MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DE BLIDA
DEPARTEMENT DE MATHEMATIQUES

Thèse de Magister

en
MATHEMATIQUES APPLIQUEES

option

MODELISATION MATHEMATIQUE
ET
TECHNIQUES DE DECISION

ORDONNANCEMENT STOCHASTIQUE OPTIMAL PAR CLASSES DE PRIORITES BORNEES DANS UN SYSTEME MULTIPROGRAMME MONOPROCESSEUR

par

Loucif ZERGUINI

Présentée devant le jury constitué de :

Président:	Mr Djedour.M. ,USTHB, Professeur
Rapporteurs:	Mr Proust.C. , Univ de Tours, Professeur
	Mr Aïssani.A. , Univ de Blida, Professeur
Examinateurs:	Mr Ksir.B. , Univ de Constantine, MC
	Mr Arrar.A. , Univ d'Annaba, MC
	Mr Ouafi.R. , USTHB, CC

Fevrier 1996

REMERCIEMENTS

Je remercie vivement monsieur Christian Proust, professeur à l'université de Tours, ainsi que monsieur Amar Aissani, professeur à l'université de Blida, pour les encouragements et l'aide qu'ils n'ont cessé de me prodiguer tout au long de la réalisation de ce travail.

Je remercie monsieur M.Djedour d'avoir bien voulu accepter de présider le jury, de même que messieurs B.Ksir, A.Arâar et R.Ouafi d'avoir bien voulu y participer.

Mes remerciements vont également à mes collègues du groupe de travail " Ordonnancement et Conduite".

Enfin, je remercie tous ceux de près ou de loin ont participé à la réalisation de ce travail.

SOMMAIRE

RESUME

Un ordonnanceur (Scheduler) est l'une des principales composantes des systèmes d'exploitation. Il est donc important de prouver l'optimalité de nouveaux algorithmes régissant son fonctionnement . Les démonstrations de L'O.P.B.M [HARO-PROUST 92] montrent les limites des mathématiques déterministes.

L'équité de L'O.P.B.M, bien que vérifiée expérimentalement , reste un problème mathématiquement ouvert.

La théorie des processus stochastiques de décision laisse entrevoir des possibilités de modélisation interessantes.

Dans cette thèse, on montre que le modèle peut être formulé comme un problème de Bandit en introduisant une structure de coût adéquate. On présente un algorithme de détermination de la stratégie optimale d'ordonnancement inter-classes.

INTRODUCTION

On dit qu'on a affaire à un problème d'ordonnancement lorsque : " on doit programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution [...]. Les tâches sont le dénominateur commun des problèmes d'ordonnancement , leur définition n'est ni toujours immédiate , ni toujours triviale" *

On notera que ces problèmes se rencontrent dans les domaines divers d'application notamment dans les systèmes informatiques et dans les systèmes de production. Seule l'importance relative de telle ou telle contrainte change alors (la préemption , l'aspect cyclique....) mais pas la façon de les aborder ni l'ensemble des outils nécessaires qui relèvent de toute évidence de la panoplie des méthodes de résolution des problèmes combinatoires.**

Une expertise en ordonnancement nécessite une prise de connaissance d'une problématique particulière pour en faire émerger les contraintes dures, fondamentales. Ceci suppose la connaissance d'une typologie , des problèmes centraux et des outils tels que ceux présentés dans G.O.Th.A***

---

*) in *J.CARLIER, P.CHRETIENNE, **Problèmes d'ordonnancement** (modélisation/ complexité/ algorithmes), Masson éd . **collection Etudes et Recherches en Informatique**, 1988.*

) Voir par exemple: *J.BLAZEWICZ, E. ECKER, G. SCHMIDT, J. WEGLARZ, **Scheduling in Computer and Manufacturing Systems, Springer- Verlag, 1993.*

***) Groupe d'Ordonnancement Théorique et Appliqué . (Pr **P.CHRETIENNE, C.HANEN, A.MUNIER:** LITP Paris VI, Pr **M-C PORTMANN** : Mines/ CRIN Nancy, **E. PINSON**: IMA/ CREAM Angers, Prs **J. CARLIER & P. VILLON, J-P. BOUFFLET**: U.T. C . Compiègne , Pr **J. ERSCHLER & P.LOPEZ:** L.A.A.S./ CNRS Toulouse, Pr **C. PROUST :** E3i/ LI Tours, **C.PRINS:** Mines Nantes).....

...G.O.Th.A, *Les problèmes d'ordonnancement , RAIRO- RO./ Operations Research vol.27, n° 1 , pp. 77-150, 1993.*

Ils existent de multiples approches de résolution *, **, ***

- Par construction progressive,
- Par Voisinage,
- Par décomposition,
- Par modification des contraintes,
- Liées à l'intelligence artificielle.....

.....qui puisent dans les techniques de l'optimisation combinatoire (programmation mathématique, dynamique, P.S.E., Théorie des graphes,..... méthodes Tabou , Récuit Simulé, Réseaux neuromimétiques, algorithmes génétiques.....)

Les problèmes liés à l'ordonnancement des processus d'un système multiprogrammé ont suscité de nombreuses études : [DEITEL 1984, KRAKOWIAK 1987, PETERSON 1983,etc.....].

Un algorithme d'ordonnancement réalise l'allocation des proceseurs libres aux processus candidats de la file d'attente , selon un critère donné qui "optimise" l'utilisation de la ressource.

L'ordonnancement préemptif est caractérisé par le fait que le processeur exécute à tout instant le processus ayant la plus haute priorité parmi ceux qui attendent d'être exécutés. Le service du processus candidat se poursuit jusqu'à la fin si et seulement s'il n'y a pas occurence d'un processus de plus haute priorité.

---

*) *" Les problèmes d'ordonnancement" , 1993, G.O.Th.A., RAIRO - RECH. Op / Operations Research, vol, n° 1, p.77 à 150.*

***) Actes des journées d'études " Ordonnancement et entreprises: applications concrètes et outils pour le futur " , GdR Automatique / pôle SED / GT3, Toulouse, 315p., juin 1994.*

****) Manufacturing Scheduling", 1993, Conférence Invitée, Actes de IEPM'93, FUCAM/ IFIP/ INRIA, Mons (Belgique), 32 p., 1993.*

La priorité d'un processsus peut être fixe ou évoluer de manière dynamique en fonction de son histoire ou de paramètres fixés par le concepteur. On distingue diverses familles d'ordonnancement regroupant les algorithmes qui implémentent des stratégies mixtes, telles que l'organisation en files multiniveaux [CRISMAN 1965] , qui réalise une partition de l'ensemble des processus en attente ou en cours d'exécution. L' algorithme de choix du scheduler élit les processus de la file non vide la plus prioritaire selon la discipline dite du tourniquet ou Round Robin [DEITEL 1984].

Dans cette thèse ,on considère plusieurs classes de processus qui s'exécutent en parallèle dans un système multiprogrammé monoprocesseur , où :

- La priorité de chaque processus évolue dynamiquement en fonction de paramètres temporels mais en restant bornée par deux réels positifs,
- Chaque classe est une file d'attente de type M/M/1 preemptive qui regroupe un ensemble de processus de même intervalle d'évolution de leurs priorités.

Ces deux points caractérisent bien le Scheduler* O.P.B.M** introduit par [Haro- Proust 1992] lorsqu'on a un très grand nombre de processus.

La structure de coût est le gain moyen pur reçu à l'instant de décision. Notre problème est d'obtenir l'ordonnancement optimal des processus qui maximise l'espérance de la valeur actuelle du gain dans un horizon infini et par suite déterminer le domaine de variation de chacune des bornes associées à chaque processus . Pour cela on on formule notre modèle comme un problème de bandit avec arrivées poissonniennes .

Plusieurs auteurs [L.E Schrage, G.P.Klimov , etc..] ont proposé , dans le cas de minimisation de l'espérance des coûts de séjour,des stratégies optimales, qui sont caractérisées par la priorité donnée par la discipline de service.

(*) Voir chap .1

(**) Voir chap.2

Quand le temps moyen de service est connu dans une file d'attente non-preemptive la discipline " le plus court temps moyen de service - premier servi " (S.E.P.T.F) est optimale. Quand le temps de service est connu dans une file d'attente préemptive la discipline " le plus court temps de service restant - le premier servi" (S.R.P.T) est optimale

L'objectif est de montrer que la problématique de l'ordonnancement dans un système multiprogrammé monoprocesseur relève de celle des problèmes de bandit [GITTINS 79,89]. Ces derniers concernent l'étude et la gestion de projets indépendants . A un instant donné, on décide de travailler sur l'un de ces projets , on reçoit un gain et le prochain état du projet est déterminé par des probabilités de transition. [GITTINS 79,89] a montré que la stratégie optimale d'ordonnancement est obtenue en élisant le projet dont l'indice d'allocation dynamique (DAI) est le plus grand. Il l'applique sur plusieurs exemples, y compris dans le cas d'ordonnancement de processus. [Whittle 1980,1981,1982] a obtenu l'expression intégrale de l'espérance de la valeur actuelle du gain et a prouvé l'optimalité d'une stratégie qu'il propose dans le cas d'arrivées continues et de non arrivées (système fermé). Beaucoup d'applications du problème de bandit sont discutées dans [T.Hirayama 1987, Z.Rosberg 1985, P.P.Varaiya, J.c Walrand et C.Buyukkoc 1985]

Quand les distributions des durées de service au sein des différentes classes de processus sont connues, la stratégie optimale est obtenue par la règle d'indice* . Pour un critère avec actualisation [HARRISON 1975] a étudié les files M/G/1 non-préemptives. On montre que l'approche de ce dernier reste aussi valable pour les files M/M/1 préemptives (à source infinie).

---

*) La règle d'indice consiste à choisir la classe qui a le plus grand indice et à l'intérieur de cette classe les processus sont élus selon la discipline FIFO.

Cette thèse , après une brève présentation des systèmes d'exploitation dans le chapitre 1, montre les avantages théoriques que procure l'utilisation des processus de décision définis dans le chapitre 2. La modélisation de l'O.P.B.M est donnée dans le chapitre 3, et la construction de la stratégie optimale dans le chapitre 4. Enfin des critiques et des possibilités d'extension de l'approche sont données en conclusion.

CHAPITRE 1

1. GENERALITES SUR LES SYSTEMES D'EXPLOITATION

1.1 INTRODUCTION

Un ordinateur est constitué, essentiellement , des périphériques et d'une unité centrale (fig 1). Cette dernière est chargée, à partir des informations lues par les éléments d'entrée, d'élaborer des résultats et de les transmettre aux éléments de sortie, elle se compose de la mémoire centrale qui mémorise les informations (données, tables, barèmes, résultats, ...), d'un processeur qui est chargé d'analyser et d'exécuter les instructions enregistrées dans la mémoire centrale

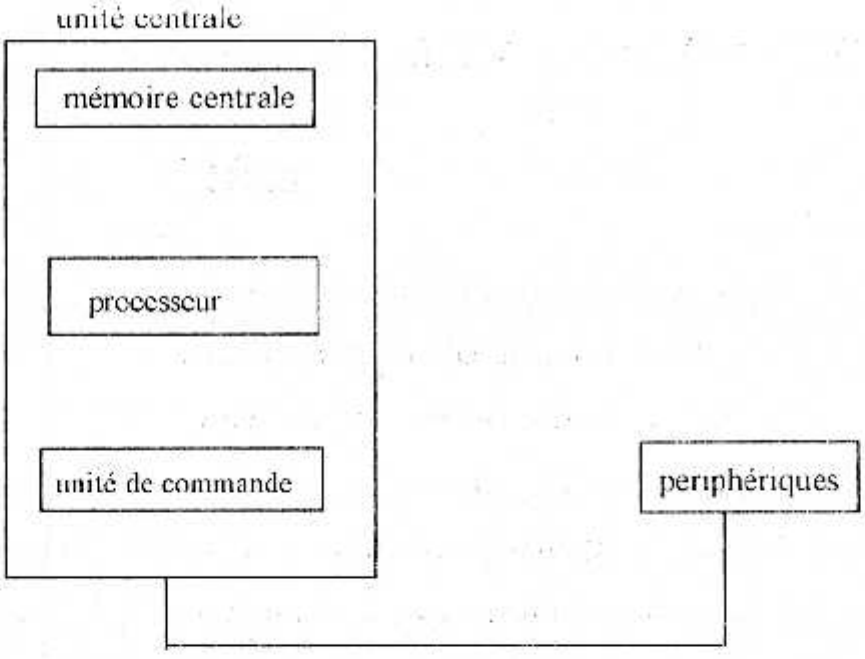


Fig 1: Schéma général d'un ordinateur

ainsi que de commander et de contrôler les différents éléments de l'unité centrale et partiellement des périphériques et enfin d'une unité de commande des entrées et sorties qui commande, coordonne et contrôle les échanges d'informations entre l'unité centrale et les unités périphériques.

Le système d'exploitation est un ensemble de programmes qui se chargent du contrôle et de la transmission des données entre les différentes parties de l'ordinateur. Son rôle est donc de gérer la machine et de fournir à l'utilisateur les moyens pour mettre en oeuvre ses applications . Pour mieux comprendre leurs objectifs et leurs structure, donnons un bref aperçu historique de leurs évolutions.

Les premières machines (1946-1955) n'ont pas de mémoire et n'exécutent qu'un seul programme à la fois. Ecrit en langage machine, le seul que l'unité centrale puisse exécuter directement, il doit être chargé manuellement dans des registres, au départ bit par bit, en manoeuvrant des interrupteurs (charger un programme pouvait prendre plusieurs jours), puis par l'intermédiaire de claviers hexadécimaux et, plus tard, en utilisant des lecteurs de cartes ou de rubans perforés. Puis l'exécution est lancée, en plaçant dans le compteur ordinal l'adresse de la première instruction . Le programmeur peut suivre cette exécution par l'allumage de boutons lumineux sur la console, d'où il peut déduire la valeur de certains registres. En cas de comportement anormal, il interrompt l'exécution et consulte les contenus de la mémoire et des registres pour essayer d'en trouver la cause .Le programmeur gère lui-même ses opérations d'entrée-sortie, un intéressant travail qui l'oblige à connaître les moindres détails du dialogue avec chaque type de périphérique et à se recycler chaque fois qu'un nouveau périphérique est introduit Ecrire et exécuter un programme est alors une tâche fastidieuse, que l'on se hâte d'adoucir en faisant réaliser automatiquement par un programme écrit une fois pour toutes, un certain nombre d'opérations. C'est ainsi que sont développés des chargeurs, des assembleurs et des éditeurs de liens, les fonctions mathématiques les plus courantes sont groupées dans des bibliothèques de programmes. Pour chaque périphérique, un conducteur de périphérique (device driver) , c'est-à-dire un programme réalisant les opérations d'entrée-sortie sur ce périphérique en fonction de quelques paramètres simples, est écrit une fois pour toutes et peut , par la suite, être utilisé par d'autres

programmes. Plus tard ,sont développés les compilateurs, qui permettent d'utiliser des langages évolués, plus puissants et considérablement plus concis que le langage machine, pour l'écriture des programmes.

L'apparition des transistors (1955-1965) modifie radicalement la situation .Au lieu d'une équipe qui fabrique et utilise son propre matériel, des industriels mettent en oeuvre la production et la commercialisation des machines. Les utilisateurs écrivent leurs programmes en fortran et en assembleur. Les différentes étapes de l'exécution d'un programme sont alors les suivantes. Les utilisateurs doivent perforer des cartes et les apporter à un opérateur. A son tour, celui-ci, si besoin est ,va chercher les cartes d'un compilateur qu'il charge. Le compilateur lit, comme donnée, le programme utilisateur et le transforme en un programme sous une forme intermédiaire, par exemple en langage d'assemblage. L'opérateur doit donc aller chercher le paquet de cartes correspondant à l'assembleur, le faire lire par le lecteur de cartes et le faire s'exécuter, avec, comme donnée, le programme produit par le compilateur. L'assembleur produit à son tour un programme en langage machine qui peut enfin être exécuté. Un problème naît des erreurs, puisque le programmeur n'est pas sur place pour tenter un diagnostic à chaud et qu'il est impossible à l'opérateur de le faire. Dans ce cas, l'opérateur imprime l'image de la mémoire sur un listing (dump), qui est fourni ultérieurement au programmeur. Même exécutés par un opérateur professionnel, les chargements successifs ralentissent le rythme des exécutions, avec pour conséquence un temps d'utilisation de l'unité centrale toujours faible. On tente donc de diminuer et d'automatiser ces chargements. Pour accélérer les opérations deux méthodes sont introduites:

a) Le traitement par lots (batch processing) consiste à regrouper et à traiter à la suite des travaux similaires. Ainsi, tous les programmes fortran sont rassemblés

dans un lot, tous les programmes cobol dans un deuxième et tous les programmes en langage d'assemblage dans un troisième.

b) L'enchaînement automatique des travaux est réalisé par un programme spécial, appelé moniteur d'enchaînement ou moniteur résident. Des cartes de type spécial appelées cartes de contrôle sont insérées au début et à la fin de chaque programme, ce qui permet au moniteur d'enchaînement de distinguer les différents programmes d'un même lot

Le moniteur est l'ancêtre des systèmes d'exploitation. Une place privilégié lui est réservée dans une partie de la mémoire centrale, qui se divise donc en deux zones: La zone moniteur et la zone utilisateur. Avec le traitement par lots, la différence énorme de vitesse de traitement, entre les périphériques d'entrée-sortie et l'unité centrale, se fait encore plus manifeste. Par exemple un programme de 250 cartes est lu en 50 secondes et assemblé en 2 secondes. Les développements suivant tentent de regagner ce temps perdu en faisant se recouvrir la lecture du programme d'un lot avec l'exécution de programmes précédents. Les périphériques d'entrée-sortie sont dotés de circuits leur permettant de transférer des informations de manière autonome, sans le secours de l'unité centrale.

L'arrivée des circuits intégrés (1965-1980), qui remplacent les transistors, abaisse le coût de fabrication des ordinateurs. Les modèles évoluent rapidement, ce qui pose le problème de leur compatibilité. L'acquisition d'un nouveau matériel nécessite parfois la réécriture totale des programmes d'exploitation. C'est pourquoi IBM lance, avec le système OS/360 , l'idée du système unique s'adaptant à plusieurs types de machines. Simultanément, l'arrivée des disques modifie complètement la situation. Le problème des bandes magnétiques est que, lorsque la tête de lecture-écriture est à une extrémité d'une bande ,il faut un délai important avant qu'elle ne puisse traiter une information à l'autre extrémité .Au contraire ,un disque est capable de déplacer sa tête de lecture-écriture très

rapidement d'un secteur à l'autre. Les disques jouent le rôle précédemment tenu par des dérouleurs de bandes magnétiques, en permettant de plus le recouvrement de phase de calcul d'un programme avec des phases d'entrée-sortie d'autres programmes, ce qui donne naissance au concept de multiprogrammation. Plusieurs programmes sont chargés sur le disque et en mémoire centrale.

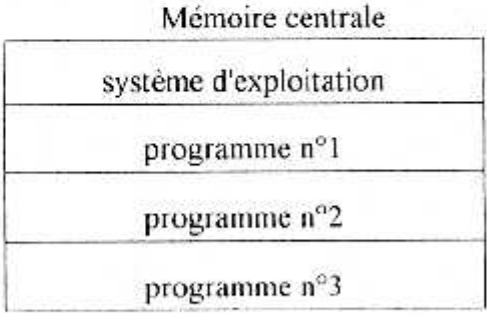


fig 2: Multiprogrammation

L'unité centrale commence par exécuter l'un des programmes. Dés que celui-ci demande l'exécution d'une opération d'entrée-sortie, l'unité centrale la lance et commence l'exécution d'un autre programme, sans attendre l'achèvement de l'opération d'entrée-sortie. Si aucun des programmes chargés en mémoire centrale n'est prêt à être exécuté, l'unité centrale en charge un à partir du disque Les exécutions se poursuivent sur ce mode: chaque fois qu'une opération d'entrée-sortie est nécessaire, l'unité centrale la lance et passe l'exécution d'un autre programme. Ainsi, l'unité centrale est-elle active de manière quasi permanente. La grande capacité de stockage du disque fait qu'il y a pratiquement toujours un programme qui puisse être exécuté. Les passages d'un programme à un autre sont provoqués par des signaux d'interruption. Bien entendu, chaque fois qu'un programme est interrompu, les informations nécessaires à sa reprise ultérieure doivent être sauvegardées. La multiprogrammation permet enfin de rétablir l'interactivité entre l'utilisateur et l'exécution de son programme, par l'intermédiaire des systèmes temps partagés, plusieurs utilisateurs travaillent

simultanément, chacun avec son propre terminal, et l'unité centrale est attribuée successivement à chacun d'eux, pendant une fraction de temps: la rapidité des temps de réponse peut donner à l'utilisateur l'impression qu'il est seul à disposer de l'ordinateur .C'est dans cet esprit qu'à été conçu le système Multics (multipliplexed information and computing service). Bien que ce système ne soit pas répandu de façon significative, les idées qu'il a introduits ont eu une influence considérable sur la conception des systèmes d'exploitation ultérieurs.

