priorité suivant la règle "premier arrivé, premier servi". Les files d'attente utilisateurs sont représentées par des chaînes monodirectionnelles à quatre points d'entrée placés tous les n éléments (n est le nombre maximum d'utilisateurs admissibles dans le système). Chaque point d'entrée est en correspondance biunivoque avec un degré de priorité et sert de délimiteur entre deux sous-chaînes. Enfin le dernier élément de chaque sous-chaîne est repéré par un pointeur.

REMARQUE : Nous introduisons un degré de récursivité supplémentaire en considérant l'ensemble des files d'attente utilisateurs comme une seule grande chaîne monodirectionnelle à R points d'entrée en correspondance biunivoque avec les ressources.

Nous n'avons pas fait d'efforts particuliers en ce qui concerne la représentation de la chaîne évènements en raison même de la définition que nous avons donnée à l'évènement : il consiste en l'activation du prochain utilisateur à satisfaire. Nous notons encore ici la correspondance biunivoque entre

évènements et utilisateurs. En conséquence, la longueur de la file des évènements est relativement peu importante et justifie une représentation du type monodirectionnel.

Description du mécanisme : Se reporter à la figure 7.

Chaque élément de la chaîne évènement représente un évènement, chaque élément de la chaîne utilisateurs représente un utilisateur.

Chaque élément de la chaîne des utilisateurs se compose d'un pointeur vers l'élément suivant et du nom d'un utilisateur qui, avec l'adresse-origine de la chaîne des évènements définit un point d'entrée dans cette chaîne.

Chaque élément de la chaîne des évènements est constitué d'un pointeur vers l'élément suivant et d'une valeur qui est l'heure d'apparition de cet évènement (i.e. l'heure à laquelle l'utilisateur qui pointe vers cet élément est activable).

Dans ces conditions, la mise à jour de l'heure d'apparition des évènements est rapide, mais la détermination de l'heure de simulation impose l' exploration de la chaine des évènements toute entière.

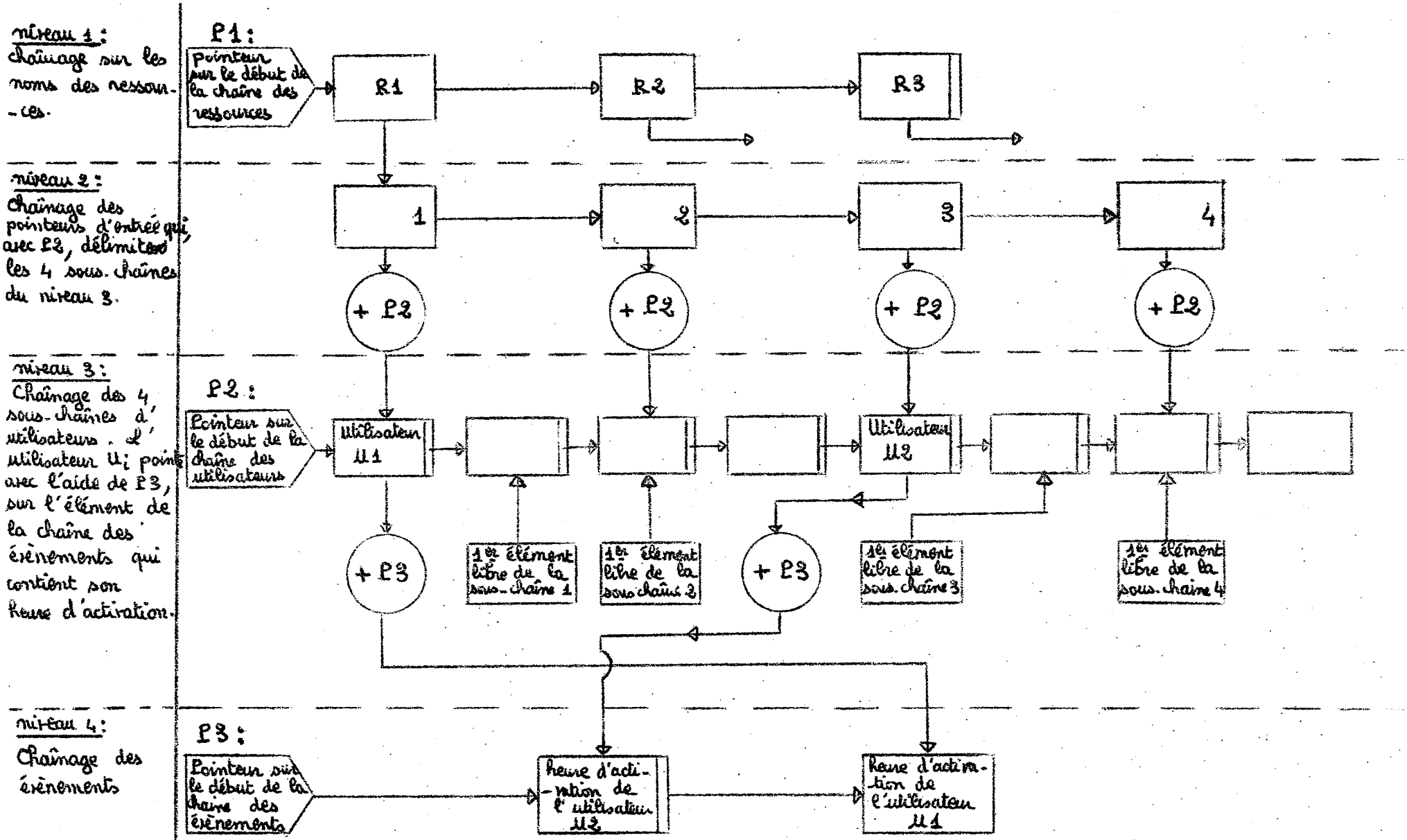


Figure 7 : Exemple de représentation des chaines utilisateurs et de la chaîne des évènements. Le système réel est dans cet exemple composé de 3 ressources R1, R2, R3 et de deux utilisateurs qui tous deux demandent l'allocation de la ressource R1. L'un (U1) a la priorité 1, l'autre (U2) a la priorité 3.

Sur le schéma ne figurent pas les files d'attente correspondant à R2 et R3, et dont la représentation est identique àcelle de R1.

A ce niveau là, nous pourrions améliorer les performances du modèle en utilisant une représentation bidirectionnelle encore que le gain acquis en efficacité ne soit pas notable en raison du nombre restreint d'utilisateurs.

6 -VALIDATION-

Des problèmes que nous avons abordés, celui de la vérification de la validité de CASSCOU reste, il faut le reconnaître, le seul a ne pas être véritablement résolu.

Comment affirmer que ce modèle est vrai ? Sur quels critères s'appuyer ? Grave question, qui met en cause l'efficacité de notre recherche, et plus généralement l'intérêt même de la simulation, puisque l'on ne peut être vraiment sûr de la véracité des résultats; (les connaissances en matière de validation, nous l'avons vu dans le chapitre précédent, sont bien limitées; il n'existe pas de "théorèmes" qui permettent d'affirmer qu'un modèle est vrai ou pas).

En ce qui nous concerne, la vraissemblance des résultats laisse à penser que 'CASSCOU' est un modèle fort digne de confiance. Nous avons vérifié que la logique du simulateur coincidait avec celle de la partition conversationnelle. Cependant, tant que le système réel ne sera pas opérationnel, nous ne pourrons pas être vraiment sûr que les libertés que nous avons prises par rapport à la réalité par souci de simplicité et d'efficacité, n'aient pas eu d'incidence notable sur la véracité des résultats.

Le problème ne semble d'ailleurs pas différent de celui de la vérification d'une hypothèse physique :

- établir des règles qui traitent ce qui est plus ou moins bien connu,
- prévoir un évènement ou un comportement,
- confirmer que cela existe physiquement, par l'expérience.

C H A P I T R E IV

REPRESENTATION DES EVENEMENTS DANS UN MODELE DE SIMULATION

INTRODUCTION

1 -LE FLUX DU TEMPS : REPRESENTATION ET MECANISMES

2 -DEUX MODELES SIMPLES

2.1 Un modèle à incréments de temps fixes(M1)

2.2 Un modèle à incrément de temps variable (M2)

3 -REPRESENTATION DU TEMPS DANS NOTRE MODELE DE SIMULATION

3.1 Représentation par le multitasking de PL/I

3.2 Représentation du flux du temps dans notre modèle (voir fig.3)

C H A P I T R E IV

REPRESENTATION DES EVENEMENTS DANS UN MODELE DE SIMULATION

Un programme de simulation doit représenter avec suffisamment de précision la logique du système réel, mais aussi prévoir la suite des évènements qui y apparaissent. L'occurrence d'un évènement provoque un changement d'état et se traduit par une action que le système effectue en fonction de cet évènement.

L'objectif de ce chapitre est de décrire la flux des évènements dans un système, leur mécanisme et leur représentation. Nous introduisons d'abord le concept fondamental de l'horloge qui intervient directement dans l'organisation du programme de simulation. Puis deux modèles simples de représentation du temps se proposeront de donner une idée des moyens dont on dispose pour simuler le flux du temps. La deuxième partie de ce chapitre sera consacrée à l'étude que nous avons menée pour la représentation de l'écoulement du temps dans notre modèle. L'existence des évènements simultanés dans le système réel que nous avons à simuler, tout comme dans la plupart des systèmes digitaux actuels, nous a amenés à poser le problème de la représentation de ces évènements. Nous avons d'abord pensé que le multitasking de PL/I fournissait une solution efficace à ce problème, en faisant dérouler en "parallèle" les taches correspondant aux éléments du système réel qui fonctionnent simultanément. Cette façon de procéder cache en fait sous le masque de la simultaneité apparente le traitement séquentiel de tous les évènements, et s'est révélée fort couteuse du point de vue de l'encombrement mémoire. Nous expliquerons ultérieurement pourquoi dans le cas présent, l'utilisation du multitasking de PL/I a tout d'une spéculation intellectuelle inutile. Aussi nous sommes nous orientés vers une représentation séquentielle des évènements qui est à la fois simple et efficace.

1 -LE FLUX DU TEMPS : REPRESENTATION ET MECANISMES.

Une considération importante dans la formulation d'un modèle de simulation concerne le choix d'une méthode pour représenter l'écoulement du temps. On dispose d'une part des méthodes à incréments de temps fixes, et des méthodes à incréments de temps variables, d'autre part. [Ref 14]

La première méthode consiste à mettre l'horloge de temps réel à l'heure à des intervalles réguliers et à rechercher dans le système de simulation les évènements qui doivent se produire à cet instant particulier.

Dans les modèles à incréments de temps variables, l'horloge de temps réel est mise à l'heure de l'évènement dont la réalisation est imminente, et les temps morts (c'est-à-dire des périodes de temps pendant lesquelles aucun évènement donc aucun changement ne se produit dans le système) sont évités. Cette méthode implique deux balayages de l'état du système : un pour mettre l'horloge à l'heure, un pour trouver tous les évènements qui se produisent à cet instant, alors que dans la première méthode un balayage suffit pour déterminer les évènements qui se produisent en coïncidence avec l'heure réelle.

Un tel balayage doit être fait de manière à perdre le moins de temps possible. Le balayage de toute la liste des évènements peut être évité et donner un gain de temps appréciable si l'on utilise une technique de tri permettant de conserver les évènements dans l'ordre de leur apparition. Le premier évènement est alors le premier de la liste. Pour gagner du temps également au moment du tri, la suite des évènements est gardée sous forme de liste. Le problème de l'insertion d'un évènement entre plusieurs n'est alors qu'un problème de déplacement de pointeurs, chose facile et peu couteuse en temps.

SIMPAC et DYNAMO sont des langages de simulation qui utilisent la méthode à incréments de temps fixes, tandis que GPSS et SIMSCRIPT fonctionnent suivant la seconde méthode.

Les systèmes dans lesquels les évènements se produisent de manière régulière peuvent être simulés efficacement par des langages qui utilisent des incréments de temps fixes. Cette méthode est moins coûteuse en opérations, et on l'a vu, fait intervenir un seul balayage dans la liste des évènements. Mais l'expérience montre que la plupart des modèles simulent des évènements dont les heures d'apparition sont aléatoires et, dans ce cas, la seconde méthode se révèle la plus efficace.

Quelle que soit l'unité de temps choisie, la méthode à incréments de temps variables a l'avantage de ne pas affecter la vitesse de déroulement du programme de simulation. Elle permet de plus une économie de temps machine, lorsque la simulation est statique pendant un long intervalle. L'article "Some Problem of Digital Systems Simulation" que présentent Conway, Johnson et Maxwell, fournit quelques algorithmes de décision de choix entre une méthode à incrément de temps fixe et une méthode à incrément de temps variable quand on désire minimiser le coût d'exécution en machine. Conway et ses collaborateurs ont démontré que l'efficacité des méthodes à incréments de temps fixes croissait avec le nombre de variables d'états (temps d'attente, longueur des files d'attente..) et que l'efficacité des méthodes à incréments de temps variables augmentait avec la durée moyenne de l'évènement. L'expérimentation des deux méthodes est en fait, pour un problème donné, la seule façon sûre de déterminer celle qui minimise le temps d'exécution en machine!

Les deux exemples qui suivent servent à souligner les différences entre ces deux mécanismes. Il s'agit de deux versions d'un modèle simple de file d'attente sur une ressource. L'une utilisant l'approche à incréments de temps fixe, l'autre utilisant l'approche à incréments de temps variable. Nous supposons que les heures d'arrivée et les temps de service sont des variables aléatoires dont les variances et les lois de distribution sont connues. Les arrivées sont satisfaites suivant la règle du "premier arrivé, premier servi". Nous supposons aussi l'existence d'un ensemble de sous-programmes de génération de variables aléatoires à partir de fonctions de distributions connues.

2 -DEUX MODELES SIMPLES

2.1 Un modèle à incréments de temps fixes (M1)

Nous définissons en premier lieu les variables et les relations fonctionnelles du système en termes d'un ensemble de symboles mathématiques. Puis nous convertirons le modèle mathématique résultant en un organigramme;

Les variables de notre modèle à incréments de temps fixes sont définies dans la liste suivante :

1°/ Variables"externes"

- AR_i = intervalle de temps qui s'écoule entre l'arrivée de l'élément i et de l'élément i+1
- ST_i = temps de service fourni par la ressource à l'élément i.

2°/ Variables d'états : (voir figure 1)

- $SUMAR_i$ = somme des heures d'arrivée des i éléments quand l'unité i+1 demande les services de la ressource.
- $ATTENTE_i$ = temps d'attente de l'élément i sur la ressource.
- $INOCC_i$ = temps pendant lequel la ressource attend l'arrivée de l'élément i.
- $TINOCC_i$ = temps total d'inactivité de la ressource à l'arrivée de l'élément i.
- $TATTENTE_i$ = temps total d'attente à l'arrivée du $i^{ème}$ élément.
- N_i = nombre d'éléments attendant d'être satisfaits par la ressource.
- HORLOGE = l'heure simulée en unités d'incréments.

En début de simulation, le temps total d'arrivée, le temps total

d'inactivité de la ressource, le temps total d'attente, la longueur de la file d'attente sont nuls.

$SUMAR_1 = AR_o = 0$

$TINOCC_1 = 0$

$TATTENTE_1 = 0$

$N_1 = 0$

$HORLOGE = 0$

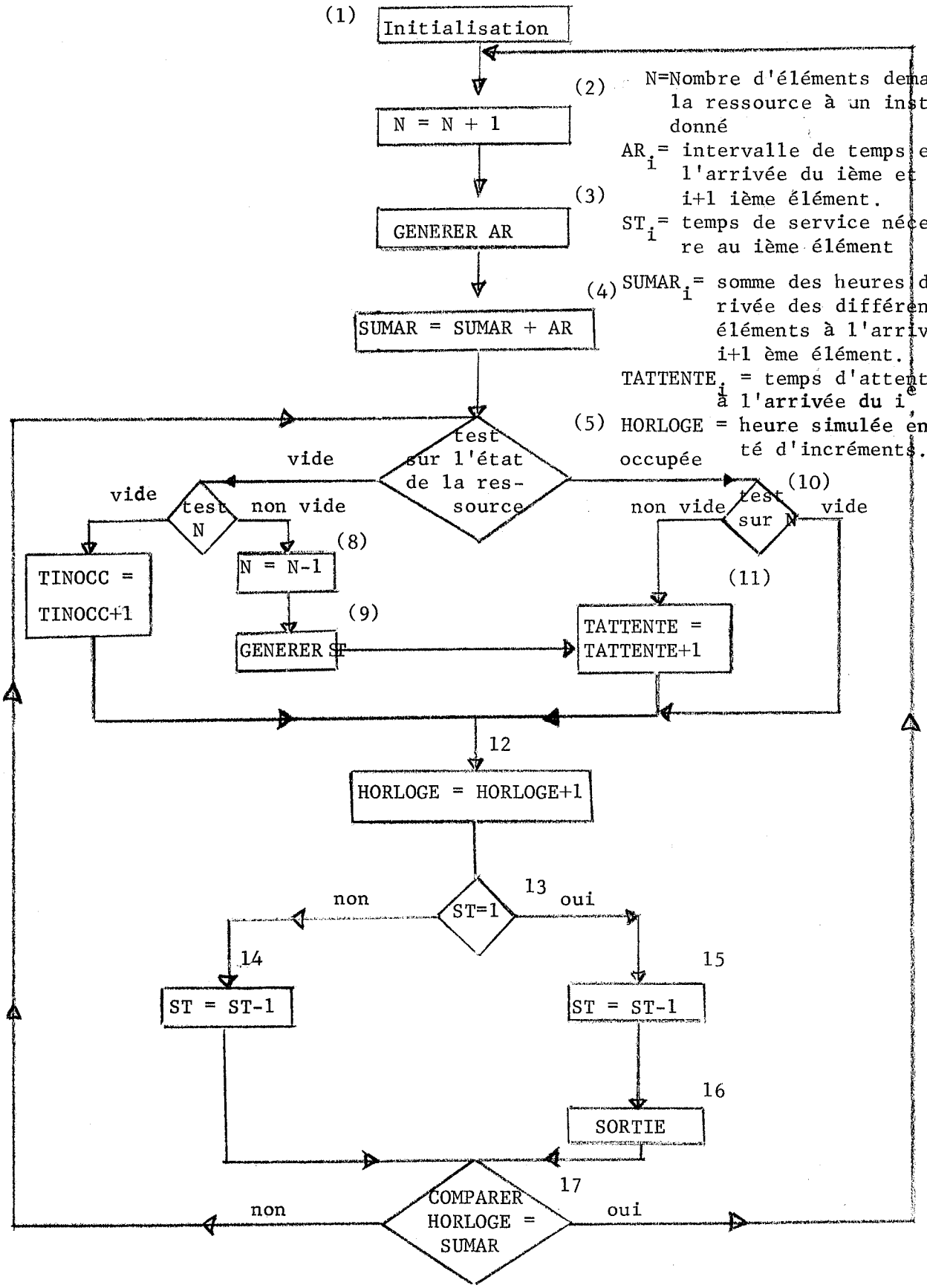


Figure 1 : un modèle à incrément de temps fixe

L'organigramme correspondant à ce modèle apparait sur la figure [1]. Le premier bloc de cette figure est une procédure d'initialisation dans laquelle SUMAR, TINOCC, TATTENTE, N et HORLOGE sont mis à zéro, comme l'indiquent les équations précédentes. Dans le second bloc, la longueur de la file d'attente est augmentée d'une unité et indique la demande de service de la ressource r par le premier élément. Puis, une heure d'arrivée AR est générée par une fonction aléatoire appartenant au système de simulation. Par exemple, si AR a une loi de distribution exponentielle négative, l'heure d'arrivée est déterminée en fonction de cette loi de distribution. A la première itération, AR indique l'intervalle de temps qui s'écoule entre l'arrivée du premier et celle du second élément. Dans le quatrième bloc l'heure d'arrivée totale est mise à jour. Puis un test détermine si la ressource est déjà occupée à cet instant précis ou si elle est inactive. Si la ressource est inactive, c'est-à-dire si aucun élément ne l'occupe, un test sur la longueur de la file d'attente indique qu'un élément est en attente de service ou pas. Dans le cas où la file d'attente contient au moins une demande, l'élément de tête vient exploiter la ressource et la longueur de la file d'attente diminue d'une unité (on applique la règle de priorité du premier arrivé, premier servi). Dans le bloc 9, une procédure génère le temps de service ST pour l'élément que la ressource vient d'admettre. Une unité de temps est ajoutée au temps d'attente de chaque élément se trouvant dans la file. Cependant, si le test du bloc 6 a trouvé que la file était vide, une unité de temps est ajoutée au temps total d'inactivité de la ressource (bloc 7). Si le test du bloc 5 indique que la ressource est déjà occupée, les éléments en attente dans la file voient leur temps d'attente augmenter de une unité de temps.

Après avoir envisagé toutes les possibilités logiques concernant l'état de la ressource ou de la file d'attente, l'horloge est avancée de une unité de temps. Le temps pendant lequel la ressource est occupée par l'élément courant est évidemment fonction de ST. Si le temps de service est supérieur à l'unité de temps, la ressource est occupée, et le temps de service pour l'élément courant est diminué de l'unité. Un temps de service nul reflète l'inactivité de la ressource. Enfin, si le temps de service est de 1 unité de temps, l'exploitation de la ressource par cet élément prend fin à cet instant. On

libère la ressource en affectant la valeur 0 au temps de service et en retirant l'élément courant de la ressource.

Le bloc 17 compare l'heure de simulation avec l'heure totale d'arrivée. Si l'heure indiquée par l'horloge est égale à l'heure totale d'arrivée, alors une nouvelle demande a été lancée par un élément. Le contrôle est donc donné au bloc 2 qui réalise l'entrée du nouvel élément dans la file d'attente. L'intervalle de temps séparant la nouvelle arrivée de la prochaine est généré par une fonction aléatoire et le processus entier se répète. Si l'horloge est inférieure à SUMAR, l'arrivée du prochain élément n'a pas encore lieu et l'on doit retourner au bloc 5 pour mettre à jour l'horloge, les temps d'attente ou les temps d'inactivité.

Ce processus peut être répété autant de fois qu'on le désire. La simulation s'achèvera par exemple à une heure fixée à l'avance ou à l'occurrence du $n^{ième}$ élément d'entrée (pour n déterminé à l'avance). Pour sortir de ce processus, il suffit d'inclure une procédure de test sur l'heure de fin de simulation ou, selon le cas sur le nombre d'occurrences prédéterminé d'éléments en entrée.

2.2 Un modèle à incrément de temps variable (M2)

La différence la plus évidente entre M1, notre modèle à incrément de temps fixe et M2 est que l'heure est traitée ici comme une variable continue. De plus dans le second modèle il n'y a pas d'horloge explicitement simulée.

Les variables et les paramètres ont la même signification que dans le modèle précédent. Toutefois, SUMAR, N et HORLOGE ne sont pas essentielles dans la logique des opérations de ce modèle. On se sert du même système de génération aléatoire des heures d'arrivée et des temps de service, et l'on applique la même règle de priorité du premier venu, premier servi.

La figure 2 décrit l'organigramme correspondant à ce modèle. Le

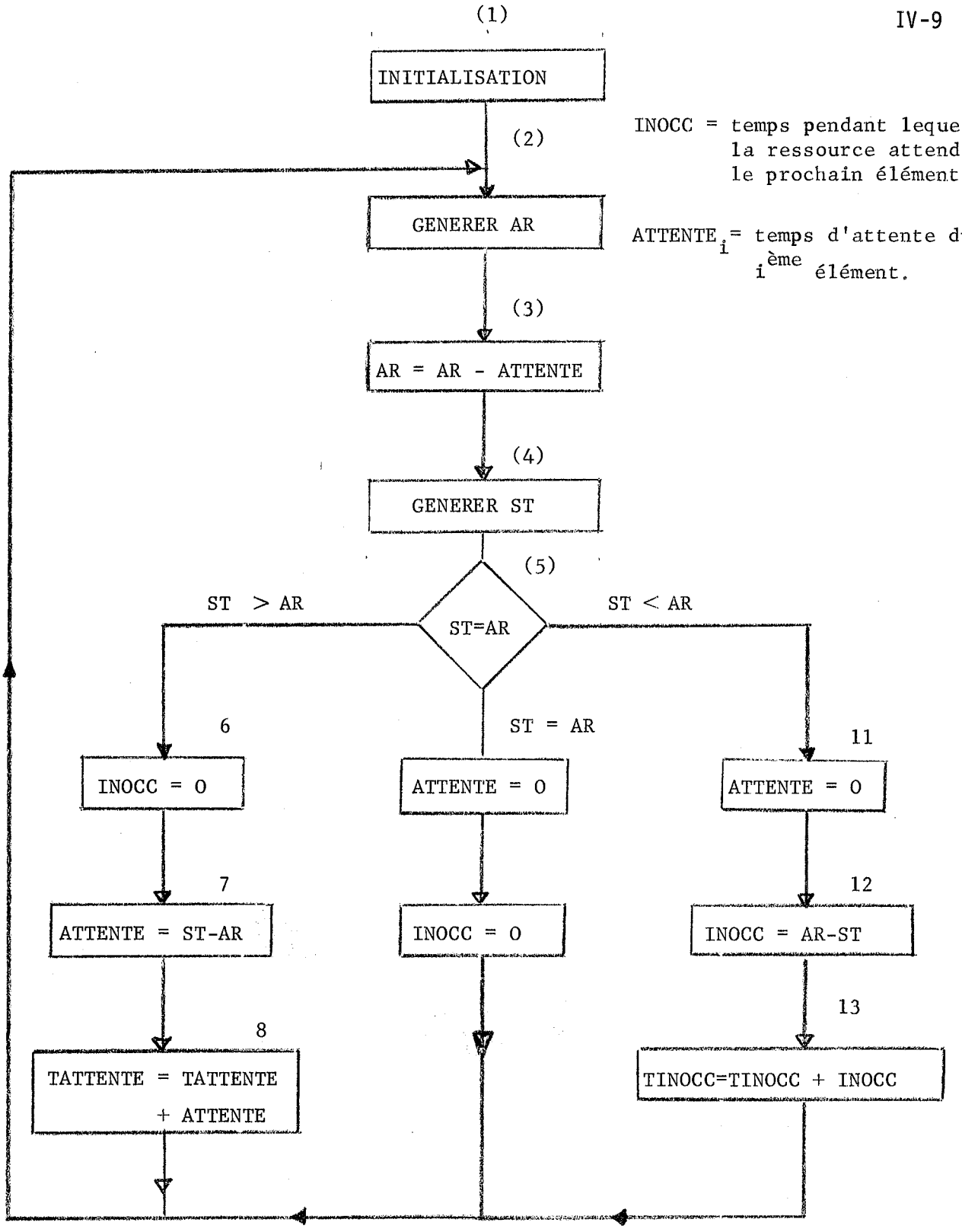


Figure 2 : un modèle d'incrément de temps variable

bloc 1 réalise les conditions initiales, à savoir : temps d'attente, temps d'inactivité etc... mis à 0,et correspond à l'arrivée du premier élément. Un second élément arrive dans le système à une heure générée par la fonction aléatoire convenable (bloc 2). Le temps d'attente qui est égal à 0 à la première itération, est soustrait à l'heure d'arrivée dans le bloc 3. Puis le temps de service est à son tour généré dans le bloc 4 et est comparé à l'heure d'arrivée. Si le temps de service dépasse l'heure d'arrivée, alors le $(i+1)^{ème}$ élément d'entrée arrive avant que la ressource ait achevé le service pour le compte du ième élément. De ce fait, l'élément i+1 tombe en attente et les temps d'attente sont mis à jour (7) et (8) tandis que le temps d'attente de la ressource pour l'arrivée du prochain élément est nul(6). Dans le cas où le temps de service est inférieur à l'heure d'arrivée du prochain élément, on comptabilise les temps d'inactivité de la ressource((12) et (13)). Enfin il n'y a aucun temps d'attente ni d'inactivité à noter dans le cas où le temps de service est égal à l'heure d'arrivée du prochain évènement.

Comme précédemment, ce processus est répété autant de fois qu'on le désire, et la fin de simulation survient par exemple à l'occurrence de la $n^{ième}$ entrée(pour n fixé à l'avance).

Les deux modèles simples que nous venons de présenter éclairaient les possibilités offertes pour représenter le flux de temps. On peut concevoir leur utilisation en tant que procédures dans les programmes complexes qui simuleraient un ensemble de ressources exploitables par plusieurs utilisateurs à la fois.

Le modèle qui représente notre système conversationnel simule les évènements dont la dates d'apparition sont aléatoires; pour cette raison, nous avons choisi de représenter le flux du temps par une méthode à incréments de temps variables. Nous évitons ainsi les nombreux temps morts qui correspondent par exemple aux instants de réflexion des utilisateurs à leurs terminaux, et nous réalisons un gain de temps appréciable au niveau de l'exécution du programme de simulation.

Cependant, nous avons retenu pour le flux du temps une représentation qui diffère de celle proposée par la méthode à incrément variable(M2).

En effet, dans la méthode à incréments de temps variables, toute la logique du système repose sur la valeur de l'intervalle de temps qui sépare l'arrivée de deux éléments. Dans un système conversationnel, il est moins naturel(et plus complexe) de générer l'intervalle de temps séparant l'arrivée d'une commande lancée par un utilisateur de l'arrivée de la prochaine commande (lancée ou non par le même utilisateur), que de déterminer (par une fonction aléatoire) le laps de temps qui s'écoule entre l'instant où l'utilisateur reçoit une réponse du système et l'instant où il envoie son prochain message. En effet, dans la méthode M2, l'intervalle de temps qui sépare deux entrées apparait comme une donnée en fonction de laquelle le système de simulation évolue, alors que dans un système conversationnel, cet intervalle de temps est un résultat ou une conséquence du comportement et de l'état de ce système.

La nature du système que nous avons à simuler impose donc un point de vue différent, présenté dans le paragraphe suivant :

3 -REPRESENTATION DU TEMPS DANS NOTRE MODELE DE SIMULATION

Dans la plupart des systèmes digitaux actuels, on tend à développer la simultaneité vraie qui permet à plusieurs éléments d'un système de fonctionner en même temps. Cette simultaneité se retrouve à différents niveaux : d'abord au niveau hardware le plus bas dans les circuits logiques(on annonce en particulier, pour la quatrième génération d'ordinateurs, des unités centrales capables d'exécuter plusieurs instructions machine à la fois), puis au niveau macroscopique du système. En ce qui concerne le second niveau, le seul à nous intéresser ici, on rencontre fréquemment, dans les configurations actuelles de la troisième génération, une unité centrale qui exécute une tâche pendant qu'un canal(ou des canaux) sélecteur(s) et multiplexeur(s) réalise(nt)

une ou des opérations d'entrées-sorties (on rappelle qu'un canal multiplexeur est capable de soutenir plusieurs entrées-sorties simultanément). Les systèmes multiprocesseurs organisés autour de plusieurs unités centrales fonctionnant en parallèle sont encore des exemples de simultaneité vraie;les systèmes de ce type les plus puissants sont sans doute ceux dont dispose la NASA pour les vols spaciaux(en particulier, cinq IBM 360/75 pour les vols APOLLO). Dans les systèmes conversationnels, comme celui que nous avons à simuler, plusieurs commandes peuvent être lancées simultanément à partir des terminaux et inversement plusieurs messages peuvent être reçus en même temps par plusieurs consoles.

Autant d'exemples qui nous amènent à poser le problème de la représentation des évènements simultanés ou plus exactement des actions parallèles. Nous nous sommes donnés une première représentation à l'aide du MULTITASKING de PL/I (voir 3.1), puis nous avons reconsidéré le problème en traitant séquentiellement les évènements simultanés et les actions parallèles par une gestion rigoureuse de l'écoulement du temps(voir 3.2).

3.1 Représentation par le multitasking de PL/I

Puisqu'il existe dans le système réel des éléments qui travaillent en simultaneité, les éléments du programme de simulation qui les représentent devront travailler au moins en simultaneité apparente. Coordination et liens existant entre ces programmes seront assurés par l'échange de signaux particuliers.

Alan Scherr propose un système de simulation(*) dans lequel chaque élément du système réel est représenté par une procédure du programme de simulation. Ces procédures sont exécutées en parallèle grâce à un petit "Operating System" qui les synchronise en gérant l'apparition des évènements. En s'inspirant de cette idée, nous avons pensé que le multitasking de PL/I fournissait un moyen pratique pour résoudre le problème de la représentation des évènements parallèles (voir Appendice B). Cette option permet en effet d'exécuter plusieurs tâches "simultanément" (il peut s'agir de simultaneité vraie mais aussi de

---

* Une extension de MAD(Michigan Algorithm Decoder)dans lequel on a introduit des instructions permettant de définir le temps et les liens entre procédures.

simultaneité apparente). Il est nécessaire de comprendre le principe du Multi-tasking de faire la distinction entre le programme et son exécution. Une tâche est dynamique et n'existe que lorsque le programme s'exécute; et un ensemble d'instructions peut être exécuté plusieurs fois dans des tâches différentes (en particulier une même procédure peut être exécutée par plusieurs tâches à la fois; une application intéressante est la suivante : pour simuler n unités de disques, il suffit d'écrire une procédure qui représente une unité de disque et de créer n taches distinctes exécutant cette procédure).

Placée dans ce contexte, chaque ressource du système est représentée par une tache et les taches ainsi créées sont exécutées en parallèle, tandis que la tache "CPU" simule le déroulement d'un programme utilisateur en mémoire centrale; parallèlement, les tâches "CHANNEL" simulent les opérations mises en jeu par les canaux réels pour réaliser les opérations d'entrées-sorties et les tâches "UNITE", suivant le type d'unité qu'elles représentent, positionnent un bras de disque ou placent une bande magnétique devant la tête de lecture.

Les liens logiques qui coordonnent ces taches sont réalisés au moyen d'un certain nombre de variables EVENT et d'instructions WAIT(attente d'un ou plusieurs évènements). Mais la nécessité de rassembler des résultats statistiques impose une synchronisation rigoureuse que l'instruction DELAY(retard imposé à une tâche) ne peut réaliser à elle seule. En effet DELAY retarde une tâche d'un certain nombre de millisecondes comptées sur l'horloge réelle du calculateur. Dans ces conditions, les résultats statistiques sont directement soumis au système d'exploitation qui déroule le programme de simulation : il devient alors nécessaire de gérer le temps par une horloge implémentée dans notre programme de simulation. Notre modèle de représentation du temps est identique à celui que nous proposons dans le paragraphe suivant, et nous n'en présentons ici que le principe.

Pour chaque utilisateur, on crée une horloge (on pourrait dire une montre) que le système de simulation met à l'heure aux instants opportuns (par exemple en fin d'exploitation d'une ressource par un utilisateur. Si l'horloge

de l'utilisateur I indique HEURE(I) avant l'exploitation de la ressource, au moment de quitter cette ressource le système de simulation la met à l'heure HEURE(I) + temps d'exploitation de la ressource par I); l'heure de simulation est le minimum des heures indiquées par les horloges des utilisateurs. Chaque tâche représentant une ressource du système réel, consulte, au moment d'activer un utilisateur choisi dans la file d'attente correspondant à cette ressource, l'horloge de cet utilisateur. Si elle est en "avance" sur l'heure de simulation, l'utilisateur n'est pas encore activable et la tâche tombe en attente jusqu'à ce que l'horloge de l'utilisateur "soit à l'heure".

Un évènement dans notre modèle de simulation est le choix du prochain utilisateur à satisfaire, et l'action entreprise est l'exploitation de la ressource par cet utilisateur . Avec cette définition, lorsque plusieurs évènements se produisent simultanément dans des ressources distinctes, le multitasking de PL/I permet d'exécution "simultanée" des taches qui représentent les ressources réelles concernées. En fait, à ce niveau-là, il s'agit d'une simultaneité apparente, puisque le petit superviseur propre au multitasking est contraint par le système d'exploitation (monoprocessor) disponible à Grenoble, de traiter les taches séquentiellement. En conséquence, les évènements simultanés du système réel sont pris en compte de manière séquentielle par le système de simulation.

Par ailleurs le multitasking limite profondément la vocation générale que nous souhaitons pour notre modèle, en limitant à 15 le nombre de tâches actives à un instant donné. Sachant qu'à chaque élément du système réel correspond une tâche du système de simulation, notre modèle doit se borner à simuler des configurations réelles relativement peu importantes.

L'expérience a montré que l'exécution des tâches en "parallèle" entrainait une demande d'allocation mémoire énorme et que de ce fait, le programme de simulation devenait inexploitable : 300 K octets de mémoire centrale étaient nécessaires au déroulement du programme de simulation de 120 K(octets) L'essence même du multitasking, qui veut que plusieurs tâches soient actives simultanément, va à l'encontre des possibilités offertes par l'OVERLAY, qui

seul, aurait permis de minimiser l'encombrement en mémoire centrale.

Toutes ces raisons nous ont amenés à reconsidérer la programmation du modèle de simulation, de façon à nous dispenser du multitasking devenu un élément encombrant. L'expérience tirée du multitasking nous a indirectement enseigné qu'il existe une représentation de l'écoulement du temps qui permet de traiter séquentiellement tous les évènements. Nous en proposons une au paragraphe suivant, qui a l'avantage de fournir un modèle de simulation moins encombrant en mémoire centrale et tout aussi(sinon plus) efficace au niveau de l'exécution.

3.2 Représentation du flux du temps dans notre modèle. (voir figure 3)

Schématiquement, le modèle de simulation voit le système digital comme un ensemble de ressources devant lesquelles s'alignent des files d'attente d'utilisateurs.

On vient de voir que chaque utilisateur dispose d'une horloge HEURE(I) et que l'évènement courant consiste à déterminer dans les files d'attente l'utilisateur dont l'horloge est à l'heure de simulation (MIN(HEURE(I)). Si plusieurs utilisateurs "sont effectivement à l'heure", nous choisissons le premier rencontré qui remplit la condition. Ce choix est arbitraire mais non aléatoire. Les indicateurs sur les heures d'arrivée dans les ressources, et de départ pour chaque utilisateur, permettent le traitement séquentiel des évènements parallèles; et le fait d'avoir satisfait un utilisateur en premier n'a pas d'incidence sur les temps d'attente des utilisateurs qui demandaient un service à la même heure.

L'action entreprise à l'occurrence de l'évènement qui a déterminé l'utilisateur courant I, est l'exploitation de la ressource par I pour une durée ARGUMEN millisecondes. Pour la ressource "unité de disques", ARGUMEN indique le temps moyen de positionnement du bras sur l'enregistrement désiré, ou le temps nécessaire à la lecture(écriture) d'un ensemble d'enregistrements.

Pour un canal, ARGUMEN représente la durée du transfert de l'information entre mémoire centrale et mémoires secondaires ou correspond à la suite des opérations signalant une demande d'entrée-sortie.

Tandis que l'utilisateur courant exploite la ressource r pour son propre compte, les utilisateurs qui se trouvent à cet instant dans la file d'at tente de r ou dans les files d'attente des ressources dont l'activité dépend de r(c'est le cas des unités de disques vis à vis du canal sélecteur auquel elles sont connectées), voient leur temps d'attente (ressource WAIT) augmenter une certaine valeur T(bloc 3), et l'évènement qui activera une ressource pour ces utilisateurs se produira T milliseconds plus tard. (Ce qui revient à dire que leur horloge est "avancée" de T millisconds) (bloc n).

