MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DE BLIDA
INSTITUT DE MATHEMATIQUES

THESE DE MAGISTER

SPECIALITE : MATHEMATIQUES APPLIQUEES

OPTION: MODELISATION MATHEMATIQUE ET TECHNIQUES DE DECISION

ETUDE ET SIMULATION D'UN ORDONNANCEMENT PAR PRIORITES BORNEES MOYENNES APPLIQUE A UN SYSTEME MULTIPROGRAMME MONOPROCESSEUR

PAR
AKRIB FOUAD

Présentée devant le Jury constitué de :

Président:	**Haddad Zahir MC, E.N.P. Alger**
Rapporteurs:	**Proust Christian Prof, E3I . Tours**
	Aissani Amar Prof, Univ . Blida
Examinateurs:	**Aissani Djamil Prof, Univ. Bejaia**
	Chouaf Benamar MC, Univ. Sidi Bel Abbès
Invitée :	**Oukid-Khouas Saliha Doct, Univ.Blida**

UNIVERSITE DE BLIDA .
Fevrier 1997

DEDICACES

A la mémoire de ma grand-mère MAMMANI.

A ma mère pour sa patience et son sacrifice.

A mon père pour son soutien.

A ma soeur Dalila et mon frère Mohamed-Amine.

A Amina pour sa confiance.

A mon Cher Ami Sofiane Bouchou.

Je dédie ce modeste travail.

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont participé de loin ou de prés à l'élaboration de ce modeste mémoire.

Je remercie en particulier le Professeur Christian-Proust d'avoir été à l'origine de ce sujet.

Mes remerciements vont également au Professeur Amar-Aissani pour ses précieuses indications tout au long de ce travail.

J'exprime toute ma reconnaissance au professeur Jean-Pierre-Asselin-De-Beauville et tout le personnel de l'école E-3I de Tours (France) de m'avoir accueilli au sein de leur établissement dans le cadre de l'accord programme avec l'institut de mathématiques de l'université de Blida (Algerie).

Mes remerciements vont également au personnel du Centre Des Etudiants Stagiaires C.I.E.S en particulier Mme Augendre de m'avoir facilité mon séjour en France.

Je n'oublie pas de rendre grâce aux responsables de la post-graduation de l'Université de Blida et de l'institut de mathématiques qui ont veillé au succès de cet accord programme.

Je remercie très sincèrement les responsables de l'I.N.E.S de Médéa plus particulièrement Dr Tchikou Ahmed Directeur de l'I.N.E.S pour ses encouragements, Mr Zouaouine Hicham responsable du matériel informatique, Mr Boudekkas Mohamed responsable de la bibliothèque pour l'aide qu'ils m'ont apporté.

Je remercie cordialement, Monsieur Gilles Nicolas, la famille Bouchou en particulier Sofiane, La famille Bendali Braham en particulier Djamel de m'avoir permis d'utiliser le matériel informatique.

Qu'il me soit permis enfin de remercier tous les membres du jury d'avoir accepté de juger mon travail.

RESUME

Un système d'exploitation est sans doute le côté logiciel le plus important et le plus déterminant de la qualité des performances d'un ordinateur, ceci est prouvé et confirmé par le formidable investissement déployé pour son étude et son amélioration. Le concepteur de ce système doit être capable de fournir un ensemble de programmes qui coopèrent à la gestion des ressources de la machine.

Le développement surprenant dans le domaine du génie des matériaux a obligé les concepteurs des systèmes informatiques à essayer d'exploiter au mieux les possibilités offertes par ces matériaux, et de concevoir des algorithmes et des stratégies d'implémentation capables de s'adapter aux exigences actuelles de plus en plus grandes. L'un des critères les plus étudiés est celui de l'équité du système. En effet les concepteurs cherchent à doter le SCHEDULER, principal processus du système d'exploitation, d'une stratégie d'allocation équitable ou à défaut asymptotiquement équitable.

Lors de ce travail, nous présentons une récente stratégie d'ordonnancement appelée « Ordonnancement Par Priorités Bornées Moyennes » ou O.P.B.M imaginée par Haro-Proust accompagnée de ses deux conjectures.

L'étude a montré qu'en fait, cette stratégie ne dépendait que d'une seule conjecture. Une modélisation en file d 'attente à système fermé a été proposée puis simulée avec une loi de service de type général, quelques exemples numériques sont présentés pour valider les résultats théoriques issus de la conjecture, notamment l'équité asymptotique de l'ordonnancement O.P.B.M.

Pour que cet ordonnancement puisse être appliqué à des systèmes réels plus simples contenant des processus de durée de vie finie, les temps d'exécution de ces processus doivent alors satisfaire une équation différentielle formulée dans cette étude explicitant le fonctionnement du système.

Enfin, nous avons pu établir une analogie entre la nature stochastique de cette méthode d'ordonnancement et les processus de renouvellement, d'où nous tirons quelques résultats intéressants renforçant l'O.P.B.M.

Mots clés: Systèmes d'exploitation, , Ordonnancement, Files d'attente, Priorités.

SOMMAIRE

INTRODUCTION

Un système d'exploitation est sans doute le côté logiciel le plus important et le plus déterminant de la qualité des performances d'un ordinateur, ceci est prouvé et confirmé par le formidable investissement déployé pour son étude et son amélioration. Le concepteur de ce système doit être capable de fournir un ensemble de programmes qui coopèrent à la gestion des ressources de la machine.

Le développement surprenant dans le domaine du génie des matériaux a obligé les concepteurs des systèmes informatiques à essayer d'exploiter au mieux les possibilités offertes par ces matériaux, et de concevoir des algorithmes et des stratégies d'implémentation capables de s'adapter aux exigences actuelles de plus en plus grandes. L'un des critères les plus étudié est celui de l'équité du système. En effet les concepteurs cherchent à doter le SCHEDULER, principal processus du système d'exploitation, d'une stratégie d'allocation des ressources équitable ou à défaut asymptotiquement équitable. Dans ce mémoire, nous présentons dans le premier et le deuxième chapitre l'évolution, les fonctions principales des systèmes d'exploitation et leurs mécanismes de base. Le troisième chapitre comprend une étude sur les systèmes à file d'attente du type M/G/1 avec priorités constantes. D'autres modèles plus complexes notamment les files d'attente à un serveur avec priorités dépendantes du temps sont introduits avec leurs principaux résultats.

Dans le quatrième chapitre une étude détaillée est menée sur le modèle à temps partagé dit modèle R.R, appliqué aux systèmes à files d'attente à un seul serveur.

Dans le cinquième chapitre, nous présentons une récente stratégie d'ordonnancement appelée « ordonnancement par priorités bornées moyennes » ou O.P.B.M imaginée par Haro-Proust accompagnée de ses deux conjectures.

L'étude a montré qu'en fait, cette stratégie ne dépendait que d'une seule conjecture. Une modélisation en file d 'attente à système fermé a été proposée puis simulée avec des lois de type général gérant le blocage des processus, quelques exemples numériques sont présentés pour valider les résultats théoriques issus de la conjecture, notamment l'équité de l'ordonnancement O.P.B.M.

Pour que cet ordonnancement puisse être appliqué à des systèmes réels plus simples contenant des processus de durée de vie finie, les temps d'exécution de ces processus doivent alors satisfaire une équation différentielle formulée dans cette étude explicitant le fonctionnement du système.

Enfin, nous avons pu établir une analogie entre la nature stochastique de cette méthode d'ordonnancement et les processus de renouvellement, d'où nous tirons quelques résultats intéressants renforçant l'O.P.B.M.

CHAPITRE 1

I.1. GENERALITE SUR LES SYSTEMES D'EXPLOITATION

I.1.1. SYSTEME D'EXPLOITATION

Un système d'exploitation est le côté logiciel le plus important et le plus valorisant de la qualité des performances d'un ordinateur électronique, c'est ce qui justifie l'effort humain et financier investi pour son amélioration tout au long de ces dernières décennies. Le concepteur doit être en mesure de fournir des programmes coopérants pour l'optimisation de la gestion des ressources de la machine physique en constance développement dans le but de satisfaire les principales fonctions.