La miniaturisation des circuits intégrés a permis la fabrication des mini-ordinateurs, qui a commencé vers 1961 avec le DEC PDP-1. Le développement des mini-ordinateurs de la famille PDP est tout à fait caractéristique de la troisième génération et c'est sur un PDP-7 que les premières versions du système UNICS (UNIPLEXED INFORMATION AND COMPUTING SERVICE) ont été réécrites. ce système rebaptisé UNIX, a été réécrit dans un nouveau langage évolué ,le C, et implanté sur de nombreux ordinateurs (VAX, SM90, SUM, etc..).

Enfin l'avènement des micro-ordinateurs (1980-1994) fait de l'informatique un produit grand public. Ces ordinateurs personnels sont caractérisés par des systèmes interactifs, d'importantes possibilités graphique et des logiciels agréables pour les utilisateurs Le système MS-DOS de microsoft est utilisé sur IBM PC ainsi que plusieurs machines ayant une unité centrale INTEL 8088.

Parallélement se développent les systèmes multiprocesseurs et des réseaux (ARPANET, SNA, TRANSPAC, ETHERNET, INTERNET), qui relient plusieurs ordinateurs, parfois géographiquement éloignés. Ils son gérés par des systèmes d'exploitations qui ont à résoudre des problèmes tout à fait inédits, résultant des délai d'acheminement non négligeable des messages. Ils permettent une meilleur utilisation des ressources et offrent de très grandes capacités de calcul et de stockage.

1.2 OBJECTIFS ET FONCTIONS ESSENTIELLES.

L'évolution historique met en lumière deux objectifs fondamentaux du système d'exploitation. D'une part, il construit, sur la machine physique telle qu'elle est livrée par le constructeur, une machine virtuelle plus facile d'emploi et plus conviviale pour l'utilisateur. D'autre part, il prend en charge la gestion complexe des ressources de l'ordinateur (processeur, mémoire ,périphériques d'entrée -sortie, etc..), en optimise l'utilisation et en permet le partage entre les utilisateurs.

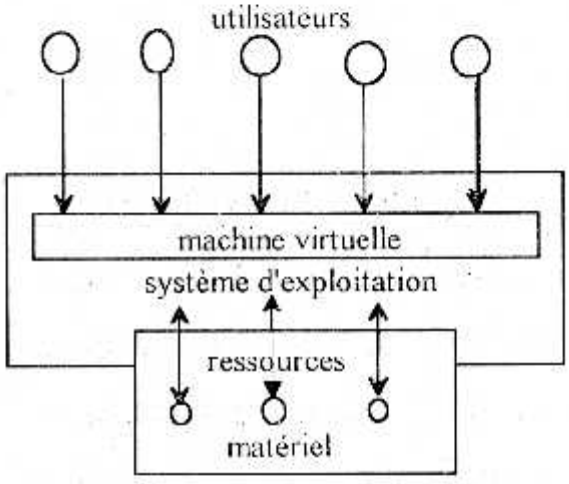


fig 3: Objectifs d'un système d'exploitation

1.3 PROCESSUS

La notion , qui éclaire le fonctionnement des systèmes multiprogrammé est celle de processus. ce terme a été introduit dans les années 60, comme abstraction de l'activité d'un processeur. La raison en est simple : Dans les systèmes multiprogrammé monoprocesseur, le processeur ne peut travailler que pour un seul utilisateur à la fois. Pendant ce temps, les autres utilisateurs sont en attente, non pour des raisons internes, mais uniquement parce que le processeur est indivisible. Le problème est donc d'exprimer qu'un programme est en cours

d'exécution, bien qu'en fait il ne progresse pas, dans l'attente du processeur. La solution consiste à distinguer un programme de son exécution, que nous appelons processus, et à introduire une notion d'état.

Un programme est une entité statique, associée à la suite des instructions qui le composent. Un processus est une entité dynamique associé à la suite des actions réalisées par un programme, lors d'une exécution particulière. Cette suite est indépendante du nombre de fois des instants où le processus a été mis en attente du processeur.

1.3.1 ETAT D'UN PROCESSUS

L'état d'un processus permet d'exprimer si le processus est réellement exécuté, s'il est seulement prêt à être exécuté c'est-à-dire en attente du processeur ou encore s'il est en attente de ressource autre que le processeur ou d'un évènement extérieure (terminaison d'une opération d'entrée-sortie par exemple).

Nous distinguerons donc les états " en exécution ", " prêt ", ce qui signifiera en attente du processeur, ou " en attente " , ce qui signifiera en attente d'une autre ressource que le processeur. Ces trois états ne sont, bien entendu, qu'une première approximation, et on peut définir d'autres, comme "extérieur" et "terminé". Comme il convient de distinguer un processus en attente d'un événement qui va se produire, d'un processus en attente d'un événement qui ne se produira jamais, nous introduirons l'état "bloqué" et nous utiliserons le terme de blocage (deadlock) pour décrire cette seconde situation. Un processus peut être mis temporairement en retrait par le système: il est alors placé dans l'état "suspendu". Les systèmes d'exploitation réels considèrent généralement plus d'états que ceux qui viennent d'être présentés.

D'une part, ils distinguent les différentes causes d'attente et, d'autre part ,l'attente en mémoire centrale et en mémoire secondaire.

affectées sont libérées, son bloc de contrôle est effacé et il disparaît des tables et des listes.

* Mettre en attente et réveiller

Lorsqu'un processus a besoin de ressources autres que l'unité centrale , pour poursuivre sa progression, et que ces ressources ne peuvent lui être affectées dans l'immédiat, il est placé par le système d'exploitation dans l'état "en attente" Lorsque, plus tard , le système d'exploitation est en mesure de lui affecter les ressources demandées, il réveille" le processus, ce qui a pour effet de le faire passer dans l'état "prêt" , signifiant que le processus dispose de toutes les ressources qui lui sont necessaires, sauf l'unité centrale.

* Suspendre et reprendre

La suspension est une opération qui amène un processus dans l'état "suspendu", où sa progression est figée et où il cesse donc d'exécuter des actions. Il s'agit d'un retrait passager de ce processus, comparable à une forme plus sévère de mise en attente. La suspension d'un processus provoque la sauvegarde de son contexte, contenant toutes les informations nécessaires à sa reprise ultérieure. Lorsque cette reprise intervient, le processus repasse à l'état "prêt" ou à l'état "en attente ", suivant l'état où était intervenue sa suspension. Lorsqu'il passe, encore plus tard, à l'état "en exécution", il retrouve le contexte qui était le sien au moment de sa suspension. Pour garantir une certaine équité entre les processus, le système d'exploitation doit éviter de laisser trop longtemps un processus dans l'état "suspendu".

Un processus, ayant fourni des résultats partiels et attendant des données de la part d'autre processus, peut être suspendu jusqu'à ce que ces données aient été calculées. Ou bien, si le système d'exploitation a admis trop de processus et si

ses performances sont de ce fait, réduites de manière importante, il peut suspendre certains.

* Changer de priorité

Changer la priorité d'un processus se fait très simplement en modifiant l'information correspondante dans son bloc de contrôle. Lorsqu'un algorithme d'ordonnancement utilise les priorités des processus comme paramètres, Le système par souci d'équité, doit augmenter périodiquement les priorités des processus qui ne sont pas choisis.

1.3.4 - PHASES DE CALCUL ET PHASE D'ENTREES-SORTIES.

L'exécution d'un programme se décompose en une alternance de périodes où l'unité centrale est utilisée et de périodes où s'accomplissent des opérations d'entrée-sortie. Nous appellerons les premières des phases de calcul et les secondes des phases d'entrée-sortie.

1.4. RESSOURCES

1.4.1 - DEFINITION

On appelle ressources les éléments qui contribuent à la progression des processus. Certains relèvent du matériel : l'unité centrale et les processeurs, la mémoire centrale et les mémoires secondaires , bandes magnétiques ou disques, les périphériques d'entrée - sortie (lecteurs, écrans, claviers, imprimantes, scanners, contrôleurs de disque ou de bande), les bus et les réseaux, etc. D'autres sont de nature plus logicielle : les programmes d'application (compilateurs, assembleurs, éditeurs de texte, tableurs, grapheurs, bibliothèques de fonction) et les processus système (ordonnanceurs, distributeurs, chargeurs, éditeurs de liens), ou des fichiers. Puisqu'un processus peut en appeler un autre, un processus est lui même une ressource.

I.4.2 - CLASSES DE RESSOURCES

Nous appelons classe de ressource le couple constitué d'une collection d'éléments regroupés sous un nom commun et de l'ensemble des opérations que l'on peut leur appliquer. Chaque élément est une unité de la ressource considérée ou, plus simplement , une ressource.

Toutes les unités d'une même classe sont supposées indiscernables : Lorsqu'un processus fait la demande d'une ressource d'une certaine classe, toute unité disponible de cette classe lui convient. Lorsqu'une ressource peut être à nouveau demandée après sa libération, elle est dite réutilisable. C'est le cas en particulier des ressources matérielles. Au contraire d'autre ressources sont consommables , c'est -à - dire qu'elle disparaissent après usage (messages). Les ressources ont des états internes qui peuvent ou non être modifiés par leur utilisateur. Si l'état d'une ressource est susceptible d'être modifié, il doit être réinitialisé après chaque libération et la ressource ne peut donc pas être partagée par plusieurs processus

Le système d'exploitation doit contrôler l'utilisation des ressources dans une table indiquant si la ressource est disponible ou non, et si elle est allouée, à quel processus. A chaque ressource est associée une file d'attente contenant les blocs de contrôle des processus qui l'attendent (appelée queue d'exploitation).

Chaque fois qu'un nouveau processus fait une demande de la ressource et que cette dernière n'est pas utilisable , son bloc de contrôle est ajouté à la file d'attente.

Dans certains cas, on autorise une opération supplémentaire sur certaines ressources: la préemption. Lorsque la ressource est allouée et qu'une demande survient de la part d'un processus prioritaire, cette ressource peut être retirée au processus en cours et attribuée au processus prioritaire. Naturellement ,l'état de

la ressource doit être sauvegardé en cas de préemption, de sorte qu'elle puisse ensuite être rendue au premier processus.

1.5 ORDONNANCEMENT

1.5.1 SCHEDULER:

Dans un système multiprogrammé monoprocesseur, le **SCHEDULER** est le processus système qui définit l'ordre dans lequel les processus prêts acquièrent la ressource unité centrale et la durée pendant laquelle ils l'utilisent. Ils s'agit d'une fonction qui est à la base du concept de multiprogrammation, puisque c'est en échangeant très rapidement les processus qui disposent de l'unité centrale, que le système d'exploitation parvient à lui conserver un taux d'utilisation élevé, augmentant ainsi les performances globales.

Le terme scheduler reçoit en fait un sens plus large pour désigner un processus qui régit l'accès à des ressources telles que : l'ensemble formé par plusieurs processus dans un système multiprocesseur, la mémoire centrale, les périphériques d'entrée-sortie, etc.

1.5.2 - FILES D'ATTENTE

Lorsque plusieurs processus demandent l'allocation d'une ressource qui n'est pas disponible, mais indispensable à leur progression, ces processus doivent attendre. Comme nous l'avons vu ci-dessus, le système d'exploitation garde trace des demandes non satisfaites et des processus mis en attente, en gérant une file d'attente. Lorsque la ressource redevient disponible, le système d'exploitation doit être capable de sélectionner un des processus en attente et de la lui affecter. Le choix du processus dépend de l'algorithme d'ordonnancement et peut varier suivant les systèmes.

1.5.3 - TYPES DE SCHEDULERS.

Un scheduler est nécessaire chaque fois que le nombre de processus demandeurs d'une ressource dépasse la capacité d'accès simultanés de cette ressource. Ainsi trouve-t-on un scheduler à différents points clés du système.

* Le scheduler à long terme (JOB SCHEDULER) détermine si un processus utilisateur qui le demande peut effectivement entrer dans le système. Cette décision dépend d'informations sur les performances du système à l'instant considéré. Par exemple, si les temps de réponse deviennent élevés, parce qu'il y a trop de processus présents dans le système, le scheduler à long terme peut choisir de différer l'entrée de nouveaux arrivants.

* Le scheduler de l'unité centrale (cpu scheduler) choisit dans la file d'attente de cette dernière le processus à qui sera allouée l'unité centrale.

Les fréquences d'appel de ces deux schedulers sont très différentes. Le scheduler de l'unité centrale est nécessairement activé au bout d'une période maximum qui correspond à la durée de la phase de calcul du processus précédent . Le processus à long terme est appelé beaucoup moins fréquemment, en fonction de l'arrivée de nouveaux utilisateurs.

Une fois le choix d'un processus effectué, le scheduler de l'unité centrale appelle un processus système nommé le **Dispatcher, dont** le rôle est de donner effectivement le contrôle de l'unité centrale au processus choisi.

1.5.4 ALGORITHMES D'ORDONNANCEMENT..

Un algorithme d'ordonnancement choisit, parmi les processus qui sont dans la file d'attente d'une ressource, le prochain à qui sera attribuée cette ressource. Les critères de choix tendent naturellement à l'utilisation la meilleure possible de la ressource et, bien entendu, diffèrent suivant le type de ressource considéré. Deux cas sont particulièrement importants : celui où la ressource est un processeur,

unique (l'unité centrale) ou non (système multiprocesseur) et celui où la ressource est une unité de disque.

L'ordonnancement des processus d'un système multiprogrammé réalise l'attribution des processus libres au processus candidats. Ce problème a suscité de nombreux travaux [KRAKOWIAK 87; PETERSON 83] décrivent les plus connus.

On peut distinguer les ordonnancements non préemptifs et préemptifs.

- Les ordonnancements non-préemptifs sont caractérisés par le fait qu'un processus qui détient un processeur le monopolise jusqu'à ce qu'il le libère volontairement ou lorsqu'il se termine .Dans cette famille on impose souvent de connaître le temps de service demandé par un processus à activer, et cette connaissance est généralement impossible. Différentes méthodes ont été proposées, qui permettent de calculer une estimation de ce temps de service .Elles conduisent à des solutions diverses, mais la plus part font intervenir l'histoire (le passé) du système pour tenter de prédire plus ou moins implicitement son futur proche .

La stratégie HIGHEST- RESPONSE- RATIO- NEXT (HRN), par exemple introduite par [BRINCH HANSEN 71], établit une relation d'ordre total entre les processus qui attendent de s'exécuter. Cette relation est définie en associant à chaque processus un nombre réel appelé sa priorité . C'est une fonction du temps de service t_s demandé par le processus et du temps t_a pendant lequel ce processus a attendu ce service. Sa priorité p est donnée par:

$$p = \frac{t_a + t_s}{t_s}$$

- Dans un ordonnancement préemptif, le processeur exécute à tout instant le processus le plus prioritaire parmi ceux qui attendent de s'exécuter. Son service se poursuit jusqu'à la fin si et seulement s'il n'y a pas occurrence de processus de

plus haute priorité. Trois situations peuvent se présenter pour les processus dont le service a été interrompu. Ce dernier est de nouveau pris en charge dés qu'il n'y a plus de processus plus prioritaire que lui:

- avec une mémorisation du service déjà acquis (preemptive-resume discipline): le service reprend là où il a été interrompu.
- avec la même durée de service dés le début (preemptive-repeat without resampling)
- avec une nouvelle durée de service indépendante (preemptive-repeat with resampling).

La priorité d'un processus peut - être fixe ou évoluer dynamiquement en fonction de son histoire passée ou de paramètres fixés par le concepteur. Là encore, de nombreux algorithmes de calcul ont été proposés.

Une famille particulière d'ordonnancements regroupe les algorithmes qui implémentent des stratégies mixtes. Ainsi, par exemple, l'organisation en files multiniveaux (multilevel queues) réalise une partition de l'ensemble des processus qui attendent le service d'un processus ou qui s'exécutent. Chaque classe est une file d'attente qui regroupe un ensemble de processus de même priorité. La relation d'ordre est donc établie sur les files de processus , et non plus sur les processus eux mêmes (DEITEL84; PETERSON 87]. L'algorithme de choix du scheduler, c'est-à-dire le distributeur de la ressource unité centrale, ou dispatcher, élit les processus de la file non vide la plus prioritaire selon la discipline dite du "tourniquet" (Round-Robin) ou RR.

Cette étude envisage une méthode d'ordonnancement préemptif par priorité dans un système multi-programmé monoprocesseur, appelée O.P.B.M (ordonnancement à priorités bornées moyenne), (voir chap.3), la discipline proposée prend en compte des paramètres temporels définis dans le passé total ou récent, de chaque processus, et en cela elle ressemble à la discipline HRN dont elle s'inspire pour le mode de calcul des priorités. Mais elle définit des

classes de processus en bornant leur priorité, ce qui permet de la doter , par exemple, des caractéristiques d'une gestion de files multiniveaux tout en étant incomparablement plus simple à implémenter. Il pose cependant un problème : Dans le cas d'un système à source fini, les démonstrations d'équité se basent sur une hypothèse principale [HARO- PROUST 92] " pendant un intervalle de temps Δt, le temps d'exécution T d'un processus τ_i $(1 \leq i \leq n)$ est proportionnel à sa priorité. Cette hypothèse n'a pu être vérifiée par les mathématiques déterministes.

C'est pour cela que le sujet a été proposé, nous cherchons à déterminer si la problématique des ordonnanceurs (Schedulers) ne relèvent pas de celle des problèmes de bandit.

CHAPITRE 2

2- PROCESSUS DE DÉCISION

2.1. PROCESSUS DE DECISION MARKOVIEN

2.1.1.Introduction :

Un système évolue dans le temps $t = 0, 1, 2, \ldots$. Soit $E = \{0, 1, 2, \ldots\}$ l'espace des états possibles du système. Après chaque observation, une décision est choisie dans un ensemble fini de décisions permise, D.

Si le système est dans l'état i à l'instant t et la décision d est prise, alors deux événements se produisent :

(i) Un gain moyen $r(i, d)$ est immédiatement engendré, tel que $|r(i,d)| \leq K$, où K est une constante.

(ii) Le prochain état du système est déterminé par la probabilité de transition $P_{ij}(d)$.

Si on note par X_t l'état du système à l'instant t, et par d_t la décision prise à l'instant t, l'hypothèse (ii) est équivalente à :

$$P\{X_{t+1} = j / X_0, d_0, X_1, d_1, \ldots, X_t = i, d_t = d\} = P_{ij}(d)$$

Ainsi, les gains et les probabilités de transition sont fonction uniquement du dernier état et de la décision antérieure.

Une stratégie pour un processus de décision Markovien est une règle quelconque qui permet de choisir une décision, à chaque instant t, en fonction de l'histoire du système.

Une stratégie randomisée est une règle qui permet de choisir une décision d avec une certaine probabilité P_d, $d \in D$.

Une stratégie est dite stationnaire si la décision prise à l'instant t dépend seulement de l'état en cours et n'est pas randomisée.

La donnée d'une stratégie stationnaire f et d'une séquence d'états $\{X_t, t = 0, 1, 2, \ldots\}$ engendre une chaîne de Markov de probabilité de transition $P_{ij} = P_{ij}[f(i)]$. Ce qui justifie l'appellation processus de décision Markovien.

Le taux d'actualisation étant $\alpha \in]0, 1[$.

Nous désignerons par $V_\pi(x)$, l'espérance de la valeur actuelle du gain total lorsqu'on par de l'état x en adoptant la stratégie π.