La présence de l'utilisateur J dans la file d'attente de r, ou des ressources dépendant de r, est déterminée par un test de comparaison entre l'horloge de J et l'heure de simulation;il ne suffit pas,en effet, qu'un utilisateur se trouve rangé dans une file d'attente pour dire que cet utilisateur est effectivement présent dans cette file d'attente à l'heure de simulation. Si HEURE(J) est inférieur à HEURE(I) alors J est effectivement arrivé dans la file d'attente avant l'utilisateur courant et T = ARGUMEN(bloc 11). Si la différence HEURE(J) - HEURE(I) est inférieure à ARGUMEN, alors J est effectivement arrivé dans la file d'attente dans l'intervalle de temps ARGUMEN et T = ARGUMEN - (HEURE(J) - HEURE(I)). (bloc 12). HEURE(J) > HEURE(I) reflète le cas où J n'est pas effectivement présent dans la file d'attente (bloc 13). Puisque l'utilisateur J a été retardé de T milliseconds, l'espoir qu'il a de devenir l'utilisateur courant est reporté T milliseconds plus tard. (C'est-à-dire HEURE(J) = HEURE(J) + T); (bloc 4);

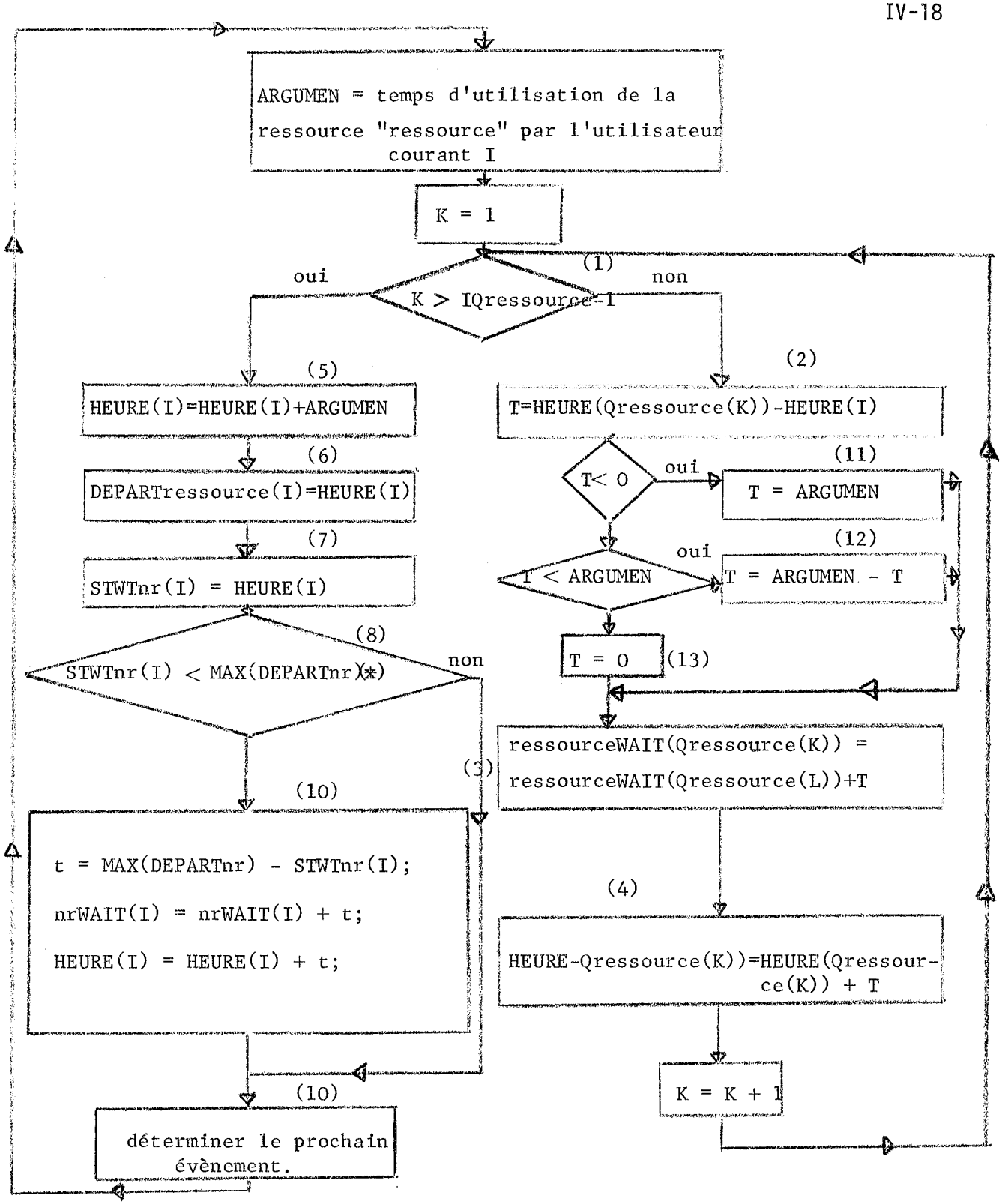
L'heure à la quelle l'utilisateur courant termine l'exploitation de la ressource est notée dans DEPART-ressource (I). Pour cet utilisateur, l'évènement qui le rendra à nouveau actif se produira au plus tôt à l'heure indiquée par sa montre : HEURE(I) = HEURE(I) + ARGUMEN. (bloc 5).

Dans la cas où la poursuite de son travail nécessite une nouvelle

ressource nr, I est placé dans la file d'attente correspondante et l'heure de son arrivée est notée dans STWT -nr(I)-(bloc 7) tandis que l'heure de départ de l'ancienne ressource est notée dans Depart ressource (I)(bloc(6)). STWT-ressource sert à déterminer entre plusieurs utilisateurs se trouvant à l'heure de simulation et qui demandent la même ressource r, celui qui le premier a posé sa candidature. STWT-nr(I) inférieure à l'heure de départ du dernier utilisateur de nr signifie que I est en fait arrivé plus tôt dans cette nouvelle file d'attente. En conséquence, I a déja passé un certain temps t à attendre la nouvelle ressource que nous comptabiliserons dans nr-WAIT(I) (bloc 9) et son horloge est mise à l'heure en l'avançant dans t milliseconds effectivement passées à attendre la nouvelle ressource.

La gestion du temps s'achève sur ces dernières mises à jour, mais le processus reprend à l'apparition du prochain évènement.

L'algorithme que nous venons de décrire est utilisé dans toutes les procédures du programme qui représentent une ressource du système réel, et nous le pensons, est applicable à n'importe quel système réel composé d'un certain nombre de ressources dans lesquelles peuvent apparaître des évènements simultanés.



K, T = variables de manoeuvres

Qressource = file d'attente sur la ressource "ressource"

IQressource = pointeur sur le sommet de Qressource

(*) MAX(DEPARTnr) = donne l'élément maximum du tableau DEPARTnr

Figure 3 : Gestion du temps dans CASSCOU

C H A P I T R E V

ANALYSE DES RESULTATS

1 -LES EXPERIENCES

2 -LES RESULTATS

2.1 Dimension de la zone de données partagée : 20 K octets.

2.1.1 Pages de 1/2 K octets

2.1.1.1 Une unité de disques

2.1.2 Pages de 1K octets.

2.1.3 Pages de 2K octets.

2.2 Dimension de la zone de données partagée : 30 K Octets.

2.2.1 Pages de 1/2 K Octets.

2.2.2 Pages de 1K Octets

2.2.3 Pages de 1/2 K Octets

2.3 Dimension de la zone de données partagée : 40 K Octets

2.3.1 Pages de 1/2 K

2.3.2 Pages de 1K

2.3.3 Pages de 2K

2.4 Dimension de la zone de données partagée : 60 K Octets

2.4.1 Dimension des pages : 2 K

3 -OBSERVATIONS

4 -PERFORMANCE DU PROGRAMME DE SIMULATION

5 -REMARQUE

1 -LES EXPERIENCES

Le modèle d'entrée permet d'abord de définir des paramètres nécessaires au fonctionnement du simulateur, et également de décrire les sessions des n utilisateurs qui ont accès à la partition conversationnelle. A la fois pour simplifier et pour nous placer dans les conditions les plus favorables, nous avons imaginé n sessions identiques. Les 24 commandes qui composent une session décrivent, nous le pensons, le comportement normal d'un utilisateur à son terminal : après avoir initialisé une session, l'utilisateur demande la compilation d'une dizaine de lignes qu'il tape au terminal. Il constitue ainsi son fichier courant. Cette compilation a de grandes chances d'être suivie par quelques commandes de modifications qui entrainent automatiquement la compilation des lignes corrigées. Suit l'exécution de ces 10 lignes qui appelle de nouveaux changements dans le fichier courant. L'utilisateur affecte des valeurs à certains paramètres et demande l'impression de certaines variables au cours de l'exécution qui va suivre. Celle-ci se révèle satisfaisante et le fichier courant est sauvegardé sur disque. L'utilisateur entre à nouveau quelques lignes sur lesquelles il effectue le même genre d'opérations en utilisant les possibilités d'insertion et d'effacement de lignes à l'intérieur du fichier courant. Il numérote puis demande la liste du fichier qu'il a constitué et le catalogue. Il demandera en dernier lieu la compilation puis l'exécution de ses fichiers rangés sur disque. Au total 30 lignes d'instructions FORTRAN.

La mémoire centrale de la partition conversationnelle (cf Appendix A) est partagée en un certain nombre de zones; celle qui nous intéresse ici est la zone de données partagée dans laquelle sont validées les pages des mémoires virtuelles des utilisateurs; c'est dans cette zone que s'exécutent les programmes utilisateurs. Sa taille est très limitée (20 K Octets) en regard des zones de données affectées sur disque aux différents utilisateurs. Nous pensons qu'elle peut être la cause d'une diminution des performances de la Partition. Aussi nous ferons varier sa dimension. (Grâce à la simulation, cette opération ne coûte pas cher!).

Un second point important concerne le système de pagination. Pour un algorithme donné, nous désirons évaluer l'influence de la taille des pages sur les temps de réponse. La dimension proposée pour la Partition Conversationnelle est 1/2K; nous la ferons évaluer jusqu'à 2K en passant par les valeurs intermédiaires 1K et 1K 1/2.

L'environnement hardware de la Partition est lui aussi relativement restreint avec une seule unité de disques à titre de mémoire secondaire. Le rapprochement avec des systèmes plus importants dans lesquels un tambour(ou à défaut une unité de disques) est spécialement utilisé pour la pagination, nous a incité à introduire une seconde unité de disques uniquement pour la pagination.

Nous rappelons le but essentiel de notre recherche : déduire des observations faites sur les temps de réponse, le nombre maximum d'utilisateurs que peut supporter la Partition. Pour cela, le programme de simulation fournit en particulier le temps de réponse de chaque commande lancée et calcule le temps de réponse moyen pour chaque valeur de n distincte.

2-LES RESULTATS

Nous avons réuni dans les tableaux qui suivent les résultats des expériences réalisées; sont entourés les résultats intéressants auxquels nous nous refèrerons dans les commentaires du paragraphe 3.

2.1. Dimension de la zone de données partagée = 20 K octets.

2.1.1 Pages de 1/2 K Octets.

2.1.1.1. Une unité de disques :

Nombre d'utilisateurs	2	3	4	5	6	7	8	9	10	11	12	13	14	15
temps de réponse	796	1172	1514	1868	2095	2369	2702	2923	3135	3436	3661	4447	4499	5317
attente sur l'unité centrale.	1	1	1	2	4	5	2	4	4	3	4	5	4	5
attente sur le canal	0	0	0	0	0	1	0	0	0	0	0	0	1	1
attente sur l'unité de disques	280	580	795	1026	1147	1317	1587	1695	1836	2283	2011	2918	2948	3630
Utilisation de l'unité centrale.	16	19	29	41	66	82	86	132	143	152	167	170	170	182
Utilisation du canal	76	83	96	1177	132	146	155	165	175	183	195	200	197	209
Utilisation des disques	357	379	436	521	578	637	672	712	750	780	830	850	838	884

Chaque valeur est un temps moyen exprimé en millisecondes.

On peut approcher la courbe des temps de réponse moyens en fonction du nombre d'utilisateurs par une droite de pente 348 .

2.1.1.2 Deux unités de disques

nombre d'utilisateurs	2	3	4	5	6	7	8	9	10	11	12	13	14	15
temps de réponse	780	1146	1377	1590	1855	2026	2366	2511	2640	2664	2712	2965	3035	3906
attente unité centrale	1	1	3	2	2	3	9	7	11	13	11	13	12	17
attente canal	4	5	8	13	16	17	21	19	26	31	34	38	40	38
attente unités de disques	260	522	652	740	918	989	1207	1411	1198	1339	1253	1520	1527	2267
utilisation U. Centrale	16	19	31	42	53	70	101	131	144	152	161	165	176	180
utilisation canal	76	81	96	118	130	146	155	166	175	182	189	194	203	206
utilisation des disques	357	379	437	524	572	636	669	713	748	779	805	823	862	874

Chaque valeur, comme dans tous les tableaux de résultats qui suivent, est un temps moyen exprimé en millisecondes.

La courbe des temps de réponse moyens peut être approchée par une droite de pente [285] . Si l'on compare avec la courbe précédente, on observe que les temps de réponse croissent moins rapidement en fonction du nombre d'utilisateurs lorsqu'on ajoute une unité de disques spécialisée pour la pagination.

2.1.2 Pages de 1 K Octets.

	1 unité de disques							2 unités de disques						
nombre d'utilisateurs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
temps de réponse moyens	997	2293	2881	3910	4058	5490	6032	992	2167	2526	3715	3812	5000	5263

La partie de la droite qui approche la courbe des temps de réponse moyens est 750 pour une unité de disques, et 650 pour 2 unités de disque !

2.1.3 Pages de 2K Octets

	2 Unités de Disques						
nombre d'utilisateurs	2	3	4	5	6	7	8
temps de réponse	2135	2662	2831	3868	3878	4668	5510

Dans ces résultats, les temps de transfert des pages de 2K n'ont pas été ajustés et ont été laissés à la valeur des temps de transfert des pages de 1 K. On verra plus loin l'intérêt de cette expérience.

2.2 Dimension de la zone de données partagée : 30 K Octets.

2.2.1 Pages de 1/2 K Octets.

	1 unité de disques							2 unités de disques						
nombre d'utilisateurs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
temps de réponse	796	1104	1425	1756	2414	2553	2801	780	1074	1357	1658	2093	2276	2380

Pente 360 pour une unité de disque, et 320 pour 2 unités de disques.

2.2.2 Pages de 1K Octets

	1 unité de disques							2 unités de disques						
nombre d'utilisateurs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
temps de réponse	818	1195	2105	2854	3934	4314	5793	812	1193	2032	2879	3173	3745	4745

2.2.3 Pages de 1K 1/2 Octets.

	1 unité de disques							2 unités de disques						
nombre d'utilisateurs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
temps de réponse	902	2110	2519	3576	4179	5604	6436	873	2101	2457	3562	4491	4795	5329

2.3 Dimension de la zone de données partagées = 40 K Octets.

2.3.1 Pages de 1/2 K

	1 unité de disques: pente 340							2 unités de disques: pente 290						
Nombre d'utilisateurs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
Temps de réponse	796	1104	1380	1554	1885	2323	2699	780	1074	1320	1500	1771	2135	2367

2.3.2 Pages de 1K

	1 unité de disques							2 unités de disques						
Nombre d'utilisateurs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
Temps de réponse	818	1190	1493	2164	3220	3984	5335	812	1168	1518	2018	2847	3411	4070

2.3.3 Pages de 2K

	1 unité de disques							2 unités de disques						
Nombre d'utilisateurs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
Temps de réponse	793	1847	2146	3054	4037	4721	5230	790	1778	2107	2843	3086	4236	4672

2.4 Dimension de la zone de données partagées = 60K Octets.

2.4.1 Dimension des pages = 2K

	2 Unités de disques						
Nombre d'utilisateurs	2	3	4	5	6	7	8
Temps de réponse	790	1130	1395	1929	2777	3415	3400

Les résultats rangés dans les tableaux précédents appuient et complètent les commentaires que nous faisons dans le paragraphe qui suit.

Les courbes suivantes mettent en évidence des résultats dont certains ont été récapitulés dans les tableaux précédents.

Sur chaque page figurent un certain nombre de points par lesquels on peut faire passer une courbe(qui n'a pas été tracée).

Nous avons adopté les conventions suivantes :

* indique les temps de réponse en fonction du nombre d'utilisateurs,

1 indique les temps d'utilisation de l'Unité centrale en fonction du nombre d'utilisateurs,

2 indique les temps d'utilisation des unités de disque en fonction du nombre d'utilisateurs,

3 indique les temps d'utilisation du canal en fonction du nombre d'utilisateurs,

A indique les temps d'attente sur l'Unité centrale en fonction du nombre d'utilisateurs,

B indique les temps d'attente sur les unités de disque en fonction du nombre d'utilisateurs,

C indique les temps d'attente sur le canal en fonction du nombre d'utilisateurs,

O indique la coïncidence de deux points.

La valeur minimale est indiquée par la valeur du point bas.
La valeur maximale est indiquée par la valeur du point haut.
Enfin le nombre d'utilisateurs varie entre 2 et 8.

S UTILISATEURS SONT DANS L UN QUELCONQUE DES 3 MODES

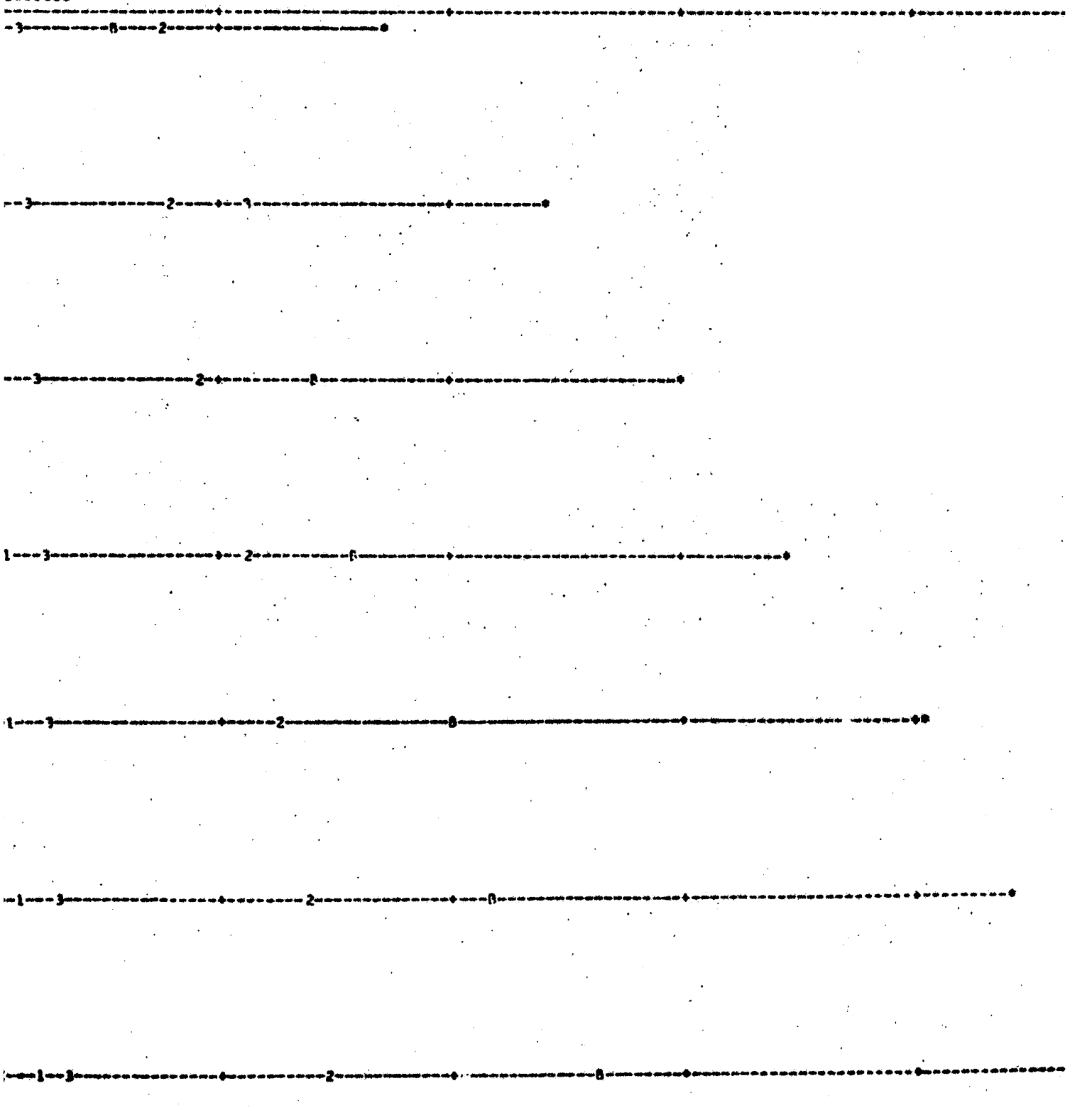
VIRONNEMENT HARDWARE: 1 CANAL SELECTEUR,2 UNITE DE DISQUE, 40 PAGES DE 1/2K CHACUNE

****

NVENTIONS :

= TPS DE REPONSE
= TPS UTILISATION DE L UNITE CENTRALE
= TPS UTILISATION DES UNITES DE DISQUE
= TPS UTILISATION DU CANAL
= TPS ATTENTE SUR L UNITE CENTRALE
= TPS ATTENTE SUR LES UNITES DE DISQUE
= TPS ATTENTE SUR LE CANAL

INT BAS = 1.00 POINT HAUT
2366.00



LES UTILISATEURS SONT DANS N IMPORTE QUEL MODE

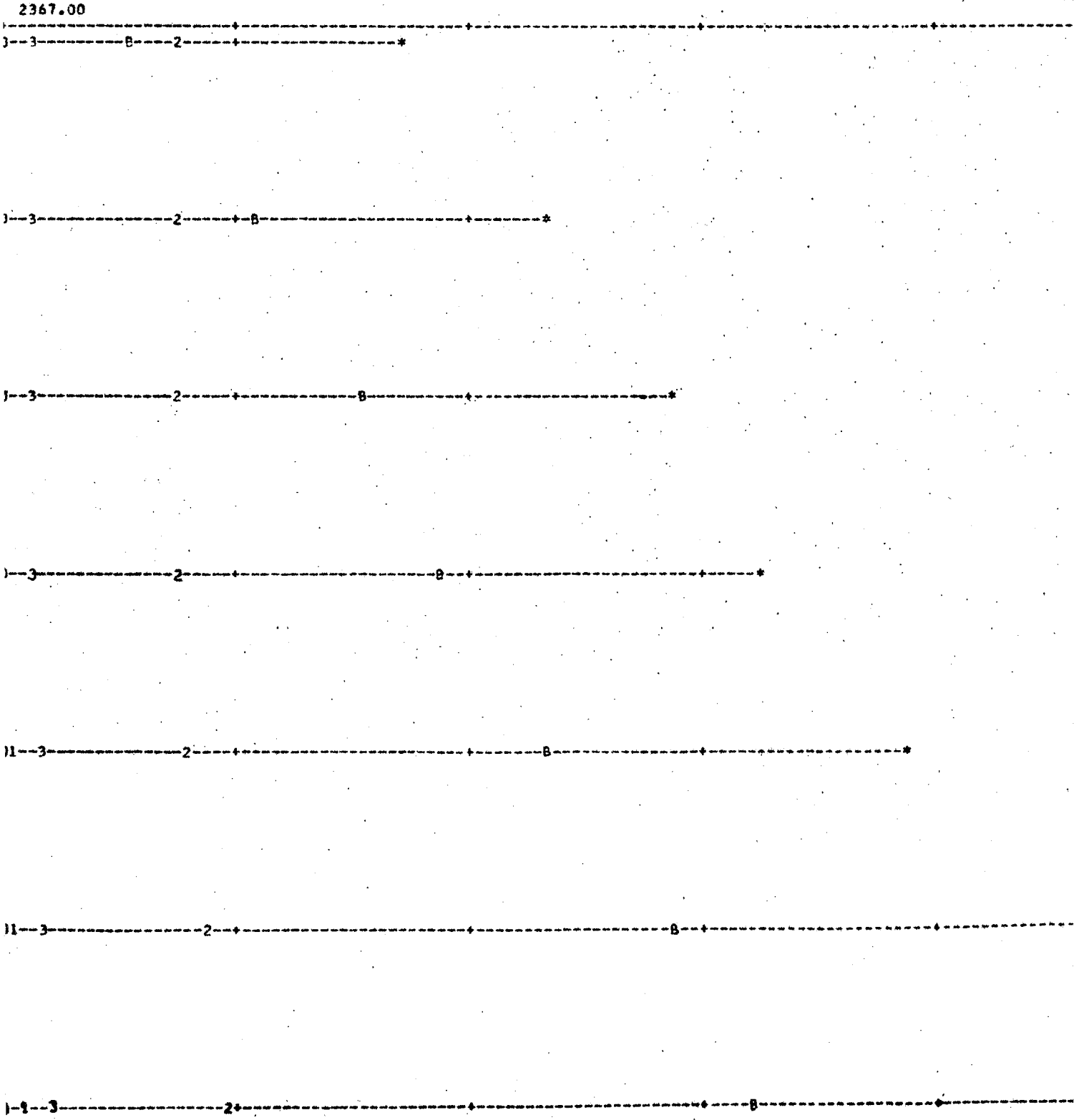
ENVIRONNEMENT HARDWARE: 1 CANAL SELECTEUR,2 UNITE DE DISQUE, 80 PAGES DE 1/2 CHACUNE

****

CONVENTIONS :

* = TPS DE REPONSE
1 = TPS UTILISATION DE L UNITE CENTRALE
2 = TPS UTILISATION DES UNITES DE DISQUE
3 = TPS UTILISATION DU CANAL
A = TPS ATTENTE SUR L UNITE CENTRALE
B = TPS ATTENTE SUR LES UNITES DE DISQUE
C = TPS ATTENTE SUR LE CANAL

POINT BAS = 1.00 POINT HAU
2367.00



5 UTILISATEURS SONT DANS N IMPORTE QUEL MODE

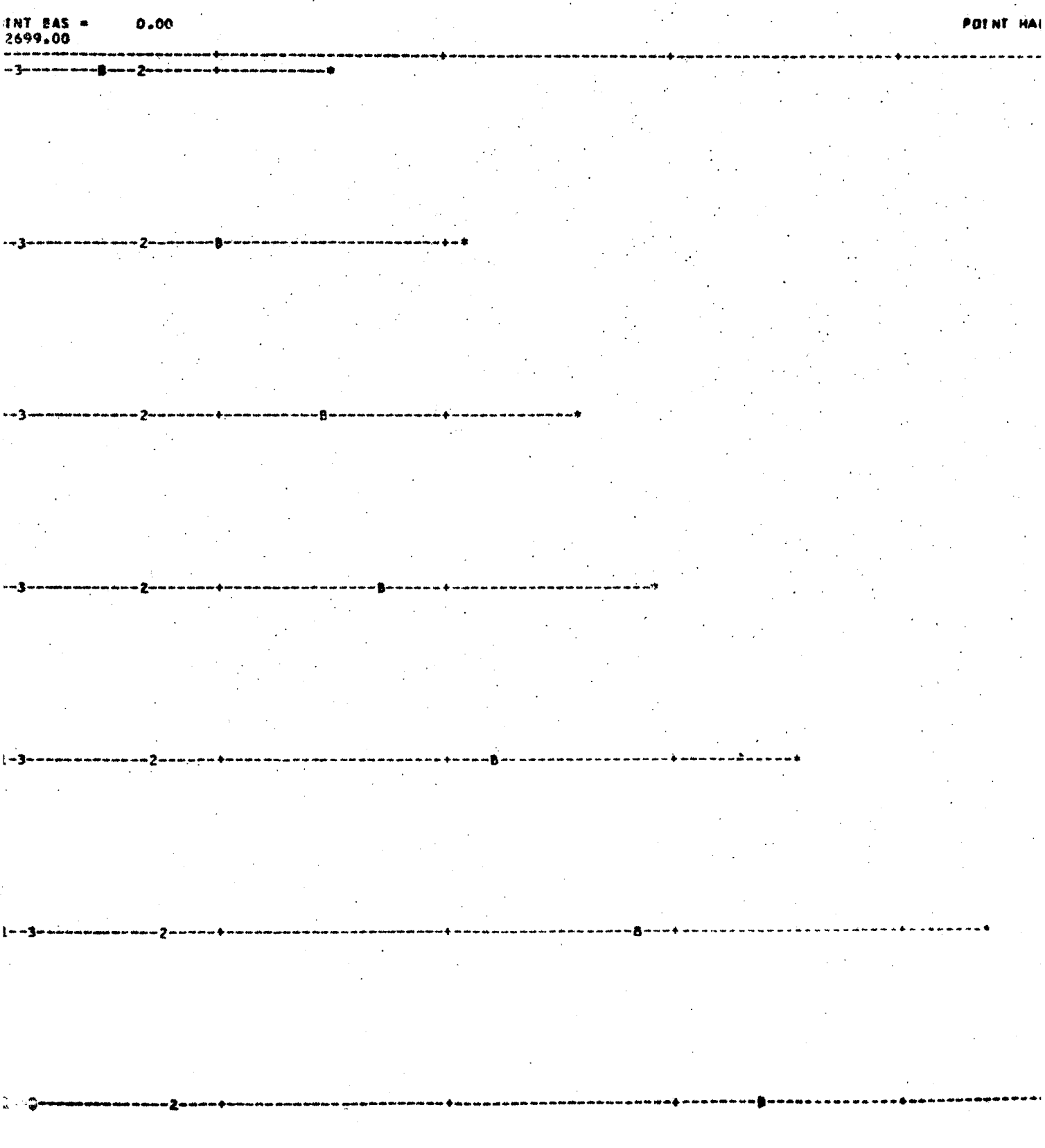
VIRONNEMENT HARDWARE: 1 CANAL SELECTEUR,1 UNITE DE DISQUE, 80 PAGES DE 1/2K CHACUNE

****

INVENTIONS :

= TPS DE REPONSE
= TPS UTILISATION DE L UNITE CENTRALE
= TPS UTILISATION DES UNITES DE DISQUE
= TPS UTILISATION DU CANAL
= TPS ATTENTE SUR L UNITE CENTRALE
= TPS ATTENTE SUR LES UNITES DE DISQUE
= TPS ATTENTE SUR LE CANAL

INT BAS = 0.00 POINT HAI
2699.00



LES UTILISATEURS SONT DANS N IMPORTE QUEL MODE

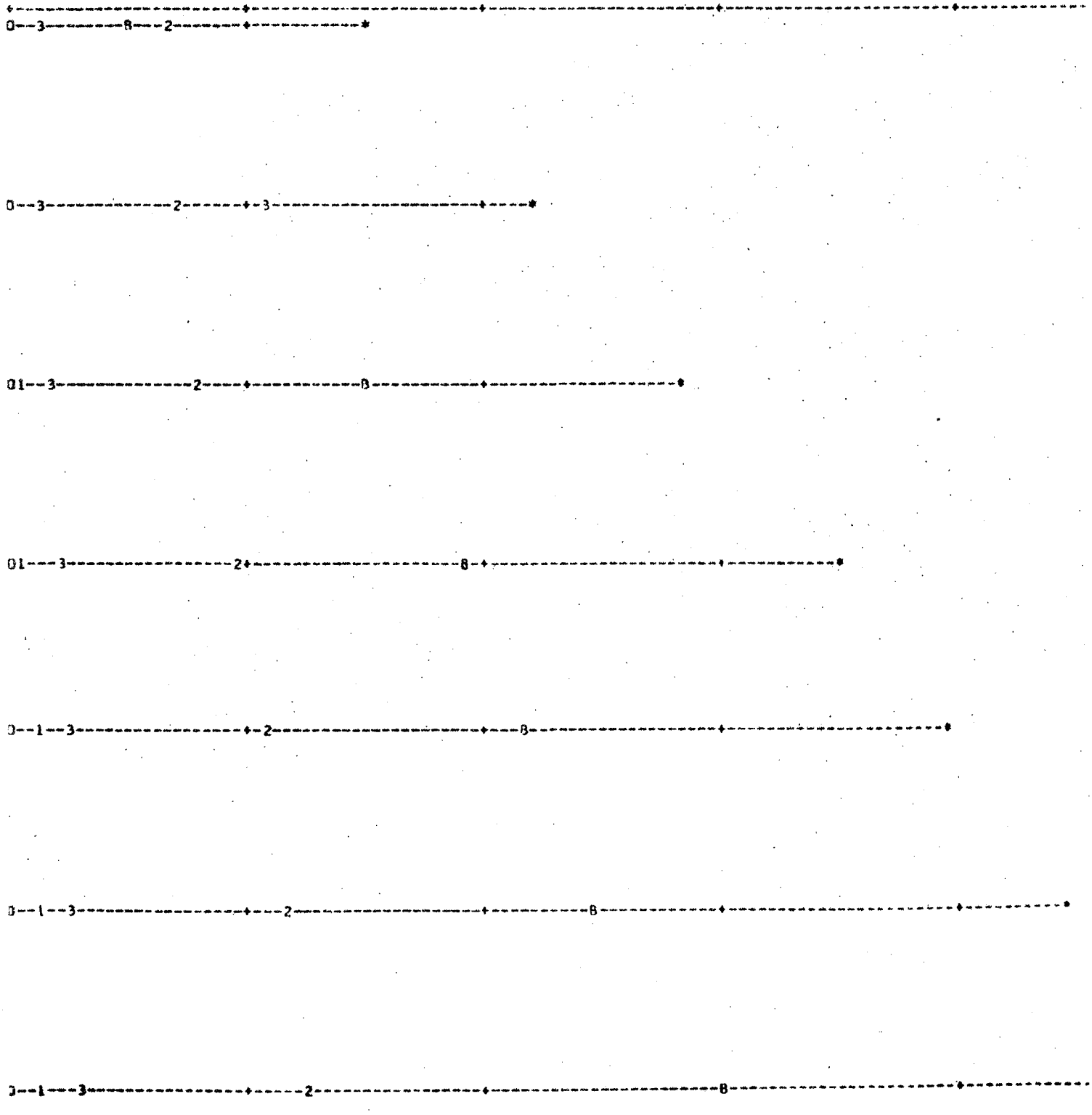
ENVIRONNEMENT HARDWARE: 1 CANAL SELECTEUR,1 UNITE DE DISQUE, 40 PAGES DE 1/2K CHACUNE

****

CONVENTIONS :

* = TPS DE REPONSE
1 = TPS UTILISATION DE L UNITE CENTRALE
2 = TPS UTILISATION DES UNITES DE DISQUE
3 = TPS UTILISATION DU CANAL
A = TPS ATTENTE SUR L UNITE CENTRALE
B = TPS ATTENTE SUR LES UNITES DE DISQUE
C = TPS ATTENTE SUR LE CANAL

POINT BAS = 0.00 POINT H
2702.00



LES UTILISATEURS SONT DANS L UN QUELCONQUE DES 3 MODES

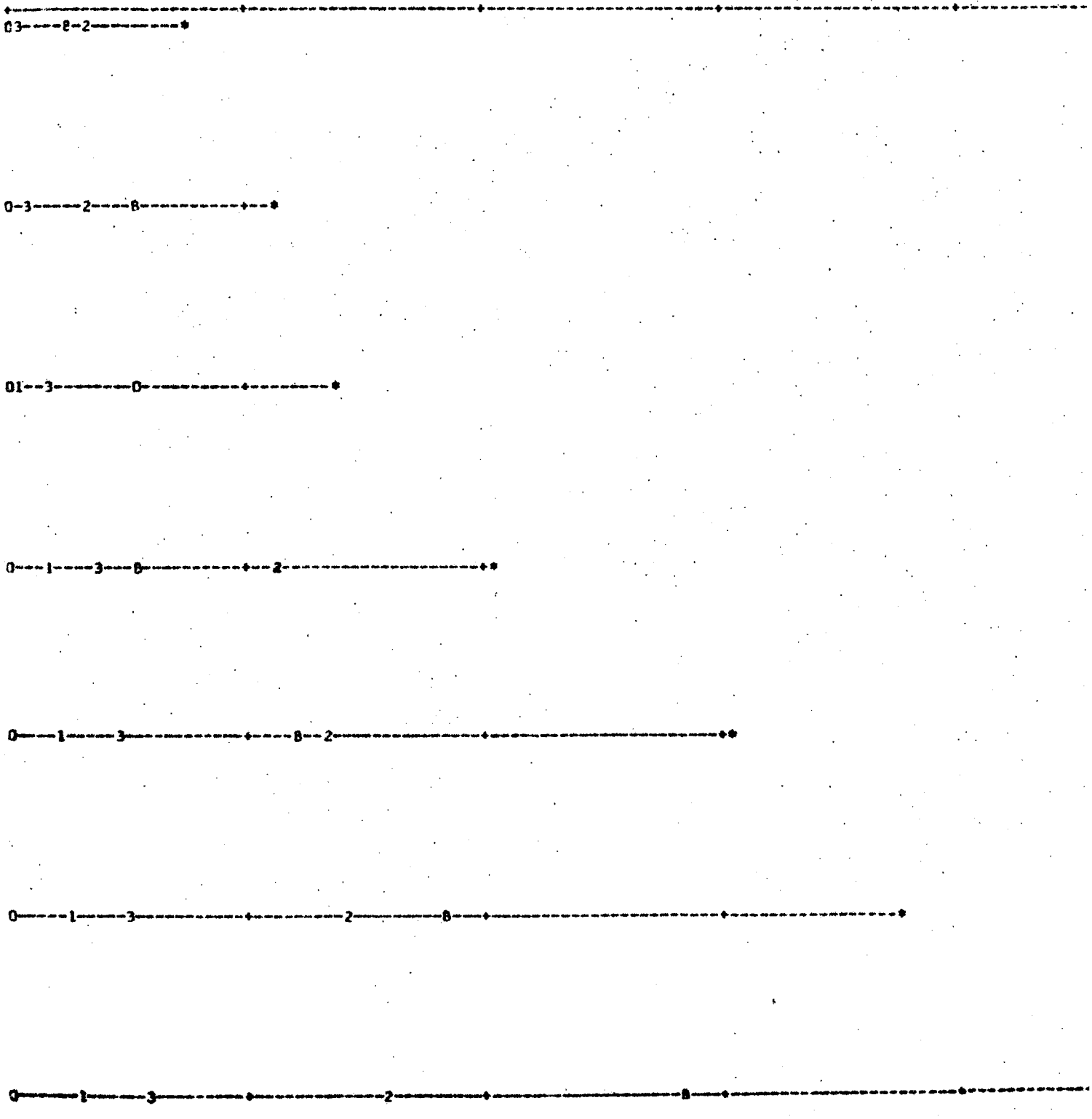
ENVIRONNEMENT HARDWARE: 1 CANAL SELECTEUR,1 UNITE DE DISQUE, 40 PAGES DE 1K CHACUNE