I.1.2. FONCTIONS D'UN SYSTEME D'EXPLOITATION

a)PARTAGE DES RESSOURCES

Un système d'exploitation doit partager les ressources de la machine entre plusieurs utilisateurs simultanés. L'objectif de cette fonction est l'optimisation de l'utilisation des ressources telles que la mémoire, l'unité centrale (ou processeur(s)), les unités d'entrées-sorties ...etc.

b)PRESENTATION D'UNE MACHINE VIRTUELLE A L'UTILISATEUR.

Cette principale fonction qui consiste en géneral à convertir un dispositif (électronique) physique en une machine virtuelle dont les caractéristiques sont plus performantes et plus significatives que celles de la machine matérielle masquée. Un exemple simple qui met en évidence la différence entre une machine virtuelle et une machine matérielle est la ressource entrée-sortie notée habituellement E/S ou I/O. Le système d'exploitation décharge l'utilisateur de la complexité de la machine matérielle ou physique et lui présente une machine dotée de performances d'E/S équivalentes mais beaucoup plus simples à utiliser. Ecrire un nom sur écran devient une simple écriture sur machine, alors que réellement c'est un phénomène plus complexe.

I.2. EVOLUTION DES SYSTEMES D'EXPLOITATION INFORMATIQUES

I.2.1. DEFINITION

Un système informatique est généralement composé de quatre parties:

- La machine physique (HARDWARE), qui comporte tout le matériel physique (électronique), qui constitue la machine.

- Le système d'exploitation qui, lui, comporte le côté logiciel capable de faire dialoguer la machine physique.
- Les programmes d'applications, qui représentent les tâches prises en charge par la machine physique dotée de son système d'exploitation.
- Les utilisateurs, qui représentent les interlocuteurs de la machine et les fournisseurs de tâches.

Brièvement, nous allons passer en revue l'évolution de ces systèmes.

I.2.2. LA PORTE OUVERTE

Les premiers systèmes informatiques étaient dépourvus de système d'exploitation. L'utilisateur devait en plus de l'exploitation réaliser les tâches d'opérateur suivantes:

1/ Placer les cartes du programme source dans l'unité d'entrée.

2/ Initialiser un programme de lecture de cartes.

3/ Lancer la compilation.

4/ Placer les cartes de données (si besoin) dans le lecteur de carte.

5/ Initialiser l'exécution du programme compilé.

6/ Extraire les résultats de l'imprimante.

I.2.3. LE MONITEUR D'ENCHAINEMENT

Une amélioration du système précèdent, consiste à faire prendre en charge par la machine sans intervention de l'opérateur, la lecture, la compilation, le chargement et l'exécution des programmes.

Le rôle de l'opérateur se réduit au changement des cartes à une extrémité et l'extraction des résultats de l'autre. Le système est nommé Moniteur d'enchaînement. Son principal inconvénient est la surcharge; En outre, ce système doit être capable de réagir en fonction des cartes de contrôle. Des erreurs peuvent alors apparaître vu la nom homogénéité des tâches demandées.

Enfin ce système est principalement dépendant et limité par les performances d'entrées-sorties, car la machine est occupée pendant l'entrée des données et la sortie des résultats.

I.2.4. Traitement par lots (système Batch)

Dans ce type de système, le moniteur d'enchaînement charge le programme et lui transfert le contrôle; une fois le chargement terminé, le programme redonne le contrôle au moniteur d'enchaînement qui continue avec la prochaine carte de contrôle, et ceci est réitéré jusqu'à l'expiration de toutes les cartes de contrôles du job courant, par la suite le moniteur passe au job suivant.

Un des inconvénients majeurs de cette technique est l'impossibilité de contrôler les programmes en cours d'exécution; s'il subsiste alors une erreur dans un programme, l'utilisateur est obligé de revoir tout le job.

I.2.5. La Multiprogrammation

Le but de cette technique est l'augmentation de l'utilisation du processeur en le forçant à toujours prendre en charge une tâche d'un programme donné, ceci implique la présence simultanée de plusieurs programmes en mémoire, ainsi le processeur est alloué à un job dès que le job en cours d'exécution nécessite une attente. Cette technique; bien qu'elle paraisse adéquate aux exigences actuelles n'est pas sans pour autant poser des problèmes aux concepteurs, entre autres, le maintien des jobs en mémoire (donc la gestion de la mémoire et les techniques d'ordonnancement des tâches). Dressons un tableau comparatif mettant en évidence les avantages et les inconvénients de la monoprogrammation et de la multiprogrammation.

Monoprogrammation	Multiprogrammation
-Système d'exploitation facile à concevoir seule contrainte protection de la partie résiduelle.	- S-E complexe. - Coût de conception élevé pour la protection de la mémoire en particulier.
- Utilisation médiocre des ressources	- Possibilité de mieux équilibrer les charges des ressources.
-Temps de réponse imposé pour les travaux longs	- Amélioration des temps de réponse des travaux courts

I.2.6. Le temps partagé

Ce mode consiste à offrir à chaque utilisateur une machine " Virtuelle " propre à lui, l'utilisateur se croit alors le seul occupant de toutes les ressources de la machine.

Ce mode privilégie l'interaction homme-machine, contrairement au mode Batch. L'unité centrale est MULTIPLEXEE entre les terminaux. Chacun d'entre eux est servi pendant un laps de temps fixé par le concepteur appelé "quantum" de temps, d'où son nom, temps partagé ou"Time sharing".

I.2.7. Système à temps réel

Ce type de système est orienté vers des applications bien spécifiques, en particulier les systèmes de contrôle (industrie, surveillance, sécurité). Des capteurs recueillent l'information qu'ils transmettent au calculateur (ordinateur) qui se charge d'analyser et d'agir en conséquence.

I.3. Conclusion

Nous pouvons conclure ce bref exposé, qu'un système d'exploitation moderne doit répondre aux exigences minimales suivantes, et c'est l'application qui détermine quelle fonction doit primer sur les autres.

- Prise en charge des jobs.
- Interprétation du langage de contrôle des jobs (compilation).
- Gestion des erreurs.
- Gestion des I/O (E/S).
- Gestion des interruptions (nous y reviendrons).
- Contrôle et évaluation des ressources.
- Protection de la mémoire.
- Accès multiples.
- Bonne interface avec l'utilisateur .

CHAPITRE II

INTRODUCTION

Tout au long de notre travail, nous allons nous intéresser à un système d'exploitation à traitement simultané, ce qui nous amène à introduire et définir quelques concepts de base.

II.1. Programme - Processeur et Processus

II.1.1. Processus

Comme on l'a déjà décrit, un système d'exploitation doit être en mesure de réaliser un ensemble d'activités. Chaque activité implique l'exécution d'un ou de plusieurs ***programmes;*** Un ***processus*** désigne justement une de ces activités, plus précisément un processus est une suite d'actions obtenues par l'exécution d'une séquence d'instructions donc un « programme », dont le résultat constitue une des fonctions du système d'exploitation

Il y a lieu de préciser qu'un processus peut faire appel à plusieurs programmes, en revanche, un programme peut être activé par plusieurs processus.

II.1.2. Processeur

Un ***processeur*** est l'agent qui active le ***processus*** afin d'exécuter le ***programme*** associé. Un processeur est généralement réalisé à partir de matériel physique (électronique), mais il arrive qu'il soit la combinaison de matériel et de logiciel.

En conclusion à ces définitions, nous pouvons dire qu'un processus est une ***action***, une ***séquence*** d'opérations qui se déroule pour réaliser une tâche déterminée, c'est aussi un programme en exécution, une entité active capable de générer des événements, tandis qu'un processeur est l'agent exécutant de cette activité.

Les concepts de processus et de processeur sont utiles pour interpréter deux caractéristiques fondamentales des systèmes d'exploitation: La simultanéité et le non-déterminisme.

II.1.3. La simultanéité

C'est l'activation de plusieurs processus, s'il y a autant de processeurs que de processus, tout se passe normalement, mais en pratique, le nombre de processeurs est inférieur à celui des processus activés; Le phénomène de simultanéité sera obtenu en faisant basculer les processeurs d'un processus à l'autre.

II.1.4. Le non - déterminisme

Comme un processus peut être considéré comme une séquence d'actions interrompables, le non-déterminisme explique l'ordre imprévisible dans lequel ces interruptions surviennent.