Formellement on a donc :

$$V_\pi(i) = E_\pi\left[\sum_{t=0}^{\infty} \alpha^t r(X_t, d_t) / X_0 = i\right] , \quad i \geq 0 \qquad (1)$$

Notons que (1) est bien définie, puisque les gains sont bornés et $\alpha < 1$. ce qui implique $|V_\pi(x)| < K/(1 - \alpha)$

2.1.2.. Equations d'optimalité et stratégie optimale :

Posons :

$$V(i) = \sup_\pi V_\pi(i) \qquad i \geq 0$$

Une stratégie π^* est dite α-optimale si

$$V_{\pi^*}(i) = V(i) \qquad \text{pour tout } i \geq 0$$

Théorème 1 [Ross 70]

La fonction $V(.)$ vérifie l'équation d'optimalité

$$V(i) = \max_d \left[r(i, d) + \alpha \sum_j P_{ij}(d)\, V(j) \right] \qquad i \geq 0 \quad (2)$$

Preuve :

" $\leq$ " Soit π une stratégie arbitraire qui choisit la décision d à l'instant 0 avec une probabilité P_d, $d \in D$, alors :

$$V_\pi(i) = \sum_{d \in D} P_d \left[r(i, d) + \sum_{j=0}^{\infty} P_{ij}(d)\, W_\pi(j) \right]$$

où $W_\pi(j)$ représente l'espérance de la valeur actuelle du gain, lorsqu'on démarre de l'état j à l'instant 1 et qu'on adopte la stratégie π. Par suite, il s'ensuit que :

$$W_\pi(j) \leq \alpha\, V(j)$$

et de plus

$$V_\pi(i) \le \sum_{d \in D} P_d \left[r(i, d) + \alpha \sum_j P_{ij}(d) V(j) \right]$$

$$\le \sum_{d \in D} P_d \max_d \left[r(i, d) + \alpha \sum_j P_{ij}(d) V(j) \right] \qquad (3)$$

$$= \max_d \left[r(i, d) + \alpha \sum_j P_{ij}(d) V(j) \right]$$

puisque π est arbitraire, (3) implique

$$V(i) \le \max_d \left[r(i, d) + \alpha \sum_j P_{ij}(d) V(j) \right] \qquad (4)$$

" ≥ " Soit d_0 tel que

$$r(i, d_0) + \alpha \sum_j P_{ij}(d_0) V(j) = \max_d \left[r(i, d) + \alpha \sum_j P_{ij}(d) V(j) \right] \qquad (5)$$

Soit π la stratégie qui choisit la décision d_0 à l'instant 0 ; et si le prochain état est j, alors considérons le processus à partir de l'état j qui suit la stratégie π_j tel que $V_{\pi_j}(j) \ge V(j) - \varepsilon$; $j \ge 0$. Par conséquent

$$V_\pi(i) = r(i, d_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(d_0) V_{\pi_j}(j)$$

$$\ge r(i, d_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(d_0) V(j) - \alpha\varepsilon$$

Puisque $V(i) \ge V_\pi(i)$, on a alors

$$V(i) \ge r(i, d_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(d_0) V(j) - \alpha\varepsilon$$

d'après (5), on obtient :

$$V(i) \ge \max_d \left\{ r(i, d_0) + \alpha \sum_{j=0}^{\infty} P_{ij}(d_0) V(j) \right\} - \alpha\varepsilon \qquad (6)$$

d'où puisque ε est quelconque, d'après (4) et (6) on a le résultat.

Théorème 2 [Ross 70]

Soit f une stratégie stationnaire, telle que

$$r[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)] V(i) = \max_d \left\{ r(i, d) + \alpha \sum_{j=0}^{\infty} P_{ij}(d) V(j) \right\} \quad , \quad i \ge 0$$

alors

$$V_f(i) = V(i) \quad \text{pour tout} \quad i \ge 0$$

et par suite f est α-optimale.

Preuve :

Soit $B(E)$ l'ensemble des fonctions bornées sur E et $t_f : B(E) \to B(E)$ l'opérateur défini par :

Pour toute stratégie stationnaire f et $u \in B(E)$

$$(T_f u)(i) = r[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)]\, u(j) \qquad (7)$$

Lemme : Pour $u \in B(E)$, et f une stratégie stationnaire on a $\lim_{n \to \infty} T_f^n u = V_f$, avec $T_f^1 = T_f$, et pour n ≥ 1 $T_f^n = T_f\left(T_f^{n-1}\right)$.

Preuve :

$$\left(T_f^2 u\right)(i) = r[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)]\, (T_f u)(j)$$

$$= r[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)] \left[r[j, f(j)] + \alpha \sum_{k=0}^{\infty} P_{jk}[f(j)]\, u(k) \right]$$

$$= r[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)]\, r[j, f(j)] + \alpha^2 \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} P_{ij}[f(i)] P_{jk}[f(j)]\, u(k)$$

$\left(T_f^2 u\right)$ représente donc l'espérance de la valeur actuelle du gain sur deux périodes à venir en suivant la stratégie f avec un gain final $\alpha^2 u$. Par récurrence, on peut voir facilement que $T_f^n u$ représente l'espérance de la valeur actuelle du gain sur n périodes à venir, en suivant la stratégie f, en terminant avec un gain final $\alpha^n u$. Puisque $\alpha < 1$ et u borné, on a le résultat.

En appliquant l'opérateur T_f à V, on obtient

$$(T_f V)(u) = r[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)]\, V(j)$$

$$= \max_d \left\{ r[i, f(i)] + \alpha \sum_{j=0}^{\infty} P_{ij}[f(i)]\, V(j) \right\}$$

$$= V(i) \text{ d'après le théorème 1}$$

Par conséquent :

$$T_f V = V$$

ce qui implique que

$$\begin{aligned} T_f^2 V &= T_f(T_f V) \\ &= T_f V \\ &= V \end{aligned}$$

et par récurrence

$$T_f^n V = V \qquad \text{pour tout n.}$$

donc pour $n \to \infty$, et d'après le lemme précédent $V_f = V$.

C.Q.F.D.

Proposition .3. [Ross 70]

V est l'unique solution bornée de l'équation d'optimalité (1).

Preuve:

Supposons que U(i), $i \geq 0$, est une fonction bornée qui satisfait l'équation d'optimalité

$$U(i) = \max_d \left[r(i, d) + \alpha \sum_j P_{ij}(d)\, U(j) \right], \quad i \geq 0$$

pour un i fixé soit $\bar{d}$ tel que

$$U(i) = r\left(i, \bar{d}\right) + \alpha \sum_j P_{ij}\left(\bar{d}\right)\, U(j)$$

par suite, puisque V satisfait l'équation d'optimalité, nous avons

$$\begin{aligned} U(i) - V(i) &= r\left(i, \bar{d}\right) + \alpha \sum_j P_{ij}\left(\bar{d}\right)\, U(j) - \\ &\quad - \max_d \left[r(i, d) + \alpha \sum_j P_{ij}(d)\, V(j) \right] \\ &\leq \alpha \sum_j P_{ij}(\bar{d}) \left[U(j) - V(j) \right] \\ &\leq \alpha \sum_j P_{ij}(\bar{d}) \left| U(j) - V(j) \right| \\ &\leq \alpha \sum_j P_{ij}(\bar{d}) \sup_j \left| U(j) - V(j) \right| \\ &\leq \alpha \sup_j \left| U(j) - V(j) \right| \end{aligned}$$

en inversant les rôles de U et V on peut conclure que:

$$V(i) - U(i) \leq \alpha \sup_j |V(j) - U(j)|$$

par suite

$$|V(i) - U(i)| \leq \sup_j |V(j) - U(j)|$$

et

$$\sup_i |V(i) - U(i)| \leq \alpha \sup_j |V(j) - U(j)|$$

ce qui implique (puisque $\alpha < 1$) que

$$\sup_j |V(j) - U(j)| = 0$$

c.q.f. d.

2.2. PROCESSUS DE DECISION SEMI-MARKOVIEN

2.2.1. Définition

Un processus de décision semi-Markovien est caractérisé par:

1/ Un espace des états Θ (supposé dénombrable) muni de la σ- algèbre χ de parties de Θ, qui contient tous les singletons de Θ.

2/ Un espace des actions A muni de la σ- algèbre ξ de parties de A, qui contient tous les singletons de A, et l'on donne pour chaque $x(\in \Theta)$ un ensemble d'actions $\Omega(x)$ $(\in \xi)$ possibles lorsque le système est dans l'état x.

3/ Une transition P de $(\Theta \times A, \chi \otimes \xi)$ dans (Θ, χ).

4/ Une durée de transition τ qui est une variable aléatoire de fonction de répartition F.

5/ Une fonction de gain r (supposé borné).

Chaque fois que le système se trouve dans l'état $x(\in \Theta)$ à l'instant t et que l'action $u(\in \Omega(x))$ est prise:

a/ Un gain $e^{-\beta t}\ r(x,u)$ ($\beta>0$) est engendré immédiatement (on suppose que les gains sont continuellement actualisés avec un taux d'intérêt β (>0), tel qu' une unité monétaire reçue à l'instant t a une valeur actuelle de $e^{-\beta t}$).

b/ L'état du système à l'instant suivant sera dans $\Gamma(\in \chi)$ avec la probabilité $P(\Gamma/x,u)$.

c/ Sachant qu'une transition de x vers y doit avoir lieu , la durée de cette transition $\tau(\in$Borélien B), est tirée au sort suivant la loi conditionnelle $F(B/x,y,u)$.

Quant le système atteint l'état y , une autre décision est prise, etc....

Les fonctions $\Omega(\bullet)$, $r(\bullet,u)$ *et* $P(\Gamma/\bullet,u)$ sont χ – *mesurables* et $F(B/\bullet,\bullet,u)$ est χ^2 – mesurable.

Afin de s'assurer qu'un nombre infini de transitions ne surviennent pas dans un intervalle de temps fini, la condition suivante est souvent imposée.

Condition A :

$\exists\delta(>0)$ et $\varepsilon(>0)$ tels que: :

$$\sum_{y\in\Theta} F(]\delta,\infty[\ /x,y,u\ )\ P(y,x,u) > \varepsilon \qquad (x \in \Theta, u \in \Omega(x)).$$

où $]\delta,\infty[$ représente l'intervalle ouvert à droite.

Sens :

Cette condition signifie que la probabilité que l'intervalle de temps séparant deux dates de décisions, et par suite entre transitions, soit plus grande que δ est toujours supérieure à ε.

Un processus de décision Markovien est un processus de décision semi-Markovien tel que pour un certain c(>0), $F(\{c\}/x,y,u) = 1, \forall x,y$ et u.

Ainsi l'intervalle de temps séparant deux dates de décisions consécutives prend une valeur constante c. Sans perte de généralité, on supposera dans la suite que c=1.

Une stratégie π, pour un processus de décision semi-Markovien, est une règle de décision (peut être randomisée), qui pour chaque instant de transition t, spécifie:

* La décision à appliquer à l'instant t (comme fonction de t),

* L'état à l'instant t

* L'ensemble des dates de décisions antérieures, les états du processus et les actions appliquées à ces instants.

Ainsi l'action à appliquer à l'instant t dépend entièrement de l'histoire antérieure du processus, dans cette situation on dira qu'elle est déterminée de manière séquentielle.

Une stratégie qui maximise l'espérance de la valeur actuelle du gain dans un horizon infini sur tout l'ensemble des stratégies, et pour tout état initial est dite optimale.

Une stratégie non randomisée est dite déterministe.

Une stratégie est dite stationnaire si elle n'inclut pas de dépendance temporelle, en d'autres termes, elle correspond à la même décision g(x) chaque fois que le système visite l'état x, où $g:\Theta \to A$. Dans ce cas on parlera de stratégie stationnaire g au lieu de π (par abus de langage).

Elle est dite Markovienne si la décision prise à l'instant t est indépendante des états et des décisions aux instants de transitions antérieurs à t.

2.2.2 Le Principe d'optimalite

Beaucoup de travaux sur le contrôle stochastique sont basés sur le principe d'optimalité. Donnons un exposé succint.

Supposons que $U^*(s), \tau \leq s \leq t$ est l'action optimale d'un problème (P) avec origine en $s = \tau$ et $X(\tau)$ donné . Considérons un sous problème avec origine en $s = \tau_1 > \tau$ alors: $u^*(s), \tau_1 \leq s \leq t$ est aussi optimale pour le problème.

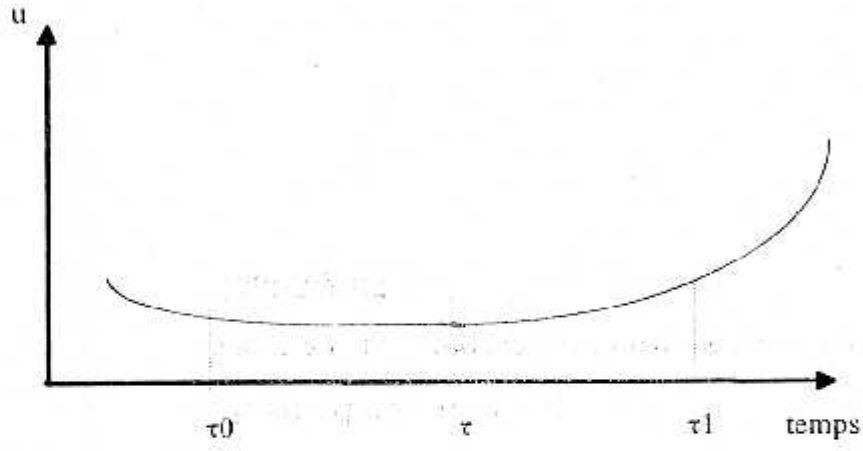


FIG 6 :Une illustration du principe d'optimalité.

La fonction u(t) est l'action optimale, fonction du temps . Si u(t) est optimale partant de τ,elle est aussi optimale partant de τ_1.

Le principe d'optimalité nous permet d'aboutir à une équation de la programmation dynamique dont la résolution permet de trouver l'action optimale.

Désignons par Z(t) le gain cumulé pendant l'intervalle [0,t] et x l'état initial. Pour une stratégie π donnée et un taux d'intérêt β (> 0) on définit

$$V_x(x) = E_\pi \left[\int_0^\infty e^{-\beta t} dZ(t) / X(0) = x \right]$$

– l'espérance de la valeur actuelle du gain dans un horizon infini lorsqu'on part de l'état initial x et qu'on suit la stratégie π.

Soit $V(x) = \sup_\pi V_\pi(x)$ la valeur optimale en cas d'actualisation.
D'après le principe d'optimalité de BELLMAN les équations fonctionnelles sont vérifiées

$$V(x) = \max_{u \in \Omega(x)} \left\{ r(x,u) + \sum_{y \in \Theta} \int_0^\infty e^{-\beta t} V(y)\, P\,[y/x,u]\, F(y, dt/x,u) \right\}$$

pour tout $x \in \Theta$. *Avec* $V: \Theta \to R$.

Ces équations expriment l'idée intuitive suivante:
La valeur optimale, partant de $x \in \Theta$, réalise l'optimum du gain immédiat ajouté à la valeur optimale, à partir du nouvel état qui suit.

Théorème 1 [S. Ross (1970)]

Si la condition A est remplie , $r(\bullet, \bullet)$ est bornée et $\Omega(x)$ est fini $\forall x \in \Theta$, *alors*
(i) Il existe une stratégie optimale qui est déterministe, stationnaire et Markovienne.

(ii) Soit π une stratégie stationnaire ,déterministe et Markovienne , et soit $V_\pi(x)$ l'espérance de la valeur actuelle du gain lorsqu'on part de l'état x et qu'on suit la stratégie π. Pour tout $x \in \Theta$ et $u \in \Omega(x)$ soit

$$V(x) = r(x,u) + \sum_{y\in\Theta} \int_0^{\infty} e^{-\beta t}\, V(y)\, P[y/x,u]\, F(y,dt\,/\,x,u)$$

si $V_\pi(x) \geq V(x)$ *pour* tout $x\in \Theta$ et $u\in \Omega(x)$, *alors* π est optimale.

On se limitera aux stratégies déterministes, stationnaires et Markoviennes. Une telle stratégie est définie par une fonction g qui est χ–mesurable sur Θ telle que $g(x) \in \Omega(x), \forall x$.

Preuve:

même principe que dans le cadre des processus de décision Markovien.

2.3. PROCESSUS DE BANDIT

2.3.1. Définition:

Un processus de bandit est un processus de décision semi-Markovien muni d'une stratégie π et $\forall x \in \Theta$, on ajoute à l'ensemble d'actions $\Omega(x)$ l'élément 0. L'action 0 gèle le processus au sens où $P(\{x\}/x,0) = 1$ et $r(x,0) = 0\ \forall x \in \Theta$.
A chaque instant de décision, soit l'action 0 ou l'action donnée par π est appliquée.

2.3.2. Famille de processus de bandit.

Une famille F de processus de bandit est dite à composantes alternatives ,si elle forme un ensemble $\{B_i, i= 1,..., n\}$ de processus de bandits ayant tous le même taux d'intérêt β (> 0) telle que:

a/ La date 0 est une date de décision pour tous les processus .A cette date l'action 0 est appliquée à toutes les composantes excepté un seul processus noté (B_i disons),pour lequel toutes les actions peuvent lui être appliquées, dans ce cas on dira que B_i est actionné, et ce jusqu'à la prochaine date de décision de B_i .

b/ A cet instant, un nouveau processus (noté B_j , B_j peut être B_i) sera déterminé pour être actionné , tandis que les autres sont gelés jusqu'à la prochaine date de décision de B_j .....

c/ L'espace des états de F est le produit de tous les espaces des états des B_i ,qui la compose. Le gain est augmenté à chaque étape uniquement par le processus en activité.

d/ L'ensemble des décisions à chaque date de décision est l'union ,à l'exclusion de l'action 0, de tous les ensembles de décisions de chaque composante de F. Ainsi sélectionnée l'action u de l'ensemble des décisions de la composante B_i signifie que la composante B_i est sélectionnée pour être actionné et l'action u est appliquée à B_i.

Ainsi à chaque étape les B_i (i = 1,..., n) sont alternativement candidats à la continuation. En l'absence de contraintes de précédences restreignant l'ensemble des processus de bandit qui peuvent être choisis à chaque instant à la continuation ,la famille F est dite simple et notée (FS).

Une stratégie pour une famille de processus de bandit à composantes alternatives est un algorithme qui fait un choix à tout instant t, du processus de bandit qui doit être actionné.

L'espérance de la valeur actuelle du gain est donc

$$E\left[\sum_{i=1}^{n}\sum_{j=1}^{\infty} e^{-\beta T_i(j)} r_{K_i(j)}/U_i(T_i(j))\right]$$ où $T_i(j)$ *et* $K_i(j)$ est la jème date de décision et la jème action à l'instant $T_i(j)$ respectivement pour le processus B_i .

U_i ($T_i(j)$) est 1 ou 0 suivant que le processus B_i est actionné ou gelé à l'instant $T_i(j)$.

Le problème d'optimisation consiste à trouver une stratégie qui maximise le revenu total moyen.

2.3.3. LA Notion de Job [Gittins 89]

- On appelle JOB, un processus de bandit possédant un état particulier (noté c) appelé état d'achèvement (complétion state). Ce modèle considère que les gains ne sont versés uniquement qu'à la fin du job.
- une famille de tels processus permet de modéliser le problème bien connu d'ordonnancement de plusieurs jobs sur une seule machine.
- Ordonnancer les jobs, signifie sélectionner à chaque date de décision, et suivant certains critères, un job incomplet jusqu'à ce que tous les jobs soient terminés.

CHAPITRE 3

3. MODELISATION DE L'O.P.B.M

Dans ce chapitre, il est question de modéliser le fonctionnement de l'O.P.B.M sous forme d'un problème de Bandit . On verra qu'en introduisant une structure de coût appropriée , le problème peut se ramener à celui de la minimisation du flow-time* pondéré.

3.1- PRESENTATION.

Soit un flot poissonnien de processus qui sont en cours d'exécution à un instant donné sur un système monoprocesseur. La structure de données qui contrôle l'allocation du processeur aux processus est une liste unique appelée QUEUE D'EXPLOITATION. Elle ordonne totalement les processus prêts à s'exécuter en fonction de leur priorité. A tout instant , c'est le processus le plus prioritaire de la queue d'exploitation qui s'exécute. L'ordonnancement considéré est donc préemptif. Lorsque plusieurs processus ,en tête de la queue d'exploitation, ont la même priorité, la discipline adoptée est RR. Chacun de ces processus reçoit alors à tour de rôle un quantum de temps processeur t_q .L a queue d'exploitation est réalisée, par exemple, par une liste doublé qui chaîne les descripteurs des processus prêts à s'exécuter [HARO.93]. Les descripteurs de la liste sont ordonnés selon les priorités décroissantes des processus qu'ils décrivent. Le processus le plus prioritaire est en tête de liste .La priorité $p_i(t)$, à tout instant t, de tout processus τ_i de la file évolue entre deux réels positifs m_i et M_i .Elle est donc caractérisée, par:

$$\forall i \ \& \ \forall t \geq 0 : 0 \leq m_i \leq p_i(t) \leq M_i \qquad (1)$$

Un processus dont la priorité est fixe est donc associé à des bornes égales. On ne s'intéresse pas ici à la façon dont sont établies les bornes m_i et M_i associées à chacun des $\tau_i$de la file .Les conditions (1) conduisent à la définition:

**Le temps de completion d'un processus de classe j représente l'instant de son départ moins l'instant de son arrivée dans la classe j. La somme de tous les temps de completions de tous les processus dans le système est appelée flow-time.*

DEFINITION N° 1

On appelle Ordonnancement par Priorites Bornees, ou O.P.B, la stratégie d'ordonnancement préemptif, par priorités, qui résulte des conditions (1).