****

CONVENTIONS :

* = TPS DE REPONSE
1 = TPS UTILISATION DE L UNITE CENTRALE
2 = TPS UTILISATION DES UNITES DE DISQUE
3 = TPS UTILISATION DU CANAL
A = TPS ATTENTE SUR L UNITE CENTRALE
B = TPS ATTENTE SUR LES UNITES DE DISQUE
C = TPS ATTENTE SUR LE CANAL

POINT BAS = 0.00 POINT HA
5335.00



LES UTILISATEURS SONT DANS L UN QUELCONQUE DES 3 MODES

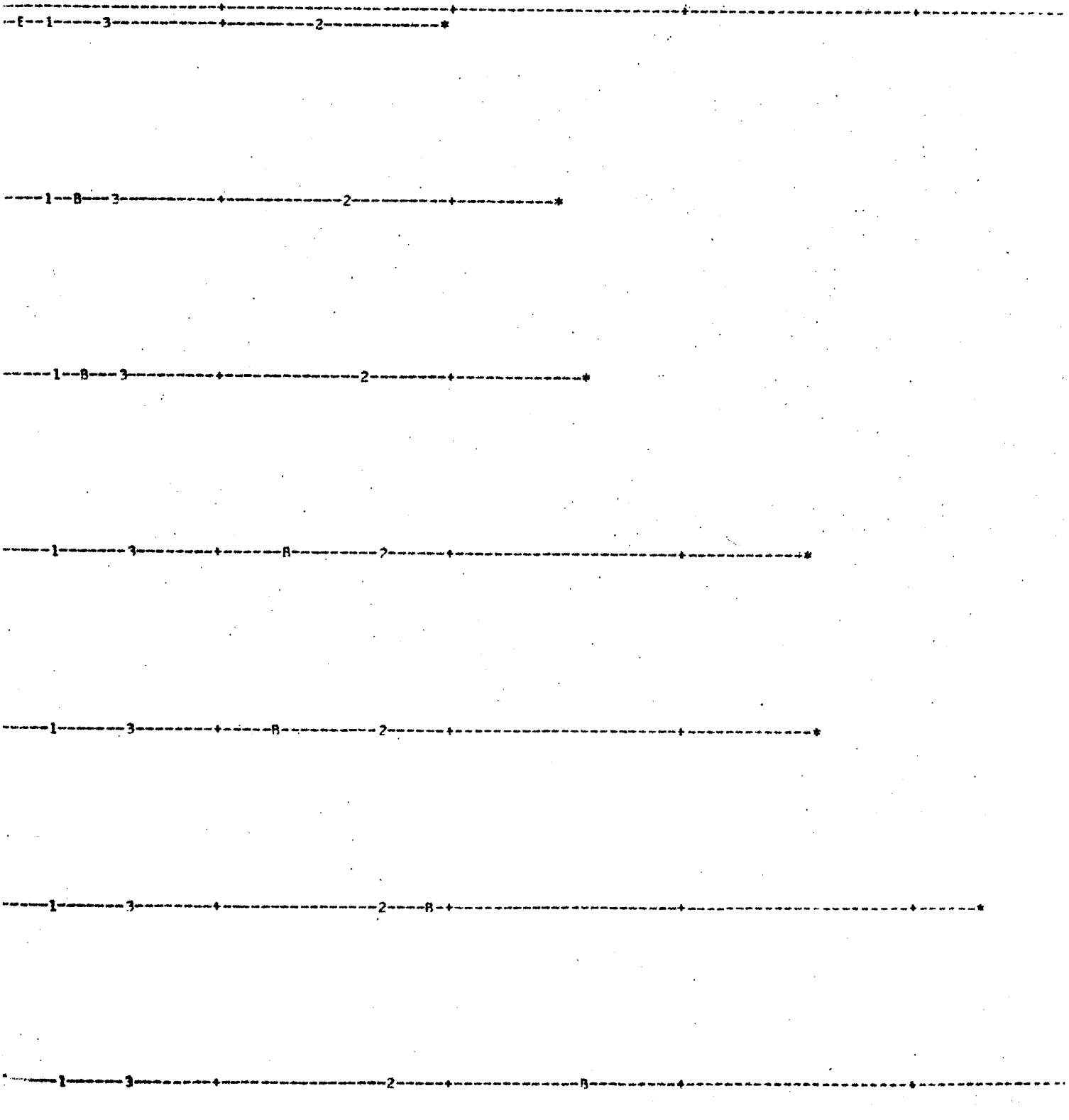
NVIRONNEMENT HARDWARE: 1 CANAL SELECTEUR,2 UNITE DE DISQUE ,10 PAGES DE 2K CHACUNE

****

ONVENTIONS :

= T. J DE REPONSE
= TPS UTILISATION DE L UNITE CENTRALE
= TPS UTILISATION DES UNITES DE DISQUE
= TPS UTILISATION DU CANAL
= TPS ATTENTE SUR L UNITE CENTRALE
3 = TPS ATTENTE SUR LES UNITES DE DISQUE
= TPS ATTENTE SUR LE CANAL

)INT BAS = 3.00 POINT HAI
5510.00



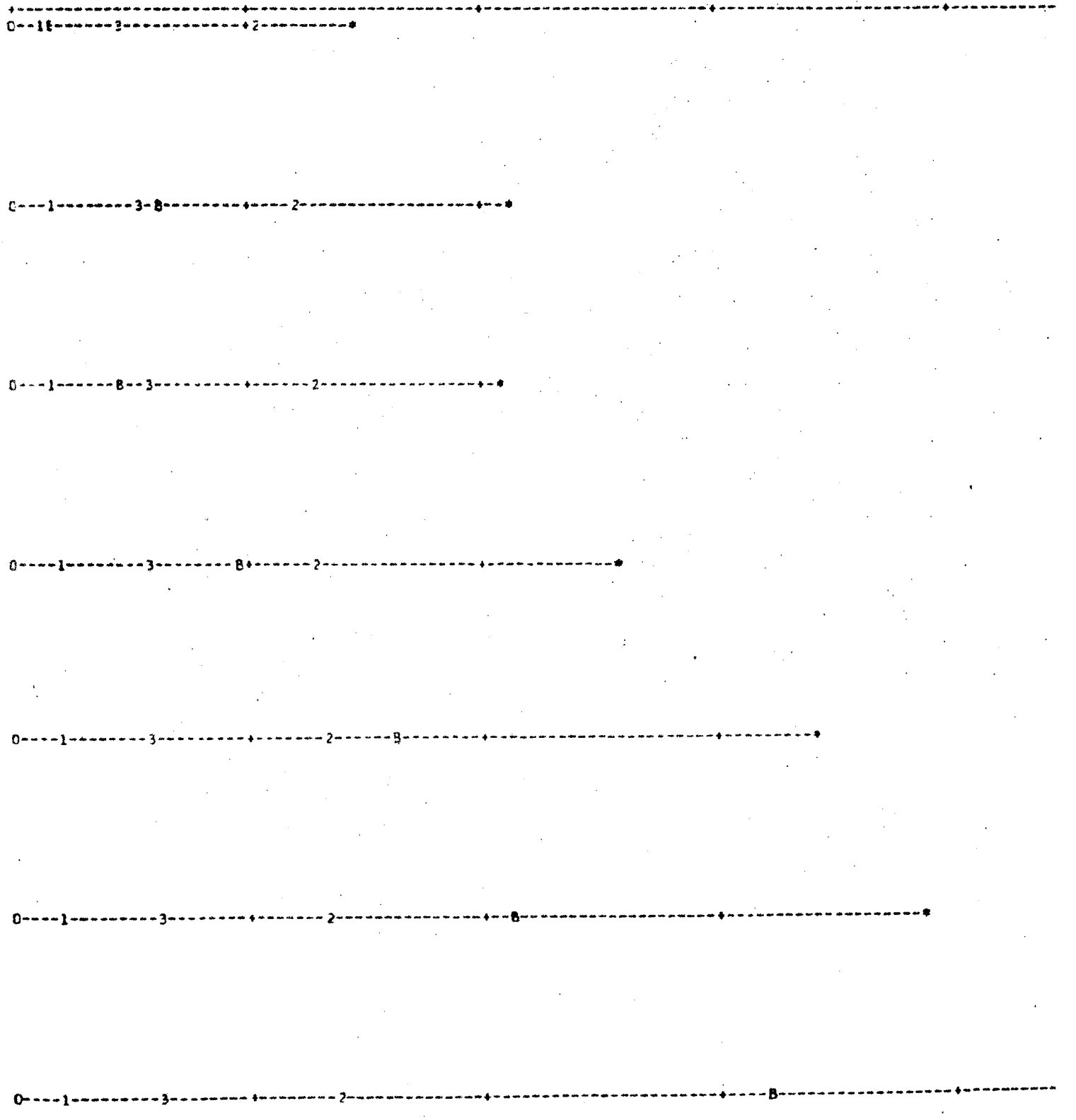
ENVIRONNEMENT HARDWARE: 1 CANAL SELECTEUR,1 UNITE DE DISCUE, 13 PAGES DE 1.5K CHACUNE

****

CONVENTIONS :

* = TPS DE REPONSE
1 = TPS UTILISATICN DE L UNITE CENTRALE
2 = TPS UTILISATION DES UNITES DE DISCUE
3 = TPS UTILISATION DU CANAL
A = TPS ATTENTE SUR L UNITE CENTRALE
B = TPS ATTENTE SUR LES UNITES DE DISCUE
C = TPS ATTENTE SUR LE CANAL

POINT BAS = 0.00 POINT HAUT
8018.00



3 -OBSERVATIONS

Nous pensons qu'un temps de réponse moyen de 3 secondes est un maximum acceptable. Ce temps moyen recouvre en effet certains cas où le temps de réponse dépasse la demi-minute; ceci se produit notamment pour l'exécution des 30 lignes d'un fichier rangé sur disque, alors que la même exécution demande au grand maximum 6 secondes lorsque deux utilisateurs seulement se partagent la partition!

Dans les conditions actuelles du projet -20K Octets pour la zone de données partagée , une unité de disques et des pages de 1/2 K Octets chacune- il apparait que la partition peut au maximum supporter 9 utilisateurs.

On s'aperçoit très vite que les temps de réponse moyens sont directement conditionnés par les temps d'attente sur l'unité de disques, alors que l'attente sur le canal et l'unité centrale sont négligeables. Le goulot d'étranglement se situe donc au niveau de l'unité de disques sur laquelle s'effectuent toutes les entrées-sorties. On peut améliorer la situation en introduisant une seconde unité sur laquelle le système ne fait que des lectures et écritures de pages. Les temps de réponse s'en trouvent sensiblement diminués, notamment lorsque les utilisateurs deviennent nombreux -environ 14% pour n=1,10; entre 20 et 25% pour n=11, 15- Dotée de cette nouvelle configuration hardware, la partition peut satisfaire convenablement 14 utilisateurs. (soit 50% de plus!) Par ailleurs les temps d'attente au niveau du canal et de l'unité centrale ne sont plus tout à fait négligeables. Ils indiquent que l'utilisation de ces deux ressources est mieux optimisée.

Une autre manière de réduire le goulot d'étranglement est de diminuer les entrées-sorties en réduisant les échanges de pages entre mémoire centrale et mémoires secondaires. Ceci n'est réalisable que si l'on augmente la dimension de la zone de données partagée . Les résultats montrent que l'amélioration est à peine perceptible et que la partition équipée de 2 unités de disques de 20K Octets de zone de données partagée est pratiquement aussi performante que celle qui possède une zone de données partagée 2 fois plus grande

et 2 unités de disque . Pour des raisons de coût, il vaut donc mieux ajouter une unité de disque supplémentaire qu'un bloc de mémoire centrale supplémentaire de 20 K octets.

Une seconde étude a trait à l'analyse des performances de la partition en fonction de la dimension des pages. Les résultats des expériences faiges avec des pages de 1K, 1K 1/2, et 2K se révèlent catastrophiques : même en triplant la dimension de la zone de données partagée initiale et en ajoutant une unité de disque pour la pagination, 6 reste le nombre maximal d'utilisateurs acceptables par la Partition lorsque la dimension des pages est fixée à 2K octets. Pour une configuration de 1 unité de disque, 20K octets de zone de donnée partagée et 1K octets pour chaque page(c'est-à-dire pour la même configuration hardware que celle proposée par le projet de la Partition à l'exception de la dimension des pages), les temps de réponse moyens sont presque doublés! Ajouter une unité de disque ou augmenter la zone de données partagée n'améliore guère la situation. Les résultats s'expliquent de la manière suivante : si x est la dimension des pages et p(x) la probabilité pour qu'un utilisateur ait besoin de valider une page, alors :

$p(x) \leqslant p(y)$ pour $x \geqslant y$. Donc les transferts disque-mémoire centrale diminuent.

Prallèlement, si q(x) est la probabilité d'avoir une page libre de dimension x en mémoire centrale,

$q(x) \leqslant q(y)$ pour $x \geqslant y$ puisque le nombre de pages en mémoire centrale diminue quand x croit. Donc les échanges mémoire centrale-disque augmentent tandis que les échanges disque-mémoire centrale diminuent quand x croit.

Dans l'une des expériences que nous avons réalisées, nous avons défini des pages de 2K octets chacune et nous avons déroulé le programme de simulation sans ajuster le temps de transfert entre mémoire-centrale et disque comme le justifie une augmentation de la taille des pages. Nous l'avons laissé égal au temps de transfert des pages de 1K. Même dans ces conditions les plus favorables, les résultats restent décevants et prouvent d'une certaine manière que q(x) décroit plus vite que p(x) quand x croit. En d'autres termes, pour cet algorithme et pour cette configuration donnés, la diminution des échanges de pages dans le sens disque-mémoire centrale ne compense pas l'accroissement des échanges dans le sens mémoire centrale-disque. Si k est la dimension de la zone de données partagée, il est aisément prévisible qu'il existe une valeur de k pour laquelle les

échanges de pages diminuent. Mais il est fort probable que cette valeur se situe au delà des 256K Octets. D'ailleurs, dans la pratique seuls les ordinateurs équipés d'une grande mémoire centrale ont des pages de 2K.

La 3ième point de l'analyse des résultats concerne les files d'attente. Il est bien évident qu'à un instant donné ces files seront remplies au maximum(n-1), puisque les n sessions sont identiques.

En résumé, la situation la plus attrayante est la suivante :

- dimension de la zone de données partagées = 20K
- nombre d'unités de disques = 2 dont 1 pour la pagination
- 1 canal sélecteur
- dimension des pages = 1/2K

Dans ces conditions, la partition est en mesure de satisfaire 14 utilisateurs simultanément.

4 -PERFORMANCE DU PROGRAMME DE SIMULATION

- 1100 instructions PL/I, y compris le générateur de données et l'algorithme de pagination.
- 200K Octets de mémoire centrale nécessaires pour son déroulement sans utiliser les possibilités de l'Overlay (PL/I remplit mal le 2ième critère de choix d'un langage de programmation : cf chapitre 3.5.1).
- 1 seconde d'unité centrale 360/67 est nécessaire pour simuler 25 minutes de session utilisateur.

5 -REMARQUE

Nous aurions pu réaliser nos expériences dans une optique différente : Imposer par exemple au système des performances précises et déterminer la configuration hardware et l'environnement software qui auraient permis la réalisation de ces performances.

C H A P I T R E VI

LES LANGAGES DE SIMULATION

1- GENERALITES - PRESENTATION GENERALE

1.1 Motivation

1.2 Définitions

1.3 L'aide apportée par ces langages

1.4 Classification

2 -CARACTERISTIQUES DE QUELQUES LANGAGES DE SIMULATION

2.1 SIMSCRIPT

2.2 GPSS

2.3 DYNAMO

3 - COMPARAISON ENTRE LES LANGAGES DE SIMULATION

3.1 Domaine d'application

3.2 Contrôle du flux du temps

3.3 Flexibilité et utilité des résultats de sortie

3.4 Efficacité

3.5 Facilité d'emploi

4- CONCLUSION

1 -GENERALITES - REPRESENTATION GENERALE

1.1 Motivation des langages de simulation

Nous avons fait le choix de programmer notre modèle de simulation dans un langage algorithmique général : PL/I. Nous en connaissons déjà les avantages essentiels : flexibilité de la description et de la formulation du modèle mathématique; flexibilité au niveau du format des entrées-sorties, flexibilité aussi dans la réalisation des expériences; enfin coût relativement faible en temps d'exécution, sinon en encombrement mémoire.

Mais ce gain s'accompagne d'une difficulté supplémentaire de programmation qui implique une mise au point souvent longue. Or l'expérience confirme que les opérations que l'on fait intervenir dans la formulation logique et dans la réalisation pratique d'un modèle, sont la plupart du temps semblables, d'un problème de simulation à un autre. Il est donc naturel de songer à la création d'un système de programmation adapté aux problèmes d'écriture des programmes de simulation.

1.2 Définitions.

Il faut souligner la distinction qui existe entre langage de simulation et système de programmation pour la simulation.

Le langage propose un certain nombre de concepts utiles et nécessaires au sujet traité; il fournit aussi un ensemble de règles concernant l'introduction de ces concepts dans les instructions qui définiront le modèle de simulation.

Le système de programmation est l'implémentation du langage sur un ordinateur particulier(ou une série d'ordinateurs); il permet de traduire les instructions du programme en un langage compréhensible par la machine, de les

exécuter et de récupérer les résultats.

Remarque : dans la pratique la plupart des langages proposés jusqu'ici ont été implémentés pour une machine, et dans le cas le plus favorable pour une série de machines. En conséquence, les langages de simulation évoluent à chaque nouvelle implémentation à partir des expériences tirées des implémentations précédentes. Ce phénomène se produira jusqu'à ce que l'on crèe un langage de simulation véritablement indépendant de la machine.

1.3 L'aide apportée par les langages de simulation.

Il est connu que les langages de simulation sont moins efficaces que les langages à vocation générale type PL/I, FORTRAN, etc... Néanmoins, tous facilitent notablement la tâche du programmeur concernant la traduction du modèle d'un système en un programme opérationnel qui génère des résultats statistiques utiles.

Chaque langage de programmation a sa "propre vision du monde" -représente et organise les éléments du monde réel d'une manière qui lui est propre-

Dans le cadre de son "point de vue", il fournit à l'utilisateur des descripteurs d'état au moyen desquels sont décrits les éléments essentiels du système à représenter. La nature de ces descripteurs tend à conditionner notre vision du système réel et donc à influencer le type et la construction de notre modèle. Ces descripteurs sont intimement liés à la structure sous-jacente des données manipulées par le langage de simulation.

L'utilisation des descripteurs d'état permet de représenter le système réel en termes plus ou moins statiques. Cependant, il est bien évident que cette description, seule, ne remplit pas les objectifs de la simulation, puisque celle-ci implique la necessité de changer l'état du système simulé. Un langage consiste à l'origine en un certain nombre de commandes à l'aide desquelles l'utilisateur peut modifier l'état de son système.Pour ces modifications,

l'utilisateur désire à la fois puissance -possibilité de réaliser des modifications étendues dans le système en un minimum d'instructions, et flexibilité -possibilité de faire ces modifications à n'importe quel niveau. Les deux objectifs sont, à certains égards incompatibles. C'est ce qui fait la différence essentielle entre les divers langages de simulation.

La simulation est donc rendue possible

. par la possibilité de décrire un système par l'application de descripteurs d'états appropriés,

. et par la possibilité de modifier l'état du système par l'utilisation des commandes appropriées.

Cependant la simulation est un processus dynamique, c'est-à-dire que l'on reste concerné par la manière dont l'état du système évolue au cours du temps. Cette évolution est souvent complexe et difficile à prévoir. Elle implique donc l'existence,dans le langage de simulation, d'un moyen souple pour contrôler la suite des changements d'états. Tous les langages de simulation possèdent une sorte d'horloge interne, ainsi qu'un mécanisme de contrôle, qui, associés aux commandes, facilitent à l'utilisateur le contrôle de ces changements d'état. L'objectif de ce mécanisme est la réalisation plus ou moins automatique du minimum nécessaire de comptabilité, en sorte que la procédure de définition d'une simulation dynamique sera accomplie aussi facilement et aussi directement que possible.

Le but essentiel d'une simulation n'est pas seulement la représentation du comportement d'un système donné, mais comporte par nécessité la récupération de données statistiques relatives aux performances du modèle. Certes, il est possible de générer, grâce à la simulation, un grand nombre de données de nature différente et en fait, c'est souvent le seul résultat que l'on obtienne. Cependant, il est important que le langage de simulation fournisse à l'utilisateur le type de données qu'il désire sous une forme claire et compréhensible.

Le dernier aspect de l'aide apportée au programmeur porte sur les facilités de mise au point et d'utilisation du modèle dans l' organisation des expériences de simulation.

L'utilisateur en effet, apprécie de pouvoir :

. introduire des modifications dans la structure à la fois statique et dynamique du modèle,

. initialiser l'état du modèle,

. changer la nature des résultats de sortie,

. rassembler plusieurs exécutions successives du même modèle.

On peut y ajouter, des tests non seulement pour détecter les erreurs de syntaxe mais surtout pour localiser les erreurs de logique du programme de simulation.

1.4 Classification des langages de simulation. [Ref 11] [Ref 25]

On distingue deux types de descripteurs d'états dans les modèles de simulation: ceux qui décrivent les éléments passifs et ceux qui décrivent les éléments actifs. Dans un contexte industriel, on pourrait dire que l'élément actif est la machine et que l'élément passif est le produit manufacturé. La relation entre eux est que le produit est manipulé par la machine qui de ce fait, le transforme au cours du temps.

On peut classer les langages de simulation suivant deux facteurs :

- la structure du programme de simulation,
- le type dominant des descripteurs d'états(machine, produit).

On peut aussi classer les langages de simulation en considérant leur

origine -extension ou non de langages algorithmiques déjà existants-

Définir un langage à partir d'un langage général comporte un certain nombre d'avantages pratiques :

- le langage général est la plupart du temps bien connu, ce qui facilite la compréhension et l'emploi du langage de simulation.
- certains problèmes de compilation (du langage de simulation) sont déjà résolus sur toutes les machines qui disposent d'un compilateur du langage de base.

Cependant cette approche comporte aussi quelques inconvénients :

- il est difficile d'introduire une efficacité software dans le codage de nouveaux concepts implicitement décrits dans le langage de base.
- on retrouve les imperfections de l'implémentation du langage général; elles sont même souvent accentuées en raison de la nature spécialisée des langages de simulation.

Nous remarquons qu'en général le développement des langages de simulation suit deux tendances différentes en Europe et aux USA : en Europe, on relève une approche orientée vers les éléments actifs (machine), extension d'Algol-, aux USA on choisit l'approche orientée vers les éléments passifs (produit manufacuré), extension de FORTRAN-

Se rapporter au tableau 1.

TABLEAU 1

Langage de simulation	GPSS	SIMPAC	SIMSCRIPT	SIMULA	CSL	GASP	ESP	GSP	SIMON	PROLLIC
Langage de base	néant	SCAT (assem-leur)	FORTRAN	ALGOL	FORTRAN	FORTRAN	ALGOL	néant	ALGOL	ALGOL

Nous présentons dans ce qui suit les caractéristiques de quelques uns des langages de simulation, choisis parmi les plus connus ou les plus originaux. Le But essentiel est de faire découvrir la manière dont ces langages conçoivent le monde réel, puis de les comparer brièvement sur les points intéressants.

2 -CARACTERISTIQUES DE QUELQUES LANGAGES DE SIMULATION : SIMSCRIPT, GPSS, DYNAMO

2.1 SIMSCRIPT

2.1.1. Les descripteurs d'état

Dans SIMSCRIPT, l'état d'un système est défini en termes d'entités (les éléments ou les objets qui composent le système), d'attributs(propriétés associées aux entités); d'ensembles (réunion d'entités ayant des caractéristiques communes). Le premier travail du programmeur est de définir explicitement toutes les entités accompagnées de la liste complète de leurs attributs et de leur appartenance aux ensembles.

Exemple :	entité	UNITE-PERIPHERIQUE
	attributs	temps d'accès, caractéristiques sur les transmissions de données
	ensemble	DISQUES, BANDES
ou bien	entité	UTILISATEUR
	attributs	commande lancée, temps de réponse, etc....
	ensemble	CONSOLES

Dans certains cas, il est possible de préciser à l'avance les individus d'une entité(par exemple, les différentes unités constamment connectées dans le système d'exploitation) Cette entité est dite permanente; elle est temporaire dans le cas contraire(nombre d'utilisateurs connectées à l'ordinateur).

2.1.2 Procédure de modification du changement d'état d'un modèle SIMSCRIPT .

A un instant donné, l'état du modèle est entièrement défini par les valeurs des attributs et le contenu des ensembles. Cet état se trouve modifié à l'occurrence d'un évènement; un évènement dans SIMSCRIPT est un sous-programme écrit par l'utilisateur à l'aide d'instructions FORTRAN et SIMSCRIPT. Il provoque l'un des changements suivants :

- création ou destruction d'un individu d'une entité temporaire,
- modification de l'appartenance à un ensemble d'un individu,
- changement des valeurs numériques d'un attribut.

2.1.3 Contrôle du temps.

Chaque évènement qui doit se produire dans le système est décrit par un sous-programme. L'heure à laquelle se produit un évènement est le plus fréquemment précisé par le programmeur sous la forme : heure courante +un certain incrément(cf l'instruction CAUSE EVENTname AT TIME + X)et l'occurrence de cet évènement est provoquée automatiquement par le système SIMSCRIPT à l'heure indiquée. Ainsi les changements d'état ont lieu automatiquement et instantanément(par rapport à l'heure simulée) en des points discrets dans le temps. Un évènement prenant fin, l'heure de simulation est automatiquement mise à jour à l'heure d'apparition du prochain évènement. En outre, les évènements dont l'apparition a été prévue à une date ultérieure peuvent être détruits avant d'être exécutés.

2.1.4 Génération des données de sortie.

L'utilisateur peut obtenir à n'importe quel instant l'état du modèle avec tous les renseignements imprimés suivant le format qu'il précise lui-même.

2.1.5 Caractéristiques opérationnelles.

Dans une première étape,un traducteur génère les routines en langage codé de gestion de la mémoire et du contrôle dynamique de la simulation. Ce traducteur transcrit chaque sous-programme SIMSCRIPT en un sous-programme FORTRAN.

Dans une seconde étape, les sous-programmes FORTRAN sont compilés; le "package" ainsi obtenu est prêt pour l'exécution. Les cartes d'initialisation placées par le programmeur à la suite du programme de simulation proprement dit servent à l'affectation des attributs permanents et à la détermination des dimensions des entités permanentes. Grâce à cette procédure d'initialisation, les modifications de données entre deux exécutions n'entrainent pas de nouvelle compilation.

2.2 GPSS

2.2.1 Les descripteurs d'état.

Etape, transactions et équipements sont les éléments clés d'un modèle GPSS.

Chaque étape représente une action particulière; cette action est caractéristique d'une opération particulière qui intervient dans un système réel. Les liens entre ces étapes indiquent la suite des actions qui se produisent dans le système. Si, à un instant donné, plusieurs actions sont possibles, plusieurs connexions sont établies à partir de cette étape pour indiquer l'existence d'un choix.

Les transactions sont les seuls éléments temporaires de GPSS. Elles se déplacent d'une étape à une autre. Chaque transaction est caractérisée par 8 paramètres et par 8 niveaux de priorités distinctes (les transactions de priorité 7 sont servies avant les transactions de priorité 6 et ainsi de suite). A l'entrée d'une transaction dans une étape, le sous-programme associé est exécuté de manière interprétative et modifie ainsi un certain nombre de descripteurs d'état. L'entrée d'une transaction dans une étape entraine donc

Exemple de transaction : les utilisateurs d'un système conversationnel.

GPSS fournit trois sortes d'équipements : FACILITY, STORAGE et AIGUILLAGE que l'on utilise pour représenter des éléments physiques ou des concepts logiques :

Exemple : FACILITY : unité centrale (une FACILITY ne peut satisfaire qu'une seule transaction, donc un seul utilisateur à la fois)

STORAGE : canal multiplexeur (une STORAGE peut satisfaire un certain nombre de transactions simultanément- nombre fixé par le programmeur).

AIGUILLAGE : console connectée ou non.

Un élément "statique" important dans GPSS est représenté par les FILE D'ATTENTE, alignées devant les éléments d'équipement (FACILITY, STORAGE). Lorsqu'une transaction ne peut pas entrer dans un élément d'équipement parce-que celui-ci est plein, GPSS le place dans la file d'attente et à la demande du programmeur comptabilise les temps d'attente, la longueur dela file d'attente et le nombre d'utilisateurs passés dans cette file d'attente.

Exemple :

L'utilisateur d'un système conversationnel qui se présente devant une ressource (représentée par une FACILITY ou une STORAGE suivant la nature de la ressource) et qui ne peut être servi sur le champ parce que la ressource est déjà occupée par un autre utilisateur, est placé automatqiuement par GPSS dans la file d'attente correspondante, alors que dans notre programme de simulation écrit en PL/I nous avons à gérer les files d'attente .

2.2.2 Procédure de modification de l'état d'un modèle GPSS.

En pratique un changement d'état intervient chaque fois qu'une transaction pénètre dans une étape. On peut avoir :

- création et destruction d'une transaction,
- changement de la valeur numérique d'un attribut,
- retard d'une transaction pour un certain temps,
- modification du parcours des transactions.

2.2.3 Contrôle du temps.

Les transactions sont regroupées à l'intérieur de chaînes. On distingue 5 types de chaines, et une transaction peut se trouver dans l'une quelconque des chaînes à un instant donné. La chaine des évènements actuels contient la liste des transactions qui doivent entreprendre une action à un moment situé à la même heure que l'heure réelle (heure de simulation), ou qui sont en retard par rapport à l'heure réelle. La chaine des évènements futurs contient la liste des transactions qui sont en avance sur l'heure réelle. Les autres chaines(chaines des interruptions, chaines de synchronisation, chaines des utilisateurs) ne sont pas essentielles dans la compréhension de l'organisation interne de GPSS.

GPSS choisit la prochaine transaction à libérer dans la chaine des évènements actuels et la fait progresser d'une étape à une autre jusqu'à ce que se présente l'un des trois cas suivants :

- elle passe dans une étape qui la retarde et qui la fait placer dans la chaine des évènements futurs,
- elle rencontre une étape qui l'arrête parce qu'une condition particulière n'est pas remplie dans l'état actuel du système.
- elle est détruite par une étape TERMINATE ou ASSEMBLE.

Il se peut aussi que l'exploration de la chaine des évènements actuels n'ait pas libéré de transaction parce que les conditions attachées à ces transactions n'étaient pas remplies. Dans ce cas l'heure de simulation est mise à jour avec l'heure de départ prévue de la première transaction placée dans la chaine des évènements futurs. Toutes les transactions qui se trouvent à la nouvelle heure de simulation sont ranées dans chaine des évènements actuels et le processus recommence.

2.2.4 Génération des résultats de sortie.

Un certain nombre de renseignements statistiques sont réunis automatiquement par GPSS: pour chaque étape le nombre total des transactions qui l'ont traversée ainsi que le nombre courant des transactions à la fin de la simulation. Ces renseignements sont cependant assez succints et le programmeur peut désirer en obtenir d'autres plus élaborés.

Aux points spécifiés par l'utilisateur, le système collecte les renseignements statistiques relatifs aux files d'attente ; longueur de la file d'attente et temps passé dans lafile d'attente par chaque transaction. Ceci est obtenu grâce aux étapes QUEUE et TABULATE.

L'utilisateur a la possibilité de demander la récupération de données statistiques dans un histogramme (fréquence d'arrivée des transactions, temps passé dans le système, utilisation moyenne et contenu maximum d'une FACILITY ou d'une STORAGE).

2.2.5 Caractéristiques opérationnelles;

GPSS est utilisable dans la série IBM/7090 mais l'est aussi dans la série IBM/360.

GPSS opère sur la définition du modèle d'une manière essentiellement interprétative; il offre une grande flexibilité pour réaliser plusieurs exécutions successives du même modèle : l'utilisateur a en effet le moyen de modifier son propre modèle après une première exécution ; une carte CLEAR qui détruit toutes les données statistiques et toutes les transactions de l'expérience précédente, suivie des cartes de redéfinition des étapes repérées symboliquement par une étiquette permet d'effectuer ce changement.

```
Exemple :   ETI     GENERATE      10   générer les 10 utilisateurs du
                                       système conversationnel.
            START   signale que le programme est complet et que la
                    simulation peut commencer.
            CLEAR
            ETI     GENERATE      15
            START   recommencer la simulation en plaçant 15 utili-
                    sateurs dans le système conversationnel.
```

2.3 DYNAMO

2.3.1 Descripteurs d'état.

DYNAMO permet de décrire des modèles de systèmes constitués par un flot d'informations indissociables. Ces modèles continus sont représentés mathématiquement par des équations différentielles qui décrivent la fréquence de hhangement des variables dans le temps.

Un modèle continu est un modèle dans lequel chaque variable fondamentale est continue et possède une dérivée première par rapport au temps. En d'autres termes, l'état du système est déterminé à chaque instant par les valeurs de ces variables continues; et l'on ne peut concevoir des changements de cet état discrets dans le temps. Une telle représentation correspond intuitivement à celle des phénomènes physiques, économiques ou sociaux.

Dans les modèles écrits à l'aide de DYNAMO, l'information relative à l'état du système à l'instant présent est utilisée pour déterminer l'état futur du système(information-feedback); en général, les états successifs de ce système ne dépendent pas de variables extérieures au système (closed loop system)- L'intérêt central est porté sur la structure interne du système et sur la manière dont les variables agissent les unes sur les autres.

2.3.2 Procédure de modification de l'état du modèle.

DYNAMO utilise des équations aux dérivées partielles, qui sont, à la limite, une approximation d'équations différentielles.

Les variables qui décrivent le système sont déterminées par ces équations : leur valeur à l'instant K dépend des valeurs à l'instant précédent J et des fonctions de changement pendant l'intervalle DT.(K = J + DT où DT est l'incrément de temps fixe standard).

Exemple :

TOTPRODUCTION.K = TOTPRODUCTION.J + (DT) (PRODUCTION.JK)

i.e la production totale est égale à ce qu'elle était à l'instant J, plus la production réalisée pendant l'intervalle DT.

DYNAMO permet aussi à l'utilisateur d'introduire de nouvelles fonctions pour faciliter le processus de description du système. Ces fonctions comprennent :

- des équations auxiliaires qui rendent possible l'introduction de variables pluscomplexes,
- des équations d'initialisation,
- des constantes,
- des fonctions qui permettent de changer l'état du système indépendamment des états présents et passés. Par exemple : R = STEP(P,Q) i.e R prend la valeur initiale donnée jusqu'à l'instant Q et à partir de cet instant, la valeur P,
- fonctions de récupération de données.

2.3.3 Contrôle dynamique du modèle.

Au début de la simulation, l'état du système est défini par les valeurs initiales ou les conditions initiales comprises dans les équations DYNAMO. Puis les valeurs des variables internes sont calculées, une fois et une seule à la fin de chaque intervalle de temps. DYNAMO dispose d'une horloge à incrément de temps fixe. Le nombre de ces incréments de temps, donc la durée de la simulation, sont évidemment précisés par le programmeur.

Une fois les variables calculées, leurs valeurs sont imprimées sous la forme d'un graphe ou d'un tableau à chaque itération et à la demande de l'utilisateur.

2.3.4 Génération des données de sortie.

A ce que nous avons dit précédemment, nous pouvons ajouter, que le programmeur indique la période durant laquelle il désire obtenir des résultats de sortie.

2.3.5 Caractéristiques opérationnelles.

DYNAMO fournit le moyen de modifier des constantes et les sorties de résultats sans qu'une compilation soit nécessaire.

3 -COMPARAISON ENTRE LES LANGAGES DE SIMULATION

	SIMSCRIPT	GPSS	DYNAMO
Unité de base	évènement	étape	équations
Le programmeur doit-il définir l'unité de base ?	oui	non	oui
Comment les unités de base sont-elles liées entre elles ?	instruction du type "CAUSE EVENT"	choix de la prochaine étape	résolution séquentielle des équations
Plusieurs occurrences d'une même unité de base peuvent-elles exister simultanément ?	oui	oui	non
Est-ce l'unité de base peut prendre du temps?	non	oui	—
horloge	l'évènement le plus imminent	l'évènement le plus imminent	incrément de temps fixe
initialisation de l'exploration	après chaque évènement	après des changements précis dans l'état du système simulé.	à chaque cycle de temps.

TABLEAU 2

SIMSCRIPT	GPSS
entité temporaire	transaction
attribut temporaire	paramètre de transaction
entité permanente	étape FACILITY STORAGE aiguillage file d'attente TABLE
attributs permanents	variables standards du système GPSS
ensemble	transactions retardées

TABLEAU 3 : terminologie -comparaison

3.1 Etendue du domaine d'application

Un facteur important pour établir le domaine d'application d'un langage de simulation est le niveau auquel sont définis les concepts du modèle. Nous considérons quatre niveaux par ordre croissant de flexibilité et d'étendue du domaine d'application.

- concepts construits dans le langage de simulation. L'utilisateur doit accepter un tel concept comme donné et entrevoir son propre problème en ces termes.
- concepts définis par l'utilisateur .Une fois défini, le concept reste fixé pour la durée de la simulation.
- concepts définis au début de la simulation ; L'utilisateur établit un concept et affecte certaines valeurs avant l'exécution du modèle proprement dit sans qu'une traduction-compilation soit nécessaire.

- concepts définis au cours de la simulation. On atteint là le maximum de flexibilité puisque les modifications sont réalisées au cours de l'exécution du modèle.

SIMSCRIPT demande à l'utilisateur de définir les unités de base(voir tableau 2). Il a de ce fait un domaine d'application largement étendu. Par contre GPSS fournit toutes les unités de base. De ceci résulte inévitablement une généralité moins grande; en échange la nature des entités et des attributs de GPSS est telle que l'utilisateur dispose encore d'une grande flexibilité(par exemple, les transactions peuvent être créées et détruites à n'importe quel moment au cours de la simulation) .On pourrait dire que GPSS fournit une réalisation spécialisée des concepts généraux accessibles dans SIMSCRIPT. Cette spécialisation restreint son domaine d'application mais lui confère du même coup une puissance supérieure de traitement des problèmes de son domaine d'application. DYNAMO entrevoit le monde de manière tout à fait différente; il est fondamentalement tourné vers la représentation des modèles continus.

3.2 Contrôle du flux du temps.

C'est probablement avec DYNAMO qu'il est le plus facile d'implémenter la contrôle dynamique désiré, bien que ce travail soit à la charge du programmeur et que cela lui demande d'avoir parfaitement assimilé le fonctionnement du système à simuler. Les équations DYNAMO sont résolues une fois et une seule à chaque incrément de temps (incrément fixe, on le rappelle). Cette approche convient évidemment à la représentation des processus continus.