II.2. Communication entre processus

Un processeur peut basculer d'un processus à un autre, et un processus peut être impliqué dans la réalisation de plusieurs tâches; cela sous-entend que les processus ne sont pas isolés et traités indépendamment les uns des autres, mais qu'ils coexistent dans l'ordinateur pour concourir à l'exploitation des tâches de l'utilisateur, et cela implique aussi qu'ils sont en compétition permanente pour l'acquisition des ressources limitées disponibles. Tout ceci exige un établissement d'un mode de communication entre processus, et ce essentiellement dans les domaines particulièrement sensibles suivants:

II.2.1. L'exclusion mutuelle

Nous distinguons deux classes de ressources d'un système informatique. Les ressources partageables pouvant être utilisées par plusieurs processus simultanément par basculement; les ressources non partageables qui ne peuvent être exploitées que par un unique processus à un moment donné, ceci est dût principalement aux contraintes de types **matériel** et **conceptuel**.

Expliquons cette deuxième contrainte à l'aide d'un exemple. Soit une zone mémoire donnée, si un processus active un processeur pour évaluer son contenu, alors qu'un autre processus l 'active pour la modifier, le résultat serait alors incohérent. Pour résoudre ce problème d'exclusion mutuelle, il faut que le système d'exploitation soit capable d 'assurer l'accessibilité aux ressources non partageables par un seul processus à la fois. Nous présenterons les techniques adéquates pour cela ultérieurement.

II.2.2. La synchronisation

Le non-déterminisme fait que la vitesse relative des processus, leurs fréquences d'interruption, leurs occupations du processeur etc..., nous obligent à synchroniser leurs mouvements afin d'assurer une coopération optimale entre ces processus en vue de minimiser les temps d'attente des résultats (response--time).

II.2.3. Le blocage [Dead Lock]

Ce cas survient lorsque plusieurs processus occupants certaines ressources, cherchent à prendre possession, au même moment, des ressources occupées par d'autres processus qui eux même demandent à acquérir les ressources occupées par les premiers processus.

II.3. Techniques et notions utiles

Nous allons présenter ci-dessous, quelques techniques utilisées pour contourner les problèmes suscités dans la section précédente, ainsi que quelques notions utiles pour la suite de ce travail.

II.3.1. les sémaphores

Introduit par Dijkstra en 1965, un sémaphore est un entier non négatif qui ayant une valeur initiale proposée et judicieusement choisie par le concepteur, ne peut être modifiée que par l'une des deux opérations suivantes:

a)- SIGNAL (nom du sémaphore: S):

L'opérateur SIGNAL appliqué sur le sémaphore S agit suivant la règle suivante: SIGNAL accroît la valeur actuelle du sémaphore S d'une unité, et d'une manière **indivis,** ce qui veut dire, que cette opération se fait d'une manière atomique. Plus précisément, quand un processus modifie la valeur du sémaphore S, aucun autre processus ne peut le faire **simultanément**.

b)- WAIT (nom du sémaphore : S) :

L'opérateur WAIT (S) décroît la valeur du sémaphore (S) d'une unité tant que sa valeur est supérieure à zéro, dans le cas contraire, le processus voulant actionner WAIT (S) est contraint d'attendre qu'un autre processus opère SIGNAL (S) sur le même sémaphore. Là encore, la remarque sur l'indivisibilité de SIGNAL (S) s'applique sur WAIT (S).

II.3.2. Activités parallèles [7]

Introduisons cette notion par un exemple instructif. Soit le programme qui lit les informations de deux périphériques, puis écrit l'information concaténée sur un troisième périphérique.

Début

lire (a) ;

lire (b) ;

c = a + b ;

écrire c ;

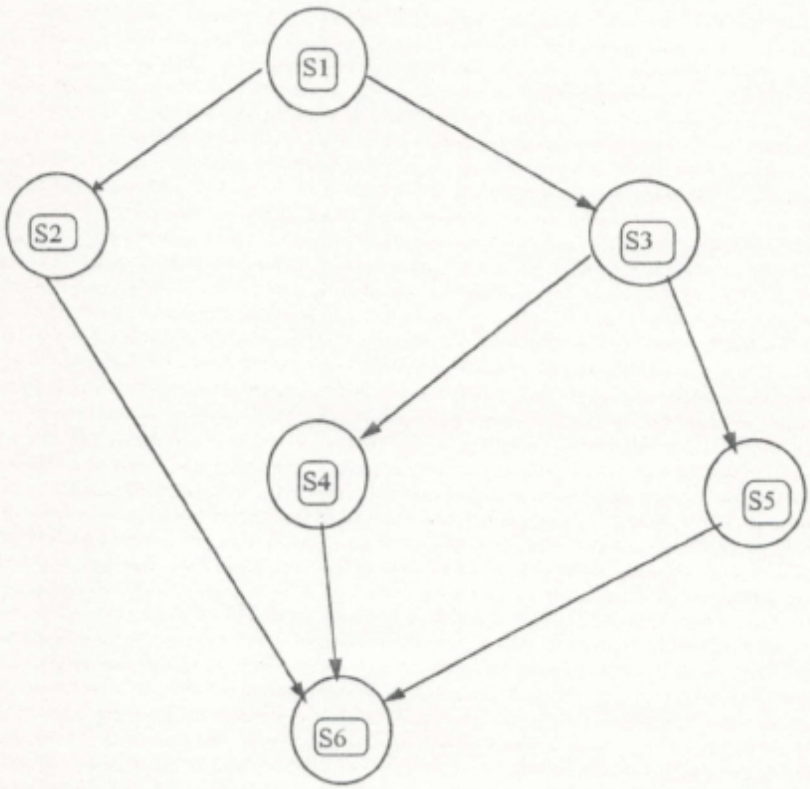
Fin.

Si on veut exécuter ces instructions simultanément, il est évident que l'instruction c = a+b ne peut avoir un sens avant la fin des deux instructions de lecture, on dit qu'on est on présence de contrainte de **précédence.**

II.3.3. Graphe de précédence

C'est un graphe orienté sans cycle dont les sommets représentent les instructions et les arcs représentent les dépendances de précedence entre les instructions. Un arc (S_i , S_j) signifie que le sommet S_j ne peut être exécuté avant la fin de l'exécution du sommet S_i.

Exemple.



S6 ne peut s'exécuter qu'après la fin de S2, S4 et S5. Remarquons que les instructions S_2 et S_3 ne sont pas liées par la contrainte de précedence, ces instructions sont appelées « instructions parallèles ». Se pose la question de savoir de façon générale comment détecter ces instructions parallèles.

II.3.3.1. Conditions de Bernstein

Soit I une instruction d'un programme, on pose R(I) l'ensemble des variables de l'instruction I qui ne changent pas par l'exécution de l'instruction I, W(I) l'ensemble des variables restantes c'est à dire les variables mises à jour par I. On dira que deux instructions I1 et I2 peuvent s'exécuter en parallèle si les conditions (de Bernstein) suivantes sont satisfaites:

a) $R(I1) \cap W(I2) = \Phi$

b) $W(I1) \cap R(I2) = \Phi$

c) $W(I1) \cap W(I2) = \Phi$

Bien que le graphe de précédence mette bien en évidence les contraintes de précédence des instructions d'un programme (des processus), il est inutilisable en programmation. D'autres techniques sont alors mises en oeuvre, la plus puissante sans doute est l'outil **FORK** et **JOINT**[5] qu'on exposera pas ici.

II.3.4. Problème de la section critique et sa résolution

Soit un système de k processus qui coopèrent. Chaque processus dispose d'un segment de code dit "section critique", dans lequel le processus peut lire des variables communes. Une caractéristique essentielle de ce système est que quand un processus est dans sa section critique aucun des autres processus ne peut exécuter la sienne, donc les exécutions des sections critiques sont **mutuellement exclusives** dans le temps.

Le problème a été résolu par [Dijkstra.1965], en introduisant les sémaphores, Dijkstra propose alors d'entourer chaque section critique par des opérations WAIT et SIGNAL sur un seul sémaphore dont la valeur initiale serait égale à 1, si chaque section critique des k processus est programmée sous la forme:

```
WAIT (Mutex)
.
. section critique.
.
.SIGNAL (Mutex).
```

Mutex: nom du sémaphore.