3.1.1- EXEMPLES DE STRATEGIES REALISABLES PAR O.P.B.

Ce paragraphe montre comment on peut doter OPB des propriétés de quelques stratégies particulières bien connues.

1°) Soient m^-et M^+ deux réels positifs tels que $m^- < M^+$ .OPB devient un simple ordonnancement par priorités dynamiques lorsque :$\forall i : m_i = m^-$ et $M_i = M^-$, car alors $\forall i : p_i \in [m^-, M^+]$.La priorité de chaque processus évolue alors selon l'algorithme adapté.

2°) L'ordonnancement est RR lorsque :$\forall i : m_i = M_i = p$ puisque les processus ont tous la même priorité dans ce cas.

3°) Un ordonnancement par priorités statiques, préemptif ou non ,est obtenu lorsque $\forall i : m_i = M_i = p_i(0)$ non obligatoirement différent de $p_j(0)$ lorsque $i \neq j$. Chaque processus dispose de sa propre priorité qui reste fixe .C'est alors un ordonnancement adapté aux systèmes temps - réel.

4°) L'ordonnancement en files multiniveaux est obtenu de la façon suivante .Soit $\Re$ la relation définie sur l'ensemble des processus par:

$$\forall i \ \& \ \forall j : \Re(\tau_i, \tau_j) \Leftrightarrow (m_i = m_j \ \& \ M_i = M_j).$$

Deux processus sont en relation par $\Re$ s'ils ont le même intervalle d'évolution de leur priorité.$\Re$ est une relation d'équivalence sur l'ensemble des processus. Soit K le nombre de classes d'équivalences. Les processus d'une classe quelconque sont ordonnancés selon leur priorité .A chaque classe k, est associé un intervalle $I_k = [m_k, M_k]$ pour $1 \leq k \leq K$.

O.P.B implémente une stratégie équivalente à un ordonnancement en files multiniveaux lorsque les $(I_k)_{k=1,...K}$ sont deux à deux disjoints,c'est-à-dire lorsque:

$$\forall k, 1 \leq k \leq K \ \& \forall h, 1 \leq h \leq K : (k \neq h \Rightarrow I_k \cap I_h = \varnothing).$$

Chaque classe modélise alors une file d'attente, mais l'ordonnancement d'une file quelconque est réalisé selon les priorités des processus de cette classe. Dans un classique ordonnancement en files multiniveaux, la stratégie d'allocation du processeur aux processus de chaque file est RR et le quantum t_q de temps CPU peut être différent d'une file à l'autre. Il est d'autant plus faible que la priorité associée aux processus de la file est plus élevée. O.P.B possède exactement ces propriétés si on force une priorité fixe , commune à tous les processus d'une file: $\forall k, 1 \leq k \leq K : m_k = M_k = p_k$. L'attribution de quanta distincts aux différentes files est un problème d'implémentation. Il est donc indépendant de O.P.B, ou de quelque stratégie que ce soit d'ailleurs. Le concepteur reste libre d'implémenter la politique d'attribution de quanta qu'il souhaite .

5°) Pour un ordonnancement en files multiniveaux avec rebouclages introduite par CTSS [CRISMAN 65] et MULTICS, les processus doivent pouvoir changer de file à mesure qu'ils utilisent le processeur [DEITEL 84; KRAKOWIAK 87]. Ceci se réalise simplement en modifiant la priorité p_k dans le descripteur du processus et en réorganisant la queue d'exploitation.

Les différentes stratégies implémentables par OPB le sont donc par une simple adaptation des bornes associées, au besoin en forçant une valeur particulière de la priorité et, éventuellement ,la valeur du quantum .Les gestionnaires des files d'attente ordonnées selon la priorité des processus, comme la queue d'exploitation par exemple, ne sont pas influencés par une modification de stratégie. Ainsi, les algorithmes d'insertion et d'extraction, par exemple restent indépendants de la stratégie particulière réalisée par OPB, et ceci grâce à une implémentation très soignée des structures de données gérées par l'ordonnanceur.

DEFINITION N° 2

On appelle Ordonnancement par Priorites Bornees Moyennes, o.u O.P.B.M, l'ordonnancement O.P B lorsque la priorité d'un processus est calculée par (3).

3.2- MODELISATION.

Les processus sont répartis en K classes disjointes de priorités:

$$\forall k, 1 \le k \le K \;\&\; \forall h, 1 \le h \le K : [m_k, M_k] \cap [m_h, M_h] = \varnothing$$

Le flot L_k, $(1 \le k \le K)$, des processus de classe k est poissonnien de paramètre λ_k , $\left(\Lambda_k = \sum_{i=1}^{k} \lambda_i\right)$..

Les flots L_1,..., L_K, sont indépendants les uns des autres.

Soit $F_k(t)$ la fonction de répartition du temps de service X_k des processus de classe k et $\psi_k(\beta) = \int_0^{\infty} e^{-\beta t} dF_k(t)$ sa transformée de Laplace Stieltjes (LST), pour un β fixé on la note simplement par ψ_k pour k=1,.....K.

Notre hypothèse concernant la distribution du temps de service est que

$$F_k(t) = 1 - e^{-\mu_k t} \qquad (\mu_k \ge 0) \qquad (1 \le k \le K) \qquad (4)$$

Les processus de classe k sont plus prioritaire que les processus de classe h si et seulement si $k < h$.

L'admission en service est garantie sur la base des priorités préemptive-resume.

On suppose que le séjour d'un processus de classe k dans le système durant une unité de temps est pénalisé par un coût h_k , l'achèvement du service d'un processus de classe k est rémunéré par un gain r_k.

Les coûts et les gains sont continuellement actualisés avec un taux d'intérêt$\beta(>0)$, telle que une unité monétaire reçue à l'instant t a une valeur actuelle de $e^{-\beta t}$.

Soit n_k le nombre de processus de classe k dans la file (en comptant également le processus en cours de service).

L'état du système est représenté par $s=(n_1,\ldots,n_K)$ et l'espace des états est $S=\{s=(n_1,\ldots,n_K)\cdot n_1+\ldots+n_K<\infty\}$

Pour un état s, l'ensemble des actions disponibles est $A(s)=\{o\}\cup\{k : n_k>0\}$, où l'action k (>0) correspond à l'affectation du processeur aux processus de classe k. Par contre l'action 0 correspond à l'option d'oisiveté; elle s'applique à l'ensemble des autres classes.

Formulons notre modèle comme un processus de décision semi-Markovien.

Les dates de décisions sont l'instant de préemption , l'instant de fin de service ou l'instant d'arrivée d'un nouveau processus dans le système vide.

Posons pour k >0,

$\delta_k=(0,\ldots..0,1,0,\ldots..0)$ est le $k^{ème}$ vecteur unité et $\delta_0=(0,\ldots.....0)$ est le vecteur nul.

Si un processus de classe k est exécuté et que durant son traitement, le vecteur arrivée des processus suivant des flots poissonniens est $a=(a_1,\ldots,a_K)$ alors l'état de la prochaine date de décision est $s-\delta_k+a$ si son service est achevé et $s+a$ sinon.

Si tous le système est oisif et le processus qui arrive est de classe i, alors le prochain état est $s+\delta_i$.

La structure de coût dans notre modèle est le gain pur r_k, qui est immédiatement reçu à la date de décision quand un processus de classe k est choisi pour être traité. Pour simplifier les notations on pose $r_0=0$.

Notons par:

$V_\pi(s)$ l'espérance de la valeur actuelle du gain ,dans un horizon infini, lorsqu'on démarre de l'état initial s et la stratégie π est employée.

On définit les fonctions suivantes:

$Q_k(t)$ = le nombre de processus de classe k dans le système au temps t.

$A_k(t)$ = le nombre de processus de classe k entrés dans la file suivant un flot poissonnien de paramètre λ_k entre les instants 0 et t, sans compter les processus initialement présents.

$D_k(t)$ = le nombre de processus de classe k sortis de la file d'attente entre les instants 0 et t

D'après ces définitions on a:

$$Q_k(t) = Q_k(0) + A_k(t) - D_k(t) \quad \text{pour tout } t > 0 \qquad (5)$$

et $$D_k(t) \le Q_k(0) + A_k(t) \qquad \text{pour tout } t > 0 \qquad (6)$$

Sous la stratégie π, l'espérance de la valeur actuelle du gain total dans un horizon infini, en démarrant d'un état initial s, dans le problème général est donnée par

$$V_\pi(s) = E_\pi\left[\sum_{k=1}^{K} r_k \int_0^\infty e^{-\beta t}\, dD_k(t) - \sum_{k=1}^{K} h_k \int_0^\infty e^{-\beta t} Q_k(t)dt\right] \qquad (7)$$

Par une intégration par partie ,on obtient la relation

$$E\int_0^\infty e^{-\beta t}Q_k(t)dt = [\frac{Q_k(0)}{\beta} + E\int_0^\infty e^{-\beta t}A_k(t)\, dt$$

$$+\beta^{-1}[e^{-\beta t}D_k(t)]_0^\infty - \beta^{-1}\int_0^\infty e^{-\beta t}dD_k(t)\ ] \qquad (8)$$

D'après (6), le 3ème terme dans la partie droite de (8) tend vers zéro et le second terme vaut

$$E\int_0^\infty e^{-\beta t}A_k(t)dt = \frac{\lambda_k}{\beta^2},\ 1 \le k \le K.$$

En substituant (8) dans (7), on obtient

$$V_\pi(s) - E_\pi\left[\sum_{k=1}^{K}(r_k + h_{k/\beta})\int_0^\infty e^{-\beta t}dD_k(t)\right] - H \qquad (9)$$

où $$H = \sum_{k=1}^{K}(h_k/\beta)(Q_k(0) + \lambda_k/\beta)$$

On transforme le gain r_k à la fin d'un traitement et le coût de séjour h_k au gain immédiat r_k^* à la date de décision comme suit:

$$r_k^* = r_k + h_k/\beta \qquad (1 \le k \le K) \qquad (10)$$

Dans le problème de bandit on se donne des gains purs, et les projets gelés ne produisent aucun gain. Grâce à cette transformation l'ordonnancement des processus qui minimise l'espérance du coût linéaire de séjour, est formulé comme un problème de bandit.

Donc nous avons :

$$V_\pi(s) = E_\pi\left[\sum_{n=1}^{\infty} e^{-\beta T(n)} r_{K(n)}\right] - H$$

$$= V_\pi^*(s) - H. \qquad (11)$$

où T(n) et K(n) (n=1,...) représentent la $n^{ème}$ date de décision et la $n^{ème}$ action à l'instant T(n) respectivement.

Puisque H est indépendante de π,en utilisant (10), le problème général et le problème de gain pur sont équivalents.

Par conséquent notre modèle est un problème de bandit avec arrivées poissonniènnes et un nombre fini de classes.

Le problème consiste donc à déterminer la stratégie optimale maximisant $V_\pi(s)$.

CHAPITRE 4

4- CONSTRUCTION DE LA STRATEGIE OPTIMALE

Ce chapitre est consacré à la construction d' une stratégie optimale d'ordonnancement de toutes les classes de processus , pour ce faire on développera une expression pour la transformée de Laplace-Stieltjes de la période d'activité générée par une classe quelconque, en se basant essentiellement sur les résultats classiques de la période d'activité pour un système M/M/1 à source infinie, moyennant certaines approximations, puisque la discipline de service considérée est supposée être préemptive- resume , ensuite on donnera l'expression générale de l'espérance de valeur actuelle du gain cumulé durant la période d'activité citée ci dessus et enfin, en utilisant quelques résultas de la programmation dynamique stochastique la condition suffisante d'optimalité de la stratégie est donnée..

4.1- PERIODE D'ACTIVITE

Dans ce paragrahe, on rappelle comment est calculée la distribution de la période d'activité dans le cas classique.

4.1.1- DEFINITION

Une période d'activité débute par l'arrivée dans le système vide d'un processus et s'achève lorsque pour la première fois le système redevient vide.

4.1.2- DISTRIBUTION

Soit B la variable aléatoire représentant la durée de la période d'activité dans un système M/GI/1 à l'état stationnaire.

On cherche la distribution $H(y) = P[B \leq y]$.

Supposons que le processus P_0 initialise la période d'activité , nécessite x unités de temps de traitement et que durant son exécution il y a eu n arrivées de processus P_1 , P_2,..., P_n.

La distribution H(y) est indépendante de l'ordre de passage des processus dans la station de service .Cet ordre va être modifié de façon à faire apparaître des sous périodes d'activités i dont le début d_i et la fin f_i sont analogues aux époques d et f de la période d'activité. Pour ce faire, après que P_0 soit complétement exécuté ,le

processeur traitera P_1 et les processus qui arriveront durant son traitement jusqu'à ce que P_2,..... P_n soient les seuls processus restant dans le système.
A ce point, de la même manière le processeur exécutera le processus P_2 et les processus qui arriveront durant son traitement jusqu'à ce que P_3 ,..., P_n soient les seuls processus restant dans le système. Ce déroulement sera répété jusqu'à l'exécution finale de P_n ,et les processus qui arriveront durant son traitement, le système redevient vide et la période d'activité s'achève.
Soit D_i le temps écoulé entre le début d'exécution de P_i et le début d'exécution de P_{i+1} ,$i<n$, et D_n l'intervalle de temps débutant avec l'exécution de P_n et se terminant avec la fin de la période d'activité. Soient X et N les variables aléatoires représentant respectivement ,la durée de traitement de P_0 et le nombre d'arrivées durant son temps d'exécution .
On pose

$F(x) = P[X \leq x]$ et $P[N=n] = e^{-\lambda x}\frac{(\lambda x)^n}{n!}$ $(n \in N)$ (nombre d'arrivées durant x).

On constate que le nombre n de sous-périodes d'activités est égal au nombre de processus qui sont arrivés pendant le service de P_0 .. Chaque sous période $d_i f_i$ est initialisée par le service d'un seul processus, tout comme la période df elle même et les services sont indépendants et identiquement distribués, il en est donc de même des n sous - périodes $d_i f_i$ et de la période entière df. Donc D_i $(1 \leq i \leq n)$ sont des variables aléatoires indépendantes, ayant toutes la même distribution de la période d'activité H(y).
Ainsi la distribution conditionnelle de la période d'activité sachant que P_0 nécessite x unités de temps de traitement et qu'il y a eu n arrivées durant son exécution, a la transformée de Laplace-Stieltjes suivante.

$$E[e^{-sB}/X = x, N = n] = E[e^{-s(x+D_1+\ldots+D_n)}] = E[e^{-sx}.e^{-sD_1}.....e^{-sD_n}]$$

Compte tenu de l'indépendance des durées X et D_i on pourra écrire:

$$E[e^{-sB}/X = x, N = n] = E[e^{-sx}] .E[e^{-sD_1}].....E[e^{-sD_n}]$$

Les D_i étant distribués comme B ; x étant une constante, il vient:

$$E[e^{-sB}/\ X=x,N=n] = e^{-sx}[E(e^{-sD_i})]^n$$

Posons

$$E[e^{-sB}] = \int_0^\infty e^{-sy}\, dH(y) = \Psi_B(s)$$ la transformée de Laplace-Stieltjes de B

On a:

$$E[e^{-sD_i}] = \Psi_B(s)$$ et par suite $E[e^{-sB}/X = x, N = n] = e^{-sx}[\Psi_B(s)]^n$

En levant la condition sur N , on a

$$1-\ E[s^{-sB}/\ X= x] = \sum_{n=0}^{\infty} E[e^{-sB}/\ X-x,N-n]\ Pr\{N = n\}$$

Les arrivées étant poissonniènnes de paramètre λ et n étant le nombre d'arrivées durant x, on a donc

$$E[e^{-sB}/X = x] = \sum_{n=0}^{\infty} e^{-sB}\,[\Psi_B(s)]^n\,\frac{(\lambda x)^n}{n!}\,e^{-\lambda x}$$

$$= \exp-\{x[s+\lambda-\lambda\Psi_B(s)]\}$$

$$2-\ \Psi_B(s) = E[e^{-sH}] = \int_0^\infty \exp-\{x[s+\lambda-\lambda\Psi_B(s)]\}dF(x$$

d'où l'équation fonctionnelle:

$$\Psi_B(s) = \Psi_X([s+\lambda-\lambda\Psi_B(s)]$$

4.1.3. REMARQUE

Si nous analysons maintenant la période fd d'inactivité du processeur, il est trivial d'affirmer qu'elle est distribuée suivant la loi des arrivés $\lambda\exp(-\lambda x)$ du fait de la nature Markovienne même du processus de ces arrivées et sa propriété d'être sans mémoire .Donc sa transformée de Laplace-Stieltjes est donnée par

$$\Psi_I(s) = \frac{\lambda}{\lambda+s}$$

4.2- FONCTION DE GAIN ET INDICE

4.2.1-CAS où $h_1 = h_2 = \ldots\ldots = h_K = 0$

L'état initial du système est noté par $s=(n_1,n_2,\ldots\ldots,n_K) \in S$,soit $B_0(s)=0$ et $B_k(s)\equiv$le premier instant (peut être ∞) où tous les processus de la classe là k, sont complètement traités. Dans le langage des files d'attente $B_k(s)$ represente la durée de la période d'activité initiale générée par la classe k.

En particulier $B_k(s)$ represente le premier instant où le processeur devient libre.

On définit :

$$\pi_0(s) = E[e^{-\beta B_0(s)}] = 1 \quad et \quad \pi_k(s) = E[e^{-\beta B_k(s)}] \quad (1 \le k \le K)$$

A présent soit:

$U_k(s)$ = l'espérance de la valeur actuelle du gain cumulé dans $[0, B_k(s)]$.

On a donc :

$$V(s) = U_K(s) + \pi_K(s)V(0) \qquad (8)$$

où 0 est un K-Vecteur à composantes toutes nulles.

Fixons k pour le moment ($1 \le k \le K$), et développons une expression pour $\pi_k(s)$.

LEMME 1:

Pour $1 \le k \le K$ fixé on a

$\pi_k(s) = \alpha_{1k}^{n_1} \ldots\ldots \alpha_{kk}^{n_k}$ où

$\alpha_{j0} = \psi_j(\beta), \quad \alpha_{jk} = \psi_j[\beta + \Lambda_k(1-\alpha_k)], \quad \Lambda_k = \sum_{i=1}^{k} \lambda_i \quad et$

α_k est l'unique solution dans (0,1) de l'équation

$$\alpha_k = \Psi_k[\beta + \Lambda_k(1-\alpha_k)]$$

avec

$$\Psi_k(\theta) = \sum_{i=1}^{k} (\lambda_i/\Lambda_k)\psi_i(\theta) \; ; \theta > 0, \; 1 \le k \le K.$$

PREUVE:

Soit k fixé pour le moment $1 \le k \le K$

Appelons :

α_{jk} la transformée de Laplace-Stieltjes (à variable β)du temps requis pour vider le système des processus des classes 1à k sachant que l'intervalle a débuté avec le service d'un processus de classe j (j=1,...,k).

Alors on a pour la discipline de service preemptive-resume, PATEROK et ETTL (eq.27, p.1154)

$$\alpha_{jk} = \psi_j[\beta + \Lambda_k(1-\alpha_k)] \quad (1 \le j \le k \le K) \tag{9}$$

où α_k est l'unique solution de l'équation fonctionnelle

$$\alpha_k = \Psi_k[\beta + \Lambda_k(1-\alpha_k)] \quad (1 \le k \le K). \tag{10}$$

avec

$$\Psi_k(\theta) = (\lambda_1/\Lambda_k)\psi_1(\theta) + \ldots + (\lambda_k/\Lambda_k)\psi_k(\theta) \quad (\theta > 0, 1 \le k \le K) \tag{11}$$

α_k represente la transformée de Laplace-Stieltjes (à variable β) de la distribution générale de la période d'activité pour notre système M/M/1(réduit à k classes.)

on peut voir facilement que:

$$0 < \alpha_k < 1 \tag{12}.$$

et

$$\pi_k(s) = E[e^{-\beta B_k(s)}]$$

$$= \prod_{j=1}^{k} \psi_j[[\beta + \Lambda_k(1-\alpha_k)]]^{n_j} \tag{13}$$

Donc (13) peut être réecrite comme suit:

$$\pi_k(s) = \alpha_{1k}^{n_1} \ldots\ldots \alpha_{kk}^{n_k} \qquad (1 \le k \le K) \tag{14}$$

Notre prochain objectif est de développer une expression générale pour $V_k(s)$.