SIMSCRIPT associe à l'exécution d'un évènement une entité temporaire dite EVENT NOTICE. Ces entités sont regroupées dans des ensembles suivis de leur occurrence d'apparition. Ceci est automatiquement réalisé par SIMSCRIPT qui choisit l'EVENT NOTICE dont l'heure d'apparition est la plus proche de l'heure de simulation. Cette fonction est très utile à l'utilisateur qui établit simplement l'apparition d'un évènement.

GPSS réalise la même fonction en employant la transaction à la fois comme entité temporaire et comme EVENT NOTICE. De plus, GPSS permet au programmeur de bloquer l'occurrence (déjà prévue) d'une entité, en fonction de l'état du système. Un changement d'état conduit GPSS à analyser de nouveau l'état de l'activité retardée et, éventuellement, provoque son apparition. GPSS est très certainement de tous les langages de simulation celui qui offre le mécanisme le mieux structuré pour l'ordonnancement des activités.

3.3 Flexibilité et utilité des résultats de sortie.

L'utilisateur peut désirer obtenir les résultats suivants :

- Rapports sur l'état du système à un instant précis.
- données relatives au comportement du système sur une suite d'instants successifs,
- statistiques déterminées en fonction des données accumulées au cours d'une longue période de simulation.

Tous les langages de simulation permettent plus ou moins d'obtenir ces résultats; mais il apparait que chacun d'entre eux est orienté vers un type particulier de sortie : SIMSCRIPT offre de loin les possibilités les plus séduisantes pour les rapports générés à un instant donné; DYNAMO est spécialisé dans l'impression des résultats à des instants successifs; GPSS offre à l'utilisateur l'approche la plus automatique et la plus complète pour l'obtention des statistiques.

3.4 Efficacité.

Deux facteurs importants interviennent dans la définition de l'efficacité : vitesse d'exécution et encombrement mémoire.

Il est certain que les langages compilés sont plus efficaces au niveau de l'exécution que les langages interprétés : en ce sens, SIMSCRIPT et DYNAMO sont plus rapides que GPSS. Le contrôle dynamique complexe que GPSS effectue automatiquement diminue encore sa vitesse d'exécution.

L'utilisation de la mémoire est optimisée par SIMSCRIPT qui récupère la place utilisée par les entités temporaires et qui effectue les réservations de mémoire pour les entités permanentes au démarrage de la simulation. Mais il faut noter aussi que les langages interprétés peuvent prendre moins de place.

En règle générale, ces langages spécialisés sont moins efficaces que les langages algorithmiques généraux.

3.5. Facilité d'emploi.

Puisque les langages de simulation sont des outils pour la mise en oeuvre de diverses expériences, il est utile (voire nécessaire) que le programmeur puisse introduire des modifications dans son modèle avec un minimum d'effort. A cet égard, les langages compilés sont désavantagés. GPSS accepte n'importe quelle modification avant le démarrage de l'exécution.du modèle; il accepte même plusieurs exécutions successives du même modèle. A l'opposé, SIMSCRIPT demande une traduction et une compilation pour la plupart des modifications. Néanmoins, grâce à sa procédure d'initialisation SIMSCRIPT permet un certain nombre de modifications entre deux exécutions. DYNAMO autorise des changements encore plus limités .

Les facilités offertes à l'utilisateur pour la mise au point de son modèle est un facteur important en programmation.A ce niveau, l'utilisateur dépend des messages et des résultats générés par le programme de simulation. L'utilisateur trouve commode de travailler au niveau du langage source. GPSS fournit des moyens sérieux en détaillant les erreurs dans le langage source. SIMSCRIPT demande à l'utilisateur de travailler en FORTRAN pour les mises au

point profondes. DYNAMO, par sa nature ne demande pratiquement pas à l'utilisateur de s'écarter du langage source.

4 -CONCLUSION

De cette analyse apparait un fait certain : chaque langage possède un point fort dans lequel il dépasse nettement tous les autres. C'est pourquoi il est difficile d'affirmer qu'un langage de simulation particulier est supérieur à un autre. C'est au programmeur qu'il revient de choisir celui qui convient le mieux au traitement de son problème ou éventuellement de sé déterminer, comme nous l'avons fait, sur un langage algorithmique général.

Est-ce utopique de penser que le langage de simulation de l'avenir doive rassembler le meilleur de chacun des langages déjà existants ? Qui ne souhaite que ce langage soit efficace, général dans son domaine d'application et facile d'emploi -c'est-à-dire facile à apprendre, et offre des possibilités puissantes de mise au point ?

A P P E N D I X A

LA PARTITION CONVERSATIONNELLE : DESCRIPTION

---

1 - PRESENTATION GENERALE DE LA PARTITION

2 -POSITION DE LA P.C PAR RAPPORT A SIRIS II

3 -STRUCTURE DE LA P.C.

- 3.1 Les modules objets-programmes
- 3.2 Zone de données partagée
- 3.3 Zones commune de contrôle
- 3.4 Zones commune de traitements de commandes
- 3.5 Zones de recouvrement.

4 -LOGIQUE DE LA P.C.

- 4.1 Fonctionnement
- 4.2 Schéma de fonctionnement.

5 -ROUTINES UTILITAIRES DE LA P.C.

- 5.1 Présentation du principe de l'adressage
- 5.2 Sous-programme de pagination
- 5.3 Algorithme de pagination.

Le but de cet Appendix n'est pas de donner une description détaillée de l'organisation de la partition conversationnelle. Le lecteur pourra consulter à cet effet les travaux de Messieurs DELPUECH & CHUPIN [Ref 2] .

Il consiste à situer la partition dans le système SIRIS II dont elle fait partie, et à montrer comment un système conversationnel peut être conçu.

Nous entrerons dans le détail sur certains points, notamment sur tout ce qui concerne la pagination de façon à faciliter la compréhension du programme de simulation.

1 - PRESENTATION GENERALE DE LA PARTITION

Le système conversationnel que nous simulons est l'une des trois partitions, série, temps réel, parallèle qui composent le système SIRIS II développé par la Compagnie Internationale pour l'Informatique. C'est pourquoi nous l'appelons indistinctement Partition Conversationnelle.

L'environnement Hardware de la partition conversationnelle comprend:

- un ordinateur IRIS 50 doté de 64 K de mémoire centrale,
- une unité de disques,
- un ruban perforé,
- et un certain nombre de consoles(ou téléimprimeurs).

Cette partition permet à un utilisateur disposant d'un téléimprimeur d'élaborer et d'exécuter des programmes écrits en FORTRAN qu'il conserve, s'il le désire, sur disque ou ruban perforé. L'intéraction qui peut s'établir entre l'utilisateur et la partition conversationnelle est réalisée au moyen d'un langage de commandes. Ces commandes sont groupées suivant leurs fonctions générales:

- Administration (connexion et déconnexion d'un utilisateur, messages adressés à l'opérateur, création et interrogation d'horloge).
- Compilation (compilation, modifications, écriture d'un fichier source)
- Exécution (exécution d'un fichier objet, affectation et impression de variables, trace et exécution pas à pas,...)
- manipulation de fichiers (duplication, changement de noms, suppression de fichiers sur disque).

A chaque groupe de commandes est associée une priorité que le système affecte aux utilisateurs et à laquelle se refère le critère d'activation d'un utilisateur.

Un trait caractéristique est que le compilateur FORTRAN est incrémentiel, la ligne d'entrée définissant l'incrément (les lignes d'entrée sont donc compilées au fur et à mesure de leur arrivée dans le système).

2 -POSITION DE LA PARTITION CONVERSATIONNELLE PAR RAPPORT A SIRIS II

Dans le système d'exploitation actuellement défini par la CII, la partition conversationnelle constitue un programme de la partition des travaux parallèles.

Si aucun autre programme n'est chargé dans cette partition, la partition des travaux parallèles s'identifie donc à la partition conversationnelle.

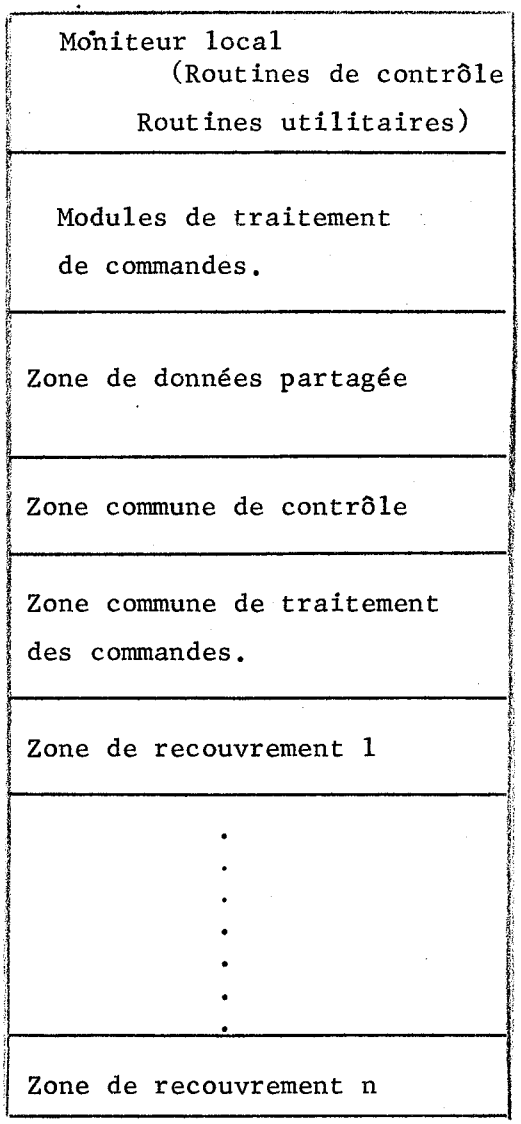
L'initialisation de la partition conversationnelle se fait par l'intermédiaire de l'opérateur : celui-ci génère une interruption au panneau de commandes et indique qu'il veut introduire un programme parallèle. Le programme de chargement des travaux parallèles, est alors activé et demande à l'opérateur le nom du programme à charger. La réponse : FORTRAN CONVERSATIONNEL IRIS 50 (FCI) provoque la recherche du programme FCI dans une bibliothèque précise : le 1er segment de ce programme est ensuite chargé en mémoire central et reçoit le contrôle.

3 -STRUCTURE DE LA PARTITION CONVERSATIONNELLE

Le programme FCI a été découpé en segments au moment de l'édition de liens. A un instant donné, la partition conversationnelle est constituée d'un certain nombre de segments chargés à partir du FCI.

La Partition se compose

- des zones programmes comprenant les modules objets programmes résidents ainsi que leurs données locales,
- une zone de données partagée,
- des zones communes de contrôle et de traitement des commandes,
- des zones de recouvrement.



3.1 Les modules objets programme.

L'ensemble de ces modules constitue

- d'une part le moniteur local de la partition avec les routines de oontrôle et les routines utilitaires,
- et d'autre part le module de traitement des commandes accompagné des sous-programmes nécessaires à ces traitements.

Les routines de contrôle sont celles qui initialisent ou clôturent la partition, celles qui effectuent la prise de contact de la partition avec

l'utilisateur, celles qui initialisent une session ou le changement de mode, enfin celles qui permettent la lecture et l'analyse d'une commande.

Les routines utilitaires effectuent les opérations de base de la partition conversationnelle telles que le calcul d'adressage, les entrées/sorties.

Le module de traitement des commandes est constitué de l'ensemble des modules permettant le traitement des diverses commandes pouvant être lancées par les utilisateurs.

Parmi les modules objets programme, certains peuvent être sollicités simultanément par plusieurs utilisateurs. Ces modules sont dits partagés et sont nécessairement réentrants. Cependant il ne s'agit pas d'une réentrance au niveau de l'instruction mais d'une réentrance "raisonnée" au niveau du groupe d'instructions se trouvant en exécution entre 2 points interruptibles. Il s'agit des routines de contrôle gérant une session, des routines utilitaires (sauf la routine de pagination) et des modules de traitement des commandes (en particulier l'analyseur-générateur et l'interpréteur).

D'autres modules comme la routine de pagination ne sont pas partagés en raison de leurs fonctions, ou bien ne sont pas partagés parce qu'ils travaillent pour l'ensemble des utilisateurs. C'est le cas des routines de contrôle qui initialisent ou clôturent la partition....

3.2 La zone de données partagée.

La structure réentrante de la plupart des modules de la partition impose que toutes les informations concernant un utilisateur de ces modules soient concentrées dans une zone de mémoire réservée à cet utilisateur.

Ainsi, lorsqu'un utilisateur a initialisé avec succès une session, une zone de données lui est affectée avec les caractéristiques suivantes :

- elle est virtuelle c'est-à-dire que pour les modules réentrants, elle sera considérée comme résidente en mémoire centrale mais qu'en fait, elle sera réservée sur disque.
- sa taille est fixe (128 K octets),
- elle est divisée en pages égales de 1/2 K octets chacune.

- elle a un format fixe quel que soit l'utilisateur, c'est-à-dire qu'elle est partagée en un certain nombre de parties, chacune d'elles regroupant les pages dans lesquelles est rangée l'information du même type : partie table des enchainements, partie pile d'exécution....

La zone de mémoire centrale disponible pour valider des informations propres à chaque utilisateur, dite "zone de données partagée" est très limitée (20 K octets environ) en regard des zones de données affectées sur disque aux différents utilisateurs.

Le but de la simulation est aussi de tester les performances de la partition en fonction de la taille de la zone des données partagée. Nous représenterons donc aussi fidèlement que possible cette organisation fondée sur la pagination.

Ainsi, à un instant donné, chacun des utilisateurs présent dans le système n'a que certaines pages de sa zone de données propre (virtuelle) dans la zone de données partagée. On dira que ces quelques pages sont valides car l'utilisateur peut disposer directement des informations qu'elles contiennent.

3.3 Zones communes de oontrôle et de traitement des commandes. Zones de recouvrement.

La zone commune de contrôle est accessible par toutes les routines du moniteur local et contient les informations nécessaires au déroulement de ces routines : informations qui servent à l'administration de la partition et à la gestion de chacune des sessions.

La zone de traitement des commandes est utilisée par les modules de traitement des commandes où ils laissent les résultats de calculs entre deux pertes de contrôle au lieu de les ranger dans la zone de données partagée de l'utilisateur pour lequel un module de traitement travaille. L'existence de cette zone offre un gain de temps appréciable au niveau de l'adressage qui est beaucoup plus rapide que celui de la zone de données partagées (du fait de la pagination).

Les zones de recouvrement servent à recevoir les différents segments de la partition.

Ces trois zones ne nous intéressent pas au niveau de la simulation,

mais il est intéressant d'en préciser l'utilisation pour éclairer la présentation de l'organisation de la partition.

4 -LOGIQUE DE LA PARTITION CONVERSATIONNELLE

4.1. Fonctionnement.

Un point que nous devons relever pour la simulation est la possibilité qu'a la partition de traiter en parallèle plusieurs processus en profitant des simultaneités entre les entrées-sorties que ce soit des entrées-sorties classiques ou des transmissions. Ce traitement parallèle étant de plus subordonné à un système de priorité.

Nous venons de parler d'une notion essentielle dans la description logique de la partition : le processus. A chaque ligne d'entrée correspond un bloc de contrôle (LCB = line control block) qui contient toutes les informations nécessaires au fonctionnement de la ligne; en particulier les paramètres des macro-instructions d'entrée-sortie sur téléimprimeur et les informations nécessaires à la reprise ou à la suppression de l'activité du processus relatif à la ligne considérée.

La création d'un processus consiste à associer, avec une certaine priorité, un LCB à une procédure (ou module). Un LCB représente en fait un utilisateur puisqu'il est admis qu'il n'y a qu'un seul téléimprimeur par ligne.

Lorsqu'il est activé, le processus effectue tous le traitements inclus dans la procédure pour le compte de l'utilisateur associé au LCB et avec la priorité qui a été donnée au processus lors de sa création. Un processus est désactivé sur le lancement d'une entrée-sortie quelconque. Le processus attend la fin de l'entrée-sortie tandis que la partition active un autre processus. Le premier processus sera de nouveau activé lorsqu'il sera de nouveau activable et que tous les processus activables à ce moment là auront une priorité inférieure à la sienne. Enfin, un processus est annulé quand tous les traitements à effectuer pour un utilisateur sont terminés.

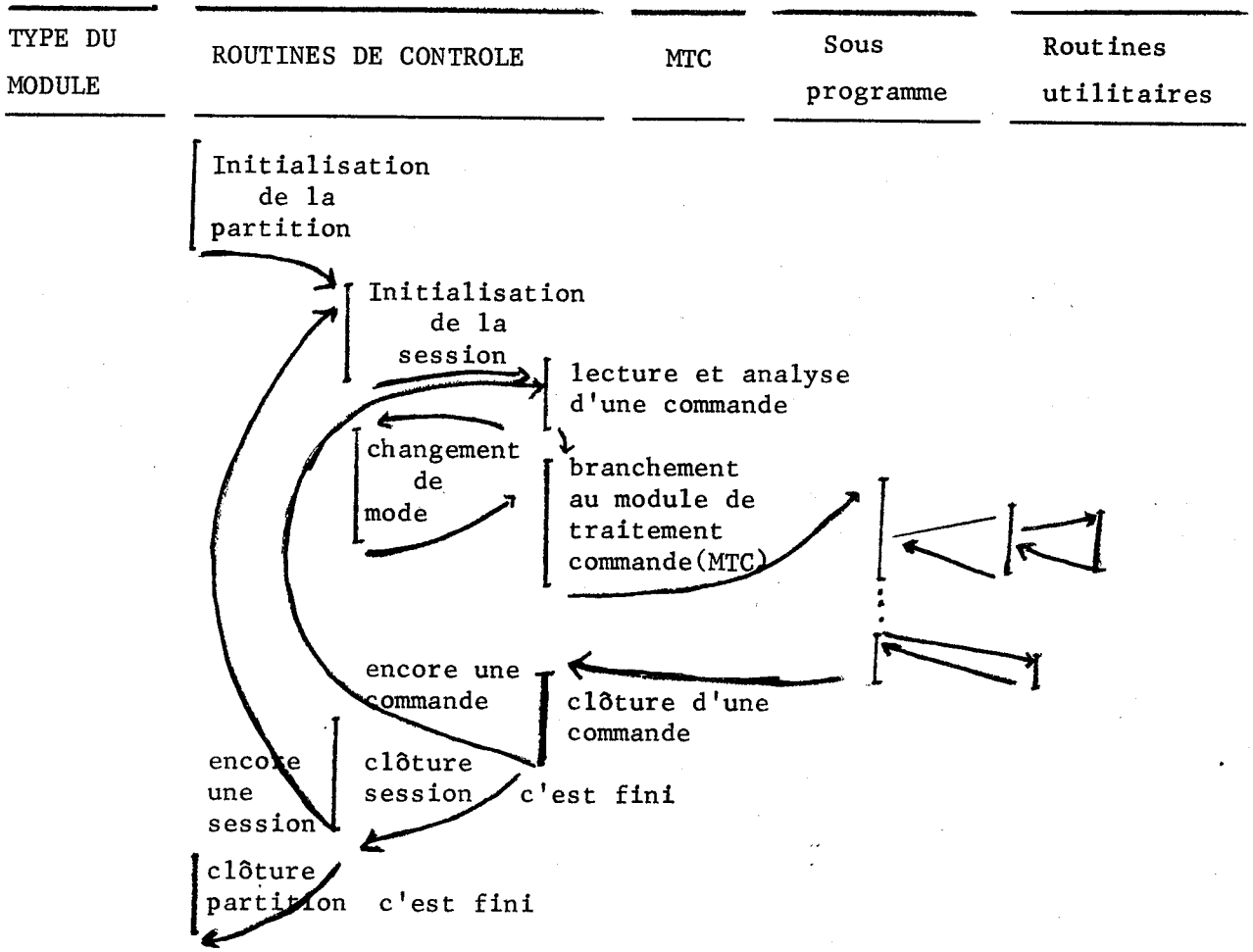
Tout traitement fait pour le compte d'un utilisateur est réalisé dans le cadre de la dernière commande lancée. Cette commande confère à la

session de cet utilisateur un certain mode et donc une certaine priorité (Administration = 1, Compilation = 2, exécution = 3, fichier = 4). Ainsi le processus qui est associé à l'utilisateur et qui doit traiter cette commande doit le faire avec la priorité associée à cette commande.

Un processus donné change de priorité chaque fois qu'un utilisateur lance une commande qui fait changer sa session de mode. Inversement chaque fois qu'un utilisateur lance une commande conservant le mode de la session, il y aura branchement au module de traitement de cette commande. Si ce module est résident, le branchement est immédiat. Si ce module n'est pas chargé en mémoire centrale, le processus correspondant est désactivé jusqu'à ce que le segment correspondant à ce module de traitement soit chargé.

Dans le programme de simulation, nous ferons correspondre à un processus, une suite d'intéractions. Une intéraction est la suite des évènements qui se produisent entre les deux instants successifs où la partition interrompt sur une entrée-sortie console ou achève l'exécution du programme utilisateur.

4.2 Schéma du fonctionnement.



5 -LES ROUTINES UTILITAIRES

Ces routines effectuent les traitements de base de la partition conversationnelle tels que l'adressage, les entrées-sorties....

Nous nous intéresserons seulement aux routines d'adressage relatives à la pagination puisque soit nous traduirons leurs effets soit nous les utiliserons fidèlement dans le programme de simulation.

5.1 Le principe de l'adressage:adresse virtuelle.

Lorsqu'ils sont appelés par un certain utilisateur, les modules de traitement de commandes utilisent la zone de données propre à cet utilisateur qu'ils adressent au moyen d'adresses virtuelles. Une adresse virtuelle est composée d'un numéro de page PAGE et d'un déplacement dans cette page ADPAGE. Une routine d'adressage teste si la page PAGE est en mémoire centrale au moyen de la table des pages de l'utilisateur. Si c'est le cas, la table des pages fournit le n° de la page réelle en zone de données partagée où cette page virtuelle est implantée. L'adresse réelle se calcule de la manière suivante :

$$\left[\begin{array}{l}\text{N° de la page réelle}\\ \text{où est validée la pa-}\\ \text{ge virtuelle PAGE}\end{array}\right] * 512 + \left[\begin{array}{l}\text{Adresse réelle absolue}\\ \text{de début de la zone de}\\ \text{données partagée de l'u-}\\ \text{tilisateur}\end{array}\right] + \text{ADPAGE.}$$

Si PAGE n'est pas validée, la routine d'adressage fait appel au sous-programme de pagination qui a pour fonction de la valider. La table des pages sera mise à jour et la traduction pourra se faire.

5.2 Le sous-programme de pagination.

Le rôle du sous-programme de pagination est de valider une page virtuelle dans une page libre de la zone de données partagée et ensuite de trouver, si besoin est, une nouvelle page libre.

Ce sous-programme est construit de telle façon qu'il y ait toujours au moins une page libre en zone de donnée partagée.

Il effectue successivement les divers traitements :

- Appel de la page virtuelle en mémoire centrale. Le n° de la page virtuelle sert à identifier l'enregistrement du fichier MEMOIRE VIRTUELLE dans lequel elle a été rangée et la page virtuelle est chargée en mémoire centrale à partir de l'adresse prise dans la table des pages libres.
- il teste ensuite si la table des pages libres est vide.
  - . s'il existe encore des pages disponibles le contrôle est rendu au sous-programme appelant.
  - . sinon il y a appel de l'algorithme de pagination dont la fonction est de déterminer des n° de pages libres pour que soit remplie à nouveau la table des pages libres.

5.3. L'algorithme de pagination

Pour déterminer les pages libres, l'algorithme utilise deux critères:

- le premier est relatif à la notion d'inactivité temporaire d'une session.
  Une session est considérée comme temporairement inactive lorsqu'une entrée-sortie est en cours au téléimprimeur correspondant et que les conditions suivantes sont respectées :
  - . en entrée : l'opération a été lancée depuis environ 5 secondes mais depuis moins de 10 secondes.
  - . en sortie : l'opération a été lancée depuis moins de 2 secondes.
- le deuxième est relatif à la notion d'inutilisation temporaire d'une page virtuelle valide.

Une page virtuelle valide est considérée inutilisée lorsqu'elle appartient à une partie de la mémoire virtuelle elle-même considérée comme inutilisée en fonction de l'historique de la session.

Cet historique est composé des modes des 3 dernières commandes lancées par l'utilisateur. Il est mis à jour à l'arrivée de chaque nouvelle commande.

Pour	Administration	on range le code	0,
	Compilation	⟶	1,
	Exécution	⟶	2,
	Fichier	⟶	4.

L'algorithme de pagination effectue la somme des valeurs différentes contenues dans l'historique. A chaque valeur obtenue correspond une suite de parties inutilisées. Connaissant la liste des pages d'une partie, la suite des pages inutilisées s'en déduit aisément.
Les différentes tables des pages sont parcourues successivement jusqu'à ce que:

* Soit
  - 5 pages virtuelles valides non modifiées aient été déterminées selon le premier critère,
  - 3 pages virtuelles valides non modifiées aient été déterminées d'après le $2^{ième}$ critère,
  - 1 page virtuelle valide modifiée ait été déterminée par le $2^{ième}$ critère.

* Soit

Toutes les tables aient été parcourues;

Au retour de cet algorithme, la table des pages libres est remplie par les numéros des pages libres (9 au maximum) dont 8 au maximum sont immédiatement disponibles et une qui le sera lorsque la page virtuelle qui s'y trouve aura été recopiée dans la mémoire virtuelle correspondante.

A P P E N D I X B

LE PROGRAMME DE SIMULATION : DESCRIPTION

---

1 -DESCRIPTION DU PROGRAMME DE SIMULATION

2 -LISTING DU PROGRAMME

L'objet de cet Appendix sans vouloir entrer dans les détails les plus intimes, consiste simplement à faciliter la lecture du programme de simulation et à montrer comment s'organise dans la pratique un tel programme.

1 -DESCRIPTION DU PROGRAMME DE SIMULATION

1.1. Présentation générale

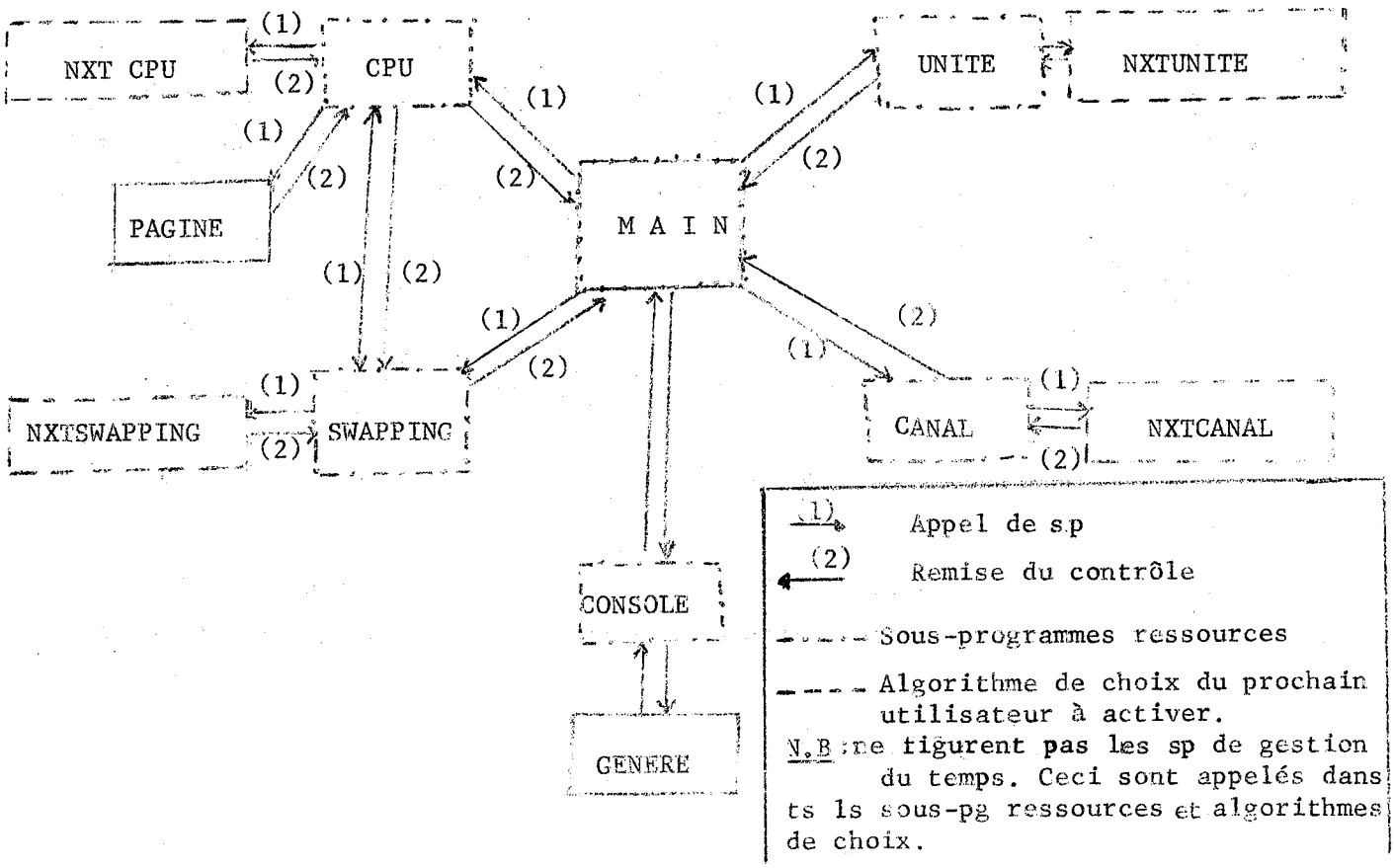
Le programme de simulation comprend :

- un programme principal, MAIN,
- un ensemble de sous-programmes par rapport à MAIN

Parmi ces sous-programmes on distingue :

- Ceux qui représentent les ressources du système réel,
- les algorithmes de choix d'utilisateurs associés à chaque ressource,
- l'algorithme de pagination,
- ceux qui servent à la gestion du temps,
- le générateur de Données, GENERE
- enfin ceux qui remplissent les fonctions utilitaires pour l'ensemble du programme de simulation telles que, suppression d'un élément d'une file d'attente, recherche d'un élément maximum dans un tableau, ou encore détermination de la longueur d'une file d'attente.

1.2 Schéma d'organisation



1.3. Une phase d'initialisation lit les différents paramètres et les codes des commandes lancées par les utilisateurs; en second lieu elle donne le contrôle

- au sous-programme CONSOLE dont la fonction est alors d'introduire les premiers utilisateurs dans le système
- puis au programme principal MAIN qui joue le rôle de superviseur pour l'ensemble de la simulation.

1.4. Le programme superviseur MAIN

Le programme superviseur MAIN réalise successivement les traitements suivants :

- Il détermine à quel niveau doit se produire le prochain évènement. Pour cela, il explore les files d'attente des utilisateurs sur l'ensemble des ressources. La première file d'attente rencontrée qui contient des utilisateurs à l'heure de simulation repère la ressource qui doit prendre le contrôle. Si plusieurs files d'attente contiennent des utilisateurs à l'heure de simulation, le programme superviseur les prend en compte séquentiellement, un système de gestion du temps permettant le réajustement des horloges(cf chapitre IV)
- il active ensuite le sous-programme qui simule la ressource activable.
- Une fois le traitement sur la ressource achevé, le contrôle est rendu au programme MAIN qui répète ce processus jusqu'à ce que la ressource CONSOLE signale que toutes les sessions sont clôturées auquel cas MAIN imprime les résultats statistiques collectés au cours de la simulation.

1.5. CONSOLE

CONSOLE boucle sur le nombre d'utilisateurs admissibles dans la partition.

Les premiers traitements de CONSOLE consistent à effectuer une série de tests :

- étant donné un utilisateur, CONSOLE regarde si cet utilisateur a déjà clôturé sa session ou s'il n'a pas initialisé de session. Si c'est le cas, CONSOLE

passe au téléimprimeur suivant.

- Inversement si l'utilisateur reste silencieux pendant plus de 5 mn ou s'il vient de clôturer sa session, CONSOLE place cet utilisateur dans la chaine des consoles déconnectées et passe à l'utilisateur suivant.
- Dans le cas où l'utilisateur est en cours de session, CONSOLE lit dans la liste des commandes propres à cet utilisateur le code qui caractérise la prochaine commande ainsi que le temps qui lui sera nécessaire pour la lancer. Le téléimprimeur de l'utilisateur est alors "bloqué" jusqu'à ce que le système tombe en attente d'entrée sur ce téléimprimeur(fin d'intéraction). CONSOLE appelle le générateur de données qui engendrera tous les renseignements nécessaires à la représentation de l'utilisateur dans le système de simulation. Ce dernier est placé dans la file d'attente correspondant à la ressource et un sous-programme de gestion du temps met à jour l'horloge propre à cet utilisateur.

Le sous-programme CONSOLE réitère cette série de traitements jusqu'à ce que toutes les consoles aient été passées en revue. Puis il rend le contrôle au programme appelant (MAIN).

1.6. La ressource CPU.

La ressource est appelée

- soit par la ressource SWAPPING qui supervise les échanges de pages entre mémoire centrale (zone de données partagée) et mémoires secondaires(fichier mémoire virtuelle),
- soit par MAIN,

Dans le premiers cas, le travail en unité centrale que demande l'utilisateur est l'exécution de l'algorithme de pagination, dit PAGINE .
La durée du déroulement de ce programme est comptabilisée au nom de l'utilisateur.

Dans le second cas, l'unité centrale travaille pour le compte d'un utilisateur qui doit être choisi dans la file d'attente de CPU, suivant un critère de priorité. Si plusieurs utilisateurs sont simultanément activables dans cette file d'attente, l'algorithme de choix permet de sélectionner celui qui

a la plus forte priorité(nous rappelons que cette priorité est affectée à l'utilisateur en fonction du type de la commande lancée).

Une fois l'utilisateur choisi, CPU détermine l'origine et l'heure de la prochaine interruption à l'aide des renseignements que chaque utilisateur porte sur lui :

. L'égalité à zéro des nombres de références aux fichiers, pages et segments marque la fin d'une intéraction. Mesure à prendre : laisser l'unité centrale travailler pour l'utilisateur pendant le temps prévu, comptabiliser ce temps et débloquer le téléimprimeur de cet utilisateur. Toutefois s'il se trouve en mode EXECUTION, l'utilisateur sera interrompu après l'exécution de 15 instructions FORTRAN et sera à nouveau placé dans la file d'attente de CPU.

. Si une entrée-sortie classique (chargement d'un segment, référence à un fichier, demande de page) doit se produire, CPU note le temps pendant lequel l'unité centrale a été occupée avant l'interruption et range l'utilisateur dans la file d'attente correspondant à la ressource demandée : CANAL pour chargement de segment et référence à un fichier, SWAPPING pour une demande de page.

1.7. La ressource CHANNEL.

L'ensemble des canaux est représenté par le sous-programme CHANNEL dont l'argument définit sans ambiguité le canal sollicité par l'utilisateur.

CHANNEL réalise successivement les traitements suivants :

- Comme dans tous les sous-programmes de ressources partageables entre plusieurs utilisateurs, une première phase consiste à choisir un utilisateur dans la file d'attente et ce choix se fait toujours suivant le même critère de priorité. L'algorithme de choix NXTCANAL détermine en outre, à partir des renseignements enregistrés dans la file d'attente, la durée du transfert de l'information ainsi que le numéro de l'unité de disque désirée par l'utilisateur.
- Il teste la nature du canal appelé. La partition conversationnelle n'a pas de canal multiplexeur : seul le type sélecteur est ici décrit.
- Chaque canal dispose d'un indicateur précisant qu'il est prêt pour la transmission de données.

. Si c'est le cas, CHANNEL comptabilise le temps de transmission des données en Mémoire centrale et disque. Il met à jour les temps d'attente dans les files des unités de disques auxquelles il a accès ainsi que les temps d'attente sur sa propre file. Il indique ensuite que l'unité de disque qui transmettait est libre pour un prochain utilisateur. Suivant la durée du transfert, CHANNEL détermine le type d'entrée-sortie qu'il vient de réaliser et met à jour, suivant le cas, les compteurs de références aux pages, aux segments ou aux enregistrements de fichier. S'il s'agit d'un transfert de page, indiquer à SWAPPING la fin de la transmission. Dernière opération : l'utilisateur est placé dans la file d'attente de CPU (avec le réajustement de l'horloge correspondante).

. Inversement, le canal n'a pas encore transmis à l'unité de disque concernée l'ordre de recherche d'un enregistrement. Le sous-programme note le temps d'utilisation du canal (1 milliseconde) nécessaire au lancement de l'entrée sortie sur disque et met à jour les temps d'attente des utilisateurs se trouvant dans sa file d'attente ainsi que les temps d'attente des éléments des files d'attente des unités de disques qui étaient prêtes à transmettre des données par l'intermédiaire de ce canal. Enfin l'utilisateur est placé dans la file d'attente de l'unité désirée. L'indicateur de fin de lancement d'entrée-sortie est positionné.

1.8 . Les ressources UNITES

Les ressources UNITES concernent uniquement les disques : la Partition conversationnelle ne dispose pas d'autres unités à l'exception du ruban perforé qui n'est pas d'un grand intérêt dans la simulation.

L'ensemble des unités de disques est représenté, comme dans le cas des canaux, par un même sous-programme dont l'argument détermine l'unité de disque activée. (L'existence de cet argument facilite l'introduction de nouvelles unités qui ne sont pas forcément des disques. Même remarque sur les canaux.)

- La première opération consiste à déterminer par le sous-programme l'utilisateur à satisfaire (même critère de priorité).
- Puis l'unité se prépare en vue de l'Entrée-sortie proprement dite : positionnement du bras sur un enregistrement dans le cas des disques qui nécessite

en moyenne 75 millisecondes (performance que nous avons pris soin de ranger au tout début de la simulation dans le tableau reflétant l'équipement Hardware de la partition conversationnelle).

- Noter le temps d'utilisation et mettre à jour le temps d'attente dans les files.
- Bloquer la ressource jusqu'à ce que le canal auquel elle est reliée la rende à nouveau libre après la transmission des données.
- Enfin placer l'utilisateur dans la file d'attente en prenant soin d'enregistrer en même temps que le nom de l'utilisateur, la durée du transfert et le numéro de l'unité qui demande le canal.

1.9. Le sous-programme SWAPPING.

SWAPPING gère les échanges de pages entre Mémoire centrale et mémoires à disques.

- La première opération consiste là encore à faire appel au sous-programme de détermination de l'utilisateur à satisfaire.
- SWAPPING teste si la page que l'utilisateur a demandée a déjà été chargée.

* Si ce n'est pas le cas, SWAPPING précise l'unité spécialisée pour la pagination (ce peut être une pile de disques différente de celles où sont enregistrés les fichiers), range l'utilisateur et tous les renseignements utiles dans la file d'attente du canal auquel cette unité est attachée, enfin rend le contrôle.

* Si la page demandée est déja validée deux possibilités se présentent.

. ou bien il n'y a plus de pages libres; auquel cas SWAPPING doit faire appel à la ressource CPU pour le déroulement de l'algorithme de pagination, PAGINE. Quand CPU rend le contrôle, SWAPPING initialise une 2^{ième} entrée-sortie de page sur le canal puis rend le contrôle (On revient toujours de PAGINE avec une page non réentrante à recopier sur disque).