Le problème de l'exclusion mutuelle serait alors résolu. De la même façon, le problème de la synchronisation évoqué plus haut, peut être résolu à l'aide de l'outil sémaphore comme suit:

Programme pour le processus A	Programme pour le processus B
.	.
.	.
.	.
.	.
.	.
.L1: WAIT (synch)	.L2: SIGNAL(synch).
.	.
.	.
.	.

Le processus A ne peut continuer au delà du point L1, sans que B ne soit arriver à L2, il suffit bien entendu que la valeur du sémaphore synch. soit initialisée à zéro.
Le blocage (Dead Lock) peut aussi être traité par ce puissant outil en évitant de disposer les sémaphores de la façon suivante:

Processus A	Processus B
.	.
.	.
WAIT (semp X)	WAIT (semp Y)
.	.
.	.
WAIT (semp Y)	WAIT (semp X).

Car si les valeurs initiales des sémaphores X et Y sont égales à 1, et chaque processus effectue sa première opération WAIT associée, ce qui les ramène aux valeurs nulles, ni l'un ni l'autre des deux processus ne peut continuer au delà, et le blocage se produit. Signalons que deux politiques sont mises en pratique pour venir à bout du blocage, la politique de prévention et la détection et traitement du blocage. Ni l'une ni l'autre des deux politiques ne peut actuellement s'affirmer comme étant la plus efficace.

II.3.5. Les moniteurs

Nous avons vu qu'une mauvaise disposition des sémaphores, peut entraîner un blocage, c'est pour cela que vers 1974 [HOAR 4] introduit l'outil MONITEUR.

II.3.5.a. Définition

Un moniteur est constitué d'un ensemble de variables d'états (données d'un objet partagé par plusieurs processus), un ensemble de procédures manipulant ces variables, certaines sont externes dites points d'entrée du moniteur, et enfin un module de programme d'initialisation de l'objet, celui-ci (programme) n'est exécuté qu'une seule fois lors de la création de l'objet.

La structure du moniteur assure le bon emplacement des sémaphores, car le compilateur d'un langage comportant des moniteurs doit vérifier que l'accès à un objet partagé ne peut être obtenu que grâce à l'appel d'une procédure du moniteur correspondant. Le compilateur doit aussi s'assurer que les procédures de chaque moniteur sont des sections critiques mutuellement exclusives.

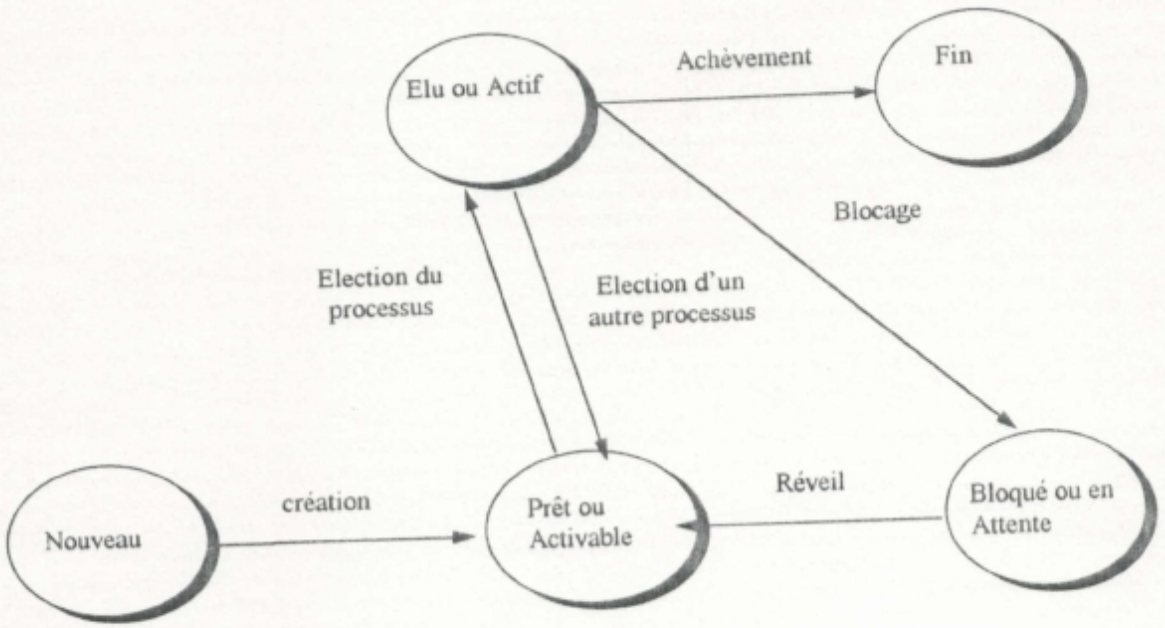
Il est clair que de cette façon le programmeur est déchargé de la responsabilité de l'exclusion mutuelle, qui est laissée à la charge du compilateur. Toute- fois les autres formes de synchronisation sont toujours sous la gérance du programmeur.

II.4. Modèle de processus et système d'interruption.

Après avoir introduit la notion de processus plus haut, nous allons expliquer plus techniquement son rôle sans avoir à s'intéresser à l'implémentation matérielle. Pour cela définissons:

II.4.1. Les différents états d'un processus

Depuis sa création jusqu'à sa destruction qui signifie sa fin, un processus mute d'état en état. La figure suivante met en évidence un graphe des transitions possibles du processus.



Etat actif ou Elu: Processus en cours d'exécution.

Etat bloqué ou en Attente: Processus en attente d'un événement (extérieur) qui puisse le débloquer pour pouvoir continuer . Ex: exécution de SIGNAL sur un sémaphore de valeur nulle.

Etat Prêt ou Activable: Processus prêt pour exécution mais reste en attente car la ressource demandée n'est pas disponible; dès qu'elle le sera, il sera pris en charge.

II.5. Les systèmes d'interruption

II.5.1. Mécanisme de changement d'état [SWAP]

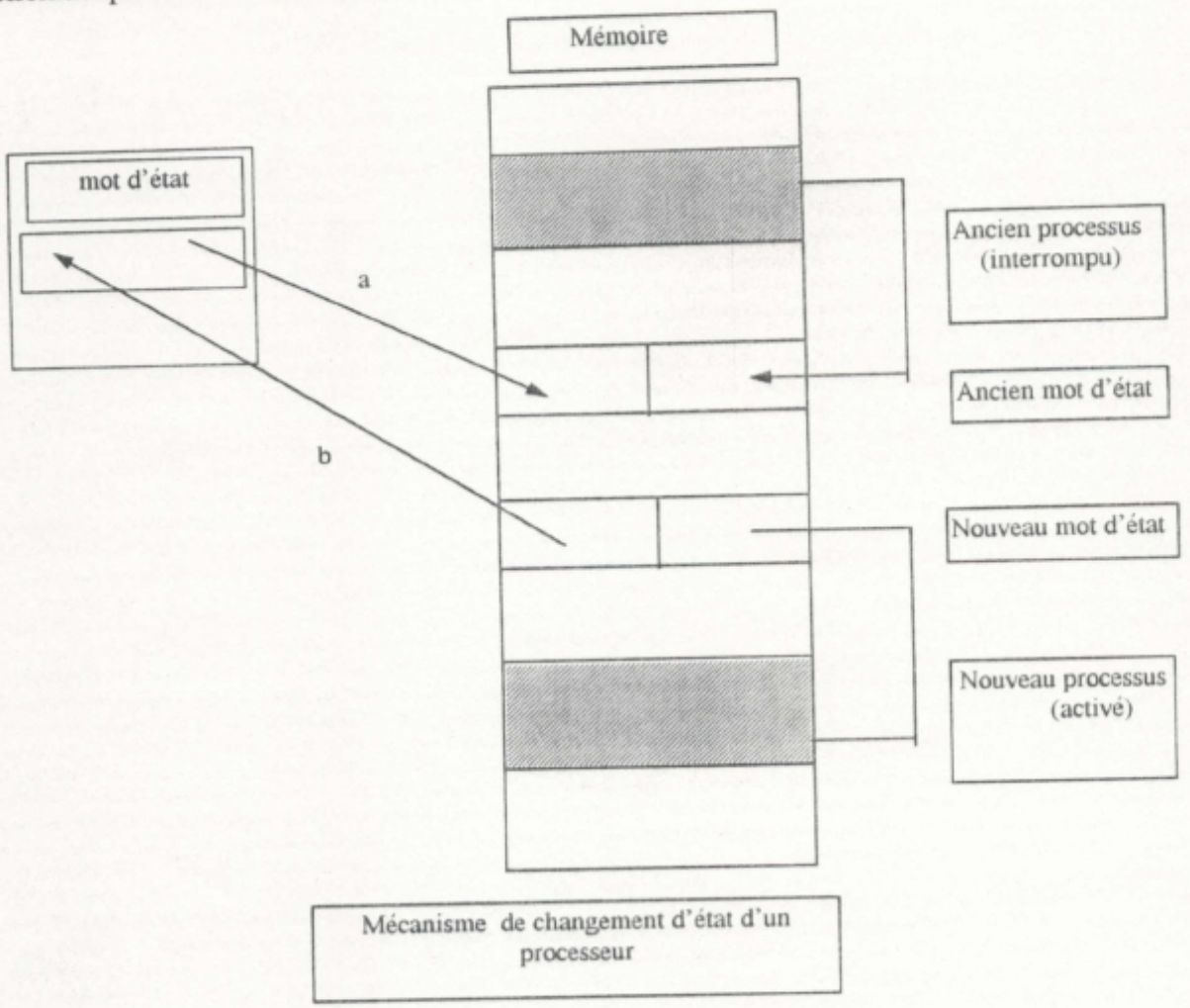
Le mécanisme de changement d'état d'un processus permet en une seule opération indivisible de:

a) Ranger dans un emplacement spécifié de la mémoire (Adresse), le contenu courant du mot d'état du processus (P, W , S).

b) Charger dans le mot d'état un nouveau contenu préparé à l'avance dans un emplacement spécifié de la mémoire .

Ce chargement n'est réalisable que si le processus se trouve dans un état bien défini, dit point observable ou point interruptible (discrétisation du temps).

Schématiquement nous avons :



Mécanisme de changement d'état d'un processeur

Le changement d'état est déclenché suivant l'état d'un indicateur (voyant) consulté par le processeur lors de l'exécution de chaque instruction.

II.5.2. Interruption

II.5.2.1. Définition

Le changement d'état de l'indicateur correspond à ce qu'on appelle une interruption. Cette interruption permet au processeur de suspendre l'exécution du programme (ou instruction, ou processus) en cours et exécuter un programme spécifique dit programme d'interruption.

Il est bon de signaler qu'il existe plusieurs types d'interruption, un mécanisme est alors mis en place pour distinguer ces niveaux d'interruptions et éventuellement les traiter. C'est ce qu'on appelle mécanisme d'interruption.

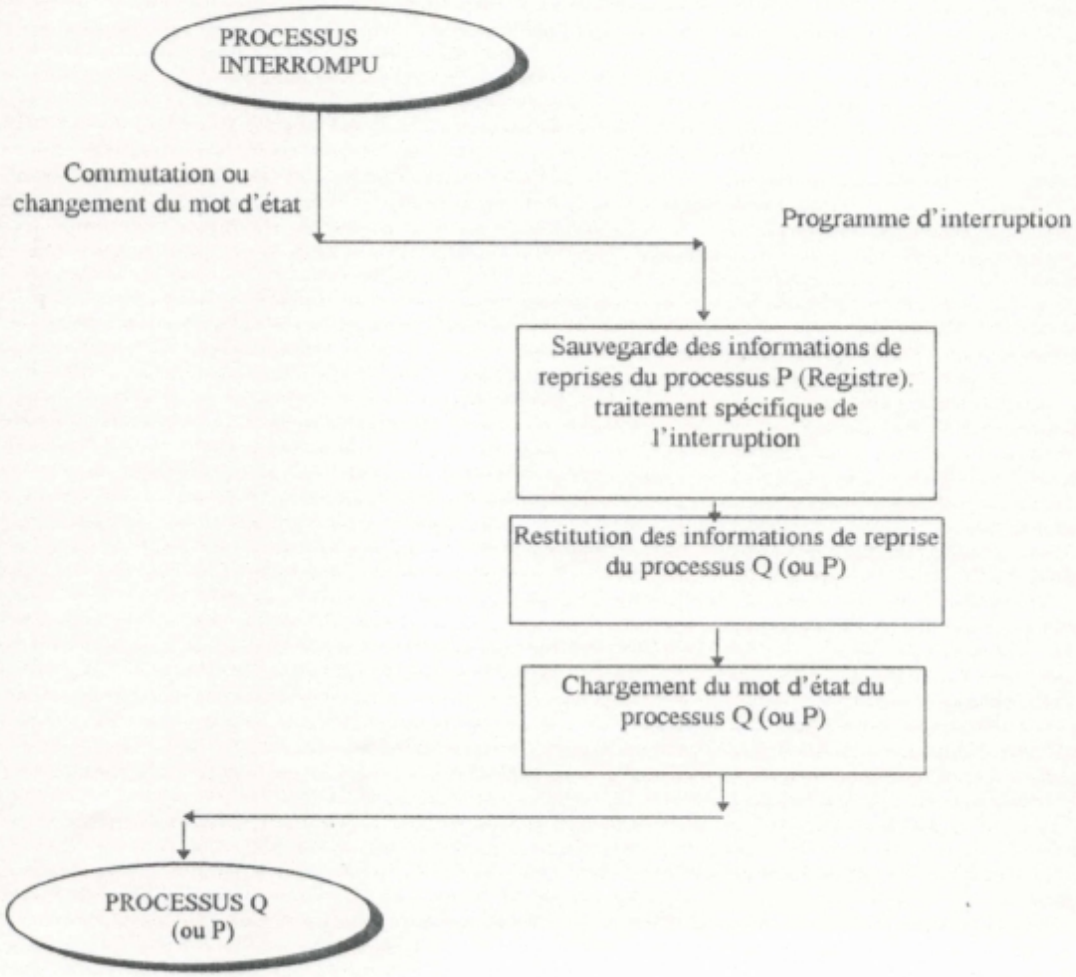
II.5.2.2. Priorité d'interruption

Le fait qu'il y ait plusieurs niveaux d'interruptions amène à considérer le cas où deux indicateurs correspondant à deux causes distinctes soient modifiés simultanément, le conflit est alors réglé en établissant un ordre de priorité entre ces causes (ou niveaux). Cet ordre de priorité peut être fixe ou modifiable.

I.5.2.3. Schéma général d'un programme d'interruption

Une interruption dans le cas général, comme on l'a déjà défini a pour fonction de forcer le processeur à réagir à un événement. L'exécution du programme en cours est suspendue. Après le traitement de cette cause d'interruption par le programme spécifique d'interruption, le programme (ou processus) repris peut oui ou non coïncider avec le programme interrompu.

II.5.2.3 SCHEMA GENERAL D'UN PROGRAMME D'INTERRUPTION



II.6. Gestion et allocation du processeur

On entend par gestion du processeur, une stratégie d'allocation qui détermine l'ordre d'exécution des processus en concurrence. Pour aborder ce sujet, nous devons présenter deux **processus systèmes** indispensables et impliqués directement dans cette stratégie.

II.6.1.LE SCHEDULER [ordonnanceur]

Le SCHEDULER est un processus système responsable des tâches suivantes:

a) Introduction de nouveau processus: Cette tâche consiste à initier un processus bloqué dés que la cause de son blocage est traitée. Par exemple: libération d'une ressource demandée par ce processus qui était déjà occupée.

b) Attribution des priorités aux processus.

C'est l'une des tâches les plus importantes qui concerne notre travail . Cette tâche consiste quant à elle à assigner les priorités aux processus suivant une règle définie par le concepteur, afin de pouvoir les exécuter sans ambiguïté et d'une façon rationnelle, dés que ces processus passent à l'état "prêt" discuté plus haut. C'est cette règle ou stratégie qui détermine l'efficacité des performances du système d'exploitation.