Pour cela définissons l'indice d'une classe k,comme suit:

$C_k = U_k(0,\ldots 0,\infty, n_{k+1}, \ldots n_K) \qquad (1 < k < K))$

C'est en fait l'espérance de la valeur actuelle du gain total lorsqu'on démarre avec une infinité de processus dans la classe k,et aucun processus dans les classes 1 à k-1.

LEMME 2 Pour tout k, $1 \le k \le K$, on a

$$U_k(s) = \sum_{i=1}^{k} [\pi_{i-1}(s) - \pi_i(s)]C_i \quad \text{avec}$$

$$C_1 = \psi_1(\beta)\,\psi_1(\mu_1)\,r_1 / [1 - \psi_1(\beta)] \quad \text{et}$$

$$C_k = \psi_k(\beta)\psi_k(\mu_k)r_k + \sum_{i=1}^{k-1} (\alpha_{k,i-1} - \alpha_{ki})C_i/(1 - \alpha_{k,k-1}) \quad (2 \le k \le K)$$

Les π_i et les α_{kj} sont données au lemme 1 pour tous i,j et k .

PREUVE:

Considérons une classe k ($1 \le k \le K$) quelconque et supposons que l'état initial est $s' = (n_1, \ldots n_{k-1}, \infty, n_{k+1}, \ldots n_K)$. Posons $U_0(s) = 0$ par convention, on a alors

$$U_k(s') = U_{k-1}(s) + \pi_{k-1}(s)C_k \qquad (15)$$

Définissons T comme étant le premier instant (peut être ∞) où le nombre cumulé de processus traités dans la classe k dépasse le nombre cumulé de processus arrivés à la classe k de n_k et le système est vidé des processus de la classe 1 jusqu'à la classe k-1. Il est clair que T a la même distribution que $B_k(s)$ et l'espérance de la valeur actuelle du gain produit jusqu'à l'instant T est $U_k(s)$.
De plus ,puisque l'espérance de la valeur actuelle du gain produit après l'instant T est C_k, nous obtenons l'expression alternée

$$U_k(s') = U_k(s) + \pi_k(s)C_k \qquad (16)$$

En combinant (15) et (16) on obtient:

$$U_k(s) = U_{k-1}(s) + [\pi_{k-1}(s) - \pi_k(s)]C_k \qquad (1 \le k \le K) \qquad (17)$$

Par récurrence on a donc

$$U_k(s) = \sum_{i=1}^{k} [\pi_{i-1}(s) - \pi_i(s)]C_i \; (1 \le k \le K) \qquad (18)$$

A présent déterminons les constantes C_i $(1 \le i \le K)$
Soit k fixé ($1 \le k \le K$)et supposons que l'état initial du système est $s = (n_1, \ldots n_K)$ avec $n_k > 0$.Supposons que le processeur soit affecté d'abord vers la classe k àl'instant zéro et ensuite l'admission en service est garantie sur la base des priorités usuelles. Si $n_1 = n_2 = \ldots = n_{k-1} = 0$, on se retrouve avec la discipline de service usuelle; en d'autres termes cette discipline représente une légère déviation de la procédure normale.
Soit Y_k ≡le temps de traitement initial écoulé jusqu'à la première interruption d'un processus de classe k.
Soit $Z_k = \min (X_k, Y_k)$. où X_k représente la durée de service d'un processus de classe k.

Posons $B'_0(s) \equiv Z_k$ et $B'_j(s)$ ≡le premier instant (peut être ∞) après Z_k où le système est vidé des processus de la classe 1 jusqu'à la classe j, $1 \le j \le K$,

$$\pi'_j(s) = E\left[e^{-\beta B'_j(s)}\right], \quad 0 \le j \le K \quad \text{et}$$

$U'_j(s)$=L'espérance de la valeur actuelle du gain produit jusqu'à l'instant $B'_j(s)$, $0 \le j \le K$.

D'après la propriété d'absence de mémoire de la loi exponentielle Y_k est identiquement distribuée que X_k.

$$Z_k = \min(X_k, Y_k)$$

$$= \begin{cases} Y_k \text{ si } Y_k \le X_k \\ X_k \text{ sinon} \end{cases}$$

on a

$$P(Y_k \le X_k) = \int_0^\infty P(Y_k \le x) dP(X_k \le x)$$
$$= \int_0^\infty F_k(x)\, dF_k(x)$$
$$= 1 - \psi_k(\mu_k)$$

donc

$$Z_k = \begin{cases} Y_k \text{ avec } 1 - \psi_k(\mu_k) \\ X_k \text{ avec } \psi_k(\mu_k) \end{cases}$$

Par suite

$$\pi'_0(s) = \psi_k(\beta) \text{ et } U'_0 = \psi_k(\beta)\, \psi_k(\mu_k) r_k$$

En utilisant les mêmes arguments que ceux pour trouver (14) on peut voir que

$$\pi'_j(s) = \begin{cases} \alpha_{kj}\pi_j(s) \text{ si } 1 \le j \le k-1 \\ \pi_j(s) \text{ si } k \le j \le K \end{cases} \qquad (19)$$

En adoptant par convention la notation $\alpha_{k0} = \psi_k(\beta)$,on peut prolonger (19) comme suit

$$\pi'_j(s) = \begin{cases} \alpha_{kj}\pi_j(s) \text{ si } 0 \le j \le k-1 \\ \pi_j(s) \text{ si } k \le j \le K \end{cases} \qquad (20)$$

A présent, en procédant exactement comme dans (17) on construit deux différentes expressions de $U_j'(s)$ dans le cas où $n_i = \infty$

En effet, on peut voir que

$$U_j'(s) = U_{j-1}'(s) + [\pi_j'(s) - \pi_j'(s)]C_j \quad (1 \le j < K) \qquad (21)$$

Par récurrence on obtient:

$$U_j'(s) = U_0'(s) + \sum_{i=1}^{j} [\pi_i'(s) - \pi_i'(s)]C_i \qquad (1 \le j \le K) \quad (22)$$

Or on a

$U_k'(\delta_k) = U_k(\delta_k)$ Et puisque d'après (14) on a

$$\pi_0(\delta_k) = \ldots\ldots = \pi_{k-1}(\delta_k) = 1 \text{ et } \pi_k(\delta_k) = \alpha_{kk}, \quad 1 \le k \le K$$

(19) nous donne

$$\pi_i'(\delta_k) = \alpha_{ki}, \quad 0 \le i \le k \le K$$

et (22) donne aussi

$$U_k(\delta_k) = \psi_k(\beta)\psi_k(\mu_k)r_k + \sum_{i=1}^{k} (\alpha_{k,i-1} - \alpha_{ki})C_i \quad (1 \le k \le K) \qquad (23)$$

Mais (18) donne aussi

$$U_k(\delta_k) = (1 - \alpha_{kk})C_k \quad (1 \le k \le K) \qquad (24)$$

En combinant (23) et (24) on obtient :

pour k=1, $C_1 = \psi_1(\beta)\psi_1(\mu_1)r_1/[1 - \psi_1(\beta)]$

par contre pour k>>1,on obtient la formule récursive:

$$C_k = \psi_k(\beta)\psi_k(\mu_k)r_k + \sum_{i=1}^{k-1} (\alpha_{k,i-1} - \alpha_{ki})C_i/(1 - \alpha_{k,k-1}) \quad (2 \le k \le K)$$

LEMME 3.

$$V(o) = \sum_{i=1}^{K} (\omega_i - \omega_{i-1})C_i/(\beta + \omega_K) \quad \text{avec}$$

$$\omega_0 = 0 \text{ et } \omega_k = \Lambda_k(1 - \alpha_k) \quad (1 \le k \le K)$$

PREUVE

Sous l'hypothèse que le système est initialement vide, la transformée de Laplace-Stieltjes (LST) de la durée où le premier processus arrive est $\Lambda_K/\Lambda_K + \beta$, et de plus la probabilité que ce dernier soit de classe k est λ_k/Λ_k, indépendamment de l'instant d'arrivée.Notons par

$U^*(0)$=l'espérance de la valeur actuelle du gain produit durant le premier cycle d'activité .

Nous avons donc

$$U^*(0)= [\Lambda_K/(\Lambda_K + \beta)]\sum_{k=1}^{K} (\lambda_k/\Lambda_K)U_k(\delta_k) \qquad (25)$$

Puisque d'aprés (18) on a

$U_K(\delta_k) = \sum_{i=1}^{K} [\pi_{i-1}(\delta_k) - \pi_i(\delta_k)]C_i$, (25) se réduit à

$$U^*(0) = \sum_{k=1}^{K} \lambda_k \sum_{i=1}^{K} [\pi_{i-1}(\delta_k) - \pi_i(\delta_k)]C_i/(\Lambda_K + \beta)$$

$$= \sum_{i=1}^{K} \left[\sum_{k=1}^{K} \lambda_k \pi_{i-1}(\delta_k) - \sum_{k=1}^{K} \lambda_k \pi_i(\delta_k) \right] C_i/(\Lambda_K + \beta) \qquad (26)$$

Définissons $\omega_0 = 0$ et $\omega_k = \Lambda_k(1 - \alpha_k)$, $(1 \le k \le K)$

en utilisant l'identité $\alpha_k = \sum_{i=1}^{k} (\lambda_i/\Lambda_k)\alpha_{ik}$ $\qquad (1 \le k \le K)$ $\qquad (27)$

on vérifie que:

$$\sum_{k=1}^{K} \lambda_k \pi_i(\delta_k) = \Lambda_K - \omega_i \ (0 \le i \le K)$$

Ainsi (26) se réduit à

$$U^*(0) = \sum_{i=1}^{K} (\omega_i - \omega_{i-1})C_i/(\Lambda_K + \beta)$$

A présent notons que la (LST) (à variable β) de la période d'activité globale pour notre système à priorité est justement α_K ;(elle est égale à la (LST) de la période d'activité pour le système M/GI/1 discuté précédemment).Ainsi la (LST) du cycle d'activité est $\Lambda_K\alpha_K/(\beta + \Lambda_K)$.Par suite,en utilisant les relations d'indépendances évidente entre les cycles d'activité successifs,on obtient

$$V(0) = \sum_{n=0}^{\infty} [\Lambda_K\alpha_K/(\Lambda_K + \beta)]^n U^*(0)$$

$$= [(\Lambda_K + \beta)/(\beta + \Lambda_K - \Lambda_K\alpha_K)]U^*(0)$$

$$= \sum_{i=1}^{K} (\omega_i - \omega_{i-1})C_i/(\beta + \omega_K).$$

4.2.2-CAS GENERAL

Notons l'état initial par $s=(n_1,\ldots,n_K)$ et donnons l'expression générale de $V(s)$. Posons

$$\Lambda_k = \sum_{i=1}^{k} \lambda_i, \quad 1 \le k \le K$$

$$\text{et} \quad \Psi_k(\theta) = \sum_{i=1}^{k} (\lambda_i/\Lambda_k)\psi_i(\theta), \theta > 0, \quad 1 \le k \le K.$$

On définit α_k comme l'unique solution positive de l'équation

$$\alpha_k = \Psi_k[\beta + \Lambda_k(1-\alpha_k)], \quad (1 \le k \le K)$$

et soit $\alpha_{j0} = \psi_j(\beta) \qquad (1 \le j \le K)$,

$$\alpha_{jk} = \psi_j[\beta + \Lambda_k(1-\alpha_k)], \quad (1 \le j,k \le K))$$

$$\pi_0(s) = 1 \text{ et } \pi_k(s) = \alpha_{1k}^{n_1} \ldots\ldots\ldots \alpha_{kk}^{n_k}, \ (1 \le k \le K)$$

$$\omega_0 = 0 \text{ et } \omega_k = \Lambda_k(1-\alpha_k) \quad (1 \le k \le K)$$

$$r_k^* = r_k + h_k/\beta \quad (1 \le k \le K)$$

On donne l'expression générale par

$$V(s) = V^*(s) - \sum_{k=1}^{K} (h_k/\beta)(Q_k(0) + \lambda_k/\beta)$$

où $\quad V^*(s) = U_K(s) + \pi_K(s)V^*(0) \qquad (28)$

$$U_K(s) = \sum_{k=1}^{K} [\pi_{k-1}(s) - \pi_k(s)]C_k \text{ et}$$

$$V^*(0) = \sum_{k=1}^{K} (\omega_k - \omega_{k-1})C_k/(\beta + \omega_k)$$

et les constantes $C_1,\ldots\ldots,C_K$ sont définies récursivements par les formules

$$C_1 = \psi_1(\beta)\psi_1(\mu_1)r_1^*/[1-\psi_1(\beta)] \quad \text{et}$$

$$C_k = \psi_k(\beta)\psi_k(\mu_k)r_k^* + \sum_{i=1}^{k-1} (\alpha_{k,j-1} - \alpha_{ki})C_i/(1-\alpha_{k,k-1}) \quad (2 \le k \le K)$$

4.3-OPTIMALITE

Soit K*, satisfaisant $1 \leq K^* \leq K$ (le cas K* =0 est exclut pour le moment).
Supposons que les classes de processus sont numérotées arbitrairement ,on définit la règle de décision

$$f(s) = \begin{cases} 0 \text{ si } n_1 = \ldots = n_{K^*} = 0 \\ \min\{k : n_k > 0\} \text{ sinon} \end{cases}$$

et g est la stratégie déterministe, stationnaire et Markovienne associée à f.

Notre objectif de départ est de développer une condition suffisante pour la β−optimalité de g. Pour cela notons par V(.) l'espérance de la valeur actuelle du gain associé à g.

Puisqu'on a transformé le problème général à un cas similaire de gain pur, la fonction V(.) n'est pas affectée par les classes K*+1,.....,K.

Ainsi en se basant sur l'équation (28) on obtient:

$$V(s) = U_{K^*}(s) + \pi_{K^*}(s)\, W_{K^*} \quad (s \in S)$$

$$\text{où} \quad U_{K^*}(s) = \sum_{k=1}^{K^*} [\pi_{k-1}(s) - \pi_k(s)] C_k \quad (s \in S) \qquad (29)$$

$$\text{et} \quad W_{K^*}(s) = \sum_{k=1}^{K^*} (\omega_k - \omega_{k-1}) C_k / (\beta + \omega_{K^*})$$

On utilisera le terme "période d'activité initiale" pour qualifier le premier instant (peut être ∞) où le système est vidé des processus de la classe 1 à la classe K*.
Donc $\pi_{K^*}(s)$ représente la (LST) de la période d'activité initiale (à variable β),et $U_{K^*}(s)$ représente l'espérance de la valeur actuelle du gain produit durant la période d'activité initiale. D'après (29) il apparaît donc que W_{K^*} est justement V(0).

A présent fixons k ($1 \leq k \leq K$), supposons que $s=(n_1,\ldots,n_K)$ où $n_k>0$, et que le processeur soit d'abord affecté vers la classe k , mais la stratégie g est suivie par la suite.

Appelons $V_k(s)$ l'espérance de la valeur actuelle du gain total qui lui correspond.

Soit Y_k le temps de traitement initial écoulé jusqu'à la première interruption d'un processus d'une classe k.

Soit $Z_k = \min (X_k , Y_k)$ ou X_k représente la durée de service d'un processus de classe k et $B'_{K^*}(s)$ dénote le premier instant (peut être $+\infty$) après Z_k où le système est vidé des processus de la classe 1 à K*.

Soit $U'_{K^*}(s)$ = l'espérance de la valeur actuelle du gain produit jusqu'à l'instant $B'_{K^*}(s)$, et définissons

$$\pi'_{K^*}(s) = E[\exp(\beta B'_{K^*}(s))].$$

Il apparaît donc que

$$V_k(s) = U'_{K^*}(s) + \pi'_{K^*}(s)\, W_{K^*} \qquad (30)$$

de plus d'après l'équation (22) on a

$$U'_{K^*}(s) = \psi_k(\beta)\,\psi_k(\mu_k) r_k^* + \sum_{i=1}^{K^*} [\pi'_{i-1}(s) - \pi'_i(s)] C_i \qquad (31)$$

où
$$\pi'_i(s) = \begin{cases} \alpha_{ki}\pi_i(s) \text{ si } 0 \le i \le k-1 \\ \pi_i(s) \text{ si } k \le i \le K \end{cases} \qquad (32)$$

En combinant (29) avec (30),(31) et (32) on obtient les propositions suivantes:

LEMME 1:

Si $1 \le k \le K^*$, $s \in S$ et $k \in A(s)$, alors :

$$V(s) - V_k(s) = (1 - \psi_k) C_1 - \psi_k(\beta)\,\psi_k(\mu_k) r_k^* - \sum_{i=1}^{k-1} (1 - \alpha_{ki}) \pi_i(s)(C_i - C_{i+1}).$$

LEMME 2:

Si $K^* \le k \le K$, $s \in S$ et $k \in A(s)$, alors :

$$V(s) - V_k(s) = (1 - \psi_k) C_1 - \psi_k(\beta)\,\psi_k(\mu_k) r_k^* - \sum_{i=1}^{K^*-1} (1 - \alpha_{ki}) \pi_i(s)(C_i - C_{i+1}) -$$

$$-(1 - \alpha_{kK^*}) \pi_{K^*}(s)(C_{K^*} - W_{K^*}).$$

A présent remarquons que $(1-\alpha_{ki}) > 0$ pour tous i et k et que $\pi_i(s)$ est non croissante en n_j pour chaque i et j.Ce qui va suivre est immédiat d'après les lemmes 1 et 2.

LEMME 3:

Supposons que $1 \le k \le K, s \in S, k \in A(s)$ et $C_1 \ge .... \ge C_{K^*} \ge W_{K^*}$, alors :

$$V(s) - V_k(s) \ge V(\delta_k) - V_k(\delta_k).$$

Remarquons à présent que

$$V(\delta_k) - V_k(\delta_k) = 0 \quad \text{si} \quad 1 \le k \le K^*. \tag{33}$$

Puisque l'admission du processeur aux processus de classe k est précisément l'action donnée par la stratégie g dans l'état δ_k.

Pour traiter le cas le plus compliqué à savoir $K^* < k \le K$, on substitue $s = \delta_k$ dans le lemme 2, et en rappelant que $\pi_0(\delta_k) = ... = \pi_{K^*}(\delta_k) - 1$, on obtient:

$$V(\delta_k) - V_k(\delta_k) = (1-\alpha_{kK^*})W_{K^*} - \psi_k(\beta)\psi_k(\mu_k)r_k^* - \sum_{i=1}^{K^*}(\alpha_{k,i-1} - \alpha_{ki})C_i \tag{34}$$

Définissons maintenant les constantes importantes suivantes:

$$Z_{kj} = \left[\psi_k(\beta)\psi_k(\mu_k)r_k^* + \sum_{i=1}^{j-1}(\alpha_{k,i-1} - \alpha_{ki})C_i\right] / (1-\alpha_{k,j-1}) \quad (1 \le j \le k \le K) \tag{35}$$

En particulier,

$Z_{k1} = \psi_k(\beta)\psi_k(\mu_k)r_k^*/(1-\psi_k)$ pour tout k,

et en comparant (35) avec (28) on voit que

$$C_k = Z_{kk} \quad (1 \le k \le K) \tag{36}$$

De même notons que (34) se réduit à

$$V(\delta_k) - V_k(\delta_k) = (1-\alpha_{kK^*})(W_{K^*} - Z_{k,K^*+1}) \text{ si } K^* < k \le K \tag{37}$$

En combinant le lemme 3 avec (33) et (37) , on a alors:

LEMME 4:

Supposons que $1 \le k \le K, s \in S$ et $k \in A(s)$.

$$\text{Si } C_1 \ge ..... \ge C_{K^*} \ge W_{K^*} \ge \max_{K^* < i \le K}[Z_{i,K^*+1}] \tag{38}$$

alors $V(s) - V_k(s) \ge 0$.