. ou bien il y a encore des pages libres en zone de données partagées; auquel cas SWAPPING effectue les traitements suivants :

- mise à jour du compteur de pages libres
- mise à jour de la table des pages de l'utilisateur, qui compte maintenant une page validée supplémentaire.
- mise à jour du nombre des références de pages
- enfin rangement de l utilisateur dans la file d attente de CPU et rendre le contrôle.

1.10. Les autres sous-programmes.

Les sous programmes utilitaires du programme de simulation ne méritent pas une attention particulière.

Les autres ont été décrits en détail dans d'autres chapitres :

- Chapitre III pour GENERE
- Appendix A pour l'Algorithme de pagination
- Chapitre IV pour les Sous-Programmes de Gestion du Temps.

2 -LE PROGRAMME DE SIMULATION [Ref 29 - Ref 30]

Nous allons définir les variables essentielles employées dans le programme de simulation :

ALGOLGO = indicateur de demande d'exécution de l algorithme de pagine.

BLOCOPEN(I) = temps total d'ouverture et de fermeture de fichiers par l'utilisateur I.

CANALWAIT = temps d'attente de l'utilisateur I sur les canaux.

CPL = compteur de pages libres en mémoire centrale.

CPUUSE = temps total d'utilisation de 1 unité centrale par l'ensemble des utilisateurs.

CPUWAIT(I) = temps d'attente de l'utilisateur I sur l'unité centrale.

DEPARTCANAL(J,I) = heure de départ de l'utilisateur I du canal J.

DEPARTUNITE(J,I) = " " " " de l'unité J.

DEPARTCONSOL(I), DEPARTCPU(I), DEPARTSWAP(I) = heure de départ de l'utilisateur de I de la console, de l'unité centrale, de la ressource de la pagination respectivement.

EJECT(I) = indicateur de réécriture d'une page sur disque comptabilisée au nom de l'utilisateur I.

ENDTERM = indicateur de fin de simulation

EQUIPMENT(I,1) = caractéristique Hardware de l'unité I

EQUIPMENT(I,2) = n° du Canal auquel l'unité est connectée.

FLAG(I) indique si le canal est utilisé pour une transmission de données ou s'il est utilisé pour signaler une demande de transfert de données.

HISTORIC(I,X) cf appendice A = 5 - 3)

IDLECANAL, IDLE,CPU servent à comptabliser le temps d'inactivité des canaux de de l'unité centrale.

INTCYC(I) = nombre d'instructions à exécuter pour le compte de l'utilisateur I

IQCANAL, IQCPU,IQSWAP, IQUNITE = pointeurs de sommets des files d'attente sur les canaux, l'unité centrale, la ressource pagination, les unités périphériques.

LFICHIERCOURANT(I) = Longueur du fichier courant I.

LISTEPARTIE = liste des parties de la zone de données partagées (mémoire virtuelle).

NBPAGEMC = nombre de pages en mémoire centrale .

NSESION(I) = longueur courante de la session de l'utilisateur I.

NXTPARTIE(I) = numéro de la prochaine partie référencée par l'utilisateur I.

ON(I) = indicateur de connexion de la console I avec l'ordinateur.

PRTY(I) = priorité courante de l'utilisateur I.

PUI = pointeur vers l'utilisateur que PAGINE va en premier lieu considérer pour le choix des pages à invalider.

QCPU, QSWAP, QUNITE, QCANAL = files d'attente sur les ressources Unité Centrale, SWAPPING, Unités et canaux respectivement.

RDY(I) = indicateur de blocage ou déblocage de la console I.

REENTRANT(I, J) = indique qu'il existe au moins une page dans la partie J non modifiée par l'utilisateur I.

REFSEG(I),
REFILE(I), REFOPEN(I), REFPAGE(I), REFPARTIE(I,J) = nombre de références à des segments, à un fichier, nombre d'ouvertures de fichier, nombre de références à ces pages distinctes, nombre de références dans la partie J par l'utilisateur I.

STWTCANAL, STWCPU, STWTUNITE, STWTSWP = heure d'arrivée dans la file d'attente canal, cpu, unité, swapping respectivement.

SWPDEVILE = n° de l'unité avec laquelle se font les échanges de pages.

SWPWAIT(I) = temps d'attente de l'utilisateur I sur la ressource de pagination.

TABLEPAGE = table des pages.

TCANAL(I), TCPU(I), TUNITE(I), TSWAP(I) = temps d'utilisation du canal cpu, unité, swapping par l'utilisateur I.

TIM(J,I) = heure relative (par rapport à l'horloge de l'utilisateur) du lancement de la $J^{ème}$ commande par l'utilisateur I.

TIMEFIL(I), TIMESEG(I) TIMEPAG(I) = heure relative (par rapport à l'horloge de l'utilisateur I) de la prochaine interruption pour une référence à un fichier, à un segment ou à une page.

TREPLY(I) = temps de réponse pour l'utilisateur I.

TTREPLY(I) = horloge de l'utilisateur I.

TYPE(J, I) = code de la $J^{ème}$ commande lancée par l'utilisateur I.

UNITEWAIT(I) = temps d'attente de l'utilisateur I sur les unités périphériques

USER = utilisateur courant.

WAITRANSMISSION(J) = indicateur d'attente du canal par l'unité J pour le transfert des données.