Plusieurs algorithmes (ou stratégies) d'évolution des priorités, remarquables ont été mis en pratique que nous présenterons plus loin.

c) Réalisation des politiques d'allocation de ressources.

Le SCHEDULER doit veiller à ce qu' aucune des ressources ne soit sur ou sous exploitée. C'est ce que l'on appelle **l'équité du système.**

L'équité signifie que les processus de même priorité doivent être servis par le processeur d'une façon équitable. C'est à dire, qu'ils disposent du processeur pendant la même fraction de temps en moyenne.

Ayant passé en revue les fonctions du SCHEDULER, nous introduisons un autre processus système intimement lié aux activités du SCHEDULER.

II.6.2. DISPATCHER

Après que le SCHEDULER ait assigné les priorités suivant la règle pré-définie; un autre processus système réalise l'élection proprement dite à la détention du processeur . Ce processus est appelé **Dispatcher** ou **Distributeur**.

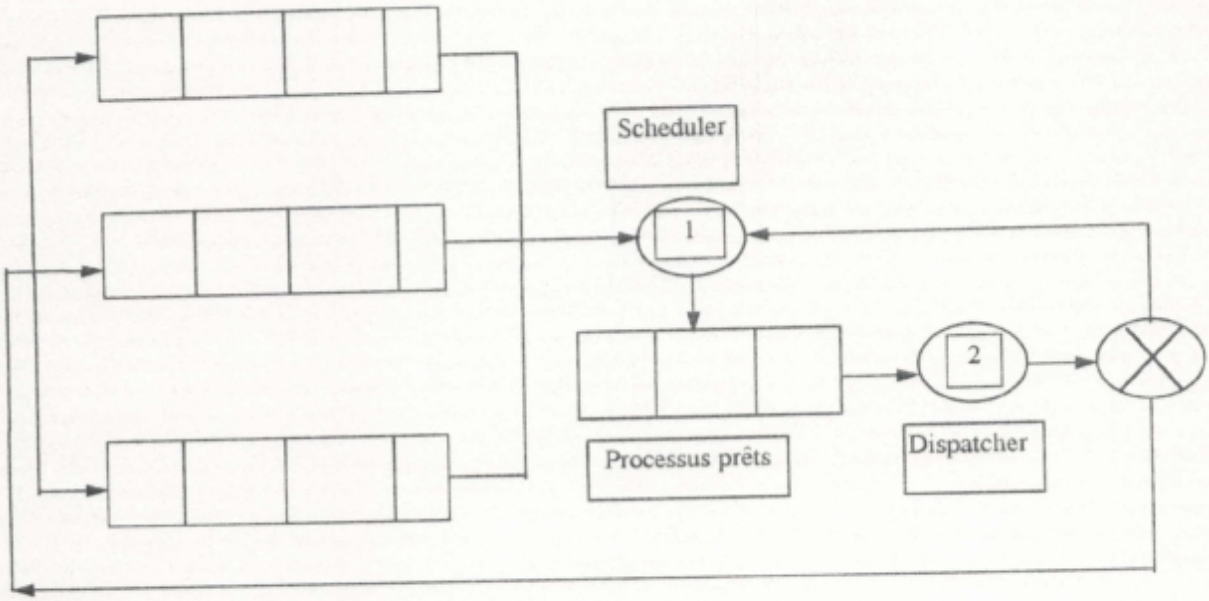
Les cas d'activation de ce processus sont les suivantes:

a) Interruption externe modifiant l'état d'un processus.

b) Blocage d'un processus.

c) Contrôle d'erreur qui suspend le processus en cours d'exécution pendant le traitement d'erreur.

Le schéma explicatif tiré du livre de [KRAKOWIAK 5] est une excellente clarification des tâches de ces deux remarquables processus.



Remarque

Comme le SCHEDULER est aussi un processus, il doit passer par le Dispatcher pour être élu à l'utilisation du processeur; il est donc nécessaire que ce SCHEDULER, se voit assigner d'une priorité supérieure à toute autre priorité afin d'assurer une réaction rapide du système d'exploitation aux différentes variations de l'environnement. Il est aussi intéressant de dresser une comparaison entre une interruption et une entrée en action du SCHEDULER .

L'interruption affecte l'exploitation des processus et l'allocation du processeur central c'est à dire, un événement très fréquent et de très bas niveau (d'un niveau très profond), tandis que l'événement associé à la réaction du SCHEDULER, est de plus haut niveau, affectant les paramètres globaux du processus tels que sa priorité et sa durée d'exécution. Cet événement est donc moins fréquent que l'apparition d'une interruption.

II.7. Algorithmes (stratégie) de scheduling (ordonnancement) [5]

Deux grandes familles d'algorithmes sont d'actualité et mises en oeuvre en pratique.

II.7.1. Les Algorithmes de scheduling sans préemption

Ces algorithmes assurent au processus en cours d'exécution le monopole du processeur jusqu'à sa terminaison (fin).

II.7.1.1. Algorithme du premier arrivé premier servi

Cet Algorithme consiste à classer les processus dans une file appelée techniquement QUEUE D'EXPLOITATION qui relie les descripteurs des processus éligibles, cette file sera ordonnée suivant les arrivées des processus de telle sorte que le processus de tête bénéficiera des services du processeur jusqu'à sa terminaison.

L'inconvénient majeur de cette stratégie d'allocation, concerne les jobs cours qui n'arrivent pas en tête de liste et qui auront donc un temps de réponse très supérieur à leur utilisation effective du processeur.

II.7.1.2 . Algorithme du plus courts job sera le premier S.J.F

Dans cette politique qui n'est valable que dans les systèmes Batch, la file est ordonnée suivant les durées d'exécutions estimées les plus faibles. Son but est de minimiser le temps moyen de réponse des jobs(processus) courts.

Là encore, cette stratégie s'avère peu efficace quand le taux de charge de la file est voisin de l'unité et présente un risque de privation pour les travaux longs. Un moyen subtile d'éviter ce risque, consiste à attribuer aux jobs une priorité croissante avec leurs temps d'attente dans la file, la priorité, initiale étant toujours fonction du temps de service estimé.

II.7.2/Algorithmes de Scheduling avec préemption

Ces algorithmes permettent à tout instant au processus le plus prioritaire de disposer du processeur, cette priorité, peut être fixe ou dynamique.

II.7.2.1.H.R.N

Pour revenir à notre S.J.F transformé comme indiqué plus haut, cette politique est appelée H.R.N. (Highest Response Ratio Next).

Ici la priorité d'un processus est calculée suivant la formule:

$$p_i(t) = \frac{w_i(t) + t_i^s(t)}{t_i^s(t)}$$

Avec : i : fait référence au $i^{ème}$ processus.

W_i (t) : temps d'attente dans la file = t - instant d'arrivée.

t_i^s (t) : temps de service estimé.

cette stratégie devra retenir notre attention car elle nous servira beaucoup dans la suite du travail.

II.7.2.2. ALGORITHME DU TOURNIQUET (ROUND-ROBIN).

Cette stratégie est utilisée dans les systèmes à temps partagé. Son implémentation se réalise de la façon suivante:

La liste des processus éligibles(ou prêts) est organisée dans une file à discipline FIFO., Un processus nouveau (dernier arrivé) est alors rattaché à la queue de la file; le SCHEDULER initialise alors l'horloge pour l'interruption après un laps de temps appelé « quantum ». Il en résulte de ce fait deux cas:

1/le temps d'exécution du job est inférieur au quantum q, alors le processus libère volontairement le processeur et le prochain processus est alors élu.