REMARQUE:

Puisque le flot total d'arrivée est poissonnien de paramètre $\Lambda_K = \sum_{i=1}^{K} \lambda_i$,

on déduit que:

$$F(t \setminus s,s',0) = \begin{cases} (\lambda_k/\Lambda_K)\left(1-e^{-\Lambda_K t}\right) & \text{si } s'=s+\delta_k \\ 0 \text{ sinon} & \end{cases} \quad ,1 \le k \le K$$

et donc

$$V_0(s) = \int_0^\infty e^{-\beta t}\left[\sum_{k=1}^{K} (\lambda_k/\Lambda_K)V(s+\delta_k)\right]\Lambda_K e^{-\Lambda_K t}dt$$

$$= \sum_{k=1}^{K} \lambda_k V(s+\delta_k)/(\Lambda_K+\beta) \quad (s \in S)$$

Notons que

$V(s+\delta_k) = V(s)$ si $K^* < k \le K$,on obtient alors:

$$(\Lambda_K+\beta)V_0(s) = \sum_{k=1}^{K^*} \lambda_k V(s+\delta_k) + (\Lambda_K - \Lambda_{K^*})V(s) \qquad (39)$$

On peut voir facilement que:

$$(\Lambda_K+\beta)V(s) = \beta V(s) + \sum_{k=1}^{K^*} \lambda_k V(s) + (\Lambda_K - \Lambda_{K^*})V(s)$$

En combinant cette expression avec (39) on obtient:

$$(\Lambda_K+\beta)[V(s) - V_0(s)] = \beta V(s) - \sum_{k=}^{K^*} [V(s+\delta_k) - V(s)] \qquad (40)$$

LEMME 5:

Pour tous $s \in S$

$$V(s) = C_1 - \sum_{j=1}^{K^*-1} \pi_j(s)(C_j - C_{j+1}) - \pi_{K^*}(s)(C_{K^*} - W_{K^*})$$

PREUVE:

En effet d'après (29) on a :

$$V(s) = \sum_{k=1}^{K^*} [\pi_{k-1}(s) - \pi_k(s)]C_k + \pi_{K^*}(s)W_{K^*}$$

$$= \pi_0(s)C_1 - \pi_1(s)C_1 + ... - \pi_{K^*-1}(s)C_{K^*} - \pi_{K^*}(s)C_{K^*} + \pi_{K^*}(s)W_{K^*}$$

$$= C_1 - \sum_{j=1}^{K^*-1} \pi_j(s)(C_j - C_{j+1}) - \pi_{K^*}(s)(C_{K^*} - W_{K^*}).$$

C.Q.F.D

LEMME 6:

Si $C_1 \geq ..... \geq C_{K^*} \geq W_{K^*}$, alors V(s) est non-décroissante par rapport à chaque composante de s, mais $V(s+\delta_k)$-V(s) est non-croissante par rapport à chaque composante de s, pour tout $k = 1,...,K^*$.

Si l'hypothèse du lemme 6 est satisfaite, il est alors immédiat que la fonction à droite de (40) est non-décroissante par rapport à chaque composante de s et elle est ainsi minimisée (pour tout $s \in S$) en posant s=0. Mais $V(0) = V_g(0)$, puisque l'oisiveté est l'action appliquée par la stratégie g dans l'état zéro. Ainsi nous avons

LEMME 7:

Si $C_1 \geq ..... \geq C_{K^*} \geq W_{K^*}$, alors

$$V(s)-V_g(s) \geq 0 \quad \text{pour tout } s \in S$$

En combinant les lemmes (4) et (7) avec le théorème 1, on obtient la condition suffisante d'optimalité de g.

THEOREME 2:

Si (38) est satisfaite (la dernière inégalité étant automatiquement satisfaite si K*=K),alors la stratégie g est β-optimale.

4.4- ALGORITHME

Nous allons présenter un algorithme à (K+1) étapes pour ordonnancer les classes de processus ,sélectionner la classe de coupure K*, et enfin ordonnancer les processus à l'intérieur de chaque classe. Il autorise en outre le concepteur des systèmes d'exploitations de choisir de façon optimale * les bornes m_i et M_i , $i=1,K$

ETAPE 0: Numéroter les classes arbitrairement.

ETAPE 1: Pour tout k=1,....,K calculer $r_k^* = r_k + h_k/\beta$

(a) pour tout k=1,...,K calculer $\alpha_{k0} = \psi_k = \psi_k(\beta)$.

(b) pour tout k=1,....,K calculer $Z_{k1} = \psi_k(\beta)\,\psi_k(\mu_k) r_k^*/(1-\psi_k)$, et renuméroter les classes telles que $Z_{11} = \max\{Z_{11}, ..., Z_{s1}\}$

(c) exécuter à chaque instant t le processus i, ayant la plus grande priorité donnée par la formule $p_i(t) = \dfrac{t_i^a(t)M_1 + t_i(t)m_1}{t}$

où $m_1$et M_1 sont deux réels positifs associés à la classe 1,tel que $m_1 \le M_1$.

(d) poser $C_1 = Z_{11}$, $\Lambda_1 = \lambda_1$ et calculer $\alpha_1 \in]0, 1[$ satisfaisant $\alpha_1 = \psi_1[\beta + \lambda_1(1-\alpha_1)]$.

ETAPE j: $(2 \le j \le K)$

(a) pour tout k=j,...,K calculer $\alpha_{k,j-1} = \psi_k[\beta + \Lambda_{j-1}(1-\alpha_{j-1})]$

(b) pour tout k=j,....,K, calculer

$$Z_{kj} = \left[\psi_k(\beta)\,\psi_k(\mu_k) r_k^* + \sum_{i=1}^{j-1} (\alpha_{k,i-1} - \alpha_{ki}) C_i \right] / (1 - \alpha_{k,j-1}),$$

et renuméroter les classes telles que $Z_{jj} = \max(Z_{jj}, ..., Z_{kj})$.

(c) exécuter à chaque instant t le processus i ,ayant la plus grande priorité donnée par la formule $p_i(t) = \dfrac{t_i^a(t)*M_j + t_i(t)*m_j}{t}$

où m_j et M_j sont deux réels positifs tels que

$m_j \le M_j < m_{j-1} \le M_{j-1} < .... < m_1 \le M_1$.

(d) poser $C_j = Z_{jj}$, $\Lambda_j = \Lambda_{j-1} + \lambda_j$ et calculer $\alpha_j \in]0, 1[$

satisfaisant $\alpha_j = (1/\Lambda_j)\sum_{i=1} \lambda_i \psi_i[\beta + \Lambda_i(1-\alpha_j)]$.

** par exemple en prenant $r_i = 0$ pour tout i , la stratégie obtenue par l'algorithme est celle qui minimise le flow-time pondéré*

ETAPE K+1:

(a) si $C_1 < 0$ (ce qui entraîne que $r_k^* < 0$ pour tout k poser K*− 0

(b) sinon, poser $\omega_0 = 0$ et calculer

$$\omega_j = \Lambda_j(1 - \alpha_j), \qquad (1 \le j \le K)$$

$$W_j = \sum_{i=1}^{j} (\omega_i - \omega_{i-1})C_i/(\beta + \omega_i) \quad (1 \le j \le K). \quad (41)$$

Soit K* le plus petit indice $j, 1 \le j \le K$, tel que $W_j > C_{j+1}$, si un tel indice existe. Sinon poser K*=K.

4.5--JUSTIFICATION DE L'ALGORITHME

On va montrer que la stratégie construite par l'algorithme satisfait le critère d'optimalité du théorème de ROSS.

LEMME 8:

Si les classes de processus sont numérotées par l'algorithme, alors

$C_1 > \ldots\ldots > C_K$

PREUVE:

On suppose que $2 \le k \le K$. Il s'ensuit par la définition (35) de $Z_{k,j}$ que

$$\begin{aligned}(1-\alpha_{k,k-1})C_k &= (1-\alpha_{k,k-1})Z_{kk} \\ &= (1-\alpha_{k,k-2})Z_{k,k-1} + (\alpha_{k,k-2}-\alpha_{k,k-1})C_{k-1} \qquad (42) \\ &= (1-\alpha_{k,k-1})C_{k-1} - (1-\alpha_{k,k-2})(C_{k-1}-Z_{k,k-1})\end{aligned}$$

L'étape (k-1) de l'algorithme attribue la $(k-1)^{ème}$ priorité de façon à satisfaire $C_{k-1} = Z_{k-1,k-1} \ge Z_{j,k-1}$, pour tout $j = k,\ldots,K$. Ainsi $C_{k-1} \ge Z_{k,k-1}$, et donc $C_k \le C_{k-1}$ par (42).

LEMME 9:

Si la classe de coupure K* déterminé par l'algorithme est supérieure à zéro, alors $C_{K^*} \ge W_{K^*}$.

PREUVE:

pour le cas K*=1, on note simplement que:

$W_1 = \omega_1 C_1/(\beta + \omega_1) < C_1$.

si $1 \le K^* \le K$, nous avons d'après la définition (41):

$$(\beta + \omega_{K^*})W_{K^*} = (\beta + \omega_{K^*-1})W_{K^*-1} + (\omega_{K^*} - \omega_{K^*-1})C_{K^*}$$

ceci implique que

$$(\beta + \omega_{K^*})(C_{K^*} - W_{K^*}) - (\beta + W_{K^*-1})(C_{K^*} - W_{K^*-1}) \quad 0. \qquad (43)$$

Or d'après l'étape K+1 on a $C_{K^*} \ge W_{K^*-1}$ et par conséquent $C_{K^*} \ge W_{K^*}$ aussi bien par (43).

THEOREME 3:

La stratégie construite par l'algorithme est optimale.

PREUVE:

l'algorithme donne $K^* = 0$ si et seulement si $r_k^* < 0$ pour toute classe k. Dans ce cas il est immédiat par le théorème 1 que la stratégie correspondant à l'option d'oisiveté perpétuelle (ou la fonction de gain total est identiquement nulle) est β–optimale.

Quand $K^*>0$, on sait d'après les lemmes 8 et 9 que notre stratégie satisfait

$$C_1 > \ldots \ge C_{K^*} \ge W_{K^*} \qquad (44)$$

et d'après l'étape K+1 et le sens de K* on a

$$W_{K^*} \geq C_{K^*+1} = \max_{K^* < i \leq K} |Z_{i,K^*+1}| \text{ si } 1 \leq K^* < K \tag{45}$$

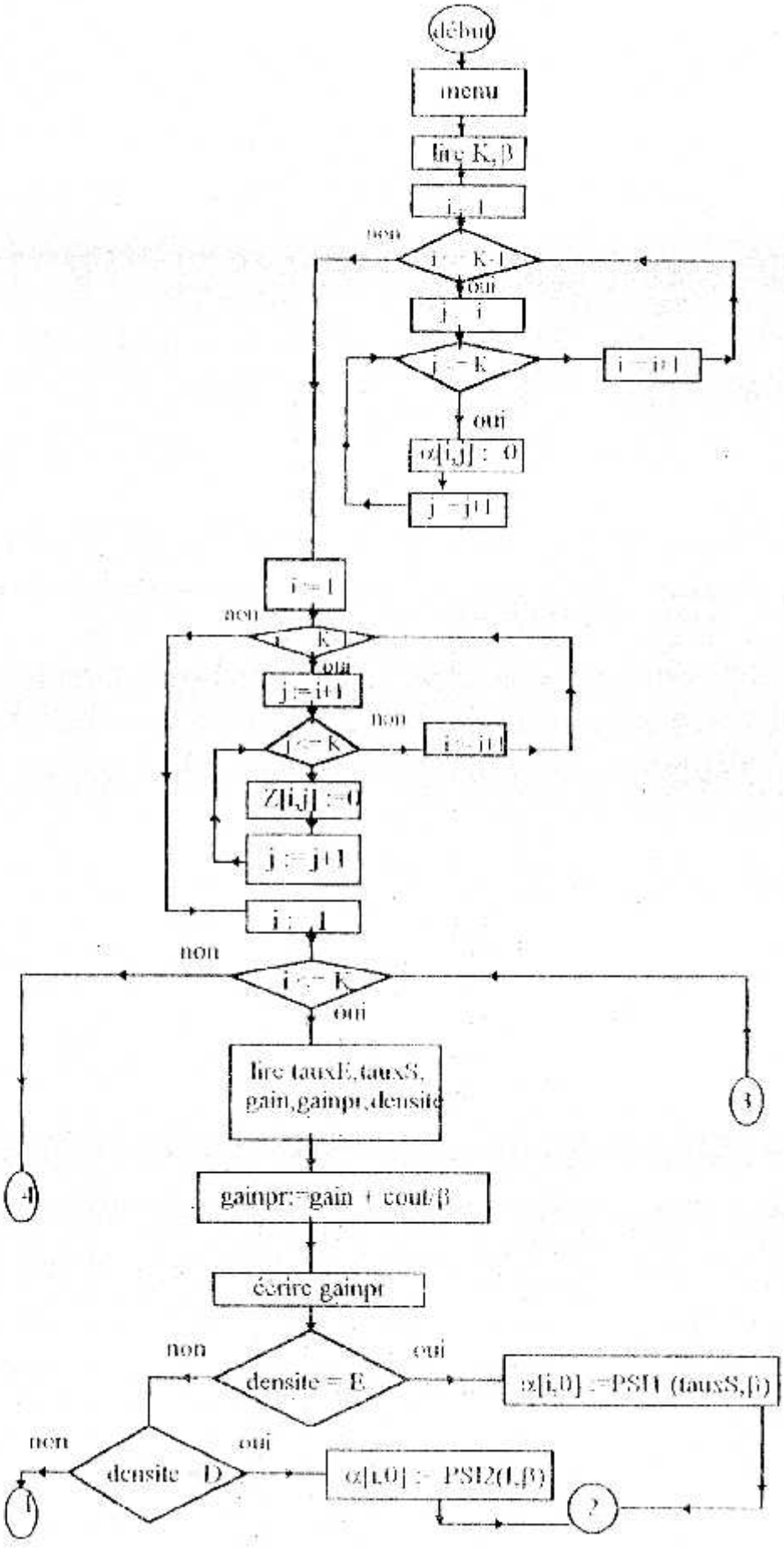
en combinant (44) et (45) on remarque que l'algorithme satisfait (38) et par suite la stratégie est β- optimale par le théorème 2.