YMRDY1(J) = indicateur de fin de transfert des données.

```
 1                (SUBSCRIPTRANGE):
                  STMUL : PROC OPTIONS(MAIN);
 2                DCL TOTO BIN FIXED(31) INIT(1);
 3                 DCL OUI(5) BIT(1) INIT((5)'1'B);
 4                 DCL SYSPRINT FILE STREAM PRINT ENVIRONMENT(F(120));
 5                 DCL SYSIN FILE STREAM ENVIRONMENT(F(80));
 6                 DCL RESULT FILE STREAM OUTPUT ENVIRONMENT(CONSECUTIVE F(80));
 7                 DCL (I,J) BIN FIXED(31);
 8                 DCL NUSER BIN FIXED(31);
 9               DCL(NCANAL,NUNITE) BIN FIXED(31) ;
10                 DCL  ISESION(8) BIN FIXED(31);
11                 DCL LSESION BIN FIXED(31);
12                 OPEN FILE(SYSIN),FILE(SYSPRINT);
13                 ON ENDFILE GOTO FINFIN;
15                 LOOP: GET LIST(NCANAL);
16                 GET LIST(NUNITE);
17                 GET LIST(NUSER);
18                 LSESION=0;
19                 DO  I=1 TO NUSER;
20                       GET LIST (ISESION(I));
21                       LSESION=MAX(LSESION,ISESION(I));
22                 END;
                   /* ISESION(I) CONTIENT LA LONGUEUR DE LA SESSION DE L'UTILISATEUR*/

23                 BEGIN;
24                 DCL CONSOLE ENTRY;
25                 DCL CPU ENTRY;
26                 DCL SWAPING ENTRY;
27                 DCL UNITE ENTRY(BIN FIXED(31));
28                 DCL CHANNEL ENTRY(BIN FIXED(31));
29                 DCL DELETE ENTRY((NUSER,4) BIN FIXED(31),BIN FIXED(31),BIN FIXED(31),
                     BIN FIXED(31));
30               DCL GENERE ENTRY(CHAR(7) VAR,BIN FIXED(31));
31                 DCL HEURE ENTRY(CHAR(9)) RETURNS(BIN FIXED(31));
32                 DCL MAX1 ENTRY( (NUSER)BIN FIXED(31)) RETURNS(BIN FIXED(31));
33                 DCL MIN1 ENTRY((NUSER)BIN FIXED(31)) RETURNS(BIN FIXED(31));
34                 DCL NXTCANAL ENTRY(BIN FIXED(31));
35                 DCL NXTUNITE ENTRY(BIN FIXED(31));
36                     DCL NEXTCPU ENTRY ;
37                     DCL NXTSWAP ENTRY ;
38                 DCL OPEN ENTRY(BIN FIXED(31));
39                 DCL PAGINE ENTRY;
40                 DCL QLENGTH1 ENTRY((NUSER,4)BIN FIXED(31),(4)BIN FIXED(31)) RETURNS
                   (BIN FIXED(31));
41                 DCL QLENGTH2 ENTRY((*,*,*,*)    BIN FIXED(31),(*,*)BIN FIXED(31),
                   BIN FIXED(31)) RETURNS(BIN FIXED(31));
42               DCL (ALGOGO ,ALGOEND)  BIT(1) INIT('0'B);
```

```
43              DCL ENDTERM BIT(1) INIT('0'B);
44              DCL ON(NUSER) BIT(1);
45              DCL PDY(NUSER) BIT(1) INIT((NUSER)'1'B);
46              DCL XMRDY1(NUSER) BIT(1) INIT((NUSER)'0'B);
47             DCL WAITRANSMISSION(NUNITE) BIT(1) INIT((NUNITE)'0'B);
48              DCL ARGUMEN BIN FIXED(31);
49              DCL BLOCOPEN(NUSER) BIN FIXED(31) INIT((NUSER)0);
50            DCL                    (CPL,CPUUSE,CPUWAIT(NUSER))   BIN FIXED(31);
51              DCL CPL1 BIN FIXED(31);
52              DCL CPUSR BIN FIXED(31);
53              DCL CHANNELUSR BIN FIXED(31);
54            DCL(CANAL(NCANAL),CANALUSR(NCANAL)) BIN FIXED(31) ;
55            DCL CANALWAIT(NUSER)   BIN FIXED(31) ;
56            DCL(DEVICE(NCANAL),DEVICE1) BIN FIXED(31) ;
57            DCL(DSKUSE1,DSKUSE2,DSKUSE3) BIN FIXED(31)   INIT(0) ;
58             DCL(DEPARTSWAP(NUSER),DEPARTCPU(NUSER),DEPARTCANAL(NCANAL,NUSER),
               DEPARTCONSOL(NUSER),DEPARTUNITE(NUNITE,NUSER)) BIN FIXED(31);
59            DCL EQUIPMENT(NUNITE,2) BIN FIXED(31) ;
60              DCL EJECT(NUSER) BIN FIXED(31) INIT((NUSER)0);
61            DCL FLAG(NCANAL) BIN FIXED(31) ;
62              DCL H BIN FIXED(31);
63             DCL HISTORIC(NUSER,3) BIN FIXED(31) INIT((NUSER*3)0);
64              DCL (HHORA,HORA) BIN FIXED(31);
65            DCL IQCANAL(NCANAL,4) BIN FIXED(31) ;
66              DCL IDLECANAL(NCANAL) BIN FIXED(31) INIT((NCANAL)0);
67              DCL IDLECPU BIN FIXED(31) INIT(0);
68            DCL IQUNITE(NUNITE,4) BIN FIXED(31) ;
               /*CONVENTION POUR IMAX:1=CPU,2=SWAP,3=UNITE,4=CANAL*/
               DCL IMAX(2) BIN FIXED(31) INIT((2)0);
70              DCL
               (IQCPU(4),IQSWAP(4)) BIN FIXED(31);
71             DCL INTCYC(NUSER) BIN FIXED(31);
72              DCL IOUT BIN FIXED(31) INIT(0);
73              DCL JMAX(NCANAL) BIN FIXED(31) INIT((NCANAL)0);
74              DCL KMAX(NUNITE) BIN FIXED(31) INIT((NUNITE)0);
75              DCL LIGNE(NUSER) BIN FIXED(31) INIT((NUSER)0);
76              DCL LFICHIERCOURANT(NUSER) BIN FIXED(31) INIT((NUSER)0);
77             DCL LISTEPARTIE(25) INIT(2,3,4,5,6,7,8,9,6,8,9,2,4,6,7,6,2,3,5,
               7,8,9,8,9,2,7) BIN FIXED(31);
78              DCL MINUS BIN FIXED(31);
79              DCL NSESION(NUSER) BIN FIXED(31);
80             DCL NBPAGEMC BIN FIXED(31);
81              DCL NXTPARTIE(NUSER) BIN FIXED(31) ;
82            DCL PRTY(NUSER) BIN FIXED(31);
83            DCL(QCPU(NUSER,4),                    QSWAP(NUSER,4))
                   BINARY FIXED(31);
```

```
85
86          DCL QCANAL(NCANAL,NUSER,4,4) BIN FIXED(31) ;
87          DCL QUNITE(NUNITE,NUSER,4,2) BIN FIXED(31) ;
88             DCL REPERE(0:6,2) BIN FIXED(31);
89            DCL REFPARTIE(NUSER,9) BIN FIXED(31);
90            DCL REFOPEN(NUSER) BIN FIXED(31);
91            DCL REENTRANT(NUSER,9) BIT(1) INIT((NUSER*9)'0'B);
          DCL(REFILE(NUSER),REFSEG(NUSER),REFPAGE(NUSER),REFSEQ(NUSER))
               BIN FIXED(31);
92            DCL SWAPUSR BIN FIXED(31) INIT(0);
93            DCL                          (SWAPUSE)
              INIT(0) BIN FIXED(31)  ,SWPWAIT
              (NUSER)    BIN FIXED(31);
94        DCL ((STWTCPU,STWTCANAL,STWTUNITE) (NUSER)) BIN FIXED(31);
95         DCL STWTSWP(NUSER) BIN FIXED(31);
96        DCL SWPDEVICE BIN FIXED(31) ;
97                DCL 1 SESSION(LSESION,NUSER),
                        2 TYPE CHAR(7) VAR,
                        2 TIM   BIN FIXED (31);
98         DCL TABLEPAGE(NUSER,9 ,2) BIN FIXED(31);
99        DCL TYPFES(NUNITE) BIN FIXED(31) ;
100       DCL TRANSFERT(NCANAL) BIN FIXED(31) ;
101         DCL (TTTREPLY(NUSER),TTREPLY(NUSER)) BIN FIXED(31);
102        DCL TTIME CHAR(9),
              (TREPLY(NUSER),TIMEPAG(NUSER),TIMEFIL(NUSER),TIMESEG(
              NUSER) ,        TCPU(NUSER),TSWAP(NUSER),TCANAL1(NUSER))
              BIN FIXED(31);
103       DCL TUNITE(NUSER) BIN FIXED(31) ;
104       DCL TCANAL(NUSER) BIN FIXED(31) ;
105       DCL UNITEUSR(NUNITE) BIN FIXED(31) ;
106       DCL UNITEWAIT(NUSER)   BIN FIXED(31) ;
107       DCL USER BIN FIXED(31)   INIT(0) ;
108         DO I=1 TO NUSER;
109               DO J=1 TO ISESION(I);
110                     GET LIST(SESSION.TYPE(J,I));
111                     GET LIST (SESSION.TIM  (J,I));
112               END;
113         END;
114         DO I=1 TO NUNITE;
115               DO J=1 TO 2;
116                     GET LIST(EQUIPMENT(I,J));
117               END;
118         END;
119         DO I=1 TO NCANAL;
120               GET LIST(CANAL(I));
121         END;
122         GET LIST(NBPAGEMC  );
123          DO I=0 TO 6;
```

```
124                 DO J=1 TO 2;
125                   GET LIST(PEPERE(I,J));
126                 END;
127               END;
128               CPL=NBPAGEMC;
129               DEPARTSWAP,DEPARTCPU=0;
130               DEPARTCANAL=0;
131               DEPARTUNITE=0;
132               TCPU,TSWAP=0;
133               CPUUSE=0 ;
134               IQCPU,IQSWAP=1;
135               IQCANAL=1;
136               IQUNITE=1;
137               NSESION=1;
138                   ON='1'B;
139               HORA,HHORA=0;
140               TABLEPAGE=0;
141               TREPLY=0;
142               TTREPLY=0;
143               TTTREPLY=0;
144               CALL CONSOLE;
145               CALL MAIN;
146               GO TO LOOP;
                 /*****************************************************************/
                 /*                                                               */
                 /*****************************************************************/
147             MAIN: PROC ;
148               DCL LAB(5) LABEL INIT(LAB1,LAB2,LAB3,LAB4,LAB5);
149               DCL (I,J,K,T,JO,IO) BIN FIXED(31);
                     /* MAIN SUPERVISE L'EMSEMBLE DU SYSTEME DE SIMULATION.
                     - IL RECOIT LE CONTROLE DES CONSOLES,AUTORISE ET INTERDIT
                     LEUR UTILISATION ;
                     - IL DETERMINE LE PROCHAIN UTILISATEUR A ACTIVER
                     - IL CONTROLE L'APPARITION DES INTERRUPTIONS(TOUS LES TYPES
                   D'E/S DISQUE ET FIN D'EXECUTION DES PROGRAMMES DES UTILISATEURS)
                     - SUIVANT LE TYPE D'INTERRUPTION,DONNE LE CONTROLE AU MODULE
                     DEMANDE--- */
                     /* LES INITIALISATIONS ET LES DECLARATIONS ONT ETE FAITES DANS
                     LE PROGRAMME PRINCIPAL */
                     /* ATTENDRE UN ORDRE UTILISATEUR,OU BIEN LE SIGNAL DE FIN
                     D'OPERATION DE SWAPPING,OU BIEN LE SIGNAL DE FIN D'ENTREE-SORTIE,
                     OU BIEN L'EVENEMENT DE FIN DE SIMULATION */
150                  IA,JA=0;
151               FTIO : IA=IA+1;

                     /* DETERMINER LE PROCHAIN EVENEMENT */
```

```
154                 I=MINI(TTREPLY);
155                 DO J=1 TO 4;
156                       DO I=1 TO IQCPU(J)-1;
157                             IF TTREPLY(QCPU(I,J))=T THEN GOTO ETI1;
159                       END;
160                       DO I=1 TO IQSWAP(J)-1;
161                             IF TTREPLY(QSWAP(I,J))=T THEN DO;
163                                   CALL SWAPING;
164                                   GOTO ETIO;
165                             END;
166                       END;
167                       DO I=1 TO NCANAL;
168                             DO K=1 TO IQCANAL(I,J)-1;
169                                   IF TTREPLY(QCANAL(I,K,J,1))=T THEN DO;
171                                         CALL CHANNEL(I);
172                                         GOTO ETIO;
173                                   END;
174                             END;
175                       END;
176                       DO I=1 TO NUNITE;
177                             DO K=1 TO IQUNITE(I,J)-1;
178                                   IF TTREPLY (QUNITE(I,K,J,1))=T THEN
179                                   IF WAITRANSMISSION(I) THEN DO;
181                                         CALL CHANNEL(EQUIPMENT(I,2));
182                                         GOTO ETIO;
183                                   END;
184                                   ELSE DO;
185                                         CALL UNITE(I);
186                                         GOTO ETIO;
187                                   END;
188                             END;
189                       END;
190                 END;
191                 IF SWAPUSR ¬=0 THEN IF TTREPLY(SWAPUSR)=T THEN DO;
194                       CALL SWAPING;
195                       GOTO ETIO;
196                 END;
197           ETI1 : JA=JA+1;
                  /* DETERMINER LE PROCHAIN UTILISATEUR A ACTIVER */
198               CALL NEXTCPU;

199               T=MAX1(DEPARTCPU);
200               IF T<STWTCPU(USER) THEN IDLECPU=IDLECPU+STWTCPU(USER)-T;
202               J=PRTY(USER) ;
                  /* ATTENDRE LA PROCHAINE INTERRUPTION(CPU,PAGINATION,
                  LOAD-SEGMENT,FICHIER) */
203            IF REFILE(USER)=0 &REFSEG(USER)=0&REFPAGE(USER)=0 THEN DO;
```

```
                /* L'EGALITE A 0 DES NOMBRES DE REFERENCES FICHIERS,PAGES ET
                SEGMENTS DISTINCTS MARQUE LA FIN D'UNE INTERACTION ; MESURES A
                PRENDRE: LAISSER LE CPU TRAVAILLER POUR L'UTILISATEUR PENDANT LE
                TEMPS PREVU,PUIS DEBLOQUER LA CONSOLE DE L'UTILISATEUR */
205                             I=PRTY(USER);
206           IF I=3 THEN IF INTCYC(USER)>1450 THEN DO;
              /*TOUT UTILISATEUR EST DESACTIVE APRES L'EXECUTION DE 15 INSTRUCTIONS
              FORTRAN CAD APRES L'EXECUTION DE 1450 INSTRUCTUONS MACHINE*/
209               ARGUMEN=(1450*5)/1000;
210               CALL CPU;
211               QCPU(IQCPU(I),I)=USER;
212               IQCPU(I)=IQCPU(I)+1;
213           IMAX(I)=MAX(IMAX(I),QLENGTH(QCPU,IQCPU));
214               STWTCPU(USER)=TTREPLY(USER);
215               T=MAX1(DEPARTCPU);
216               IF STWTCPU(USER)<T THEN DO;
218                     CPUWAIT(USER)=CPUWAIT(USER)+T-STWTCPU(USER);
219                     TTREPLY(USER)=TTREPLY(USER)+T-STWTCPU(USER);
220               END;
221               GOTO ETIO;
222           END;
223                                 ARGUMEN=(INTCYC(USER)*5)/1000;

224                                 CALL CPU;
225                                 RDY(USER)='1'B;
226                                 TREPLY(USER)=CPUWAIT(USER)+
                                    SWPWAIT(USER)  +UNITEWAIT(USER)+
                                    CANALWAIT(USER)  +TCPU(USER)+
                          BLOCOPEN(USER)+
                                    TUNITE(USER)         +TCANAL (USER);
                                    /* ECRITURE DES STATISTIQUES */
227                                 GOTO STATISTIC;
228               END ;
229               IF REFPAGE(USER)¬=0 & REFSEG(USER)¬=0 &REFILE(USER)¬=0 THEN DO;
231                         MINUS=MIN(TIMEPAG(USER),TIMESEG(USER),TIMEFIL(USER))
                            ;
232                         IF TIMEPAG(USER)=MINUS THEN GOTO LABEL1;
234                         IF TIMESEG(USER)=MINUS THEN GOTO LABEL3;
236                         GOTO LABEL2;
237                 END;
238               IF REFPAGE(USER)¬=0 & REFSEG(USER)¬=0 THEN
239                         IF TIMEPAG(USER)<TIMESEG(USER) THEN GOTO LABEL1;
241                         ELSE GOTO LABEL3;
242               IF REFPAGE(USER)¬=0 & REFILE(USER)¬=0 THEN
243                         IF TIMEPAG(USER)<TIMEFIL(USER) THEN GOTO LABEL1;
                          ELSE GOTO LABEL2;
```

```
247                         IF TIMESEG(USER)<TIMEFIL(USER) THEN GOTO LABEL3;
249                         ELSE GOTO LABEL2;
250             IF REFPAGE(USER)¬=0 THEN GOTO LABEL1;
252             IF REFSEG(USEP)¬=0 THEN GOTO LABEL3;
254             GOTO LABEL2;
255         LABEL1 :

                ARGUMEN =TIMEPAG(USER);
256             CALL CPU;
                            /* UNE INSTRUCTION DEMANDE EN MOYENNE 5
                            MICROSECONDES */
                                    /* ON DEMANDE DU SWAPPING */
                                    /* METTRE L'UTILISATER DANS LA QUEUE DES
                                    DEMANDES DE SWAPPING CORRESPONDANT A SA
                                    PRIORITE */
257                                 STWTSWP(USER)=TTREPLY(USER);
258                                 T=MAX1(DEPARTSWAP);
259                                 IF STWTSWP(USER)<T THEN DO;
261                                 SWPWAIT(USER)=
                                    SWPWAIT(USER)+T-STWTSWP(USER);
262                                 TTREPLY(USER)=TTREPLY(USER)+T-STWTSWP(USER)
                                    ;
263                                 END;
264                                 QSWAP(IQSWAP(J),J)=USER ;
265                                 IMAX(2)=MAX(IMAX(2),QLENGTH1(QSWAP,IQSWAP))
                                    ;
266                                 IQSWAP(J)=IQSWAP(J)+1 ;
267                                 GOTO ETIO;
268         LABEL2 :

                ARGUMEN =TIMEFIL(USER);
                                    /* ON DEMANDE E/S FICHIER */
                                    /* METTRE L'UTILISATEUR DANS LA QUEUE
                                    DES DEMANDES E/S DISQUE */
                                    /*INDIQUER L'UNITE REQUISE*/
269                                 DEVICE1=1;
270                                 JO=PRTY(USER);
271                                 IO=EQUIPMENT(DEVICE1,2);
272                                 QCANAL(IO,IQCANAL(IO,JO),JO,3)=17;
273                                 GOTO ET12 ;
274         LABEL3 :

                /* ON DEMANDE UN LOAD-SEGMENT */
                ARGUMEN =TIMESEG(USER);
            /* INDIQUER L'UNITE REQUISE */
275         DEVICE1=1 ;
276         JO=PRTY(USER) ;
```

```
277                 IO=EQUIPMENT(DEVICE1,2) ;
                    /* INDIQUER LA DEMANDE D'UN LOAD SEGMENT */
278                 QCANAL(IO,IQCANAL(IO,JO),JO,3)=42 ;
                        /* METTRE L'UTILISATEUR DANS LA QUEUE DES DEMANDES E/S
                        DISQUE */
279               ETI2:

                        CALL CPU;
280                     STWTCANAL(USER)=TTREPLY(USER);
281                     T=0;
282                     DO I=1 TO NUSER;
283                             IF T<DEPARTCANAL(IO,I)|T=0 THEN T=DEPARTCANAL(IO,I);
285                     END;
286                     IF STWTCANAL(USER)<T THEN DO;
288                             CANALWAIT(USER)=CANALWAIT(USER)+
                        T-STWTCANAL(USER);
289                             TTREPLY(USER)=TTREPLY(USER)+T-STWTCANAL(USER);
290                     END;
291                 QCANAL(IO,IQCANAL(IO,JO),JO,1)=USER ;
292                 QCANAL(IO,IQCANAL(IO,JO),JO,2)=DEVICE1 ;
293                 QCANAL(IO,IQCANAL(IO,JO),JO,4)=0 ;
294                 IQCANAL(IO,JO)=IQCANAL(IO,JO)+1 ;
295                 JMAX(IO)=MAX(JMAX(IO),QLENGTH2(QCANAL,IQCANAL,IO));
296                 GOTO ETIO;
297               STATISTIC :
                    I=PPTY(USER);
298                 PUT EDIT ((130)'*') (SKIP,A);
299                 PUT EDIT('UTILISATEUR NO ',USER)(SKIP(2),A,F(1,0));
300                 PUT EDIT ((120)'*') (SKIP,A);



301                PUT SKIP;
302                PUT EDIT('TREPLY=',TREPLY(USER)) (A,F(8,0)) ;
```

```
303                GOTO LAB(TOTO);
304               LAB1 :  IF OUI(TOTO) THEN DO;PUT FILE(RESULT) EDIT('1D,10 PAGES,2< ('
307               (A); OUI(TOTO)='0'B; END; GOTO ICI;
310              LAB2 : IF OUI(TOTO) THEN DO;PUT FILE(RESULT) EDIT('1 D,20 P,2 K')(A);
313                OUI(TOTO)='0'B; END; GOTO ICI;
316              LAB3: IF OUI(TOTO) THEN DO; PUT FILE(RESULT) EDIT('2D,10P,2<')(A);
319               OUT(TOTO)='0'B; END; GOTO ICI;
322              LAB4 : IF OUI(TOTO) THEN DO;PUT FILE(RESULT) EDIT('2D,20P,2<')(A);
325              OUI(TOTO)='0'B; END; GOTO ICI;
328              LAB5 : IF OUI(TOTO) THEN DO; PUT FILE(RESULT) EDIT('2D,30P,2<') (A);
331               OUI(TOTO)='0'B; END; GOTO ICI;
334               ICI:
                  PUT FILE(RESULT) EDIT(USER) (F(8,0));
335                PUT FILE(RESULT) EDIT(CPUWAIT(USER),UNITEWAIT(USER),CANALWAIT(USER)
                   ,TCPU(USER),TCANAL(USER),TUNITE(USER),TCANALI(USER)) ((7)(F(8,0)));
336                PUT FILE(RESULT) EDIT(TREPLY(USER))(F(8,0));
337                CALL CONSOLE;
338                GOTO ETIC;
339              FIN1: /* IMPRESSION DE STATISTIQUES FIN DE SIMULATION */
```

```
                TOTO=TOTO+1;
340              PUT EDIT('FIN DE SIMULATION') (SKIP,A);
341              PUT EDIT ((130)'*') (SKIP,A);
342              PUT EDIT(                'SWAPUSE= ',SWAPUSE,'DSKUSE1= ',DSKUSE1,
                 'DSKUSE2= ',DSKUSE2,'DSKUSE3= ',DSKUSE3) ((5) (SKIP,A,F(8,0)));
343              PUT EDIT('CPUUSE= ',CPUUSE) (SKIP,A,F(8,0));
344              PUT EDIT('IMAX= ')(SKIP,A);
345              PUT EDIT(IMAX) (SKIP,(2)(F(8,0)));
346              PUT FILE(RESULT) EDIT(IMAX)(F(8,0));
347              PUT EDIT('LENGTH-QCANAL') (SKIP,A);
348              PUT SKIP;
349              DO I=1 TO NCANAL;
350              PUT EDIT(JMAX(I)) (F(8,0));
351              PUT FILE(RESULT) EDIT(JMAX(I))(F(8,0));
352              PUT EDIT(IDLECANAL(I)) (F(8,0));
353              PUT FILE(RESULT) EDIT(IDLECANAL(I)) (F(8,0));
354              END;
355              PUT EDIT('LENGTH-QUNITE') (SKIP,A);
356              PUT SKIP;
357              DO I=1 TO NUNITE;
358              PUT EDIT(KMAX(I)) (F(8,0));
359              PUT FILE(RESULT) EDIT(KMAX(I)) (F(8,0));
360              END;
361              PUT FILE(RESULT) EDIT(IDLECPU) (F(8,0));
362              PUT EDIT(IDLECPU) (F(8,0));
363            END MAIN ;
                /*****************************************************************/
                /*                                                               */
                /*****************************************************************/
364            CPU: PROC ;
365              DCL (I,K,T) BIN FIXED(31);
               /* ''CPU'' SIMULE L'UNITE CENTRALE DEROULANT UN PROGRAMME UTILISATEUR
               OU EXECUTANT L'ALGORITHME DE SWAPPING */
               /* INITIALISATIONS ET DECLARATIONS ONT ETE FAITES DANS LE
               PROGRAMME PRINCIPAL */
               /* ATTENDRE LE CONTROLE */

366              CPUSR=USER;

367                  IF ALGOGO THEN DO;

                                 /* DEMANDE D'EXECUTION DE L'ALGORITHME DE SWAPPING */
369                              ALGOGO='0'B;
370                              CALL PAGINE;

                                 /* EXECUTION DE L'ALGORITHME DE SWAPPING */
```

```
                                    DO I=1 TO IQCPU(K)-1;
373                                      T=TTREPLY(QCPU(I,K))-TTREPLY(SWAPUSR);
374                                      IF T<0 THEN T=H ;ELSE IF T <H THEN
377                                      T=H -T;ELSE T=0;
379                                      CPUWAIT(QCPU(I,K))  =CPUWAIT(QCPU(I,K))+
                                         T;
380                                      TTREPLY(QCPU(I,K))=TTREPLY(QCPU(I,K))+T;
381                                 END;
382                            END;
383                            CPUUSE=CPUUSE+H;
384                            TTREPLY(SWAPUSR)=TTREPLY(SWAPUSR)+H;
385                            DEPARTCPU(SWAPUSR)=TTREPLY(SWAPUSR);
386                            TCPU(SWAPUSR)=TCPU(SWAPUSR)+H;
387                            ALGOEND='1'B;
                               /* LIBERER LE CPU */
388                            GOTO FIN5;
389                       END ;
                          /* ON DEMANDE LE CPU POUR L'EXECUTION D'UN PROGRAMME
                          UTILISATEUR */

                          /* INTERRUPTION DE FIN D'EXECUTION PROGRAMME UTILISATEUR */
390                            DO K=1 TO 4;
391                                 DO I=1 TO IQCPU(K)-1;
392                                      T=TTREPLY(QCPU(I,K))-TTREPLY(CPUSR);
393                                      IF T<0 THEN T=ARGUMEN;ELSE IF T<ARGUMEN
396                                      THEN T=ARGUMEN-T;ELSE T=0;
398                                 CPUWAIT(QCPU(I,K))  =CPUWAIT(QCPU(I,K))+
                                         T;
399                                      TTREPLY(QCPU(I,K))=TTREPLY(QCPU(I,K))+T;
400                                 END;
401                            END;
402                       CPUUSE=CPUUSE+ARGUMEN;
403                       TCPU(CPUSR)=TCPU(CPUSR)+ARGUMEN;
404                       TTREPLY(CPUSR)=TTREPLY(CPUSR)+ARGUMEN;
405                       DEPARTCPU(CPUSR)=TTREPLY(CPUSR);
406                       INTCYC(CPUSR)=INTCYC(CPUSR)-ARGUMEN*200;

407                  FIN5: END CPU ;
                      /*****************************************************/
                      /*                                                   */
                      /*****************************************************/
408                   SWAPING: PROC ;
409                     DCL (I,J,I3,J3,T,K) BIN FIXED(31);
                      /* ''SWAPING'' RECOIT LE CONTROLE DE ''MAIN'' QUAND EST SIGNALEE
                            UNE DEMANDE DE PAGE */
                      /* LES DECLARATIONS ET LES INITIALISATIONS ONT ETE FAITES DANS LE
                      PROGRAMME PRINCIPAL */
```

```
                /* ATTENDRE LE CONTROLE */

                     /* DETERMINER LE PROCHAIN UTILISATEUR A SATISFAIRE */
410                   CALL NXTSWAP ;

411                  IF XMRDY1(SWAPUSR) THEN GOTO ETI14;
                     /* SATISFAIRE LA DEMANDE DE L'UTILISATEUR SELECTE */
413                  J=PRTY(SWAPUSR) ;
                     /* METTRE L'UTILISATEUR DANS LA QUEUE DES DEMANDES
                     SERVICE-DISQUE POUR LA TRANSMISSION D'UNE PAGE */
414                ETI11 :

                   /*PAGE A JETER OU A CHARGER:METTRE L'UTILISATEUR DANS LA QUEUE
                       SERVICE-DISQUE */
                   /* OBTENIR LE CANAL AUQUEL L'UNITE DEMANDEE EST CONNECTEE */
                   /* INDIQUER L'UNITE REQUISE */
                   DEPARTSWAP(SWAPUSR)=TTREPLY(SWAPUSR);
415                          SWPDEVICE=NUNITE;
416                STWTCANAL  (SWAPUSR)=TTREPLY(SWAPUSR);
417                T=0;
418                I3=EQUIPMENT(SWPDEVICE,2) ;
419                DO I=1 TO NUSER;
420                     IF T<DEPARTCANAL(I3       ,I) |T=0 THEN T=DEPARTCANAL(I3,I);
422                END;
423                IF STWTCANAL(SWAPUSR)<T THEN DO;
425                          CANALWAIT(SWAPUSR)=CANALWAIT(SWAPUSR)+
                   T-STWTCANAL(SWAPUSR);
426                          TTREPLY(SWAPUSR)=TTREPLY(SWAPUSR)+T-STWTCANAL(SWAPUSR);
427                END;
                   /* METTRE LA DEMANDE D'E/S DANS LA QUEUE CANAL AUQUEL
                   SWAPDEVICE EST CONNECTEE */
428                J3=PRTY(SWAPUSR) ;
429                QCANAL(I3,IQCANAL(I3,J3),J3,1)=SWAPUSR ;
430                QCANAL(I3,IQCANAL(I3,J3),J3,2)=SWPDEVICE ;
431                                          QCANAL (I3,IQCANAL(I3,J3),J3,3)=48;
432                QCANAL(I3,IQCANAL(I3,J3),J3,4)=0 ;
433                IQCANAL(I3,J3)=IQCANAL(I3,J3)+1 ;
434                JMAX(I3)=MAX(JMAX(I3),QLENGTH2(QCANAL,IQCANAL,I3));
                   /* LANCER LA DEMANDE D'E/S */
435                    GOTO FIN4;
436              ETI14: XMRDY1(SWAPUSR)='0'B;

437                     IF EJECT(SWAPUSR)¬=0 THEN DO;
439                              EJECT(SWAPUSR)=0;

440                    END:
```

```
                    /* SIGNALER AU MODULE ''MAIN'' QU'ON A BESOIN DU CPU POUR
                    L'EXECUTION DE L'ALGORITHME DE SWAPPING */
443                 ALGOGO='1'B;
444                 T=MAX1(DEPARTCPU);
445                 IF TTREPLY(SWAPUSR)<T THEN DO;
447                       CPUWAIT(SWAPUSR)=CPUWAIT(SWAPUSR)+T-TTREPLY(SWAPUSR);
448                       TTREPLY(SWAPUSR)=T;
449                 END;
450                 CALL CPU;
                    /* ATTENDRE LA FIN DE L'EXECUTION DE L'ALGORITHME DE SWAPPING */

451               IF CPL1=0 THEN GOTO ETI20;
453               EJECT(SWAPUSR)='1'B;
454               GOTO ETI11;
                    /* L'OPERATION DE SWAPPING EST ACHEVEE : LE SIGNALER AU MODULE
                    ''MAIN'' */
455               END;
456                 ETI20 :
                  CPL=CPL-1;
457               J=NXTPARTIE(SWAPUSR);
458               TABLEPAGE(SWAPUSR,J,1)=TABLEPAGE(SWAPUSR,J,1)+1;
459               IF REENTRANT(SWAPUSR,J) THEN TABLEPAGE(SWAPUSR,J,2)=TABLEPAGE(
                  SWAPUSR,J,2)+1;
461               REFPARTIE(SWAPUSR,J)=REFPARTIE(SWAPUSR,J)-1;
                    /* METTRE L'UTILISATEUR DANS LA QUEUE SERVICE-CPU */
462                 IMAX(1)=MAX(IMAX(1),QLENGTH1(QCPU,IQCPU));
463                 IF TIMEFIL(SWAPUSR) THEN
464                       TIMEFIL(SWAPUSR)=TIMEFIL(SWAPUSR)-1;
465                 IF TIMESEG(SWAPUSR) THEN
466                       TIMESEG(SWAPUSR)=TIMESEG(SWAPUSR)-1;
467                 REFPAGE(SWAPUSR)=REFPAGE(SWAPUSR)-1;
468                 IF REFPAGE(SWAPUSR)>0 THEN DO;
470                 IF INTCYC(SWAPUSR)< 400 THEN TIMEPAG(SWAPUSR)=0;
472                 ELSE TIMEPAG(SWAPUSR)=1;
                  /* DETERMINER LA PROCHAINE PARTIE UTILISEE PAR SWAPUSR*/
473               IF REFPARTIE(SWAPUSR,J)=0 THEN
474                 DO K=J+1 TO 9;
475                       IF REFPARTIE(SWAPUSR,K)¬=0 THEN  DO;
477                             NXTPARTIE(SWAPUSR)=K; GOTO SUITE;
479                       END;
480                 END;
481               END;
482                 ELSE TIMEPAG(SWAPUSR)=0;
483               SUITE:
                    DEPARTSWAP(SWAPUSR)=TTREPLY(SWAPUSR);
                    STWTCPU(SWAPUSR)=TTREPLY(SWAPUSR);
485                 T=MAX1(DEPARTCPU);
```

```
486                 IF STWTCPU(SWAPUSR)<T THEN DO;
488                       CPUWAIT(SWAPUSR)=CPUWAIT(SWAPUSR)+
                    T-STWTCPU(SWAPUSR);
489                   TTREPLY(SWAPUSR)=TTREPLY(SWAPUSR)+T-STWTCPU(SWAPUSR);
490                 END;
491                 J=PRTY(SWAPUSR);
492                 QCPU(IQCPU(J),J)=SWAPUSR ;
493                 IQCPU(J)=IQCPU(J)+1 ;

494           FIN4: END SWAPING ;
                  /*****************************************************************/
                  /*                                                               */
                  /*****************************************************************/
495           CHANNEL: PROC(ICANAL) ;
                    /* CHANNEL(ICANAL) SIMULE LE FONCTIONNEMENT DU CANAL ICANAL */
496                 DCL(ICANAL) BIN FIXED(31) ;
497                 DCL(K,J2    ,I,J,T) BIN FIXED(31);
                    /* ATTENDRE LA FIN DE SIMULATION OU LE MOMENT DE L'UTILISATION
                    DE ICANAL */
                    /* TEST SUR LE TYPE DU CANAL */
498                 IF CANAL(ICANAL)=0 THEN GOTO MULTIPLEX ;
                    /* SINON ICANAL EST SELECTEUR */
                        /* DETERMINER LE PROCHAIN UTILISATEUR A SATISFAIRE */
500                 CALL NXTCANAL(ICANAL) ;
501                 CHANNELUSR=CANALUSR(ICANAL) ;

502                 T=0;
503                 DO I=1 TO NUSER;
504                     IF T<DEPARTCANAL(ICANAL,I)|T=0 THEN T=DEPARTCANAL(ICANAL,I);
506                 END;
507                 IF T<STWTCANAL(CHANNELUSR) THEN IDLECANAL(ICANAL)=IDLECANAL(ICANAL)+
                    STWTCANAL(CHANNELUSR)-T;

509                 IF FLAG(ICANAL)=1 THEN DO ;
511                    TCANAL1(CHANNELUSR)=TCANAL1(CHANNELUSR)+TRANSFERT(ICANAL);
512                    DO J=1 TO 4 ;
513                    DO I=1 TO IQCANAL(ICANAL,J)-1 ;
514                       T=TTREPLY(QCANAL(ICANAL,I,J,1))-TTREPLY(CHANNELUSR);
515                       IF T<0 THEN T=TRANSFERT(ICANAL);ELSE IF T<TRANSFERT(ICANAL)
518                       THEN T=TRANSFERT(ICANAL)-T;ELSE T=0;
520                      CANALWAIT(QCANAL(ICANAL,I,J,1))  =CANALWAIT(QCANAL(ICANAL,
                       I,J,1))  +T;
521                       TTREPLY(QCANAL(ICANAL,I,J,1))=TTREPLY(QCANAL(ICANAL,I,J,1))+T;
522                       END ;
523                       DO I=1 TO NUNITE ;
524                       IF EQUIPMENT(I,2)=ICANAL THEN DO ;
```

```
                    TTREPLY(QUNITE(I,K,J,1))=TTREPLY(CHANNELUSR);
528                 IF T<0 THEN T=TRANSFERT(ICANAL);ELSE IF T<TRANSFERT(ICANAL)
531                 THEN T=TRANSFERT(ICANAL)-T;ELSE T=0;
533                 UNITEWAIT(QUNITE(I,K,J,1))  =UNITEWAIT(QUNITE(I,K,J,1))+T;
534                 TTREPLY(QUNITE(I,K,J,1))=TTREPLY(QUNITE(I,K,J,1))+T;
535                 END ;
536                 END ;
537                 END ;
538                 END ;
539               TUNITE(CHANNELUSR)=TUNITE(CHANNELUSR)+TRANSFERT(ICANAL) ;
540               TTREPLY(CHANNELUSR)=TTREPLY(CHANNELUSR)+TRANSFERT(ICANAL);
                  /* INDIQUER QUE LA TRANSMISSION EST ACHEVEE */
541               DEPARTCANAL(ICANAL,CHANNELUSR)=TTREPLY(CHANNELUSR);
542               WAITRANSMISSION(DEVICE(ICANAL))='0'B;
543               DEPARTUNITE(DEVICE(ICANAL),CHANNELUSR)=TTREPLY(CHANNELUSR);
544               IF TRANSFERT(ICANAL)=42 THEN GOTO SEG ;
546               IF TRANSFERT(ICANAL)=17 THEN GOTO FIL ;
                  /* ON VIENT DE REALISER UN TRANSFERT DE PAGE */

548               XMRDYI(CHANNELUSR)='1'B;
549                STWTSWP(CHANNELUSR)=TTREPLY(CHANNELUSR);
550                T=MAXI(DEPARTSWAP);
551                IF STWTSWP(CHANNELUSR)<T THEN DO;
553                      SWPWAIT(CHANNELUSR)=SWPWAIT(CHANNELUSR)+T-STWTSWP(
                       CHANNELUSR);
554                      TTREPLY(CHANNELUSR)=TTREPLY(CHANNELUSR)-STWTSWP(
                       CHANNELUSR)+T;
555                END;
556                J=PRTY(CHANNELUSR);
557                QSWAP(IQSWAP(J),J)=CHANNELUSR;
558                IMAX(2)=MAX(IMAX(2),QLENGTHI(QSWAP,IQSWAP));
559                IQSWAP(J)=IQSWAP(J)+1;
560               GOTO FINCANAL;
561          SEG: /* ON VIENT DE REALISER UN CHARGEMENT DE SEGMENT */
                  /* MISE A JOUR DES INTERVALLES D'INTERRUPTION */

                  CALL OPEN(REFOPEN(CHANNELUSR));
562               IF TIMEFIL(CHANNELUSR) THEN
563               TIMEFIL(CHANNELUSR)=TIMEFIL(CHANNELUSR)-300;
564               IF TIMEPAG(CHANNELUSR) THEN TIMEPAG(CHANNELUSR)=
                  TIMEPAG(CHANNELUSR)-TIMESEG(CHANNELUSR) ;
566               REFSEG(CHANNELUSR)=REFSEG(CHANNELUSR)-1 ;
567               IF REFSEG(CHANNELUSR) > 0 THEN TIMESEG(CHANNELUSR)=3
                  /* INITIALISER UN NOUVEAU TIMESEG */ ;
569               ELSE TIMESEG(CHANNELUSR)=0 ;
570               GOTO CAN3 ;
571          F L: /* ON VIENT DE REALISER LE TRANSFERT D'UN ENREGISTREMENT DE
```

```
                  FICHIER */

                     /* MISE A JOUR DES INTERVALLES D'INTERRUPTION */
572                  IF TIMESEG(CHANNELUSR) THEN TIMESEG(CHANNELUSR)=
                     TIMESEG(CHANNELUSR)-TIMEFIL(CHANNELUSR) ;
573                  IF TIMEPAG(CHANNELUSR) THEN TIMEPAG(CHANNELUSR)=
                     TIMEPAG(CHANNELUSR)-TIMEFIL(CHANNELUSR) ;
575                  REFILE(CHANNELUSR)=REFILE(CHANNELUSR)-1 ;
576                  IF REFILE(CHANNELUSR) > 0 THEN
                        /* INITIALISER UN NOUVEAU TIMEFIL */
577                  TIMEFIL(CHANNELUSR)=0;
578                  ELSE DO; TIMEFIL(CHANNELUSR)=0; CALL OPEN(REFOPEN(CHANNELUSR));
581                  END;
582               CAN3: /* REMETTRE L'UTILISATEUR DANS QCPU */

                     J2=PRTY(CHANNELUSR) ;
583                  STWTCPU(CHANNELUSR)=TTREPLY(CHANNELUSR);
584                  T=MAX1(DEPARTCPU);
585                  IF STWTCPU(CHANNELUSR)<T THEN DO;
587                        CPUWAIT(CHANNELUSR)=
                     CPUWAIT(CHANNELUSR)+T-STWTCPU(CHANNELUSR);
588                        TTREPLY(CHANNELUSR)=TTREPLY(CHANNELUSR)+T-STWTCPU(
                        CHANNELUSR);
589                  END;
590                  QCPU(IQCPU(J2),J2)=CHANNELUSR ;
591                  IQCPU(J2)=IQCPU(J2)+1 ;
592                  IMAX(1)=MAX(IMAX(1),QLENGTH1(QCPU,IQCPU));
593                  GOTO FINCANAL;
594               END ;
                  /* FLAG(ICANAL) EST NUL: ON A BESOIN DU CANAL POUR LE LANCEMENT
                  DE L'E/S SUR DEVICE(ICANAL) */

                  /* NOTER LE TEMPS D'UTILISATION DU CANAL PAR L'UTILISATEUR COURANT
                  AINSI QUE LES TEMPS D'ATTENTE DANS QCANAL */
595               TCANAL(CHANNELUSR)=TCANAL(CHANNELUSR)+1 ;
596               DO J=1 TO 4 ;
597                  DO I=1 TO IQCANAL(ICANAL,J)-1 ;
598                     T=TTREPLY(QCANAL(ICANAL,I,J,1))-TTREPLY(CHANNELUSR);
599                     IF T>0 THEN T=0; ELSE T=1;
602                     TTREPLY(QCANAL(ICANAL,I,J,1))=TTREPLY(QCANAL(ICANAL,I,J,1))+T;
603                     CANALWAIT(QCANAL(ICANAL,I,J,1))  =CANALWAIT(
                        QCANAL(ICANAL,I,J,1))+T;
604                  END ;
605               DO I=1 TO NUNITE;
606               IF EQUIPMENT(I,2)=ICANAL THEN IF WAITRANSMISSION(I) THEN DO;
609               DO K=1 TO IQUNITE(I,J)-1:
```

```
611                 IF T>0 THEN T=0; ELSE T=1;
614                 TTREPLY(QUNITE(I,K,J,1))=TTREPLY(QUNITE(I,K,J,1))+T;
615                 UNITEWAIT(QUNITE(I,K,J,1))=UNITEWAIT(QUNITE(I,K,J,1))+T;
616               END;
617               END;
618               END;
619               END ;
                  /* METTRE L'UTILISATEUR SUR LA FILE D'ATTENTE DE L'UNITE DESIREE */
620               TTREPLY(CHANNELUSR)=TTREPLY(CHANNELUSR)+1;
621               DEPARTCANAL(ICANAL,CHANNELUSR)=TTREPLY(CHANNELUSR);
622               STWTUNITE(CHANNELUSR)=TTREPLY(CHANNELUSR);
623               J=DEVICE(ICANAL);
624               T=0;
625               DO I=1 TO NUSER;
626                     IF T<DEPARTUNITE(J,I) |T=0 THEN T=DEPARTUNITE(J,I);
628               END;
629               IF STWTUNITE(CHANNELUSR)<T THEN DO;
631                           UNITEWAIT(CHANNELUSR)=UNITEWAIT(
                  CHANNELUSR)+T-STWTUNITE(CHANNELUSR);
632                           TTREPLY(CHANNELUSR)=TTREPLY(CHANNELUSR)+T-STWTUNITE(
                            CHANNELUSR);
633               END;
634               K=PRTY(CHANNELUSR) ;
635               QUNITE(J      ,IQUNITE(J              ,K),K,1)=CHANNELUSR;
636               QUNITE(J      ,IQUNITE( J             ,K),K,2)=TRANSFERT(ICANAL);
637               IQUNITE(J,K)=IQUNITE(J,K)+1;
638               KMAX(J)=MAX(KMAX(J),QLENGTH2(QUNITE,IQUNITE,J));
639             MULTIPLEX:

                FINCANAL:
                END CHANNEL ;
                 /*****************************************************************/
                 /*                                                               */
                 /*****************************************************************/
640             UNITE: PROC(IUNITE) ;
                  /* UNITE(IUNITE) SIMULE L'UNITE IUNITE */
641               DCL IUNITE BIN FIXED(31) ;
642               DCL(DEVICEUSR,TEM,I,J,J1,I1,T) BIN FIXED(31);
643               DCL K BIN FIXED(31);
                  /* ATTENDRE LA FIN DE LA SIMULATION OU LA DEMANDE D'EXPLOITATION
                  DE IUNITE */
644             UN1: /* DETERMINER LE PROCHAIN UTILISATEUR A SATISFAIRE */

                  CALL NXTUNITE(IUNITE) ;
645               DEVICEUSR=UNITEUSR(IUNITE) ;
```

```
                SI IUNITE EST UNE PILE DE DISQUES) */
646             TEM=EQUIPMENT(IUNITE,1) ;
                /* NOTER LES TEMPS D'UTILISATION DE IUNITE PAR L'UTILISATEJR COURANT
                ET METTRE A JOUR LES TEMPS D'ATTENTE DANS LES FILES */
647             TUNITE(DEVICEUSR)=TUNITE(DEVICEUSR)+TEM ;
648             K=EQUIPMENT(IUNITE,2);
649             DO J=1 TO 4 ;
650                DO I=1 TO IQUNITE(IUNITE,J)-1 ;
651                   T=TTREPLY(QUNITE(IUNITE,I,J,1))-TTREPLY(DEVICEUSR);
652                   IF T<0 THEN T=TEM;ELSE IF T<TEM THEN T=TEM-T;ELSE T=0;
657                   UNITEWAIT(QUNITE(IUNITE,I,J,1))  =UNITEWAIT(QUNITE
                      (IUNITE,I,J,1))+T;
658                   TTREPLY(QUNITE(IUNITE,I,J,1))=TTREPLY(QUNITE(IUNITE,I,J,1))+T;
659                END ;
660             END ;
                /* METTRE L'UTILISATEUR DANS LA FILE D'ATTENTE DU CANAL AUQUEL
                IUNITE EST CONNECTEE */
661             J1=PRTY(DEVICEUSR) ;
662             I1=EQUIPMENT(IUNITE,2) ;
663             TTREPLY(DEVICEUSR)=TTREPLY(DEVICEUSR)+TEM;
664             STWTCANAL(DEVICEUSR)=TTREPLY(DEVICEUSR);
665             T=0;
666             DO I=1 TO NUSER;
667                IF T<DEPARTCANAL(I1,I) THEN T=DEPARTCANAL(I1,I);
669             END;
670             IF STWTCANAL(DEVICEUSR)<T THEN DO;
672                   CANALWAIT(DEVICEUSR)=CANALWAIT(
                DEVICEUSR)+T-STWTCANAL(DEVICEUSR);
673                   TTREPLY(DEVICEUSR)=TTREPLY(DEVICEUSR)+T-STWTCANAL(
                      DEVICEUSR);
674             END;
675             QCANAL(I1,IQCANAL(I1,J1),J1,1)=DEVICEUSR ;
676             QCANAL(I1,IQCANAL(I1,J1),J1,2)=IUNITE ;
677             QCANAL(I1,IQCANAL(I1,J1),J1,3)=TYPEES(IUNITE) ;
678             QCANAL(I1,IQCANAL(I1,J1),J1,4)=1 ;
679             IQCANAL(I1,J1)=IQCANAL(I1,J1)+1 ;
680             JMAX(I1)=MAX(JMAX(I1),QLENGTH2(QCANAL,IQCANAL,I1));
                /* LANCER LA DEMANDE D'UTILISATION DU CANAL */
681             WAITTRANSMISSION(IUNITE)='1'B;
682             DEPARTUNITE(IUNITE,DEVICEUSR)=TTREPLY(DEVICEUSR);

683          END UNITE ;
             /*****************************************************************/
             /*                                                               */
             /*****************************************************************/
684          CONSOLE: PROC ;
```

```
                /* ''CONSOLE'' SIMULE L'ENSEMBLE DES CONSOLES */
686             ETI4 : IE=IE+1;

687                  DO I=1 TO NUSER  ;
688                           IF ¬ON(I) THEN GOTO ETI5;
690                                    IF RDY(I) THEN DO;
692                                    IF NSESION(I)>ISESION(I) THEN GOTO DEHORS;
694                                    IF TIM(NSESION(I),I)<300000 THEN GOTO SUIT;
696                                    DEHORS: ON(I)='0'B;
                                       /* METTRE L'UTILISATEUR DANS LA CHAINE
                                       DES CONSOLES DECONNECTEES */
697                                    ON(I)='0'B;
698                                    IOUT=IOUT+1 ;
699                                    IF IOUT=NUSER THEN DO;
701                                    ENDTERM='1'B;
702                                    GOTO FIN2;
703                                    END ;
704                                    GOTO ETI5;
705                                    SUIT:
                                       /* PREVOIR L'HEURE D'ENVOI DU PROCHAIN
                                       MESSAGE DE L'UTILISATEUR I */
                                       TTREPLY(I)=TTREPLY(I)+
                                       TIM(NSESION(I),I);

                                       /* ATTENDRE QUE LE MAIN AIT RECU LE
                                       MESSAGE */
706                                    REFSEG(I),REFPAGE(I),REFILE(I),REFOPEN(I)=
                                       0;
707                                    BLOCOPEN(I)=0;
708                                    CALL GENERE(TYPE(NSESION(I),I),I);
                                       /* BLOQUER LA CONSOLE DE L'UTILISATEUR */
709                                    RDY(I)=0;
710                                    CPUWAIT(I),SWPWAIT(I)=0;
711                                    TUNITE(I),TCANAL(I),TCANAL1(I)=0;
712                                    STWTCPU(I)=TTREPLY(I);
713                                    UNITEWAIT(I),CANALWAIT(I)=0;
714                                    TCPU(I)=0;
715                                    T=MAX1(DEPARTCPU);
716                                    IF STWTCPU(I)<T THEN DO;
718                                         CPUWAIT(I)=T-STWTCPU(I);
719                                         TTREPLY(I)=TTREPLY(I)+T-STWTCPU(I);
720                                    END;
721                                    J=PRTY(I);
722                                    QCPU(IQCPU(J),J)=I ;
723                                    IMAX(1)=MAX(IMAX(1),QLENGTH(QCPU,IQCPU));
724                                    IQCPU(J)=IQCPU(J)+1 ;
725                                    END ;
```

```
726             ETI5: END ;
727               FIN2:

                 END CONSOLE;
                 /*****************************************************************/
                 /*                                                               */
                 /*****************************************************************/
                 /*****************************************************************/
                 /*                                                               */
                 /*****************************************************************/
728             NEXTCPU: PROC ;
729               DCL (I,J,T,K,L) BIN FIXED(31);
                /* SELECTIONNE LE PROCHAIN UTILISATEUR A SATISFAIRE DANS LE
                SERVICE-CPU */
                      /* VERIFIER L'ETAT DES CONSOLES */
730                   T=0;
731                   DO I=1 TO 4 ;
732                         DO J=1 TO IQCPU(I)-1;
733                               IF T>STWTCPU(QCPU(J,I))|T=0 THEN DO;
735                                     T=STWTCPU(QCPU(J,I));
736                                     USER=QCPU(J,I);
737                                     K=I;
738                                     L=J;
*739                        END;
740                               END;
741                   END ;
742                   CALL DELETE(QCPU,IQCPU(K),K,L);
743              END NEXTCPU;
                 /*****************************************************************/
                 /*                                                               */
                 /*****************************************************************/
744             NXTSWAP: PROC ;
745               DCL (I,J,T,K,L) BIN FIXED(31);
                /* SELECTIONNE LE PROCHAIN UTILISATEUR A SATISFAIRE DANS LE
                      SERVICE-SWAPPING */
746                   T=0;
747             DO I=1 TO 4 ;
748                         DO J=1 TO IQSWAP(I)-1;
749                               IF T>STWTSWP(QSWAP(J,I))|T=0 THEN DO;
751                                     T=STWTSWP(QSWAP(J,I));
752                                     SWAPUSR=QSWAP(J,I);
753                                     K=I;
754                                     L=J;
755                               END;
756                         END;
757             END ;
```

```
                END NXTCANAL ;
                /*****************************************************************/
                /*                                                               */
                /*****************************************************************/
760             NXTCANAL: PROC(ICANAL) ;
                  /* DETERMINE LE PROCHAIN UTILISATEUR A SATISFAIRE */
761               DCL ICANAL BIN FIXED(31) ;
                  /* -CRITERE DE CHOIX: DEMANDE LA PLUS ANCIENNE DE PRIORITE
                     LA PLUS HAUTE */
762               DCL(I,J,K,L,M,N,T) BIN FIXED(31);
763               T=0;
764               DO J=1 TO 4 ;
765                   DO I=1 TO IQCANAL(ICANAL,J)-1;
766                         IF T>STWTCANAL(QCANAL(ICANAL,I,J,1))|T=0 THEN DO;
768                               T=STWTCANAL(QCANAL(ICANAL,I,J,1));
769                               M=I;
770                               N=J;
771                         END;
772                   END;
773               END;
774                     DEVICE(ICANAL)=QCANAL(ICANAL,M,N,2) ;
775                     CANALUSR(ICANAL)=QCANAL(ICANAL,M,N,1) ;
776                     FLAG(ICANAL)=QCANAL(ICANAL,M,N,4) ;
777                     TRANSFERT(ICANAL)=QCANAL(ICANAL,M,N,3) ;
778                     IQCANAL(ICANAL,N)=IQCANAL(ICANAL,N)-1 ;
779                     DO K=M TO IQCANAL(ICANAL,N)-1 ;
780                     DO L=1 TO 4 ;
781                     QCANAL(ICANAL,K,N,L)=QCANAL(ICANAL,K+1,N,L) ;
782                     END ;
783                     END ;
784                     QCANAL(ICANAL,IQCANAL(ICANAL,N),N,*)=0;
785             END NXTCANAL ;
                /*****************************************************************/
                /*                                                               */
                /*****************************************************************/
786             NXTUNITE: PROC(IUNITE) ;
                  /* DETERMINE LE PROCHAIN UTILISATEUR A SATISFAIRE - MEME CRITERE
                  DE CHOIX QUE POUR NXTCANAL */
787               DCL IUNITE BIN FIXED(31) ;
788               DCL(I,J,K,L,T,M,N) BIN FIXED(31);
789               T=0;
790               DO J=1 TO 4 ;
791                   DO I=1 TO IQUNITE(IUNITE,J)-1;
792                         IF T>STWTUNITE(QUNITE(IUNITE,I,J,1)) |T=0 THEN DO;
794                               T=STWTUNITE(QUNITE(IUNITE,I,J,1));
795                               M=I;
796                               N=J;
```

```
797                     END;
798                END;
799             END;
800                   UNITEUSR(IUNITE)=QUNITE(IUNITE,M,N,1) ;
801                   TYPEES(IUNITE)=QUNITE(IUNITE,M,N,2) ;
802                   IQUNITE(IUNITE,N)=IQUNITE(IUNITE,N)-1 ;
                      /* DESENPILER L'UTILISATEUR SELECTE */
803                   DO K=M TO IQUNITE(IUNITE,N)-1 ;
804                   DO L=1 TO 2 ;
805                   QUNITE(IUNITE,K,N,L)=QUNITE(IUNITE,K+1,N,L) ;
806                   END ;
807                   END ;
808                   QUNITE(IUNITE,IQUNITE(IUNITE,N),N,*)=0;
809           END NXTUNITE ;
              /*****************************************************************/
              /*                                                               */
              /*****************************************************************/
810           DELETE :PROC(PILE,UPBOUND,J,L);
              /* SUPRIME L'ELEMENT QUI SE TROUVE AU FOND DE LA PILE */
811                DCL(    UPBOUND,PILE(*,*),L    ) BIN FIXED(31);
812                DCL J BIN FIXED(31);
813                DCL K BIN FIXED(31);
814                            UPBOUND=UPBOUND-1 ;
815                            DO K=L TO UPBOUND-1;
816                                    PILE(K,J)=PILE(K+1,J) ;
817                            END ;
818                PILE(UPBOUND,J)=0;
819           END DELETE ;
              /*****************************************************************/
              /*                                                               */
              /*****************************************************************/
820           MAX1 :PROC(ARRAY)BIN FIXED(31);
821                DCL (I,J,ARRAY(NUSER)) BIN FIXED(31);
822                  J=0;
823                  DO I=1 TO NUSER;
824                       IF J<ARRAY(I) THEN J=ARRAY(I);
826                  END;
827                  RETURN(J);
828            END MAX1;
              /*****************************************************************/
              /*                                                               */
              /*****************************************************************/
829            MIN1 : PROC(ARRAY) BIN FIXED(31);
830                  DCL (I,J,ARRAY(NUSER)) BIN FIXED(31);
831            J=0;
832            DO I=1 TO NUSER;
```

```
836                END;
837                     RETURN(J);
838                END MIN1;
                   /*****************************************************************/
                   /*                                                               */
                   /*****************************************************************/
839                QLENGTH1 :PROC(ARRAY,IARRAY) BIN FIXED(31);
840                     DCL (ARRAY(NUSER,4),IARRAY(4)) BIN FIXED(31);
841                     DCL(I,J,T) BIN FIXED(31);
842                             HORA=MIN1(TTREPLY);
843                             T=0;
844                             DO J=1 TO 4;
845                                  DO I=1 TO IARRAY(J)-1;
846                                       IF TTREPLY(ARRAY(I,J))<=HORA THEN T=T+1;
848                                  END;
849                             END;
850                             RETURN(T);
851                END QLENGTH1;
                   /*****************************************************************/
                   /*                                                               */
                   /*****************************************************************/
852                OPEN:PROC(NOPEN);
                    /*OPEN MET A JOUR LES TEMPS D'ATTENTE DUS AUX NOPEN OUVERTURES OU
                    FERMETURES DE FICHIERS*/
853                 DCL (I,J,T,NOPEN) BIN FIXED(31);
                    /* TOUS LES UTILISATEURS SONT BLOQUES PENDANT NOPEN*300  MILLISEC*/
854                 DO I=1 TO NUSER;
855                      IF I¬=CHANNELUSR THEN IF ON(I) THEN DO;
858                      T=TTREPLY(I)-TTREPLY(CHANNELUSR);
859                      IF T<0 THEN T=NOPEN*300; ELSE IF T<300*NOPEN THEN T=300*NOPEN-T;
863                      ELSE T=0;
864                      BLOCOPEN(I)=BLOCOPEN(I)+T;
865                        TTREPLY(I)=TTREPLY(I)+T;
866                      END;
867                 END;
868                 TTREPLY(CHANNELUSR)=TTREPLY(CHANNELUSR)+NOPEN*300;
869                 BLOCOPEN(CHANNELUSR)=BLOCOPEN(CHANNELUSR)+NOPEN*300;
870                END OPEN;
                   /*****************************************************************/
                   /*                                                               */
                   /*****************************************************************/
871                 HEURE:PROC(TPS) BIN FIXED(31);
                   /*DONNE L'HEURE EN MILLISEC*/
872                 DCL TPS CHAR(9);
873                 DCL I BIN FIXED(31);
874                      I=SUBSTR(TPS,1,2)+3600000+SUBSTR(TPS,3,2)*600000+SUBSTR(TPS,5,2)*
                         1000+SUBSTR(TPS,7,3);
```

```
875                 RETURN(I);
876           END HEURE;
              /*************************************************************************/
              /*                                                                       */
              /*************************************************************************/
877           QLENGTH2 : PROC(ARRAY,IARRAY,NO) BIN FIXED(31);
878                 DCL(NO,ARRAY(*,*,*,*    ),IARRAY(*,*)) BIN FIXED(31);
879                 DCL (I,J,T) BIN FIXED(31);
880                       HORA=MIN1(TTREPLY);
881                       T=0;
882                       DO J=1 TO 4;
883                             DO I=1 TO IARRAY(NO,J)-1;
884                                   IF TTREPLY(ARRAY(NO,I,J,1))<=HORA THEN T=T+1;
886                             END;
887                       END;
888                       RETURN(T);
889           END QLENGTH2;
              /*************************************************************************/
              /*                                                                       */
              /*************************************************************************/
890           PAGINE : PROC;
891           DCL STOPY BIN FIXED(31);
892           DCL USR BIN FIXED(31);
893           DCL (CPL5,CPL3) BIN FIXED(31);
894            DCL(I,J,T) BIN FIXED(31);
895           DCL (H1,H2) CHAR(9);
896                CPL1=0;CPL3,CPL5=0;
898                USR=PUI;
899                H1=TIME;
900           PAG1 : T=TTREPLY(USR)-MIN1(TTREPLY);
901                IF TCPU(USR)=0&TUNITE(USR)=0&TCANAL(USR)=0&(T<5|T>10) THEN DO;
                         /*L'UTILISATEUR EST TEMPORAIREMENT INACTIF*/
903                      USR=USR+1;
904                      IF USR>NUSER THEN USR=1;
906                      IF USR¬=PUI THEN GOTO PAG1;
908                      IF CPL5+CPL3+CPL1=0 THEN DO;
910                            DO USR=1 TO NUSER;
911                            DO J=2 TO 9;
912                                  IF TABLEPAGE(USR,J,1)>0 THEN DO;
914                                        CPL1=1;
915                                        TABLEPAGE(USR,J,1)=TABLEPAGE(USR,J,1)-1;
916                                        GOTO PAG12;
917                                  END;
918                            END;
919                      END;
920                            END;
```

```
                    END;
                    /*L'UTILISATEUR N'EST PAS TEMPORAIREMENT INACTIF*/
                    /*PARCOURIR LA TABLE DES PAGES VALIDES*/
923                 DO J=2 TO 9;
                    /*BOUCLE SUR LES PARTIES*/
                    /* NE PAS PRENDRE DE PAGES DANS LA PARTIE CONTROLE*/
924                 IF CPL5=5 THEN GOTO PAG2;
926                 IF TABLEPAGE(USR,J,2)>0 THEN DO;
928                 IF TABLEPAGE(USR,J,2)>(5-CPL5) THEN DO;
                         /*INVALIDER LES PAGES PRISES*/
930                      TABLEPAGE(USR,J,2)=TABLEPAGE(USR,J,2)-(5-CPL5);
931                      TABLEPAGE(USR,J,1)=TABLEPAGE(USR,J,1)-(5-CPL5);
932                      CPL5=5;
933                 END;
934                 ELSE DO;
                         /*INVALIDER LES PAGES PRISES*/
935                      CPL5=CPL5+TABLEPAGE(USR,J,2);
936                      TABLEPAGE(USR,J,1)=TABLEPAGE(USR,J,1)-TABLEPAGE(USR,J,2);
937                      TABLEPAGE(USR,J,2)=0;
938                 END;
939                 END;
940                 END;
941           PAG2: IF CPL3=3 THEN IF CPL1=1 THEN GOTO PAG10;
                    /* CALCULER L'HISTORIQUE DE LA SESSION DE L'UTILIATEJR*/
944                 STORY=HISTORIC(USR,3);
945                 IF STORY=4 THEN GOTO PAG3;
947                 IF HISTORIC(USR,2)¬=STORY THEN STORY=STORY+HISTORIC(USR,2);
949                 IF HISTORIC(USR,2)¬=HISTORIC(USR,1) &HISTORIC(USR,1)¬=
950                 HISTORIC(USR,3) THEN STORY=STORY+HISTORIC(USR,1);
951                 IF STORY=7 THEN GOTO PAG11;
953           PAG3 : /* DETERMINER LA SUITE DES PARTIES INUTILISEES*/
                    DO J=REPERE(STORY,1) TO REPERE(STORY,2);
954                      I=LISTEPARTIE(J); /*I=NO DE PARTIE INUTILISEE*/
955           PAG4 :     IF TABLEPAGE(USR,I,2)=0 THEN DO;
                         /* PAS DE PAGES REENTRANTES DANS I*/
957                      IF CPL1¬=1 THEN DO;
959                           IF TABLEPAGE(USR,I,1)¬=0 THEN DO;
961                          CPL1=1;
962                          TABLEPAGE(USR,I,1)=TABLEPAGE(USR,I,1)-1;
963                          END;
964                      END;
965                      GOTO PAG7;
966                      END;
967                      IF CPL3 =3 THEN GOTO PAG6;
                         /* INVALIDER LA PAGE PRISE*/
969                      TABLEPAGE(USR,I,2)=TABLEPAGE(USR,I,2)-1;
970                      TABLEPAGE(USR,I,1)=TABLEPAGE(USR,I,1)-1;
```

```
 971                    CPL3=CPL3+1;
 972                    GOTO PAG4;
 973          PAG6 :    IF CPL1¬=1 & TABLEPAGE(USR,I,2)=0 THEN GOTO PAG4;
 975          PAG7:  END;
 976               IF CPL3¬=3 |CPL1¬=1 THEN GOTO PAG11;
 978          PAG10 :IF CPL5=5 THEN DO;
 980                    PUI=USR;
 981                    GOTO PAG12;
 982                END;
 983          PAG11 :/* PRENDRE LE PROCHAIN UTILISATEUR*/
                     USR=USR+1;
 984                 IF USR>NUSER THEN USR=1;
 986                 IF USR¬=PUI  THEN GOTO PAG1;
 988                      IF CPL5+CPL3+CPL1=0 THEN DO;
 990                           DO USR=1 TO NUSER;
 991                           DO J=2 TO 9;
 992                                IF TABLEPAGE(USR,J,1)>0 THEN DO;
 994                                     CPL1=1;
 995                                     TABLEPAGE(USR,J,1)=TABLEPAGE(USR,J,1)-1;
 996                                     GOTO PAG12;
 997                                END;
 998                           END;
 999                      END;
1000                      END;
1001          PAG12 : CPL=CPL5+CPL3+CPL1;
1002          H2=TIME;

1003               H=HEURE(H2); H=50;
1005          END PAGINE;
              /*****************************************************************/
              /*                                                               */
              /*****************************************************************/
1006          GENERE: PROC(GRP,NBUSER);
1007            DCL (I,J,K) BIN FIXED(31);
1008            DCL GRP CHAR(7) VAR,NBUSER BIN FIXED(31);
1009            DCL OK BIT(1) INIT('1'B);
1010            DCL N BIN FIXED(31);
1011            DCL FIT(0:12) LABEL INIT(C,M,IN,F,L,NU,EX,ST,CA,S,D,DJ,EC);
1012            TIMEFIL(NBUSER),TIMESEG(NBUSER)=0;
                /* MISE A JOUR DE L'HISTORIQUE;*/
1013            I=SUBSTR(GRP,1,1);
1014            PRTY(NBUSER)=I;
1015            DO K=1 TO 2;
1016                HISTORIC(NBUSER,K)=HISTORIC(NBUSER,K+1);
1017            END;
1018            IF PRTY(NBUSER)=4 THEN HISTORIC(NBUSER,2)=4;
```

```
                  /*GENERER LE MODELE D'ENTREE EN FONCTION DE LA COMMANDE LANCEE*/
1021              J=SUBSTR(GRP,2,1);
1022              REFPARTIE(NBUSER,*)=0;
1023              REENTRANT(NBUSER,*)=0;
1024              REFPARTIE(NBUSER,1)=1;
1025              IF I=3 THEN DO;
1027                  REFPARTIE(NBUSER,7),REFPARTIE(NBUSER,8),REFPARTIE(NBUSER,9)=1;
1028                  REENTRANT(NBUSER,7)='1'B; GOTO ETI(6+J);
1030              END;
1031              IF I=4 THEN DO;
1033                  REFSEG(NBUSER)=1; REFPARTIE(NBUSER,6)=1; GOTO ETI(8+J);
1036              END;
1037              REFPARTIE(NBUSER,4)=1; GOTO ETI(J);
1039            C: /*COMPILER*/
1040              REFPARTIE(NBUSER,3)=1; REFPARTIE(NBUSER,7)=2;
1041              N=/*NOMBRE DE LIGNES A COMPILER*/ SUBSTR(GRP,4);
1042              LFICHIERCOURANT(NBUSER)=LFICHIERCOURANT(NBUSER)+N;
1043              REFPARTIE(NBUSER,4)=LFICHIERCOURANT(NBUSER)/11+1;
1044              IF SUBSTR(GRP,3,1)='F' THEN DO;
1046                  REFSEG(NBUSER)=1;
1047              REFOPEN(NBUSER)=1; REFILE(NBUSER)=N;
1049              END;
1050              INTCYC(NBUSER)=1000*N;
1051              GOTO TEST;
1052              M: /* MODIFIER UNE LIGNE*/
                  REFPARTIE(NBUSER,3),REFPARTIE(NBUSER,4)=1;
1053              REFPARTIE(NBUSER,7)=2;
1054              INTCYC(NBUSER)=300*SUBSTR(GRP,3)+1000;
                /*SUBSTR(GRP,3) INDIQUE LE NOMBRE DE FOIS QUE L'UTILISATEUR MODIFIE
                  LA LIGNE*/
1055              GOTO TEST;
1056            IN : /*INSERER*/
1057              REFPARTIE(NBUSER,3)=1; INTCYC(NBUSER)=300;
1058              GOTO TEST;
1059            E: /* EFFACER*/
1060              REFPARTIE(NBUSER,3)=1; REFPARTIE(NBUSER,7)=2;
1061              INTCYC(NBUSER)=800;
1062              GOTO TEST;
1063            L : /* LISTER*/
                  N=/* NB DE LIGNES A LISTER*/ SUBSTR(GRP,3);
1064              REFPARTIE(NBUSER,3)=/*1 PAGE SUPPLEMENTAIRE TOUTES LES 50 INSTRUCTIO
                  S*/ (N/50)+1;
1065              REFPARTIE(NBUSER,4)=/*1 PAGE SUPPLEMENTAIRE TOUTES LES 11 INSTRUCTIO
                  S*/ (N/11)+1;
1066              INTCYC(NBUSER)=100*N;
1067              REENTRANT(NBUSER,3),REENTRANT(NBUSER,4)='1'B;
1068              GOTO TEST;
```

```
1069            NU : /* NUMEROTER*/
                  N=/* NB DE LIGNES A NUMEROTER*/
                  SUBSTR(GRP,3);
1070              REFPARTIE(NBUSER,3)=(N/50)+1;/* 1 PAGE SUPPLEMENTAIRE TOUTES LES 50
                  LIGNES*/
1071              GOTO TEST;
1072            EX :/*EXECUTER*/
                  N=SUBSTR(GRP,5);/*NB DE LIGNES A EXECUTER*/
1073              REFPARTIE(NBUSER,3)=1; REENTRANT(NBUSER,3)='1'B;
1075              IF SUBSTR(GRP,4,1)='N' THEN DO     ;
1077              LIGNE(NBUSER)=LIGNE(NBUSER)+N;
1078              REFPARTIE(NBUSER,5)=LIGNE(NBUSER)/11+1;
1079              END;
1080              ELSE;REFPARTIE(NBUSER,5)=N/11+1;
1082              REENTRANT(NBUSER,5)='1'B;
1083              IF SUBSTR(GRP,3,1)='F' THEN DO;
1085                  REFPARTIE(NBUSER,6)=1; REENTRANT(NBUSER,6)='1'B;
1087                  REFSEG(NBUSER)=1; REFOPEN(NBUSER)=1; REFILE(NBUSER)=N;
1090              END;
1091              INTCYC(NBUSER)=100*N; GOTO TEST;
1093            ST : /* STORE OU DISPLAY*/
                  REFPARTIE(NBUSER,3),REFPARTIE(NBUSER,4),REFPARTIE(NBUSER,5)=1;
1094              REENTRANT(NBUSER,9)='1'B;
1095              INTCYC(NBUSER)=25;
1096              GOTO TEST;
1097            CA : /* CATALOGUER*/
1098              INTCYC(NBUSER)=5; GOTO FINGENERE;
1099            S : /* SAUVEGARDER*/
                  N=SUBSTR(GRP,4);/*NB DE LIGNES A SAUVEGARDER*/
1100              REFPARTIE(NBUSER,4)=(N/15)+1; REFPARTIE(NBUSER,3)=(N/15)+1;
1102              REENTRANT(NBUSER,4),REENTRANT(NBUSER,3)='1'B;
1103              REFOPEN(NBUSER)=1; REFILE(NBUSER)=N;
1105              IF SUBSTR(GRP,3,1)='D' THEN INTCYC(NBUSER)=60+180*N;
1107              ELSE INTCYC(NBUSER)=180*N;
1108              GOTO TEST;
1109            D :/* DEFINIR*/
1110              INTCYC(NBUSER)=60; GOTO TEST;
1111            DU : /* DUPLIQUER*/
                  N=SUBSTR(GRP,3);
1112              REFOPEN(NBUSER)=2; REFILE(NBUSER)=2*N; INTCYC(NBUSER)=60;
1115              GOTO TEST;
1116            EC : /* ECRASER*/
1117              REFSEG(NBUSER)=0; INTCYC(NBUSER)=4;
1118            TEST : DO J=1 TO 9;
1119             REFPARTIE(NBUSER,J)=REFPARTIE(NBUSER,J)/4+1;
1120                  IF REFPARTIE(NBUSER,J)=0 THEN IF REFPARTIE(NBUSER,J)
```

```
1123                    REFPARTIE(NBUSER,J)=REFPARTIE(NBUSER,J)-TABLEPAGE(NBUSER,J
                        ,1);
1124                    REFPAGE(NBUSER)=REFPAGE(NBUSER)+REFPARTIE(NBUSER,J);
1125                    IF OK THEN DO;NXTPARTIE(NBUSER)=J;OK='0'B; END;
1130                END;
1131              END;
1132              TIMEPAG(NBUSER)=0;
1133              IF REFSEG(NBUSER)¬=0 THEN DO;
1135                  TIMESEG(NBUSER)=REFPAGE(NBUSER);
1136                  TIMEFIL(NBUSER)=TIMESEG(NBUSER)+300;
1137              END;
1138            FINGENERE : NSESION(NBUSER)=NSESION(NBUSER)+1;
1139            END GENERE;
                /*****************************************************************/
                /*                                                               */
                /*****************************************************************/
1140              END;
1141            FINFIN :
               END SIMUL;
```

A P P E N D I X C

LE PROGRAMME DE SIMULATION EN TERMES DE MULTITASKING

---

1 -INTRODUCTION

1.1 Notion de tâche

1.2 Création de tâche

1.3 Synchronisation et coordination des tâches

1.4 Conclusion

2 -LE PROGRAMME DE SIMULATION

Le programme PL/I peut faire usage des possibilités de multitasking du système (exécution simultanée d'un certain nombre d'opérations), ceci à l'aide de la ressource du langage PL/I appelée Multitasking.[Ref 29],[Ref 30] .