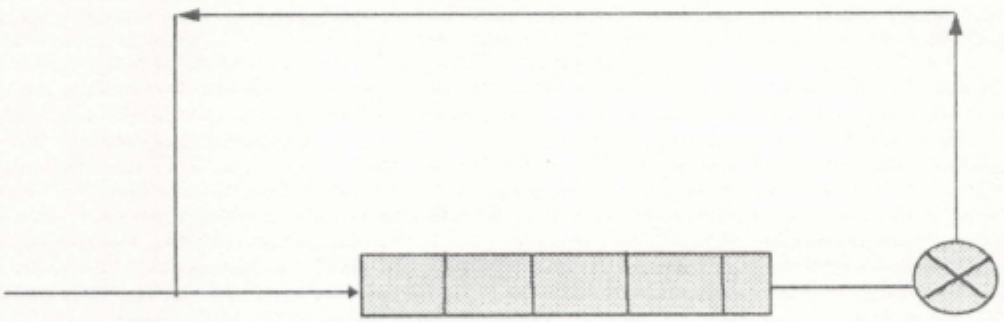
b)le temps d'exécution est supérieur au quantum. Alors le processus consommera son quantum et l'interruption de l'horloge fait intervenir le SCHEDULER pour:

La sauvegarde dans le bloc de contrôle de processus des informations actuelles.

Le replacement du processus à la fin de la liste.

L'élection du prochain processus.

Schématiquement:

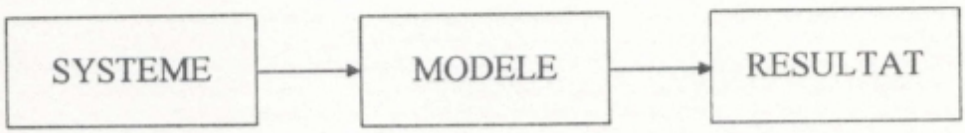


L'aspect mathématique sera étudié en détail ultérieurement.

CHAPITRE III

III. INTRODUCTION

L'étude d'un système réel n'est pas souvent réalisable dans un contexte opérationnel, et ce pour plusieurs considérations, telles que le coût, le temps et les moyens de mesures qui souvent font défaut au chercheur, ou vont à l'encontre de ces moyens. C'est pour cela qu'il est nécessaire de contourner ces problèmes, en représentant le fonctionnement d'un système d'une manière assez précise en s'appuyant sur des outils descriptifs (graphe) mathématiques (probabilités) ou autres, qui nous permettent d'approcher le comportement réel du système. Cette phase de représentation est appelée modélisation. Le résultat de cette opération fournit un symbolisme (code) de description des opérations du système.



Nous précisons que la modélisation peut être exploitée aussi bien pendant les phases de conception que durant les phases d'exploitations opérationnelles; cette phase bien qu'elle soit schématique ou symbolique nécessite un support théorique solide et c'est ce qui détermine un modèle par rapport à un autre. En ce qui nous concerne les systèmes informatiques deviennent de plus en plus complexes. Il est actuellement presque impossible de dimensionner précisément de tels systèmes, car il faudrait optimiser trop de paramètres ce qui mènerait à un modèle impossible à résoudre.

Une première façon de procéder est de décomposer le problème global en des sous problèmes correspondant à des sous-systèmes et de bien veiller à ce que cette décomposition ne fausse pas le contexte opérationnel du système tout entier.

Dans un premier temps il faudrait substituer le sous système par un modèle résolvable soit par la méthode analytique soit par simulation. La méthode analytique la plus utilisée est sans doute le système à file d'attente.

En effet les ressources d'un tel système sont limitées, les jobs vont alors entrer en conflit d'attente pour l'accès à ces ressources ce qui va engendrer des files.

III.1 PHENOMENES D'ATTENTE ET OUTILS DE FILE D'ATTENTE.

Le modèle général système d'attente est constitué comme suit:

Des clients arrivent dans le système et réclament un service, les instants d'arrivée et les durées de services sont souvent des quantités aléatoires. Un système d'attente comprend une zone de service où les clients sont pris en charge et un espace d'attente pour les clients qui ne peuvent accéder à la zone de service. L'objectif principal de cet outil est la description des performances de ce genre de système.

III.1.1 CLASSIFICATION DES SYSTEMES D'ATTENTE.

Pour identifier un système d'attente on a besoin des spécification suivantes.

a/ La nature stochastique du processus d'arrivée, qui est complètement définie par la distribution des intervalles séparant deux arrivées consécutives.

b/ La distribution du temps aléatoires de service.

c/ le nombre de serveurs dans l'espace de service.

d/ la capacité de l'espace d'attente.

Pour cette classification nous convenons d'utiliser la notation symbolique due à Kendall: A/B/S/N/POLITIQUE.

La politique fait référence aux techniques d'ordonnancement de la file, en définissant l'algorithme de choix du prochain client à servir et la façon de disposer les nouveaux clients dans la file.

A: fait référence à la distribution des temps interarrivées.

B: fait référence à la distribution des durées de service.

S: fait référence au nombre de serveur.

N: fait référence à la capacité du système (file +s).

A et B prendront en général leurs formes parmi les distributions très répandues suivantes:

Exponentielle, Hyper-exponentielle, Constante, d'Erlang, de Cox, Déterministe et Générale.

Dans ce chapitre nous étudions une classe particulière de ces systèmes, appelée files d'attente avec priorités.

L'étude des files d'attente avec priorité à fait son apparition au milieu des années cinquante, Harrison 58, Aczel 60 - Avi-itzhak 63- Etschmair 65 - et a été complétée au fur et à mesure du besoin de ces modèles, notamment la monographie de (Jaiswall 68) de (Gnedenko & Al 73) ,(Jackson 72), (Kleinrock 76) et bien beaucoup d'autres travaux qui sont venus enrichir cette branche des files d'attente.

La nécessite d'introduire ces modèles est en premier lieu due à la non homogénéité des jobs traités, le serveur de la file qui se doit dans beaucoup d'applications pratiques attribuer des préférences aux clients servis, cette préférence justement s'interprète par une attribution d'une valeur au client appelée sa priorité qui sera proportionnelle à celle-ci, ainsi le serveur ne suit plus un automatisme dans le service (fifo,lifo) mais doit à chaque interruption (différentes sources d'interruptions) évaluer la priorité des clients présents dans le système et servir le plus prioritaire.

Le but de ces modèles est principalement la recherche de « l'équité » dans le traitement pour palier au phénomène de privation, en prenant en considération la proportionnalité entre l'importance et la durée d'attente du service demandé.

Dans les modèles les plus connus, la modélisation propose une répartition des clients suivant des classes, et chaque classe se voit attribuer sa priorité qui peut être « constante » ou « dynamique », « bornée » ou « non bornée ».

Après avoir introduit ces notions, nous distinguons deux cas de modèles de priorité.

III. 1.1. PRIORITE RELATIVE [18] (NON PREEMPTIVE OU SANS PREEMPTION).

Si un client prioritaire se présente devant la file, le serveur ne le prend en charge qu'après avoir achevé le service du client en cours et qu'aucun client plus prioritaire que celui ci ne s'est présenté pendant la durée résiduelle du service en cours.

III 1.2 PRIORITE ABSOLUE (PREEMPTIVE, AVEC PREEMPTION)

Le client prioritaire arrivant est pris en charge immédiatement par le serveur, dans ce cas deux sous modèles se présentent.

a/ Le service abandonné est repris lorsque son tour viendra là où il était (preemptive resume).

b/ Au contraire, le service abandonné reprendra de zéro (premptive repeat).

III. 2 FILE D'ATTENTE AVEC PRIORITES CONSTANTES

III.2.1. FILE M/G/1 AVEC PRIORITES CONSTANTES [14]

III.2.1.1. PRESENTATION DU MODELE

Nous supposons dans ce système qu'au temps t > o, des clients de différentes priorités arrivent suivant un processus de Poisson de densité λ. Les priorités des arrivées sont des V.A identiquement distribuées et mutuellement indépendantes les unes des autres et indépendantes des instants d'arrivées.

Soit P(y) « la probabilité qu'une arrivée ait une priorité ≤ y ». Nous convenons que le client de plus petite valeur en priorité est plus prioritaire que celui assigné d'une valeur plus grande.

les clients sont servis par un seul serveur suivant les priorités croissantes.

Les temps de service sont supposés des V.A mutuellement indépendantes entre elles et indépendantes des instants d'arrivés.

Notons $H_p(x)$ « la probabilité que le temps de service du client de priorité p est ≤ x ».

Posons $\alpha_p^r = \int_0^\infty x^r \, dH_p(x)$ (1) moment d'ordre r

En particulier $\alpha_p^1 = \alpha_p$

Définissons { η^*_n (p)} » le temps d'attente dans la file du nième client si sa priorité est p ». Nous allons montrer que dans les deux cas si $\lambda \int_0^p \alpha y \, dPy < 1$ alors la distribution stationnaire de { η^*_n (p)} existe et est unique, et que dans le cas contraire la distribution stationnaire n'existe pas.