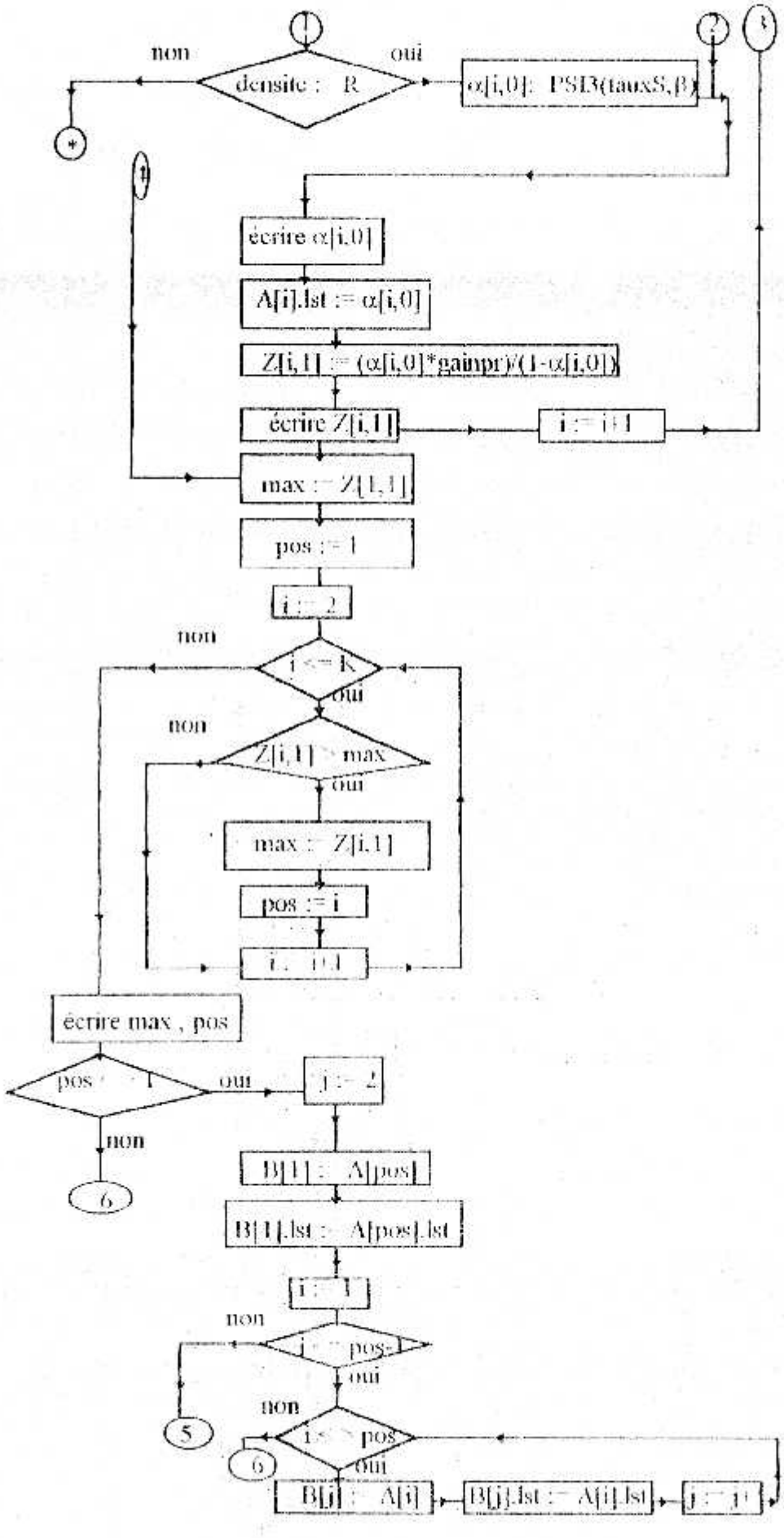
CONCLUSION

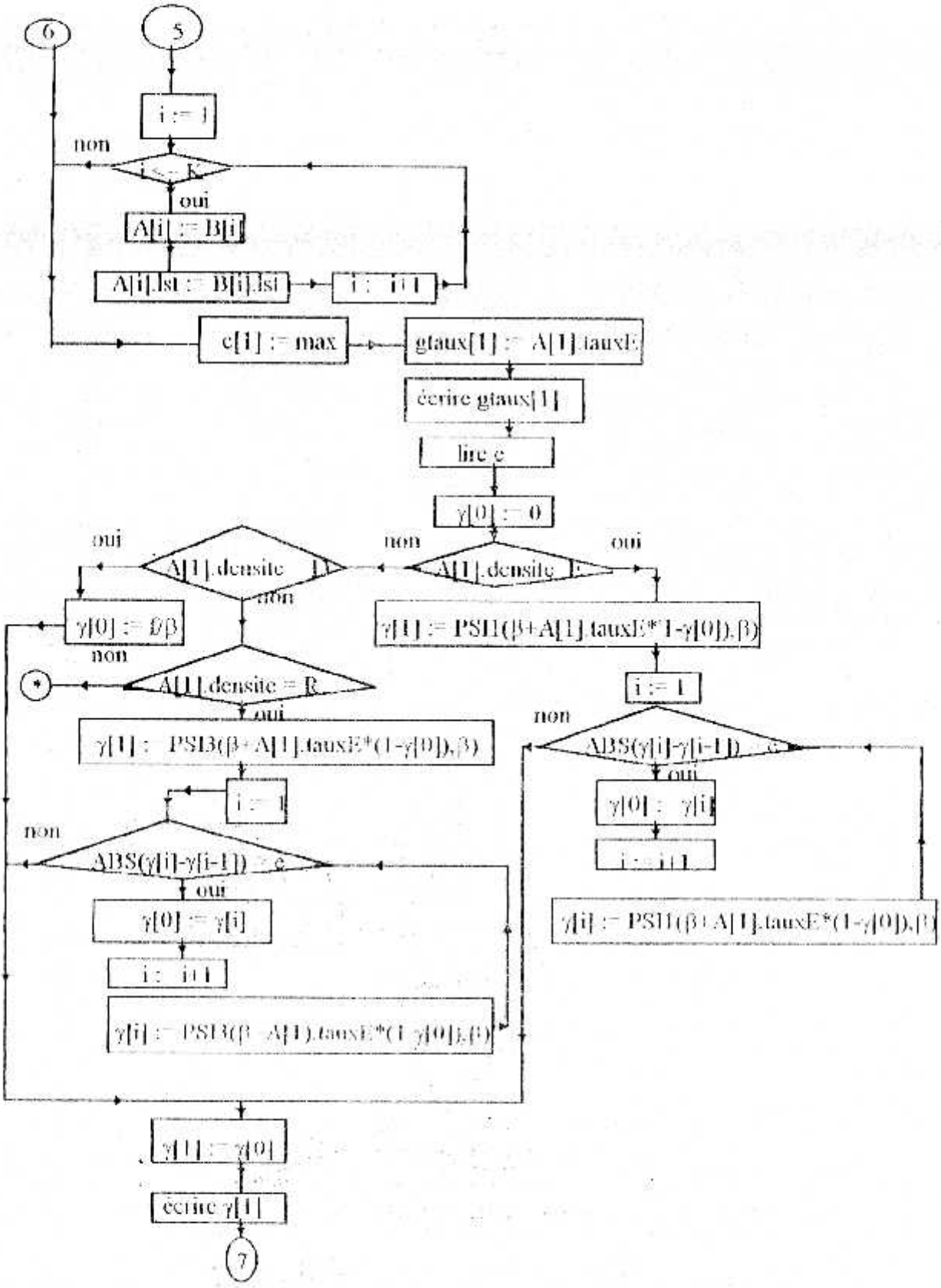
5- CONCLUSION

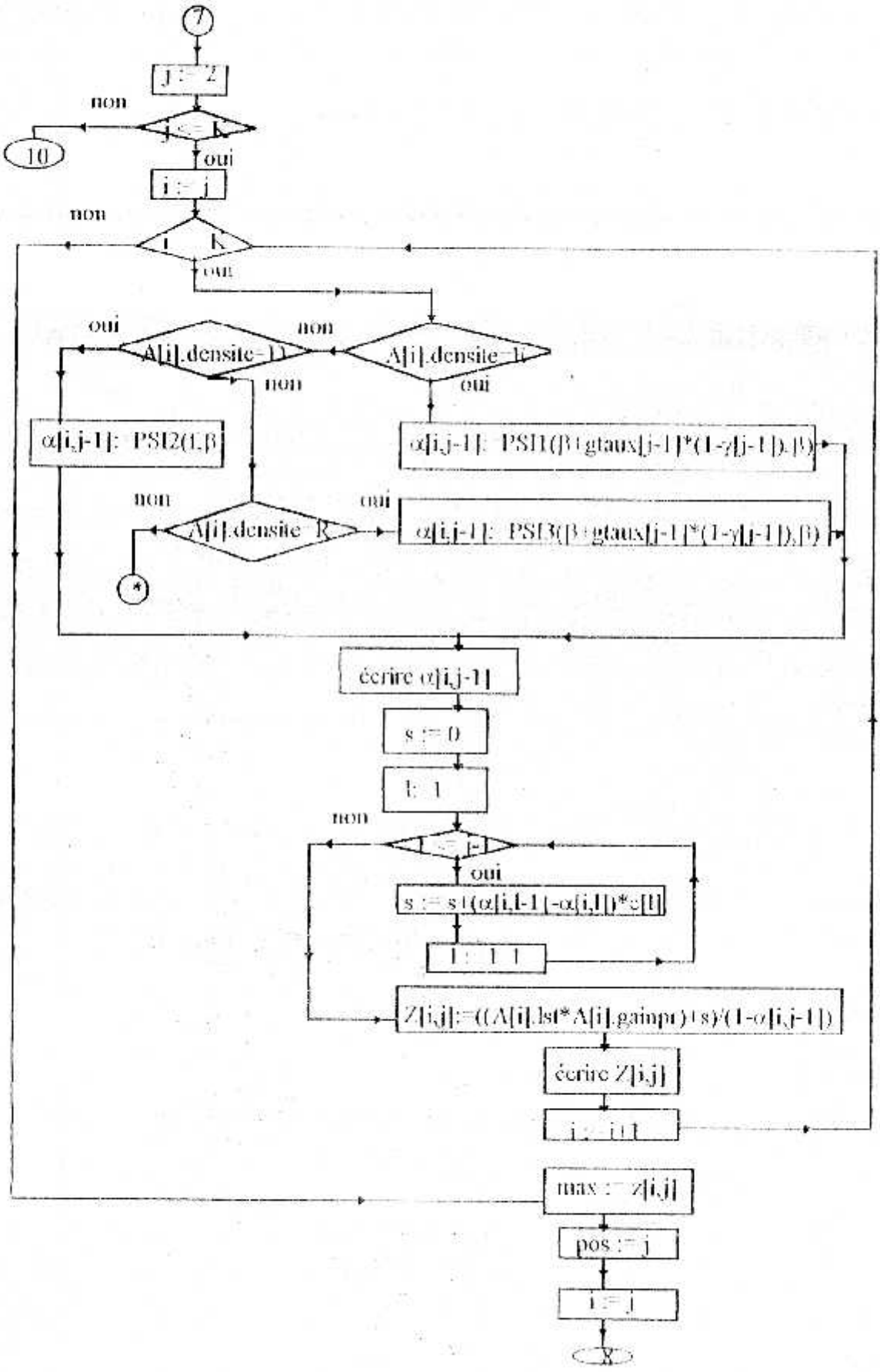
Cette étude montre sur l'exemple de la stratégie OPBM que la problématique du scheduler dans les systèmes d'exploitation peut être formulée en termes de processus de décision stochastique , en particulier les processus de bandit. Cependant , une question reste encore mal élucidée , il s'agit en particulier , du problème de l'équité qui n'a pas été prouvé de manière satisfaisante dans [HARO & PROUST 92] . De plus, l'algorithme proposé utilise les résultats classiques pour la période d'activité d'un système M/G/1 avec source infinie. Cependant il est aisé de l'étendre , tout au moins pour le cas de deux classes, au cas plus réaliste d'une source finie.

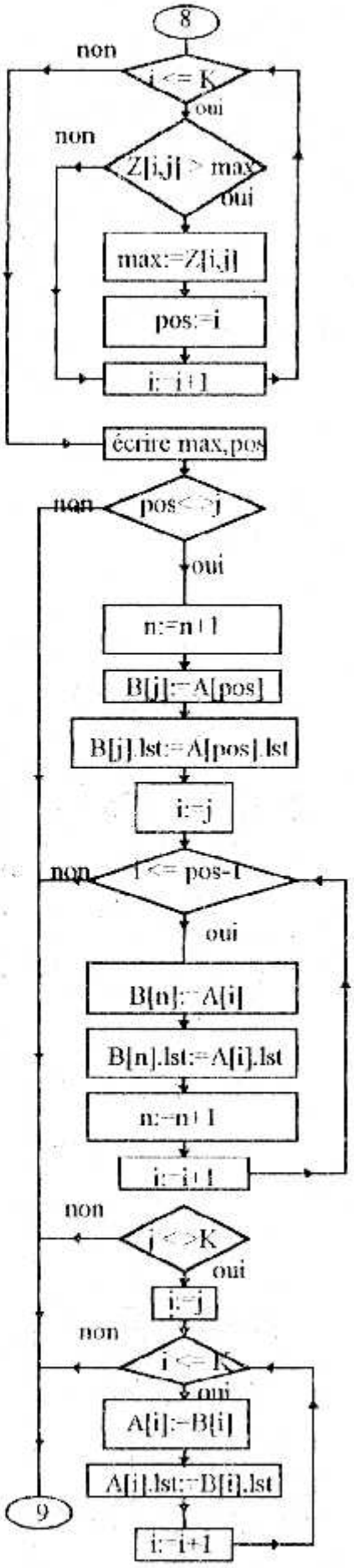
ORGANIGRAMME

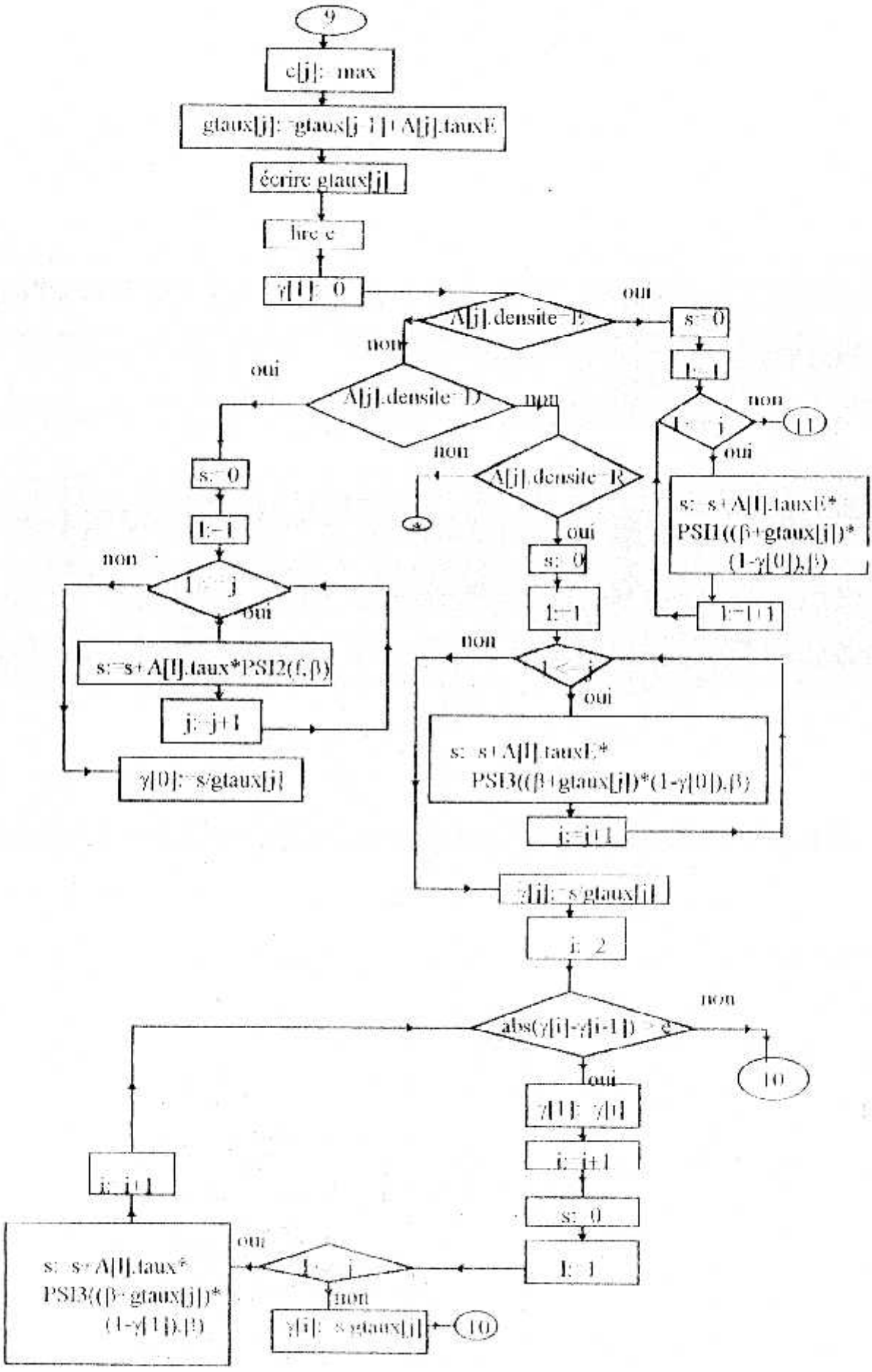


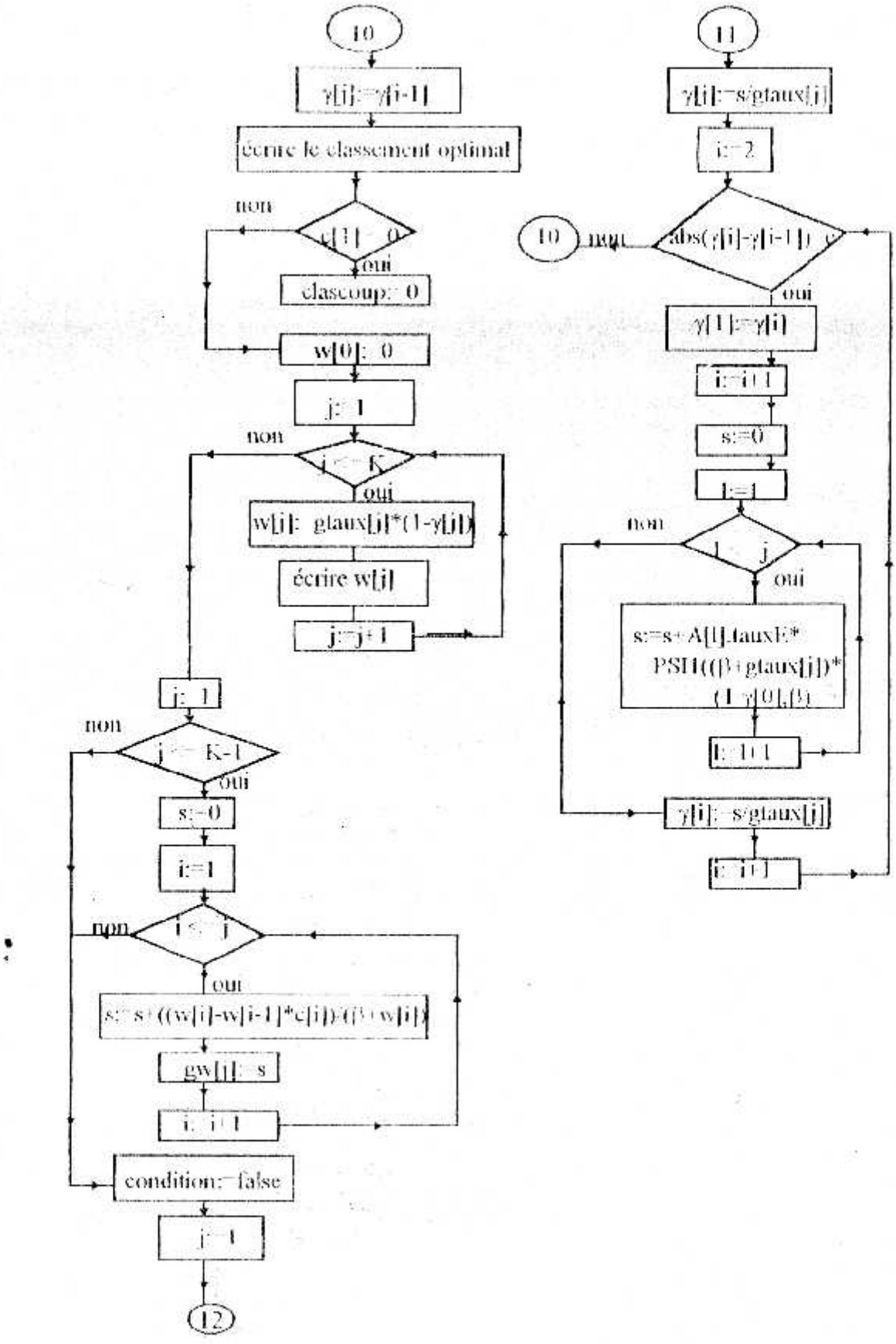

densite : R
non
oui
α[i,0]: PSI3(tauxS,β)
écrire α[i,0]
A[i].lst := α[i,0]
Z[i,1] := (α[i,0]*gainpr)/(1-α[i,0])
écrire Z[i,1]
i := i+1
max := Z[1,1]
pos := 1
i := 2
i <= K
Z[i,1] > max
max := Z[i,1]
pos := i
écrire max , pos
pos <> 1
j := 2
B[1] := A[pos]
B[1].lst := A[pos].lst
i := 1
i <> pos-1
i <= pos
B[j] := A[i]
B[j].lst := A[i].lst
j := j+1

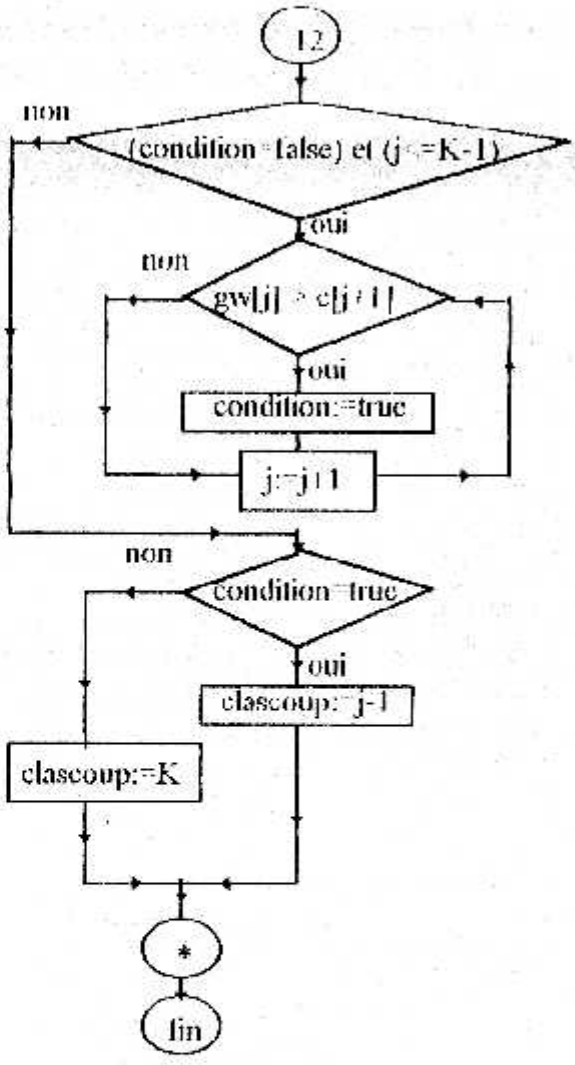

6
5
i := 1
non
i <= K
oui
A[i] := B[i]
A[i].lst := B[i].lst
i := i+1
c[i] := max
gtaux[1] := A[1].tauxE
écrire gtaux[1]
lire e
γ[0] := 0
oui
A[1].densite
non
oui
non
γ[0] := β/β
γ[1] := PSI1(β+A[1].tauxE*1-γ[0]),β)
non
A[1].densite = R
i := 1
oui
γ[1] := PSI3(β+A[1].tauxE*(1-γ[0]),β)
non
ABS(γ[i]-γ[i-1])
oui
γ[0] := γ[i]
i := i+1
γ[i] := PSI1(β+A[1].tauxE*(1-γ[0]),β)
i := 1
non
ABS(γ[i]-γ[i-1]) <= e
oui
γ[0] := γ[i]
i := i+1
γ[i] := PSI3(β+A[1].tauxE*(1-γ[0]),β)
γ[1] := γ[0]
écrire γ[1]
7


7
j := 2
non
j <= K
10
oui
i := j
non
i K
oui
oui
A[i].densite=1)
non
A[i].densite=1(
non
oui
α[i,j-1]: PSI2(1,β
α[i,j-1]: PSI1(β*gtaux[j-1]*(1-γ[j-1]),β)
non
A[i].densite K
oui
α[i,j-1]: PSI3(β*gtaux[j-1]*(1-γ[j-1]),β)
*
écrire α[i,j-1]
s := 0
l: 1
non
l <= i-1
oui
s := s+(α[i,l-1]-α[i,l])*c[l]
l := l+1
Z[i,j]:=((A[i].lst*A[i].gainpu)+s)/(1-α[i,j-1])
écrire Z[i,j]
i := i+1
max := z[i,j]
pos := j
i := j
X


8
non
i <= K
oui
non
Z[i,j] > max
oui
max:=Z[i,j]
pos:=i
i:=i+1
écrire max,pos
non
pos<>j
oui
n:=n+1
B[j]:=A[pos]
B[j].lst:=A[pos].lst
i:=j
non
i <= pos-1
oui
B[n]:=A[i]
B[n].lst:=A[i].lst
n:=n+1
i:=i+1
non
j <>K
oui
i:=j
non
i <= K
oui
A[i]:=B[i]
A[i].lst:=B[i].lst
i:=i+1
9


9
c[j]:=max
gtaux[j]:=gtaux[j-1]+A[j].tauxE
écrire gtaux[j]
lire e
γ[1]:=0
A[j].densite=E
oui
s:=0
l:=1
non
A[j].densite=D
oui
non
l=i
non
11
oui
A[j].densite=R
non
s:=0
l:=l+1
s:=s+A[l].tauxE*
PSI1((β+gtaux[j])*
(1-γ[0]),β)
oui
s:=0
l:=1
non
l=j
oui
s:=s+A[l].taux*PSI2(f,β)
j:=j+1
γ[0]:=s/gtaux[j]
non
l<=i
oui
s:=s+A[l].tauxE*
PSI3((β+gtaux[j])*(1-γ[0]),β)
j:=j+1
γ[i]:=s/gtaux[i]
i:=2
non
abs(γ[i]-γ[i-1])<e
10
oui
γ[1]:=γ[i]
i:=i+1
s:=0
l:=1
l=i
oui
non
s:=s+A[l].taux*
PSI3((β+gtaux[j])*
(1-γ[1]),β)
γ[i]:=s/gtaux[i]
10
i:=i+1