1 -LES ELEMENTS ESSENTIELS DU MULTITASKING ET LEUR UTILISATION POSSIBLE DANS UN PROGRAMME DE SIMULATION.

1.1. Notion de tâche.

L'exécution d'un programme PL/I constitue une ou plusieurs tâches chacune devant être identifiée par un nom de tâche différent. Une tache est une notion dynamique, son existence est liée à l'exécution du programme. En parlant de multitasking, on devra distinguer le programme PL/I de son exécution. Le même ensemble d'instructions peut être exécuté plusieurs fois dans des tâches différentes.

Un programme PL/I qui n'utilise pas le multitasking constitue une seule tâche. Si une procédure en appelle une autre, le contrôle est passé à la procédure appelée et l'exécution de la procédure appelante est suspendue jusqu'à ce que la procédure appelée rende le contrôle. Ce type d'opération séquentielle est dit synchrone.

Avec le multitasking, la procédure appelante ne prend pas le contrôle au profit de la procédure appelée. Au lieu de cela un contrôle parallèle est établi si bien que les deux procédures peuvent être exécutées en parallèle. Ce type d'opération est dit asynchrone. Chaque tâche peut appeler autant de tâches qu'elle le souhaite (fig.1).

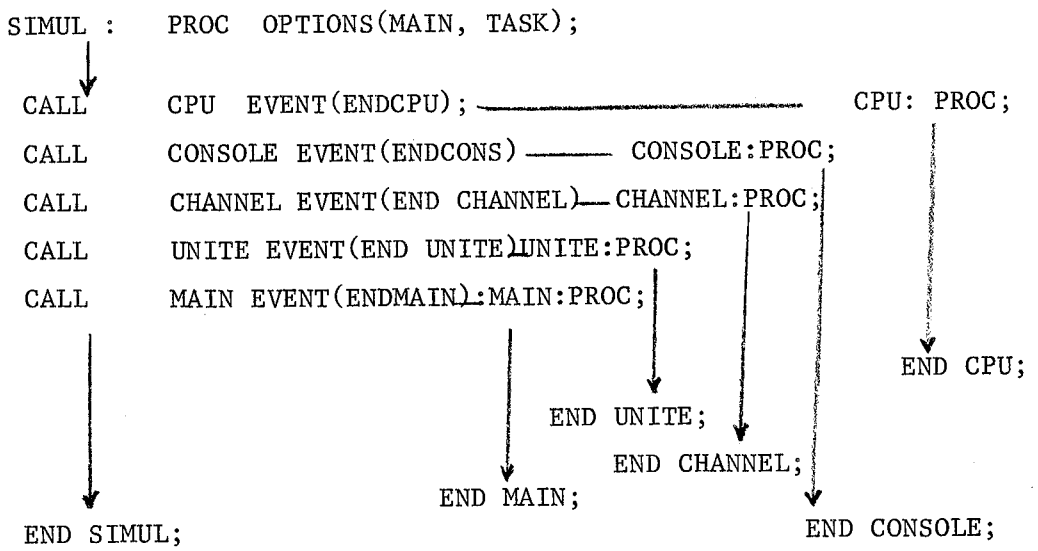


Figure 1 : Déroulement asynchrone de SIMUL, UNITE, CHANNEL, CONSOLE, CPU.
(SIMUL demande l'exécution en parallèle de toutes les tâches qui décrivent les ressources du système).

Remarque : Lorsque plusieurs procédures sont exécutées comme des tâches asynchrones, les instructions ne sont pas nécessairement exécutées simultanément par différentes tâches. Cette éventualité dépend de l'état des ressources du système dans lequel ces tâches se déroulent. Si par exemple nous disposions d'un système multiprocessor dans lequel une unité centrale déroulerait la tâche qui représente le CPU et une autre unité centrale qui exécuterait celles relatives aux entrées-sorties, nous aurions un programme de simulation extrêmement efficace. (une simultanéité vraie dans la représentation des entrées-sorties et du travail en unité centrale). Puisque ce n'est pas le cas, à n'importe quel instant le système peut être amené à choisir l'activité suivante parmi différentes tâches. Chaque tâche a une priorité qui dirige ce processus de sélection. A la limite, le programmeur, s'il le désire , contrôle la priorité des tâches, sinon le système donne auomatiquement une valeur à cette priorité. Dans notre programme de simulation toutes les tâches créées ont même priorité.

1.2. Création de tâches.

Le programmeur crée une tâche en utilisant dans l'instruction classique d'appel CALL, une ou plusieurs des options du Multitasking.

Les options qui nous intéressent dans le programme de simulation (parmi d'autres proposées par le multitasking) sont les suivantes :

TASK et EVENT

- TASK(nom de tâche) sert à donner un nom à une tâche : c'est une possibilité qui ne nous concerne pas. Nous utilisons TASK uniquement dans la déclaration de la procédure principale pour indiquer au système que nous désirons un déroulement asynchrone des tâches créées au cours de la simulation :

```
SIMUL : PROC      OPTIONS(MAIN, TASK);
```

- EVENT(nom d'évènement) permet d'associer un nom d'évènement à la fin d'exécution de la tâche créée par une instruction CALL. Ce nom d'évènement peut être indicé :

```
CALL        CPU          EVENT(ENDCPU);

CALL        CHANNEL(ICANAL) EVENT(ENDCHANNEL(ICANAL));
END;
```

A la tâche CPU est attachée l'évènement de fin d'exécution de CPU, ENDCPU;

A chaque tâche CHANNEL(il y en a NCANAL NCANAL = nombre de canaux dans la Partition) est attaché un évènement différent de fin d'exécution.

Une variable évènement a deux valeurs distinctes. Une valeur de fin qui indique que la tâche est terminée ou pas, et une valeur d'état qui indique si la tâche a été terminée normalement ou pas (La seconde valeur n'est pas utilisée dans le programme de simulation). A l'exécution d'un CALL, la valeur de fin de la variable évènement est posée à 0. A la fin d'exécution de la tâche créée, la valeur de fin est posée à 1.

1.3 Synchronisation et coordination des tâches.

1.3.1. Principe. Il se peut qu'une tâche soit exécutée indépendamment des autres pendant quelques instants, puis devienne dépendante d'autres tâches (par exemple, une tâche attend le résultat d'une autre tâche).

Pour permettre ceci, on a prévu qu'une tâche puisse, avant de continuer, attendre la fin d'une opération d'une autre tâche. Ce processus est dit synchronisation des tâches. L'information sur l'état d'une opération peut être donnée par une variable évènement référencée par un nom de variable évènement. En indiquant un nom de variable évènement dans une instruction WAIT, le programmeur peut, avant de continuer, faire attendre la tâche pour que l'opération associée se termine.

Exemple d'application : L'option EVENT rend possible le déroulement asynchrone d'une Entrée-sortie et de la tâche qui l'a initialisée. Après cette initialisation la tâche peut attendre la fin de cette opération : une tâche donnant un message à l'opérateur, au lieu d'attendre la réponse, continue un travail et s'arrête plus tard pour le traitement de la réponse.

1.3.2. Application au programme de simulation.

Nous ne reviendrons pas sur les traitements qu'effectuent les diverses tâches-ressources.

Ce programme moins élaboré que celui qui n'utilise pas le multitasking ne comprend pas de ressource CANAL.

1.3.2.1. Phase d'initialisation.

Le programme de simulation SIMUL crée toutes les tâches qui représentent une ressource de la partition conversationnelle. Ces tâches se déroulent en "parallèle" (simultanéité apparente) et toutes, sauf CONSOLE, tombent en attente.

1.3.2.2. CONSOLE.

Pour cette tâche, une variable évènement essentielle ; celle qui indique que les consoles sont bloquées. CONSOLE tombe en attente si toutes les consoles sont bloquées. Dès qu'un utilisateur est autorisé à lancer une nouvelle commande, CONSOLE effectue les traitements nécessaires à l'entrée de cet utilisateur dans le système et "bloque" la oonsole de cet utilisateur. La tâche CONSOLE se termine quand toutes les sessions sont cloturées. A ce moment là, elle envoie le message de clôture de la Partition conversationnelle à la tâche superviseur, MAIN.

1.3.2.3. MAIN

La première instruction de MAIN correspond à l'attente de l'arrivée d'un évènement (traduit par une variable évènement) en provenance de la tâche CONSOLE, CPU, SWAPPING ou DISQUE. Selon l'origine de cet évènement, MAIN libère la ressource dont l'utilisation est demandée. Cependant si cet évènement est le message de clôture de la Partition conversationnelle, MAIN le transmet aux différentes tâches ressources qui à leur tout se terminent.

1.3.2.4. CPU

Cette tâche attend l'un des trois évènements suivants :

- l'arrivée du message de clôture de la partition,
- l'arrivée du message d'exécution de l'Algorithme de pagination,
- le message d'exécution d'un programme utilisateur.

L'arrivée d'un de ces évènements provoque l'évènement : "CPU occupée" que traduit, une variable EVENT. CPU travaille pour le compte d'un utilisateur jusqu'à l'arrivée de la prochaine entrée-sortie qu'elle signale à la tâche MAIN : déblocage d'une console s'il s'agit de la fin d'une intéraction, activation de la tâche SWAPPING pour une demande de page, activation de la ressource DISQUE pour le chargement d'un segment ou la manipulation d'un enregistrement de fichier.

Une fois les divers traitements assumés, CPU retombe en attente sur les 3 mêmes évènements.

1.3.2.5. SWAPPING

SWAPPING attend en premier lieu :

- le message de clôture ou
- le signal d'un travail pour le compte d'un utilisateur.

Dans le second cas, après avoir placé l'utilisateur dans la file des demandes service-disque et après avoir signalé cette demande, SWAPPING tombe en attente jusqu'à ce que la transmission de la page ait été réalisée. Si besoin est, elle signale à la tâche MAIN que l'Algorithme de pagination doit être exécuté; MAIN transmettra le message à la tâche CPU, dès que cette ressource sera libre. Pendant ce temps, SWAPPING attend le message de MAIN qui lui signalera la fin d'exécution de l'Algorithme. Ce traitement est suivi d'une reitération sur la demande d'entrée-sortie page. Cette seconde transmission une fois réalisée, SWAPPING se remet en attente de l'un des deux évènements qu'elle attendait initialement.

1.3.2.6. DISQUE

Cette tâche attend l'un des trois évènements :

- le premier en provenance de MAIN se rapportant à la clôture de la partition,
- le second provoqué par SWAPPING pour le transfert d'une page,
- le dernier en provenance de CPU pour le chargement d'un segment ou la manipulation d'un enregistrement de fichier.

DISQUE après avoir effectué les traitements qu'on lui demandait, envoie par l'intermédiaire de MAIN un signal-retour de fin de transmission à la ressource qui l'avait sollicité, puis retombe en attente.

1.4. Conclusion

Dans ce qui précède, nous n'avons pas parlé des différentes contraintes imposées par l'utilisation du multitasking tels que le partage d'un fichier entre plusieurs tâches, l'interdiction de créer une tâche encore active etc.... ni de la fin d'exécution d'une tâche. Ce sont des détails qui n'entrent pas dans la philosophie qu'il faut pouvoir tirer du Multitasking et l'idée essentielle est la suivante : Le Multitasking, en réduisant les temps d'attente permet une utilisation plus efficace de l'Unité Centrale et des canaux d'entrée-sortie. Ceci ne veut pas dire nécessairement qu'un programme asynchrone sera plus efficace que le programme synchrone correspondant équivalent (qui d'ailleurs est plus facile à écrire). En fait, ceci dépend du degré possible de simultanéité entre les opérations de l'Unité Centrale et des E/S. Si la simultaneité est faible, le multitasking peut être le moins efficace des méthodes en raison des charges supplémentaires imposées au système. En ce qui concerne la simulation et notamment la représentation des évènements parallèles, le multitasking est un outil précieux et intéressant dans la mesure où le système d'exploitation dont on dispose est multiprocessor. Dans un tel système, on déroulerait en simultanéité vraie les tâches qui représentent les ressources-travail Unité Centrale d'une part et d'autre part les tâches qui décrivent les ressources d'entrée-sortie.

2 -SIMUL : LE PROGRAMME DE SIMULATION UTILISANT LE MULTITASKING

On pourra se reporter utilement à l'Appendix B pour la définition des variables employées dans le programme.

```
'SUBSCRIPTRANGE):

     (SUBSCRIPTRANGE):
   SIMUL  : PROC OPTIONS(MAIN,TASK);
    DCL (ENDCLK1,ENDCLK3,ENDCLK2) EVENT;
    DCL (I,J) BIN FIXED(31);
    DCL COUNT BIN FIXED(31);
    DCL (QCPUP(3,4,17),QDISKP(8,4,17)) BIN FIXED(31);
   DCL (A,B,C,D,E ,AP,BP,CP,DP,EP) CHAR(30) VAR;
    DCL NUSER BIN FIXED(31);
    DCL  ISESION(8) BIN FIXED(31);
    DCL LSESION BIN FIXED(31);
    GET LIST(NUSER);
   LSESION=0;
    DO  I=1 TO NUSER;
          GET LIST (ISESION(I));
          LSESION=MAX(LSESION,ISESION(I));
    END;
    /* ISESION(I) CONTIENT LA LONGUEUR DE LA SESSION DE L'UTILISATEUR
    BEGIN;
DCL CLOCK ENTRY(BIN FIXED(31));
   DCL CLOCK1 ENTRY(BIN FIXED(31));
   DCL HEURE ENTRY(CHAR(9)) RETURNS(BIN FIXED(31));
       DCL CHECKU ENTRY ;
   DCL DELETE ENTRY((NUSER,4) BIN FIXED(31),BIN FIXED(31),
     BIN FIXED(31));
DCL GENERE ENTRY(BIN FIXED(31),BIN FIXED(31));
DCL NXTDISK ENTRY ;
       DCL NEXTCPU ENTRY ;
       DCL NXTSWAP ENTRY ;
DCL(ALGOGO ,ALGOENC,ALGORIT) EVENT ;
DCL(CPUFREE,CPUGO,CPUBSY,CPUSTOP) EVENT ;
DCL(ENDTERM,ENDTERM1,ENDTERM2 ENDTERM3) EVENT;
   DCL GO(NUSER) EVENT;
DCL(  ,GATE1,GATE2,GATE3,GATE4,GATE5,GATE6,GATE7) EVENT ;
   DCL GATE8 EVENT;
   DCL NEEDCPU EVENT;
  DCL OK(NUSER) EVENT;
   DCL RDY(NUSER) EVENT;
DCL(SWAPGO,SWAPDON) EVENT ;
DCL(XMIT1,XMIT2,XMRDY1,XMRDY2) EVENT ;
DCL(ASKSWAP,ASKSEG) BIT(1) INIT('0'B), ARGUMEN BIN FIXED(31);
DCL CHANGE BIT(1),(CYCDON,CPUUSE,CPUWAIT(NUSER,4)) BIN FIXED(31);
   DCL CPUSR BIN FIXED(31);
DCL(DSKUSE1,DSKUSE2,DSKUSE3) BIN FIXED(31)   INIT(0) ;
DCL DSKUSR INIT(0) BIN FIXED(31),
  (DUREE,DSKWAIT(NUSER,4)) BIN FIXED(31);
   DCL EJECT BIN FIXED(31);
DCL(FORSWAP(NUSER,4),FORSEG(NUSER,4))  BIN FIXED(31);
```

```
SUBSCRIPTRANGE):

      DCL(H1,H2,H3,H4) CHAR(9) ;
        DCL (HHORA,HORA) BIN FIXED(31);
       /*CONVENTION POUR IMAX:1=CPU,2=SWAP,3=DISQUE*/
      DCL IMAX(4,4) BIN FIXED INIT((16)0),(IQCPU(4),IQDISK(4),
          IQSWAP(4),IFOR(4),JFOR(4),ITSEG(NUSER),ITPAGE(NUSER),
          ITFILE(NUSER),IREFSEQ(NUSER))                BIN FIXED(31),
          INTCYC(NUSER) BIN FIXED(31);
        DCL MINUS BIN FIXED(31);
        DCL                          (NXTAPE(NUSER),NXTIME) BIN FIXED(31);
        DCL NEEDUSR BIT(1) INIT('0'B);
        DCL NSESION(NUSER) BIN FIXED(31);
        DCL OUT(NUSER) BIN FIXED(31);
      DCL PRTY(NUSER) BIN FIXED(31);
      DCL(QCPU(NUSER,4),QDISK(NUSER,4),QSWAP(NUSER,4))
          BINARY FIXED(31);
      DCL(REFILE(NUSER),REFSEG(NUSER),REFPAGE(NUSER),REFSEQ(NUSER))
          BIN FIXED(31);
        DCL SWAPUSR BIN FIXED(31) INIT(0);
        DCL                          (SWAPUSE,START,STARTT,STARTEX)
        INIT(0) BIN FIXED(31)  ,((SWPWAIT,STWTCPU,STWTDSK,STWTSWP)
        (NUSER,4)) BIN FIXED(31)  ,SSGROUP(NUSER) BIN FIXED(31);
            DCL 1 SESSION(LSESION,NUSER),
                  2 TYPE BIN FIXED (31),
                  2 SSGP BIN FIXED (31),
                  2 TIM   BIN FIXED (31);
        DCL TEST BIT(1) INIT('0'B);
        DCL THEURE BIN FIXED(31);
        DCL (TTTREPLY(NUSER),TTREPLY(NUSER)) BIN FIXED(31);
      DCL(TTIME,TTTIME,TIMER,TTIMER,TTTIMER) CHAR(9),
          (TREPLY(NUSER),TIMEPAG(NUSER),TIMEFIL(NUSER),TIMESEG(
          NUSER),TEM,TEM1,TCPU(NUSER),TSWAP(NUSER),TDISK(NUSER),TEXEC)
          BIN FIXED(31);
      DCL USER BIN FIXED(31)   INIT(0) ;
        DO I=1 TO NUSER;
            DO J=1 TO ISESION(I);
                GET LIST(SESSION.TYPE(J,I));
                GET LIST (SESSION.SSGP(J,I));
                GET LIST (SESSION.TIM  (J,I));
            END;
        END;
        COMPLETION(ALGOGO ),COMPLETION(ALGOEND),COMPLETION(ALGORIT)='0'B ;
        COMPLETION(CPUFREE)='1'B ;
        COMPLETION(CPUGO),COMPLETION(CPUBSY),COMPLETION(CPUSTOP)='0'B ;
        COMPLETION(ENDTERM)='0'B ;
        COMPLETION(ENDTERM1)='0'B;
        COMPLETION(ENDTERM2)='0'B;
        COMPLETION(ENDTERM3)='0'B;
```

```
      COMPLETION(GO)='0'B ;
      COMPLETION(GATE1),COMPLETION(GATE2),COMPLETION(GATE3),
      COMPLETION(GATE4),COMPLETION(GATE5),COMPLETION(GATE6),
      COMPLETION(GATE7)='0'B ;
         COMPLETION(GATE8)='0'B;
      COMPLETION(NEEDCPU)='0'B;
      COMPLETION(RDY)='1'B ;
      COMPLETION(SWAPDO),COMPLETION(SWAPDON)='0'B ;
      COMPLETION(XMIT1),COMPLETION(XMIT2),COMPLETION(XMRDY1),
      COMPLETION(XMRDY2)='0'B ;
      FORSWAP,FORSEG=0;
      EJECT=0;
      TCPU,TSWAP,TDISK=0;
      TEXEC=0;
      CPUUSE=0 ;
      IQCPU,IQDISK,IQSWAP,IFOR,JFOR=1 ;
      NSESION=1;
      NXTAPE=0 ;
      OUT=0;
      HORA,HHORA=0;
      TREPLY=0;
      TTREPLY=0;
      TTTREPLY=0;
      CALL CONSOLE EVENT(ENDCONS) ;
      CALL CPU EVENT(ENDCPU);
      CALL DISQUE EVENT(ENDDISK);
      CALL SWAPING EVENT(ENDSWP);
      CALL MAIN EVENT(ENDMAIN);
      WAIT (ENDMAIN,ENDCPU,ENDCONS,ENDSWP,ENDDISK);
     /*****************************************************************
     /*                                                               *
     /*****************************************************************
    MAIN: PROC ;
      DCL (I,J,K) BIN FIXED(31);
          /* MAIN SUPERVISE L'EMSEMBLE DU SYSTEME DE SIMULATION.
          - IL RECOIT LE CONTROLE DES CONSOLES,AUTORISE ET INTERDIT
          LEUR UTILISATION ;
          - IL DETERMINE LE PROCHAIN UTILISATEUR A ACTIVER
          - IL CONTROLE L'APPARITION DES INTERRUPTIONS(TOUS LES TYPES
    D'E/S DISQUE ET FIN D'EXECUTION DES PROGRAMMES DES UTILISATEURS)
          - SUIVANT LE TYPE D'INTERRUPTION,DONNE LE CONTROLE AU MODULE
          DEMANDE--- */
          /* LES INITIALISATIONS ET LES DECLARATIONS ONT ETE FAITES DANS
          LE PROGRAMME PRINCIPAL */
          /* ATTENDRE UN ORDRE UTILISATEUR,OU BIEN LE SIGNAL DE FIN
          D'OPERATION DE SWAPPING,OU BIEN LE SIGNAL DE FIN D'ENTREE-SORTI
          OU BIEN L'EVENEMENT DE FIN DE SIMULATION */
```

```
(SUBSCRIPTRANGE):

        IA,JA=0;
ETIO  : IA=IA+1;
 A= 'N ATN AVANT LE WAIT 1 ' ;
        WAIT(GO,SWAPDON,XMRDY2,ENDTERM,NEEDCPU) (1);
        IF COMPLETION(ENDTERM) THEN DO;
                        /*FIN DE SIMULATION*/
                        COMPLETION(ENDTERM)='1'B;
                        GOTO FIN1;
        END;
        IF COMPLETION(SWAPDON) THEN DO ; COMPLETION(SWAPDON)= '0'B;
        GOTO ETI1 ; END ;
        IF COMPLETION(NEEDCPU) THEN DO;
                        NEEDUSR='0'B;
                        GOTO ETI3;
        END;
        IF COMPLETION(XMRDY2) THEN COMPLETION(XMRDY2)='0'B;
        /* DETERMINER LE PROCHAIN UTILISATEUR A ACTIVER */
ETI1  : JA=JA+1;
        CALL NEXTCPU;
        IF USER=0 THEN IF COMPLETION(NEEDCPU) THEN GOTO ETI3;
        ELSE DO;
                        NEEDUSR='1'B;
                        GOTO ETIO;
        END;
        /* ATTENDRE QUE L'UNITE CENTRALE SOIT LIBRE AVANT DE LA RELANCER
        POUR LE COMPTE DU NOUVEL UTILISATEUR */
A='MAIN AVANT LE WAIT 2';
        WAIT(CPUFREE) ;
        COMPLETION(CPUGO)= '1'B;
A='MAIN AVANT LE WAIT 3';
        WAIT(CPUBSY) ;
        /* SYNCHRONISATION AVEC ''CPU'' */
        COMPLETION(GATE4)= '1'B;
 A='MAI NWAIT 4';
        WAIT(GATE5) ;
        COMPLETION(GATE5)= '0'B;
        J=PRTY(USER) ;
        /* ATTENDRE LA PROCHAINE INTERRUPTION(CPU,PAGINATION,
        LOAD-SEGMENT,FICHIER) */
 IF REFILE(USER)=0 &REFSEG(USER)=0&REFPAGE(USER)=0 THEN DO;
        /* L'EGALITE A 0 DES NOMBRES DE REFERENCES FICHIERS,PAGES ET
        SEGMENTS DISTINCTS MARQUE LA FIN D'UNE INTERACTION ; MESURES A
        PRENDRE: LAISSER LE CPU TRAVAILLER POUR L'UTILISATEUR PENDANT LE
        TEMPS PREVU,PUIS DEBLOQUER LA CONSOLE DE L'UTILISATEUR */
A='MAIN WAIT 5';
                        WAIT(CPUFREE);
                                COMPLETION(CPUSTOP)= '1'B;
```

```
      A='MAIN WAIT 6';
                                    WAIT(GATE6) ;
                                    COMPLETION(GATE6)= '0'B;
                                    COMPLETION(OK(USER))='1'B;
                                    COMPLETION(RDY(USER))='1'B;
                                    TTIME=TIME ;
                                    JAP=JA;JCP=JC;JDP=JD;
                                    AP=A;BP=B;CP=C;DP=D;EP=E;
                                    IAP=IA;IBP=IB;ICP=IC;IDP=ID;IEP=IE;
                                    THEURE=HEURE(TTIME);
                                    I=PRTY(USER);
                                    TREPLY(USER)=CPUWAIT(USER,I)+
                                    SWPWAIT(USER,I)+DSKWAIT(USER,I)+
                                    TCPU(USER)+TDISK(USER);
                                    /* ECRITURE DES STATISTIQUES */
                                    GOTO STATISTIC;
          END ;
          IF REFPAGE(USER)¬=0 & REFSEG(USER)¬=0 &REFILE(USER)¬=0 THEN DO;
                        MINUS=MIN(TIMEPAG(USER),TIMESEG(USER),TIMEFIL(USER)
                        ;
                        IF TIMEPAG(USER)=MINUS THEN GOTO LABEL1;
                        IF TIMESEG(USER)=MINUS THEN GOTO LABEL3;
                        GOTO LABEL2;
            END;
          IF REFPAGE(USER)¬=0 & REFSEG(USER)¬=0 THEN
                        IF TIMEPAG(USER)<TIMESEG(USER) THEN GOTO LABEL1;
                        ELSE GOTO LABEL3;
          IF REFPAGE(USER)¬=0 & REFILE(USER)¬=0 THEN
                        IF TIMEPAG(USER)<TIMEFIL(USER) THEN GOTO LABEL1;
                        ELSE GOTO LABEL2;
          IF REFSEG(USER)¬=0 & REFILE(USER)¬=0 THEN
                        IF TIMESEG(USER)<TIMEFIL(USER) THEN GOTO LABEL3;
                        ELSE GOTO LABEL2;
          IF REFPAGE(USER)¬=0 THEN GOTO LABEL1;
          IF REFSEG(USER) THEN GOTO LABEL3;
          GOTO LABEL2;
     LABEL1 : CALL CLOCK(TIMEPAG(USER)) EVENT(ENDCLK1);
          WAIT(ENDCLK1);
          ARGUMEN=TIMEPAG(USER);
          DO K=1 TO 4;
            DO I=1 TO TOCPU(K)-1;
                 T=TTREPLY(I)-TTREPLY(USER);
                 ELSE T=0;
                 IF T<=0 THEN T=ARGUMEN;ELSE IF T<ARGUMEN THEN T=ARGUMEN-
                 CPUWAIT(OCPU(I,K),K)=CPUWAIT(OCPU(I,K),K)+T;
                 TTREPLY(I)=TTREPLY(I)+T;
            END;
            END;
          COMPLETION(CPUSTOP1)='1'B;
                                    /* ON DEMANDE DU SWAPPING */
                                    /* METTRE L'UTILISATEUR DANS LA QUEUE DES
```

```
SUBSCRIPTRANGE):
                                        DEMANDES DE SWAPPING CORRESPONDANT A SA
                                        PRIORITE */
                                        QSWAP(IQSWAP(J),J)=USER ;
                                        TTIME=TIME ;
                                        IMAX(2,J)=MAX(IQSWAP(J),IMAX(2,J)) ;
                                        IQSWAP(J)=IQSWAP(J)+1 ;
                                        /* ACTIVER LE MODULE DE SWAPPING */
                                        COMPLETION(SWAPGO)='1'B ;
                                        IF COMPLETION(NEEDCPU) THEN GOTO ETI3;
                                        ELSE GOTO ETI1 ;
   LABEL2 : CALL CLOCK(TIMEFIL(USER)) EVENT(ENDCLK2);
            ARGUMEN=TIMEFIL(USER);
            DO K=1 TO 4;
              DO I=1 TO IQCPU(K)-1;
                 T=TTREPLY(I)-TTREPLY(USER);
                 ELSE T=0;
                 IF T<=0 THEN T=ARGUMEN;ELSE IF T<ARGUMEN THEN T=ARGUMEN-T;
                 CPUWAIT(QCPU(I,K),K)=CPUWAIT(QCPU(I,K),K)+T;
                 TTREPLY(I)=TTREPLY(I)+T;
            END;
            END;
          COMPLETION(CPUSTOP2)='1'B;
                                        /* ON DEMANDE E/S FICHIER */
                                        /* METTRE L'UTILISATEUR DANS LA QUEUE
                                        DES DEMANDES E/S DISQUE */
                                        GOTO ETI2 ;
   LABEL3 : CALL CLOCK(TIMESEG(USER)) EVENT(ENDCLK3);
            /* ON DEMANDE UN LOAD-SEGMENT */
            WAIT(ENDCLK3);
            ARGUMEN=TIMESEG(USER);
            DO K=1 TO 4;
              DO I=1 TO IQCPU(K)-1;
                 T=TTREPLY(I)-TTREPLY(USER);
                 ELSE T=0;
                 IF T<=0 THEN T=ARGUMEN;ELSE IF T<ARGUMEN THEN T=ARGUMEN-
                 CPUWAIT(QCPU(I,K),K)=CPUWAIT(QCPU(I,K),K)+T;
                 TTREPLY(I)=TTREPLY(I)+T;
              END;
              END;
            COMPLETION(CPUSTOP1)='1'B;
            WAIT(CPUSTOP1);
          FORSEG(JFOR(J),J)=USER;
          JFOR(J)=JFOR(J)+1;
          /* METTRE L'UTILISATEUR DANS LA QUEUE DES DEMANDES E/S
          DISQUE */
   ETI2:  QDISK(IQDISK(J),J)=USER ;
          TTIME=TIME;
          IMAX(3,J)=MAX(IQDISK(J),IMAX(3,J)) ;
          IQDISK(J)=IQDISK(J)+1;
          /* ACTIVER LE MODULE DISQUE */
          COMPLETION(XMIT2)='1'B ;
          IF COMPLETION(NEEDCPU) THEN GOTO ETI3;
          GOTO ETI1 ;
   ETI3:  /* LE MODULE ''SWAPING'' DEMANDE L'EXECUTION DE L'ALGORITHME DE
                    PAGINATION */
```

```
(SUBSCRIPTRANGE):

                    WAIT(CPUFREE) ;
                    COMPLETION(ALGOGO )='1'B ;
   A='MAIN WAIT 7';
                    WAIT(ALGOEND) ;
                    COMPLETION(ALGOEND)='0'B ;
                    COMPLETION(NEEDCPU)='0'B;
                    COMPLETION(ALGORIT)='1'B ;
                    GOTO ETIL ;      *
STATISTIC :
   I=PRTY(USER);
   PUT EDIT ((130)'*') (SKIP,A);
   PUT EDIT('UTILISATEUR NO ',USER)(SKIP(2),A,F(1,0));
   PUT EDIT ((130)'*') (SKIP,A);
   PUT EDIT('HEURE DU READY=',THEURE)(SKIP,A,F(8,0));
   PUT EDIT('CPUWAIT= ',CPUWAIT(USER,I))(SKIP(2),A,F(8,0));
   PUT EDIT ('DSKWAIT= ',DSKWAIT(USER,I),'SWPWAIT= ',SWPWAIT(USER,I),
   'TSWAP= ',TSWAP(USER),'TDISK= ',TDISK(USER),'TCPU= ',TCPU(USER),
   'TREPLY=',TREPLY(USER))
   ((6) (SKIP(1),A,F(8,0)));
   PUT SKIP;
   PUT EDIT('OUT = ') (A);
   DO I=1 TO IOUT-1;
         PUT EDIT (OUT(I)) (F(8,0));
   END;
   PUT EDIT('QUEUE DISK AU MOMENT DU READY')(SKIP,A);
   DO J=1 TO 4;
         PUT SKIP;
         DO I=1 TO IQDISK(J)-1;
                  PUT EDIT(QDISK(I,J)) (F(8,0));
         END;
   END;
   PUT EDIT('QUEUE CPU AU MOMENT DU READY ')(SKIP,A);
   DO J=1 TO 4;
         PUT SKIP;
         DO I=1 TO IQCPU(J)-1;
                  PUT EDIT( QCPU(I,J)) (X(1),F(8,0));
         END;
   END;
   PUT EDIT('QUEUE SWAP AU MOMENT DU READY')(SKIP,A);
   DO J=1 TO 4;
         PUT SKIP;
         DO I=1 TO IQSWAP(J)-1;
                  PUT EDIT(QSWAP(I,J)) (F(8,0));
         END;
   END;
   PUT EDIT('****ARRET MAIN',A,IA,JA) (SKIP,A,A,F(8,0),F(8,0));
   PUT EDIT('AP,IAP,JAP',AP,IAP,JAP)(SKIP,A,A,F(8,0),F(8,0));
```

```
SUBSCRIPTRANGE):

   PUT EDIT('****ARRET CPU',B,IB)(SKIP,A,A,F(8,0));
   PUT EDIT('BP,IBP',BP,IBP)(SKIP,A,A,F(8,0));
   PUT EDIT('****ARRET CONSOL',E,IE)(SKIP,A,A,F(8,0));
   PUT EDIT('EP,IEP',EP,IEP) (SKIP,A,A,F(8,0));
   PUT EDIT('****ARRET SWAP',D,ID,JD)(SKIP,A,A,F(8,0),F(8,0));
   PUT EDIT('DP,IDP,JDP',DP,IDP,JDP)(SKIP,A,A,F(8,0),F(8,0));
   PUT EDIT('****ARRET DISQUE',C,IC,JC)(SKIP,A,A,F(8,0),F(8,0));
   PUT EDIT('CP,ICP,JCP',CP,ICP,JCP)(SKIP,A,A,F(8,0),F(8,0));
       IF COMPLETION(ENDTERM) THEN DO;
                   COMPLETION(ENDTERM1)='1'B;
                   GOTO FIN1;
       END;
   GOTO ETI1;
FIN1: /* IMPRESSION DE STATISTIQUES FIN DE SIMULATION */
   PUT EDIT('FIN DE SIMULATION') (SKIP,A);
   PUT EDIT ((130)'*') (SKIP,A);
   PUT EDIT('TEXEC= ',TEXEC,'SWAPUSE= ',SWAPUSE,'DSKUSE1= ',DSKUSE1,
   'DSKUSE2= ',DSKUSE2,'DSKUSE3= ',DSKUSE3) ((5) (SKIP,A,F(8,0)));
   PUT EDIT('CPUUSE= ',CPUUSE) (SKIP,A,F(8,0));
   PUT EDIT('IMAX= ')(SKIP,A);
   DO I=1 TO 3;
         PUT EDIT(IMAX(I,*)) (SKIP,(4)(F(8,0)));
   END;
END MAIN ;
 /*****************************************************************/
 /*                                                               */
 /*****************************************************************/
CPU: PROC ;
   DCL (I,J,K) BIN FIXED(31);
/* ''CPJ'' SIMULE L'UNITE CENTRALE DEROULANT UN PROGRAMME UTILISATEUR
OU EXECUTANT L'ALGORITHME DE SWAPPING */
/* INITIALISATIONS ET DECLARATIONS ONT ETE FAITES DANS LE
PROGRAMME PRINCIPAL */
   DCL KI BIN FIXED(31);
/* ATTENDRE LE CONTROLE */
       IB=0;
ETI13 : IB=IB+1;
B='CPU WAIT 1';
 WAIT(CPUGO,ALGOGO,ENDTERM1) (1);
 CPUSR=USER;
       IF COMPLETION(ENDTERM1) THEN GOTO FIN5; /* FIN SIMULATION */
       /* INDIQUER QUE LE CPU EST OCCUPE */
       COMPLETION(CPUFREE)='0'B ;
       COMPLETION(CPUBSY)='1'B ;
       IF COMPLETION(ALGOGO ) THEN DO ;
                   /* DEMANDE D'EXECUTION DE L'ALGORITHME DE SWAPPING */
                   COMPLETION(ALGOGO )='0'B ;
```

```
(SUBSCRIPTRANGE):
                        TTTIMER=TIME ;
                        STARTEX=HEURE(TTTIMER) ;
                        /* EXECUTION DE L'ALGORITHME DE SWAPPING */
                        DELAY(50) ;
                        DO K=1 TO 4;
                            DO I=1 TO IQCPU(K)-1;
                                CPUWAIT(QCPU(I,K),K)=CPUWAIT(QCPU(I,K),K)
                                50;
                            END;
                            DO I=1 TO IQSWAP(K)-1;
                                SWPWAIT(QSWAP(I,K),K)=SWPWAIT(QSWAP(I,K),
                                +50;
                            END;
                        END;
                        CPUUSE=CPUUSE+50;
                        TCPU(SWAPUSR)=TCPU(SWAPUSR)+50;
                        TTTIMER=TIME ;
                        TEXEC=TEXEC+HEURE(TTTIMER)-STARTEX ;
                        KI=SUBSTR(TTTIMER,7,3);
                        EJECT=1;
                        COMPLETION(ALGOEND)='1'B ;
B='CPU WAIT 2';
                        WAIT(GATE3) ; /* SYNCHRONISATION */
                        COMPLETION(GATE3)='0'B ;
                        /* LIBERER LE CPU */
                        COMPLETION(CPUBSY)='0'B ;
                        COMPLETION(CPUFREE)='1'B ;
                        GOTO ETI13;
      END ;
      /* ON DEMANDE LE CPU POUR L'EXECUTION D'UN PROGRAMME
      UTILISATEUR */
      COMPLETION(CPUGO)='0'B ;
      TTTIMER=TIME ;
B='CPU WAIT 3';
      WAIT(GATE4) ; /* SYNCHRONISATION AVEC ''MAIN'' */
      COMPLETION(GATE5)='1'B ;
      COMPLETION(GATE4)='0'B ;
      STARTT=HEURE(TTTIMER) ;
 IF REFILE(CPUSR)¬=0 |REFSEG(CPUSR)¬=0|REFPAGE(CPUSR)¬=0 THEN DO;
B='CPU WAIT 4';
      WAIT(CPUSTOP);ARGUMEN=MIN(TIMEPAG(CPUSR),TIMEFIL(CPUSR),
   TIMESEG(CPUSR));* INTERRUPTION EN PROVENANCE DU ''MAIN'' */
                        COMPLETION(CPUSTOP)='0'B ;
                        COMPLETION(CPUBSY)='0'B ;
                        INTCYC(CPUSR)=INTCYC(CPUSR)-ARGUMEN*200;
                        /* UNE INSTRUCTION DEMANDE EN MOYENNE 5
                        MICROSECONDES */
```

```
(SUBSCRIPTRANGE);
                    CPUUSE=CPUUSE+ARGUMEN ;
                    TCPU(CPUSR)=TCPU(CPUSR)+ARGUMEN;
                    COMPLETION(CPUFREE)='1'B ;
                    GOTO ETI13;
      END ;
      /* INTERRUPTION DE FIN D'EXECUTION PROGRAMME UTILISATEUR */
      TEMI=INTCYC(CPUSR);
      ARGUMEN=(TEMI*5)/100;
      CALL CLOCK(ARGUMEN) EVENT(ENDCLK4);
      WAIT(ENDCLK4);
    DO K=1 TO 4;
      DO I=1 TO NOCPU(K)-1;
           T=TTREPLY(I)-TTREPLY(USER);
           ELSE T=0;
           IF T<=0 THEN T=ARGUMEN;ELSE IF T<ARGUMEN THEN T=ARGUMEN-T;
           CPUWAIT(OCPU(I,K),K)=CPUWAIT(OCPU(I,K),K)+T;
           TTREPLY(I)=TTREPLY(I)+T;
      END;
      END;
      COMPLETION(CPUFREE)='1'B ;
      COMPLETION(CPUBSY)='0'B ;
      CPUUSE=CPUUSE+ARGUMEN;
      TCPU(CPUSR)=TCPU(CPUSR)+ARGUMEN;
      COMPLETION(GATE6)='1'B ;
 B='CPU WAIT 5';
      WAIT(CPUSTOP) ;
      COMPLETION(CPUSTOP)='0'B ;
      GOTO ETI13 ;
FIN5: END CPU ;
 /*************************************************************/
 /*                                                           */
 /*************************************************************/
SWAPING: PROC ;
  DCL (I,J) BIN FIXED(31);
/* ''SWAPING'' RECOIT LE CONTROLE DE ''MAIN'' QUAND EST SIGNALEE
     UNE DEMANDE DE PAGE */
/* LES DECLARATIONS ET LES INITIALISATIONS ONT ETE FAITES DANS LE
PROGRAMME PRINCIPAL */
/* ATTENDRE LE CONTROLE */
  ID,JD=0;
 ETI10 : ID=ID+1;
 D='SWAP WAIT 1';
  WAIT(SWAPGO,ENDTERM2) (1);
      IF COMPLETION(ENDTERM2) THEN GOTO FIN4;
      COMPLETION(SWAPGO)='0'B ;
      /* DETERMINER LE PROCHAIN UTILISATEUR A SATISFAIRE */
 ETI12 : JD=JD+1;
       CALL NXTSWAP ;
      IF SWAPUSR=0 THEN  GOTO ETI10;
                  /* ATTENDRE L'ARRIVEE DE NOUVELLES DEMANDES */
```

```
(SUBSCRIPTRANGE):

        /* SATISFAIRE LA DEMANDE DE L'UTILISATEUR SELECTE */
        J=PRTY(SWAPUSR) ;
        TTIMER=TIME ;
        /* METTRE L'UTILISATEUR DANS LA QUEUE DES DEMANDES
        SERVICE-DISQUE POUR LA TRANSMISSION D'UNE PAGE */
        START=HEURE(TTIMER) ;
    ETI11 :
    /*PAGE A JETER OU A CHARGER:METTRE L'UTILISATEUR DANS LA QUEUE
        SERVICE-DISQUE */
        FORSWAP(IFOR(J),J)=SWAPUSR ;
        IFOR(J)=IFOR(J)+1 ;
        QDISK(IQDISK(J),J)=SWAPUSR ;
        IMAX(3,J)=MAX( IMAX(3,J),IQDISK(J)) ;
        IQDISK(J)=IQDISK(J)+1;
        TTIMER=TIME ;
        COMPLETION(XMIT1)='1'B;
        /* ATTENDRE LA FIN D'E/S DISQUE */
D='SWAP WAIT 2';
        WAIT(XMRDY1) ;
        COMPLETION(XMRDY1)='0'B ;
        /* SYNCHRONISATION AVEC LE MODULE ''DISQUE'' */
        COMPLETION(GATE1)='1'B ;
        IF EJECT¬=0 THEN DO;
                         EJECT=0;
                         GOTO ETI20;
        END;
        /* SIGNALER AU MODULE ''MAIN'' QU'ON A BESOIN DU CPU POUR
        L'EXECUTION DE L'ALGORITHME DE SWAPPING */
        COMPLETION(NEEDCPU)='1'B;
        /* ATTENDRE LA FIN DE L'EXECUTION DE L'ALGORITHME DE SWAPPING *
D='SWAP WAIT 3';
        WAIT(ALGORIT) ;
        COMPLETION(ALGORIT)='0'B ;
        COMPLETION(GATE3)='1'B ; /* SYNCHRONISATION */
        IF EJECT¬=0 THEN GOTO ETI11;
        /* L'OPERATION DE SWAPPING EST ACHEVEE : LE SIGNALER AU MODULE
        ''MAIN'' */
        ETI20 :
        /* METTRE L'UTILISATEUR DANS LA QUEUE SERVICE-CPU */
        TTIMER=TIME ;
        DUREE=HEURE(TTIMER)-START ;
        IMAX(1,J)=MAX( IMAX(1,J),IQCPU(J)) ;
        TSWAP(SWAPUSR)=TSWAP(SWAPUSR)+DUREE ;
        SWAPUSE=SWAPUSE+DUREE ;
        IF TIMEFIL(SWAPUSR) THEN
                    TIMEFIL(SWAPUSR)=TIMEFIL(SWAPUSR)-TIMEPAG(SWAPUSR);
        IF TIMESEG(SWAPUSR) THEN
```

```
                    TIMESEG(SWAPUSR)=TIMESEG(SWAPUSR)-TIMEPAG(SWAPUSR);
        REFPAGE(SWAPUSR)=REFPAGE(SWAPUSR)-1;
        IF REFPAGE(SWAPUSR)>0 THEN
        TIMEPAG(SWAPUSR)=3;                    /*INITIALISER 1 NOUVEAU
        TIMEPAG*/
        ELSE TIMEPAG(SWAPUSR)=0;
        QCPU(IQCPU(J),J)=SWAPUSR ;
        IQCPU(J)=IQCPU(J)+1 ;
   /* SIGNALER LA FIN DU SWAPPING SI MAIN ATTEND UN UTILISATEUR*/
        IF NEEDUSR THEN DO;
                    NEEDUSR='0'B;
                    COMPLETION(SWAPDON)='1'B;
        END;
        GOTO ETI12 ;
FIN4: END SWAPING ;
 /*************************************************************************/
 /*                                                                       */
 /*************************************************************************/
DISQUE: PROC ;
  DCL (I,J,K) BIN FIXED(31);
/* ''DISQUE'' SIMULE LES DISQUES */
/* LES INITIALISATIONS ONT ETE FAITES DANS LE PROGRAMME PRINCIPAL */
/* ATTENDRE LE CONTROLE */
  IC,JC=0;
 ETI6 : IC=IC+1;
C='DISK WAIT 1';
  WAIT(XMIT1,XMIT2,ENDTERM3) (1);
        IF COMPLETION(ENDTERM3) THEN GOTO FIN3;
        /* DETERMINER LE PROCHAIN UTILISATEUR A SATISFAIRE */
 ETI7 : JC=JC+1;
  ASKSWAP='0'B;
  ASKSEG='0'B;
  CALL NXTDISK;
        IF DSKUSR=0 THEN DO ;
                    /* ON N'A PLUS D'UTILISATEUR DANS LA QUEUE-DISQUE:
                    SE METTRE EN ATTENTE */
                    COMPLETION(XMIT2)='0'B ;
                    GOTO ETI6 ;
        END ;
        IF ASKSWAP THEN DO ;
                    /* TRANSMISSION D'UNE PAGE */
                    COMPLETION(XMIT1)='0'B ;
                    TEM=102 ;
                    DSKUSE1=DSKUSE1+TEM ;
                    DELAY(TEM) ;
                   DO K=1 TO 4;
                        DO I=1 TO IODISK(K)-1;
```

```
UBSCRIPTRANGE):

                        DSKWAIT(QDISK(I,K),K)=DSKWAIT(QDISK(I,K),K)+
                        TEM;
                        END;
                        DO I=1 TO IQSWAP(K)-1;
                        SWPWAIT(QSWAP(I,K),K)=SWPWAIT(QSWAP(I,K),K)+
                        TEM;
                        END;
                    END;
                     COMPLETION(XMRDYI)='1'B ;
                     /* LA TRANSMISSION EST TERMINEE */
                     /* RENDRE LE CONTROLE AU MODULE DE SWAPPING ET
                     ATTENDRE QUE CELUI L'AIT PRIS */
                              TDISK(DSKUSR)=TDISK(DSKUSR)+TEM;
C='DISQ WAIT 2';
                     WAIT(GATE1) ; /* SYNCHRONISATION AVEC LE MODULE
                     ''SWAPING'' */
                     COMPLETION(GATE1)='0'B ;
                              GOTO ETI7;
      END ;
      COMPLETION(XMIT2)='0'B;
      IF ASKSEG THEN DO ;
                     /* TRANSMISSION D'UN SEGMENT */
                     TEM=120 ;
                     DSKUSE3=DSKUSE3+TEM ;
                     DELAY(TEM) ;
                    DO K=1 TO 4;
                        DO I=1 TO IQDISK(K)-1;
                        DSKWAIT(QDISK(I,K),K)=DSKWAIT(QDISK(I,K),K)+
                        TEM;
                        END;
                        DO I=1 TO IQSWAP(K)-1;
                        SWPWAIT(QSWAP(I,K),K)=SWPWAIT(QSWAP(I,K),K)+
                        TEM;
                        END;
                    END;
                     TTTIME=TIME;
                     IF TIMEFIL(DSKUSR) THEN
                     TIMEFIL(DSKUSR)=TIMEFIL(DSKUSR)-TIMESEG(DSKUSR);
                     IF TIMEPAG(DSKUSR) THEN
                     TIMEPAG(DSKUSR)=TIMEPAG(DSKUSR)-TIMESEG(DSKUSR);
                     REFSEG(DSKUSR)=REFSEG(DSKUSR)-1;
                     IF REFSEG(DSKUSR)>0 THEN DO;
                     TIMESEG(DSKUSR)=3;
                     /*INITIALISER UN NOUVEAU TIMESEG*/
                     END;
                     ELSE TIMESEG(DSKUSR)=0;
                     GOTO ETI8 ;
```

```
        END ;
                    /* TRANSMISSION D'UN ENREGISTREMENT DE FICHIER */
                    TEM=95 ;
                    DSKUSE2=DSKUSE2+TEM ;
                    DELAY(TEM) ;
                   DO K=1 TO 4;
                        DO I=1 TO IODISK(K)-1;
                        DSKWAIT(QDISK(I,K),K)=DSKWAIT(QDISK(I,K),K)+
                        TEM;
                        END;
                        DO I=1 TO IOSWAP(K)-1;
                        SWPWAIT(QSWAP(I,K),K)=SWPWAIT(QSWAP(I,K),K)+
                        TEM;
                        END;
                   END;
                    TTTIME=TIME;
                    IF TIMEPAG(DSKUSR) THEN
                    TIMEPAG(DSKUSR)=TIMEPAG(DSKUSR)-TIMEFIL(DSKUSR);
                    IF TIMESEG(DSKUSR) THEN
                    TIMESEG(DSKUSR)=TIMESEG(DSKUSR)-TIMEFIL(DSKUSR);
                    REFILE(DSKUSR)=REFILE(DSKUSR)-1;
                    IF REFILE(DSKUSR)>0 THEN
                    /*INITIALISER UN NOUVEAU TIMEFIL*/
                    TIMEFIL(DSKUSR)=3;
                    ELSE TIMEFIL(DSKUSR)=0;
ETI8:  /* REMETTRE L'UTILISATEUR DANS LA QUEUE DES DEMANDES CPU */
       J=PRTY(DSKUSR) ;
       QCPU(IQCPU(J),J)=DSKUSR ;
       IMAX(1,J)=MAX(IQCPU(J),IMAX(1,J)) ;
       IQCPU(J)=IQCPU(J)+1 ;
       /* NOTER LE TEMPS D'UTILISATION DES DISQUES PAR L'UTILISATEUR */
       TDISK(DSKUSR)=TDISK(DSKUSR)+TEM ;
       /* SIGNALER LA FIN DE LA TRANSMISSION SI MAIN ATTEND UN UTILISA
       -TEUR*/
       IF NEEDUSR THEN DO;
                       NEEDUSR='0'B;
       COMPLETION(XMRDY2)='1'B ;
       END;
       GOTO ETI7 ;
FIN3:  END DISQUE ;
 /*****************************************************************/
 /*                                                               */
 /*****************************************************************/
CONSOLE: PROC ;
  DCL (I,J,K) BIN FIXED(31);
/* ''CONSOLE'' SIMULE L'ENSEMBLE DES CONSOLES */
IOUT=1 ;
```

```
(SUBSCRIPTRANGE):

        IE=0;
        ETI4 : IE=IE+1;
             NXTIME=0 ;
             DO I=1 TO NUSER  ;
                        DO J=1 TO IOUT ;
                                   IF I=OUT(J    ) THEN GOTO ETI5 ;
                        END ;
                        IF COMPLETION(GO(I)) THEN GOTO ETI5;
                        IF NXTAPE(I)=0 THEN
                                   IF COMPLETION(RDY(I)) THEN
                                   IF  TIM(NSESION(I),I)>=300000
                                   |NSESION(I)>ISESION(I) THEN DO;
                                   /* METTRE L'UTILISATEUR DANS LA CHAINE
                                   DES CONSOLES DECONNECTEES */
                                   OUT(IOUT)=I ;
                                   IOUT=IOUT+1 ;
                                   IF IOUT > NUSER  THEN DO;
                                   COMPLETION(ENDTERM)='1'B ;
                                   COMPLETION(ENDTERM2)='1'B;
                                   COMPLETION(ENDTERM3)='1'B;
                                   E='ENDTERM POSITIONE A 1';
                                   GOTO FIN2 ;
                                   END ;
                                   ELSE GOTO ETI5 ;
                                   END ;
                                   ELSE DO ;
                                   /* PREVOIR L'HEURE D'ENVOI DU PROCHAIN
                                   MESSAGE DE L'UTILISATEUR I */
                                   NXTAPE(I)=TTREPLY(I)+
                                   TIM(NSESION(I),I);
                                   PRTY(I)=TYPE(NSESION(I),I);
                                   SSGROUP(I)=SSGP(NSESION(I),I);
                                   NSESION(I)=NSESION(I)+1;
                                   GOTO ETI5 ;
                                   END ;
                                   ELSE GOTO ETI5 ;
                        ELSE DO;
                        /* TESTER L'HEURE D'ENVOI DU PROCHAIN MESSAGE */
                                   IF NXTAPE(I)>HORA THEN GOTO ETI5;
                                   ELSE DO ;
                                   /* ENVOYER LE MESSAGE */
                                   TTREPLY(I)=HORA;
                                   NXTAPE(I)=0;
                                   COMPLETION(GO(I))='1'B ;
                                   /* ATTENDRE QUE LE MAIN AIT RECU LE
                                   MESSAGE */
                                   END ;
```

```
SCRIPTRANG3):

          END;
ETI5:  END ;
       DO I=1 TO NUSER;
              IF NXTAPE(I)¬=0 THEN
                                IF NXTIME > NXTAPE(I)|NXTIME=0 THEN DO ;
                           NXTIME=NXTAPE(I);
                           K=I;
              END;
       END;
       IF NXTIME=0 THEN
 DO;
 F='CONS WAIT 1';
                    WAIT(RDY)(1) ;
                    GOTO ETI4;
 END;
       ELSE DO ;
       /* ATTENDRE L'HEURE DU PROCHAIN ENVOI DE MESSAGE OU LE DEBLOCAGE
       D'UNE CONSOLE */
       IF NXTIME>HORA THEN DO;
                    HHORA=NXTIME;
                    NXTIME=NXTIME-HORA;
                    HORA=HHORA;
         CALL CLOCK1(NXTIME) EVENT(ENDCLK5);
 F='CONS WAIT 2';
       WAIT(ENDCLK5,OK) (1);
       IF COMPLETION(ENDCLK5) THEN GOTO ETI4;
       DO I=1 TO NUSER;
              IF COMPLETION(OK(I)) THEN DO;
                   COMPLETION(OK(I))='0'B;
                   NXTAPE(K)=0;
                   HORA=TTREPLY(I)+
                   GOTO ETI4;
              END;
       END;
       END;
       END ;
FIN2:  END CONSOLE ;
 /*****************************************************************/
 /*                                                               */
 /*****************************************************************/
 CLOCK1 : PROC(I);
/* UNE HORLOGE COMPTANT I MILLISECONDES */
       DCL I BIN FIXED(31);
       DCL (J,K) BIN FIXED(31);
       DO J=1 TO I;
              DO K=1 TO NUSER;
                   TEST=TEST|COMPLETION(OK(K));
```

(SUBSCRIPTRANGE):

```
                END;
                IF TEST THEN GOTO FIN11;
                ELSE DELAY(1);
        END;
        FIN11 : TEST='0'B;
    END CLOCK1;
CLOCK : PROC(I) ;
        DCL I BIN FIXED(31);
        /* UNE HORLOGE COMPTANT I MILLISECONDES*/
        DELAY(1);
END CLOCK ;
 /*****************************************************************
 /*
 /*****************************************************************
NEXTCPU: PROC ;
  DCL (I,J) BIN FIXED(31);
/* SELECTIONNE LE PROCHAIN UTILISATEUR A SATISFAIRE DANS LE
SERVICE-CPU */
        NEEDUSR='0'B;
        /* VERIFIER L'ETAT DES CONSOLES */
        CALL CHECKU ;
        IF CHANGE THEN GOTO FIN6 ;
        DO I=1 TO 4 ;
                IF IQCPU(I)-1¬=0 THEN DO;
                        USER=QCPU(1,I) ;
                        CALL DELETE(QCPU,IQCPU(I),I) ;
                        GOTO FIN6 ;
                END ;
        END ;
        USER=0 ;
FIN6: END NEXTCPU ;
 /*****************************************************************
 /*
 /*****************************************************************
CHECKU: PROC ;
  DCL (I,J) BIN FIXED(31);
/* PERMET DE VERIFIER L'ETAT DES CONSOLES ET DE METTRE A JOUR LA QUE
SERVICE-CPU */
        CHANGE='0'B ;
        DO J=1 TO NUSER  ;
                IF COMPLETION(RDY(J)) & COMPLETION(GO(J)) THEN DO;
                        I=PRTY(J);
                        CALL GENERE(SSGROUP(J),J);
                        /* BLOQUER LA CONSOLE DE L'UTILISATEUR *
                        COMPLETION(RDY(J))='0'B ;
                        COMPLETION(GO(J))='0'B;
                        H?=TIME ;
```

```
JBSCRIPTPAUGE):
                                        CPUWAIT(J,I),DSKWAIT(J,I),SWPWAIT(J,I)=0 ;
                                        TCPU(J),TDISK(J)=0;



                                        QCPU(IQCPU(I),I)=J ;
                                        IMAX(1,I)=MAX(IMAX(1,I),IQCPU(I)) ;
                                        IQCPU(I)=IQCPU(I)+1 ;
                        END ;
        FIN8: END;
 END CHECKU;
 /*************************************************************************/
 /*                                                                       */
 /*************************************************************************/
NXTDISK: PROC ;
  DCL (I,J) BIN FIXED(31);
/* SELECTIONNE LE PROCHAIN UTILISATEUR A SATISFAIRE DANS LE SERVICE-
DISQUE */
        DO I=1 TO 4 ;
                                        IF IQDISK(I)-1¬=0 THEN DO;
                                        DSKUSR=QDISK(1,I) ;
                                        CALL DELETE(QDISK,IQDISK(I),I) ;
                                        IF FORSWAP(1,I)=DSKUSR THEN DO;
                                        /* ALORS DU SWAPPING EST DEMANDE */
                                        ASKSWAP='1'B ;
                                        CALL DELETE(FORSWAP,IFOR(I),I);
                                        END ;
                                        ELSE
                                        IF FORSEG(1,I)=DSKUSR   THEN DO;
                                        ASKSEG='1'B ;
                                        CALL DELETE(FORSEG,JFOR(I),I) ;
                                        END ;
                                        GOTO FIN9 ;
                        END ;
        END ;
        DSKUSR=0 ;
FIN9: END NXTDISK ;
 /*************************************************************************/
 /*                                                                       */
 /*************************************************************************/
NXTSWAP: PROC ;
  DCL (I,J) BIN FIXED(31);
/* SELECTIONNE LE PROCHAIN UTILISATEUR A SATISFAIRE DANS LE
        SERVICE-SWAPPING */
DO I=1 TO 4 ;
```

```
(SUBSCRIPTRANGE):

                                IF IQSWAP(I)-1¬=0 THEN DO;
                        SWAPUSR=QSWAP(1,I) ;
                        CALL DELETE(QSWAP,IQSWAP(I),I) ;
                        GOTO FIN7 ;
        END ;
END ;
/* IL N'Y A AUCUN UTILISATEUR DANS LA QUEUE SERVICE-SWAPPING */
SWAPUSR=0 ;
FIN7: END NXTSWAP ;
 /*****************************************************************/
 /*                                                               */
 /*****************************************************************/
DELETE: PROC(PILE,UPBOUND,J) ;
/* SUPRIME L'ELEMENT QUI SE TROUVE AU FOND DE LA PILE */
        DCL(    UPBOUND,PILE(*,*)       ) BIN FIXED(31);
        DCL J BIN FIXED(31);
        DCL K BIN FIXED(31);
                        UPBOUND=UPBOUND-1 ;
                        DO K=1 TO UPBOUND-1;
                                PILE(K,J)=PILE(K+1,J) ;
                        END ;
        PILE(UPBOUND,J)=0;
END DELETE ;
 /*****************************************************************/
 /*                                                               */
 /*****************************************************************/
HEURE: PROC(TEMPS) BIN FIXED(31);
/* DONNE L'HEURE EN MILLISECONDES */
        DCL TEMPS CHAR(9) ;
        DCL ITIME BIN FIXED(31);
                        ITIME=SUBSTR(TEMPS,1,2)*3600000+
                                SUBSTR(TEMPS,3,2)*60000+
                                SUBSTR(TEMPS,5,2)*1000+SUBSTR(TEMPS,7,3)
                        RETURN(ITIME) ;
END HEURE ;
 /*****************************************************************/
 /*                                                               */
 /*****************************************************************/
GENERE: PROC(GRP,NBUSER);
          DCL GRP BIN FIXED(31);
          DCL NBUSER BIN FIXED(31);
          DCL ETI(3) LABEL INIT(ETI30,ETI40,ETI50);
          GOTO ETI(PRTY(NBUSER));
ETI30:/* MODE ADMINISTRATION*/
          REFPAGE(NBUSER)=0;
          REFSEG(NBUSER)=0; REFILE(NBUSER)=0;
          TIMESEG(NBUSER)=0; TIMEFIL(NBUSER)=0;
```

```
SOUSCRIPTRANGE):

          TIMEPAG(NBUSER)=0;
          INTCYC(NBUSER)=1200;
          GOTO FIN;
ETI40: /* MODE COMPILATION*/
          REFSEG(NBUSER)=2;REFILE(NBUSER)=3;
          TIMESEG(NBUSER)=1; TIMEFIL(NBUSER)=2;
/*QWERTYUIOP*/
          REFPAGE(NBUSER)=3;
          TIMEPAG(NBUSER)=3;
          INTCYC(NBUSER)=4400;
          GOTO FIN;
ETI50: /* MODE EXECUTION*/
          REFSEG(NBUSER)=0; REFILE(NBUSER)=0;
          TIMESEG(NBUSER)=0; TIMEFIL(NBUSER)=0;
          REFPAGE(NBUSER)=0;
          INTCYC(NBUSER)=1200;
          TIMEPAG(NBUSER)=0;
FIN : END GENERE;
 /*************************************************************************/
 /*                                                                       */
 /*************************************************************************/
  END;
END STMUL;
```

B I B L I O G R A P H I E

* * * *

On trouvera dans ce qui suit un commentaire sur chaque ouvrage ou article auquel nous nous sommes referé au cours de notre recherche.

Les étoiles placées entre crochets indiquent l'intérêt que nous avons porté aux documents :

[* * *] reflète la richesse des éléments que l'on peut trouver dans l'article.

[* *] indique un document intéressant et qui mérite d'être lu.

[*] correspond à un ouvrage interessant mais non fondamental dans notre recherche.

[ ] correspond à un article pouvant intéresser certains lecteurs.

1 - [* *] BOKSENBAUM Université de Grenoble

"A proposal for the simulation of CP 67" non publié
On relève dans cet article une idée intéressante concernant la charpente générale d'un modèle de simulation d'un système : découpage du système réel en un certain nombre de ressources devant lesquelles s'alignent des files d'attente d'utilisateurs rendus activables par un algorithme de choix.

2 - [* * *] CHUPIN & DELPUECH IMAG

"Avant projet de spécification du FORTRAN conversationnel sur IRIS 50". Communication privée.

3 - [*] DYNKIN

"Markov processes" Volume 1.
Noter la définition des processus markoviens.

4 - [*] DYNKIN Dunod

"Théorie des processus markoviens".
Noter la définition des processus markoviens.

5 - [*] FALKOFF

"Criteria for a system design Language".
Working paper for the NATO Sciences commitee conference on techniques in Software engineering, Rome, Italy, Octobre 1969.

Le langage formel qui a pour objet la description d'un système doit pouvoir exprimer de manière claire, concise et précise un certain nombre d'idées ou de principes inhérents au système. Le choix d'un tel langage est fondé sur certains critères cités dans cet article et accompagnés d'exemples décrits à l'aide d'APL/360.

6 - [* *] G.H.FINE and Paul V.Mc ISAAC

"Simulation of Time-sharing system" .
Management Science Vol 12 /n°6/ February 1966

Le modèle décrit par FINE et Mc ISAAC simule le système à temps partagé SDC. Originalité du modèle : implémentation en langage assembleur et simulation des "pannes" du système réel à l'aide d'un générateur de mauvais fonctionnement.

7 - [*] HERSCOVITCH, SCHNEIDER

"Expended general purpose simulator"

Vol /n°3/ 1968 / IBM System Journal
Cet article peut servir d'introduction à l'étude de GPSS.

8 - [****] J.J. KATZ

"An experimental model of System 360 "
Com, ACM /Vol 10/ n°11 / November 1967

Cet article décrit avec clarté le modèle SIMSCRIPT représentant le système 360. Les paramètres utilisés reprennent fidèlement tous les concepts que l'on rencontre dans le système 360. Il contient des idées intéressantes sur la façon de représenter la configuration Hardware d'un système.

9 - [* *] J.J. KATZ

"Simulation of a multiprocessor Computer System".
Proc.AFIPS 1966 Spring joint comp.conf. Vol 28

Ce système de simulation comprend :

- le générateur de travaux qui fournit un ensemble de paramètres caractéristiques pour chaque travail,

-le simulateur, décrit à l'aide de SIMSCRIPT, qui reçoit en entrée le produit du générateur de travaux et des paramètres de contrôle de la simulation proprement dite. Originalité : dans ce simulateur est inclus un analyseur d'OVERHEAD construit à partir de mesures opérées sur le système réel et l'évolution dynamique du modèle est sous le contrôle d'une matrice de précédence.

10 - [ ] KOLSKY

"Problem formulation using APL"
IBM System journal /n°3/ 1969

Ce papier se propose de formuler un problème donné à l'aide d'APL et de montrer comment des erreurs de logique sont évitées grâce à la concision inhérente à ce langage. Une comparaison est ensuite faite avec une formulation en FORTRAN du même problème.

11 - [***] KRASNOW, MERIKALLIO.

"The past, present and future of general simulation languages".
Management Science /Vol 11/ n°2/ Novembre 1964.
HOWARD et MERIKALLIO résument et comparent les caractéristiques essentielles de cinq langages de simulation (CSL, SIMSCRIPT, GPSS, SIMPAC et DYNAMO). L'exposé est clair mais le lecteur peut regretter l'absence de tableaux récapitulatifs.

12 - [* *] H.MARKOWITZ The rand corporation

"SIMSCRIPT a simulation programming language".
Un ouvrage clair qui permet d'acquérir facilement tous les concepts de SIMSCRIPT.

13 - [*] J. MAUBLANC

"Etude statistique d'un système à temps partagé".
Thèse de la Faculté des Sciences de Clermont-Ferrand Février 1970.

Développement d'un modèle mathématique du système à temps partagé RAX, à partir des hypothèses, émises au cours d'une précédente étape de mesures réalisées sur le système réel.

14 - [* *] T.NAYLOR, BALINGTFY, BURDICK, KONGCHU

"Computer Simulation techniques"

Cet ouvrage traite des problèmes de la simulation en général, mais il reste orienté vers l'étude des modèles économiques. On y trouve des techniques de génération de nombres aléatoires, de files d'attente....Les auteurs consacrent un chapitre entier aux langages de simulation. Ils soulèvent aussi le problème de la validité d'un modèle et indiquent les moyens existant actuellement pour les besoins de la vérification.

15 - [***] N.R. NIELSEN

"The simulation of time-sharing systems".

Com. ACM /Vol 10/ n°7/ juillet 1967
NIELSEN décrit le modèle qu'il a conçu pour simuler un certain nombre de systèmes à temps partagé. Ce modèle peut aussi être utilisé pour développer de nouveaux algorithmes et de nouvelles techniques de "time-sharing". Par souci d'efficacité et pour étendre son domaine d'application, le modèle a été implémenté en FORTRAN. Il a été employé en particulier pour la simulation du IBM-360/67-TSS. En dehors de la description proprement dite du modèle, le lecteur trouve des remarques intéressantes

- sur les raisons qui rendent la simulation nécessaire dans le cas des systèmes d'exploitation complexes,

- sur les caractéristiques générales d'un modèle de simulation des systèmes d'exploitation.
- sur le degré de détail du modèle.
- sur la représentation des JOBS.
- sur la façon de conduire une expérimentation à l'aide de la simulation.

16 - [* *] NYGAARD

"Simula -An algol based simulation language".

CACM septembre 1966

17 - [***] A.L. SCHERR

"An analysis of time-shared computer systems".

MIT research monograph n°36
L'intérêt de cet ouvrage est fondamental pour la simulation des systèmes conversationnels. Allan SCHERR s'est défini un langage de simulation -extension du langage MAD (Michigan Algorithm Decoder) Enfin et surtout le lecteur trouve en Appendice le programme complet du modèle de simulation dont il peut tirer quelques idées constructives.

18 - [*] SEAMAN

"Role of Digital Simulation"

IBM system journal /Vol5/ n°3/ 1966.
Ce papier commente les différents critères qui interviennent dans le choix d'un langage à vocation générale ou d'un langage spécialisé pour la simulation d'un système.

19 - [* *] J.L. SMITHS

"An analysis of time-sharing computing systems using Markov models"

Proc. AFIPS 1966 Spring joint computer conf. vol 28.
Ce papier décrit la manière de représenter un système à temps partagé au moyen d'un modèle de Markov. Ce procédé implique l'approche bien connue de la théorie des files d'attente avec l'introduction de variables stochastiques. Ce modèle appliqué à la simulation d'un système à temps partagé simple a donné satisfaction mais son manque de souplesse limite sa généralité et sa puissance.

20 - [***] STANLEY and ISERTEL

"Statistics gathering and Simulation for the Appollo real time Operating system"

IBM system journal n°2 1962.
Article intéressant par le sujet d'actualité auquel il se rapporte. Stanley et Hertel présentent RTOS/360 (real time operating system/360) qui consiste en un OS modifié (efficacité accrue + adaptation aux contraintes du temps réel). L'évolution de RTOS/360 s'est faite grâce à un modèle de simulation qui comprend d'une part un système de récupération dynamique de données statistiques sur RTOS/360, d'autre part un programme GPSS de prédiction des performances d'une mission donnée (activité unité centrale -temps de réponse- utilisation des canaux).
Cet article est riche d'enseignements intéressants, notamment sur l'interface entre RTOS/360 et le système de récupération de données.

21 - [*] SANDRA REHMAN et SHERBIE

" A simulation study of resource management in a time-sharing system".

Il s'agit d'un modèle GPSS de description d'un système à temps

partagé simple (sans pagination). Il fait intervenir de nombreuses simplifications sur la réalité au niveau des Entrées-sorties et de la génération des travaux des utilisateurs.

22 - [* *] MERIKALLIO & F.C.HOLLAND

"Simulation design of multiprocessing system".

Fall joint comp. 1968.
Cet article contient à la fois une présentation d'un système multiprocesseur avec les avantages que ce système d'exploitation possède et la description du modèle de simulation correspondant (implémenté en GPSS puis en CSS). Il donne en particulier un résumé des différentes approches possibles pour réaliser une simulation: expérimentation sur le système réel, analyse mathématique, modèle de simulation discret.

23 - [* *] ROSENFELD - LEHNAN

"Performance of a simulated multiprogramming system".

Fall joint Computer conf. 1968
Etude des performances de MVT sur un ordinateur IBM 360/65 à l'aide d'un modèle de simulation.

24 - [ ] GREENBERGER

"A new methodology for computer simulation" MIT octobre 1964

Introduction et idées générales sur la manière de collecter des informations et de les transmettre au modèle de simulation par l'intermédiaire de variables.

25 - [* *] TOCHER

"Review of simulation techniques". Operational research
Vol 16/N°2/ Juin 1965

Les remarques faites pour l'article suivant peuvent être reprises pour Review of Simulation techniques.
Tocher fait une comparaison entre GPSS, SIMPAC, SIMSCRIPT, SIMULA, CSL, ESP, GSP, MONTECODE et SIMON; son papier est cependant moins facile à lire que "Computer simulation discussion of the technique and comparison of languages".

26 - [***] D.TEICHROEW and J.F.LUBIN

"Computer simulation -Discussion of the technique and comparison of Languages"

Com.ACM/Vol 9/N°10/ Octobre 1966.

Le but de ce papier est de comparer divers langages de simulation parmi les plus connus : SIMSCRIPT, CLP, CSL, GASP, GPSS et SOL.
La qualité essentielle de l'article est sa clarté et sa concision : les caractéristiques de chaque langage sont rassemblées dans des tableaux qui facilitent l'analyse et l'étude de ces langages de simulation.

27 - [*] "Simulation programming Languages"

(Proceding of the IFIP working conference on simulation programming Languages OSLO 1967)

On y trouve une bonne définition de la simulation (page 2), un exemple de langage de simulation incrémentielle (OPS-4), divers langages de simulation dérivés de PL/I ou d'Algol et la présentation d'un langage général de la 3e génération, GPL, qui n'aurait pas l'inconvénient des langages de la deuxième génération, orientés eux vers le traitement de problèmes trop particuliers.

28 - [* *] General purpose simulator system/360 user's manual H20-0326-2

Le système de simulation GPSS/360 fournit un certain nombre de concepts de base dont l'agencement judicieux permet de décrire aisément un modèle, de suivre le déroulement d'une expérience et d'en tirer les leçons grâce aux résultats statistiques obtenus.

Ce langage est relativement aisé à apprendre mais la façon dont le programmeur doit aborder son problème s'écarte des approches utilisées d'ordinaire en programmation courante.

29 - [* *] PL/I reference manual IBM System 360 3360-29

30 - [* *] PL/I(F) programmer's guide IBM System 360 3605 - NL - 511

31 - [* *] APL/360 Primer student text IBM

APL/360 est un système à temps partagé qui permet à un utilisateur de définir des programmes APL à partir de terminaux.
Caractéristiques d'APL : Puissance, simplicité, concision.
Le document est facile à lire et constitue un excellent apprentissage à l'utilisation d'APL/360.

32 - [ ] D.PARNAS

"On simulating networks of parallel processes in which simultaneous events may occur".
Com. ACM Vol/2/ N°9/ septembre 1969.

Les systèmes au sens général sont représentés par des réseaux d'opérations séquentielles interconnectés . Parnas propose une représentation de ces réseaux en utilisant certains éléments de la théorie des systèmes combinatoires.

33 - [*] Géneral purpose simulator system/360. User's manual H20-0326-2

34 - [*] DAVID Université de Grenoble

"Fiabilité" 1967-68 Polycopié
Contient les renseignements utiles à la définition et à l'emploi des processus de Markov.

35 - [*] COUTAZ Université de Grenoble

"Simulation d'un système conversationnel".
Congrès AFCET-Septembre 1970

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

VU, et permis d'imprimer

Le Recteur de l'Académie de GRENOBLE