10
γ[i]:=γ[i-1]
écrire le classement optimal
non
c[1] = 0
oui
clascoup:= 0
w[0]:= 0
j:= 1
non
j <= K
oui
w[j]:= gtaux[j]*(1-γ[j])
écrire w[j]
j:=j+1
j:= 1
non
j <= K-1
oui
s:=0
i:=1
non
i <= j
oui
s:=s+((w[i]-w[i-1]*c[i])/(1-w[i])
gw[j]:= s
i:= i+1
condition:=false
j:=1
12
11
γ[i]:=s/gtaux[j]
i:=2
10
non
abs(γ[i]-γ[i-1]) <
oui
γ[1]:=γ[i]
i:=i+1
s:=0
l:=1
non
l = j
oui
s:=s+A[l].tauxE*
PSI((β+gtaux[j])*
(1-γ[l])/β)
l:=l+1
γ[i]:=s/gtaux[j]
i:=i+1


12
non
(condition=false) et (j<=K-1)
oui
non
gw[j] > c[j+1]
oui
condition:=true
j:=j+1
non
condition=true
oui
clascoup:=j-1
clascoup:=K
*
fin

PROGRAMME

EXEMPLE D'EXECUTION

```
PROGRAM
USES CRT;
TYPE classe=record
                    lauxI,lauxS,gain,gainpr,cout:real;
                    densite:char;
                    lst:real;
             end;
      liste=array[1..30] of classe;
VAR   A,B:liste;
      alpha:array[0..30,0..30] of real;
      Z:array[1..30,1..30] of real;
      s,max,beta,f,e:real;
      n,i,j,k,l,pos,clascoup:integer;
      c,gtaux,gw:array[1..20] of real;
      w:array[0..30] of real;
      gamma:array[0..30] of real;
      condition:boolean;
      rep:char;
PROCEDURE menu;
  BEGIN
    clrscr;
    gotoxy(12,2);
    write(chr(201));
    for i:=1 to 55 do
    write(chr(205));
    writeln(chr(187));
    gotoxy(12,3);
    write(chr(186),'  Loi                    ');
    write(' Symbole       Transformée Laplace     ');
    writeln(chr(186));
    gotoxy(12,4);
    write(chr(200));
    for i:=1 to 55 do
    write(chr(205));
    writeln(chr(188));
    gotoxy(12,5);
    write(chr(201));
    for i:=1 to 55 do
    write(chr(205));
    writeln(chr(187));
    for i:=0 to 7 do
    begin
      gotoxy(12,i+6); write(chr(186));
      gotoxy(68,i+6); write(chr(186));
    end;
    writeln;
    gotoxy(12,14);
    write(chr(200));
    for i:=1 to 55 do
    write(chr(205));
    writeln(chr(188));
    gotoxy(13,6);
    write(' EXPONENTIELLE          E          µ/(µ+s)');
    gotoxy(13,8);
    write(' DETERMINISTE           D          e^-s');
    gotoxy(13,10);
    write(' ERLANG(n)              E          (µ/(µ+s))^n');
    gotoxy(13,12);
    write(' COX2                   C          µ1/(s+µ1)+
```

```
    gotoxy(13,13);
    write('                                          ');
    write(' (μ2/(s+μ2)-μ1/(s+μ1))a');
    gotoxy(12,15);
    write(chr(201));
    for i:=1 to 55 do
    write(chr(205)),
    writeln(chr(187));
    gotoxy(12,16);
    write(chr(186),'                    TRANSFORMEE DE LAPLACE'),
    write('                ');
    write(chr(186));
    gotoxy(12,17);
    write(chr(200));
    for i:=1 to 55 do
    write(chr(205));
    writeln(chr(188));
  END;
FUNCTION PSI1(mu,beta:real):real;
  BEGIN
    PSI1:=mu/(mu+beta);
  END;
FUNCTION PSI2(f,beta:real):real;
  BEGIN
    PSI2:=f/beta;
  END;
FUNCTION puis(x:real;n:integer):real;
  VAR y:real;
  BEGIN
    y:=1;
    for i:=1 to n do
    y:=y*x;
   puis:=y;
 END;
FUNCTION PSI3(mu,beta:real):real;
  BEGIN
    PSI3:=puis(mu/(mu+beta),n);
  END;
BEGIN {corps du programme}
 repeat
  clrscr;
  menu;
  writeln;
  writeln('le nombre de classes est:');
  readln(k);
  writeln('beta =?');
  readln(beta);
  for i:=1 to k-1 do
  for j:=i to k do
  alpha[i,j]:=0;
  for i:=1 to k-1 do
  for j:=i+1 to k do
  Z[i,j]:=0;
  for i:=1 to k do
  begin
   readln;clrscr;
    writeln(' POUR LA CLASSE ',i);
    with A[i] do
```

```
  begin
    writeln('donner le taux d''entrée tauxE'),
    readln(tauxE);
    writeln('donner le taux de service tauxS');
    readln(tauxS);
    writeln('donner le gain');
    readln(gain);
    writeln('donner le cout');
    readln(cout);
    writeln('donner la densite');
    readln(densite);
    gainpr:=gain+(cout/beta);
    writeln('gainpr =',gainpr:6:2);
    case densite of
         'E','e': alpha[i,0]:=PSI1(tauxS,beta);
         'D','d': begin
                    writeln('la loi deterministe égale A 1  ');
                    readln(f);
                    alpha[i,0]:=PSI2(f,beta);
                  end;
         'R','r': begin
                    writeln('LA LOI D''ERLANG EST D''ORDRE n= ');
                    readln(n);
                    alpha[i,0]:=PSI3(tauxS,beta);
                  end;
    end;
    writeln('alpha[',i,',0]  ',alpha[i,0]:6:2),
    A[i].lst:=alpha[i,0];
    Z[i,1]:=(alpha[i,0]*gainpr)/(1-alpha[i,0]);
    writeln('z[',i,',1]= ',z[i,1]:6:2);
  end;
  writeln;
end;
readln;
clrscr;
max:=Z[1,1];
pos:=1;
for i:=2 to k do
if Z[i,1]>max
then  begin
        max:=Z[i,1];
        pos:=i;
      end;
writeln('max=',max:6:2);
writeln('pos=',pos);
if pos<>1
then  begin
        j:=2;
        B[1]:=A[pos];
        B[1].lst:=A[pos].lst;
        for i:=1 to pos-1 do
        if i<>pos
        then  begin
                B[j]:=A[i];
                B[j].lst:=A[i].lst;
                j:=j+1;
              end;
        for i:=1 to k do
        A[i]:=B[i];
        A[i].lst:=B[i].lst;
      end;
```

```
c[1]:=max;
gtaux[1]:=A[1].tauxE;
writeln('gtaux[1]=',gtaux[1]:6:2);
writeln;
writeln('donner la precision  e');
readln(e);
gamma[0]:=0;
case A[1].densite of
  'E','e':
    begin
      gamma[1]:=PSI1(beta+A[1].tauxE*(1-gamma[0]),beta);
      i:=1;
      while abs(gamma[i]-gamma[i-1])>e do
      begin
        gamma[0]:=gamma[i];
        i:=i+1;
        gamma[i]:=PSI1(beta+A[1].tauxE*(1-gamma[0]),beta);
      end;
    end;
  'D','d':gamma[0]:=f/beta;
  'R','r':
    begin
      gamma[1]:=PSI3(beta+A[1].tauxE*(1-gamma[0]),beta);
      i:=1;
      while abs(gamma[i]-gamma[i-1])>e do
      begin
        gamma[0]:=gamma[i];
        i:=i+1;
        gamma[i]:=PSI3(beta+A[1].tauxE*(1-gamma[0]),beta);
      end;
    end;
end;
gamma[1]:=gamma[0];
writeln('gamma[1]=',gamma[0]:6:2);
writeln;
writeln('****************fin de la premiére partie**************');
readln;
clrscr;
{étape j}
for j:=2 to k do
begin
  readln;
  clrscr;
  for i:=j to k do
  begin
    case A[i].densite of
      'E','e':
        begin
          alpha[i,j-1]:=PSI1(beta+gtaux[j-1]*(1-gamma[j-1]),beta);
          writeln('alpha[',i,',',j-1,']= ',alpha[i,j-1]:6:2);
        end;
      'D','d':
        begin
          alpha[i,j-1]:=PSI2(f,beta);
          writeln('alpha[',i,',',j-1,']= ',alpha[i,j-1]:6:2);
        end;
      'R','r':
        begin
          alpha[i,j-1]:=PSI3(beta+gtaux[j-1]*(1-gamma[j-1]),beta);
          writeln('alpha[',i,',',j-1,']= ',alpha[i,j-1]:6:2);
        end;
    end;
```

```
  s:=0;
  for l:=1 to j-1 do
  s:=s+(alpha[i,l-1]-alpha[i,l])*c[l];
  z[i,j]:=((A[i].lst*A[i].gainpr)+s)/(1-alpha[i,j-1]);
  writeln('Z[',i,',',j,']=',Z[i,j]:6:2);
end;
max:=Z[j,j];
pos:=j;
for i:=j to k do
if z[i,j]>max
then begin
       max:=z[i,j];
       pos:=i;
     end;
writeln('max=',max:6:2);
writeln('pos=',pos);
if pos<>j
then begin
       n:=j+1;
       B[j]:=A[pos];
       B[j].lst:=A[pos].lst;
       for i:=j to pos-1 do
       begin
         B[n]:=A[i];
         B[n].lst:=A[n].lst;
         n:=n+1;
       end;
       if j<>k
       then
       for i:=j to k do
       begin
         A[i]:=B[i];
         A[i].lst:=B[i].lst;
       end;
     end;
     c[j]:=max;
     gtaux[j]:=gtaux[j-1]+A[j].tauxE;
     writeln('gtaux[',j,']=',gtaux[j]:6:2);
     writeln;
     writeln('donner la precision e');
     readln(e);
     gamma[1]:=0;
     case A[j].densite of
       'E','e':
         begin
           s:=0;
           for l:=1 to j do
           s:=s+A[l].tauxE*PSI1((beta+gtaux[j])*(1-gamma[0]),beta);
           gamma[j]:=s/gtaux[j];
           i:=2;
           while abs (gamma[i]-gamma[i-1])>e do
           begin
             gamma[1]:=gamma[i];
             i:=i+1;
             s:=0;
             for l:=1 to j do
             s:=s+A[l].tauxE*PSI1((beta+gtaux[j])*(1-gamma[1]),beta);
             gamma[i]:=s/gtaux[j];
           end;
         end;
```

```
          'D','d':begin
                    s:=0;
                    for l:=1 to j do
                    s:=s+A[l].tauxE*PSI2(f,beta);
                    gamma[0]:=s/gtaux[j];
                  end;
          'R','r':
            begin
              s:=0;
              for l:=1 to j do
              s:=s+A[l].tauxE*PSI3((beta+gtaux[j])*(1-gamma[0]),beta);
              gamma[j]:=s/gtaux[j];
              i:=2;
              while abs (gamma[i]-gamma[i-1])>e do
              begin
                gamma[1]:=gamma[i];
                i:=i+1;
                s:=0;
                for l:=1 to j do
                s:=s+A[l].tauxE*PSI3((beta+gtaux[j])*(1-gamma[1]),beta)
                gamma[i]:=s/gtaux[j];
              end;
            end;
        end;
        gamma[j]:=gamma[i-1];
        writeln('gamma[',j,']=',gamma[i-1]:6:2);
  end;
  readln;
  clrscr;
  writeln;
  writeln('LE CLASSEMENT DONNE :');
  for i:=1 to k do
  with A[i] do
  begin
    writeln;
    writeln('POUR LA CLASSE ',i,':');
    writeln('le taux d''entrée est: ',tauxE:2:0);
    writeln('le taux de service est: ',tauxS:2:0);
    writeln('le gain est: ',gain:2:0);
    writeln('le cout est: ',cout:2:0);
    writeln('la densite est: ',densite);
  end;
  writeln;
  writeln('**************fin de la deuxiéme partie***************');
  readln;
  clrscr;
{étape k+1}
  if c[1]<0
  then begin
         clascoup:=0;
         writeln;
         writeln('la classe de coupure est:',clascoup);
       end
  else begin
         w[0]:=0;
         for j:=1 to k do
         begin
           w[j]:=gtaux[j]*(1-gamma[j]);
           writeln('w[',j,']=',w[j]:6:2);
         end;
         writeln;
         for j:=1 to k-1 do
```

```
        begin
          s:=0;
          for i:=1 to j do
          begin
            s:=s+((w[i]-w[i-1])*c[i])/(beta+w[i]);
            gw[j]:=s;
          end;
          writeln('gw[',j,']=',gw[j]:6:2);
        end;
        writeln;
        condition:=false;
        j:=1;
        while (condition=false) and (j<=k-1) do
        begin
          if gw[j]>c[j+1]
          then condition:=true;
          j:=j+1;
        end;
        if condition=true
        then  begin
                clascoup:=j-1;
                writeln;
                writeln('la classe de coupure est:',clascoup);
              end
        else  begin
                clascoup:=k;
                writeln;
                writeln('la classe de coupure est:',clascoup);
              end;
        end;
    writeln;
    writeln('voulez-vous recommencer ?(y/n)');
    readln(rep);
  until rep in['N','n'];
END.{fin du programme}
```

Loi	Symbole	Transformée Laplace
EXPONENTIELLE	E	$\mu/(\mu+s)$
DETERMINISTE	D	C/S
ERLANG(n)	E	$(\mu/(\mu+s))^n$
COX2	C	$\mu1/(s+\mu1)+$ $(\mu2/(s+\mu2)-\mu1/(s+\mu1))a$

TRANSFORMEE DE LAPLACE

```
le nombre de classes est:
3
beta =?
1


 POUR LA CLASSE 1
donner le taux d'entrée tauxE
2
donner le taux de service tauxS
0.5
donner le gain
10
donner le cout
0
donner la densite
E
gainpr = 10.00
alpha[1,0]=   0.33
z[1,1]=   5.00


 POUR LA CLASSE 2
donner le taux d'entrée tauxE
1
donner le taux de service tauxS
1
donner le gain
5
donner le cout
0
donner la densite
E
gainpr =  5.00
alpha[2,0]=   0.50
z[2,1]=   5.00
```

```
 POUR LA CLASSE 3
donner le taux d'entrée tauxE
3
donner le taux de service tauxS
2
donner le gain
15
donner le cout
0
donner la densite
E
gainpr = 15.00
alpha[3,0]=   0.67
z[3,1]=  30.00


max= 30.00
pos=3
gtaux[1]=  3.00

donner la precision  e
0.05
gamma[1]=  0.68

****************fin de la premiére partie***************

alpha[2,1]=   0.66
Z[2,2]= -4.43
alpha[3,1]=   0.66
Z[3,2]=  7.87
max=  7.87
pos=3
gtaux[2]=  4.00

donner la precision e
0.05
gamma[2]=  0.61


alpha[3,2]=   0.72
Z[3,3]= 16.73
max= 16.73
pos=3
gtaux[3]=  6.00

donner la precision e
0.05
gamma[3]=  0.71
```

```
LE CLASSEMENT DONNE :

POUR LA CLASSE 1:
le taux d'entrée est:  3
le taux de service est:  2
le gain est: 15
le cout est:  0
la densite est: E

POUR LA CLASSE 2:
le taux d'entrée est:  1
le taux de service est:  1
le gain est:  5
le cout est:  0
la densite est: E

POUR LA CLASSE 3:
le taux d'entrée est:  2
le taux de service est:  1
le gain est: 10
le cout est:  0
la densite est: E

**************fin de la deuxiéme partie***************



w[1]=  0.88
w[2]=  1.55
w[3]=  1.75

gw[1]= 14.01
gw[2]= 16.08


la classe de coupure est:1

voulez-vous recommencer ?(y/n)
```

BIBLIOGRAPHIE

[BRINCH HANSEN 71] BRINCH HANSEN P., *Short-term scheduling in multiprogramming systems* : Third ACM symposium on Operating Systems Principles, Stanford University, Oct. 1971, pp. 103- 105.

[COFFMAN 73] COFFMAN E.G et Denning P.J ., *Operating Systems Theory*: Prentice Hall, 1973.

[CONWAY 67] CONWAY.R.W , MAXWELL.W.L And MILLER.W., *Theory of Scheduling* : Addison- Wesley, Reading Mass.

[CRISMAN 65] CRISMAN P.A.D., *The compatible time-sharing system*, Mit Press, 1965.

[GITTINS 79] GITTINS J.C., *Bandit Processes and Dynamic Allocation Indices*: J.Roy.Stat.Soc.Ser.B , vol. 41 ,1979 , pp148- 177.

[GITTINS 89] GITTINS J.C., *Multi-armed Bandit Allocation Indices*: Jonh Wiley & Sons, 1989.

[GNEDENKO 68] GNEDENKO B ., KOVALENKO I., *Introduction to queueing theory*: Weiner Bindery, 1968.

[HARO 92a] HARO C., PROUST C., *Un ordonnancement équitable par priorités bornées*: Laoratoire Informatique , Université de Tours, Rapport Interne n° 119, TOURS, avril 1992.

[HARO 92 b] HARO C., PROUST C., *Un ordonnancement équitable par priorités bornées*: Actes du colloque international "Méthodes et Outils d'Aide à la Décision" (MOAD'92), Bejaia, Algérie, Déc.92, pp.46-49.

[HARO 93] HARO.C., PROUST.C., *A Multitasking executive for opérating systems courses* : Operating Systems Review., Vol.27, n°3,1993, pp.97-108.

[HARRISON 75] HARRISON J.M., *A Priority Queue With Discounted Linear Cost* : Opns. Res., 23,1975 , 260- 269.

[HARRISON 75] HARRISON J.M .,*Dynamic Scheduling of a Multiclass Queue: Discount Optimality* : Opns. Res., 23,1975 , 270- 282

[HIRAYAMA 87] HIRAYAMA T., *Nonpreemptive Scheduling of a Finite-Source Queue With Two Customer Class* : J. Opr.Res.Soc . Japan., 30, 1987, 200-217.

[JAISWAL 68] JAISWAL N.K , *Priority Queues*: Academic Press, NewYork.

[KLEINROCK 75] L. KLEINROCK, *Queuing Systems* , Tome1: Theory,Wiley Interscience, 1975.

[KLIMOV 74] KLIMOV.G.P , *Time Sharing Service Systems I* : Theory Prob.Appl., 19, 1974, 532-551

[KLIMOV 78] KLIMOV.G.P , *Time Sharing Service Systems II*: Theory Prob.Appl., 23, 1978, 314- 321.

[KRAKOWIAK 87] , KRAKOWIAK S., *Principes des Systèmes d'Exploitation des Ordinateurs*, Dunod, 1987.

[PATEROK 94] PATEROK . M et ETTL . M ., *Sojourn Time and Waiting Time Distributions For M/GI/1 Queues With Preemption- Distance Priorities*: Opns.Res. Soc.of. America., vol 42, n° 6, 1994, 1146-1161.

[PETERSON 83] PETERSON J., SILBERSCHATZ A.., *Operating Systems Concepts*: ADDISON- WESLEY, 1983.

[ROSBERG 85] ROSBERG . Z ., *Process Scheduling in Computer System*: IEEE Trans , On Comp., 34, 1985, 633-645.

[ROSS 70] ROSS.S.M., *Applied Probability Models With Optimisation Applications*: Holden- Day, San Francisco,1970.

[ROSS 83] ROSS.S.M., *Introduction to Stochastic Dynamic Programming*: Academic Press , 1983.

[SCHRAGE 67] SCHRAGE.L.E., *A Proof of The Optimality of The SRPT Discipline*: Opns.Res., 16, 1968, 687-690.

[TAKÄCS 61] TAKÄCS . L ., *The Probability Law of The Busy Period For Two Types of Queuing Processes* : Opns.Res.9, 1961, 402-407.

[VARAIYA 85] VARAIYA.P.P ., WALRAND.J.C and BUYUKKOC.C ., *Extension of the Multi-armed Bandit Problems., The Discounted Case*: IEEE Auto. Cont., 30, 1985, 426- 439.

[WEISS 92] WEISS GIDEON., *A Tutorial in Stochastic Scheduling* : School Of ISYE, Georgia Tech Atlanta, GA 30332, July 10, 1992.

[WELCH 64] WELCH.P.D., *On Pre-Emptive Resume Priority Queues*: Anns.Math.Stat., 1964, 600-612.

[WHITTLE 80] WHITTLE .P ., *Multi- Armed Bandits and the Gittins Index*: J.Roy.Statist.Soc., Ser.B ,42, 1982, 143-149.

[WHITTLE 81] WHITTLE .P ., *Arm- acquiring Bandits*: Ann. Prob.,9,1981, 284-292.

[WHITTLE 82] WHITTLE.P., *Optimisation Over Time* : Vol.1, Wiley,1982.