Notons que si $\lambda \int_0^p \alpha_y \, dPy < 1$ alors dans les deux cas de service [preemptif et non preemptif] $\lim_{n\to\infty} P\{\eta^*_n(p) \le x\}$ existe et est indépendante de la distribution initiale, et de plus elle coïncide avec la distribution stationnaire.

III. 2.1.2. NOTATIONS ET CONSIDERATIONS RELATIVES AU MODELE

Les arrivées des clients de priorité ≤ p suivent un processus de Poisson de densité: $\lambda_p = \lambda \; P(p)$. Sous la condition que les clients entrants dans le système possèdent la priorité ≤ p, la probabilité que le temps de service soit ≤ x est donnée par :

$$A_p(x) = \frac{1}{P(p)} \int_0^p H_y(x)\ d\ P(y) \quad (2)$$

Soit alors $\alpha_p(s) = \int_0^\infty e^{-s\,x}\ d\ A_p(x)$ transformée de Laplace-Stieltjes

Et soit $a_p^r = \int_0^\infty x^r d\ A_p(x)$ moment d'ordre r

et en particulier $a_p = a_p^1$ et $a^r = a_\infty^r$ et $a = a^1$

Considérons aussi que les arrivées des clients de priorité (strictement) < p suivent un processus de Poisson de densité $\mu_p = \lambda\ P(p-0)$.

Sous cette condition la probabilité « le temps de service ≤ x » est notée et donnée par :

$$B_p(x) = \frac{1}{P(p-0)} \int_0^{p-0} H_y(x)\ d\ P(y) \quad (3)$$

Soit alors $\beta_p(s) = \int_0^\infty e^{-s\,x}\ d\ B_p(x)$ transformée de Laplace – Stieltjes.

Et soit $b_p^r = \int_0^\infty x^r d\ B_p(x)$ moment d'ordre r

et en particulier $b_p = b_p^1$

Définissons $q = \inf \{ p , \lambda_p a_p \geq 1 \}$

$\forall\ p$

$q = +\infty$ si $\lambda_p a_p < 1$

Les arrivées des clients de priorité appartenant à [p,q] suivent un processus de Poisson de densité $\upsilon_p = \lambda\ (P(q-0)-P(p))$. Sous ces conditions, un client ayant une priorité $\in$ [p,q] a un temps de service ≤ x avec la probabilité :

$$C_p(x) = \frac{1}{P(q-0) - P(p)} \int_0^{p-0} H_y(x)\, d\, P(y) \quad (4)$$

Soit alors $\nu_p(s) = \int_0^{\infty} e^{-s\,x}\, d\, C_p(x)$ transformée de Laplace – Stieltjes.

$c_p^r = \int_0^{\infty} x^r d\, C_p(x)$ moment d'ordre r

et en particulier $c_p = c_p^1$

Finalement notons la transformée de Laplace-Stieltjes de la distribution stationnaire de $\{\eta^*_n(p)\}$ représentée par $W^*_p(x)$ par:

$\Omega^*_p = \int e^{-sx}\, dW^*_p(x)$ et son riem moment $W^{r*}_p = \int_0^{\infty} x^r\, dW^*_p(x)$

Notons au passage que les transformées de Laplace-Stieltjes convergent si $R(s) \geq 0$.

III. 2.1.3 DETERMINATION DE $W^*_p(x)$

Pour un p fixé, dans les deux cas de service on définit un processus modifié comme suit:

Les clients de priorité ≤ p sont servis dans l'ordre de leurs arrivées sans se préoccuper de la valeur de leur priorité, pourvue qu'elle soit inférieure à p. Leurs temps de service sont des V.A i.i.d suivant la distribution $A_p(x)$, autrement toutes les suppositions faites plus haut sont maintenues.

Dans ce processus modifié, notons $\eta_n(p)$ = « le temps d'attente du nième client si sa priorité est p ».

Nous allons montrer que dans les deux politiques de service on a: $\lim_{n\to\infty} P\{\eta_n(p) \leq x\}$ existe et est indépendante de la distribution initiale, et de plus $\{\eta_n(p)\}$ possède une distribution stationnaire qui concorde avec la distribution limite. dans le cas contraire (i.e. $\lambda_p\, a_p \geq 1$) $\lim_{n\to\infty} P\{\eta_n(p) \leq x\} = 0$.

De plus $\{\eta_n(p)\}$ne possède pas une distribution limite.

Dans les deux cas de service, notons par $\{W_p(x)\}$ la distribution stationnaire de $\{\eta_n(p)\}$ et par : $\nu_p(s) = \int e^{-sx}\, dW_p(x)$ (5)

Dans les deux cas, la distribution de $W_p^*(x)$ sera facilement exprimable à l'aide de $W_p(x)$.

III.2.1.4 .THEOREME [Takacs 63] [14]

Dans les deux cas de service si $\lambda_p a_p < 1$ alors $\{\eta^*_n(p)\}$ a une distribution stationnaire appropriée $W_p^*(x)$ de transformée de L-S donnée sous la forme :

$$\Omega_p^*(s) = \Omega_p(s + \mu_p[1 - \delta_p(s)]) \quad (6)$$

ou $\Omega_p(s)$ est donné e par (5) et $\delta_p(s)$ est la solution de plus petite valeur absolue de l'equation en z : $Z = \beta_p(s + \mu_p(1 - Z))$ (7)

Si $\lambda_p a_p \geq 1$ alors $\{\eta^*_n(p)\}$n'a pas de distribution stationnaire.

III.2.1.5 PREUVE

Le temps d'attente $\eta^*_n(p)$ se voit décomposer en :

a) temps de service des clients de valeur de priorité ≤ p qui étaient présents dans le système juste avant la nième arrivée (plus le temps résiduel du service en cours si on est dans le cas 1).

b) Les durées de service des clients arrivants au cours de son attente (a) et qui ont une valeur de priorité strictement inférieure à p.

La première composante est identiquement distribuée avec $\eta_n(p)$, et s'il y a $j \geq 0$ arrivées de valeur de priorité < p durant cet intervalle de temps, cette durée est identiquement distribuée avec la durée totale des j indépendantes périodes « d'activité » par rapport au système M/G/1, où le processus de Poisson est de densité μ_p , et les temps de service sont i.i.d de distribution $B_p(x)$.

Si nous notons par $D_p(x)$ la fonction de distribution de la période d'activité dans ce processus modifié et $D_p^j(x)$ la j ième convolution de $D_p(x)$ avec elle même, nous pouvons alors écrire dans le cas stationnaire :

$$W_p^*(x) = \left[\sum_{j=0}^{\infty} \int_0^x e^{-\mu_p y} \frac{(\mu_p y)^j}{j!} \, dW_p(y)\right] * D_p^j(x)$$

représente le produit de convolution .

Si $\delta_p(s) = \int_0^{\infty} e^{-sx} \, D_p(x)$ alors : $\Omega_p^*(s) = \Omega_p[s + \mu_p(1 - \delta_p(s)]$.

D'après [Takacs16] il est démontré que δ_p (s) est la racine de plus petite valeur absolue de (7).

Si $\{\eta_n (p)\}$ n'a pas de solution stationnaire alors $\{\eta^*_n (p)\}$ n'en a pas aussi.

Posons: $\Delta^r_p = \int_0^\infty x^r d\ D_p(x)$ (8)

Si b^r_p est fini alors Δ^r_p l'est aussi. D'après [Takacs 15] Δ^r_p est explicitement donné pour r = 1,2... en particulier :

$$\Delta^1_p = \frac{b_p}{1-\mu_p b_p}$$

$$\Delta^2_p = \frac{b^{(2)}_p}{(1-\mu_p b_p)^2}$$

$$\Delta^3_p = \frac{b^{(3)}_p}{(1-\mu_p b_p)^4} + \frac{3\mu_p (b^{(2)}_p)^2}{(1-\mu_p b_p)^5}$$

Si on connaît les moments d'ordre r de W_p alors $W^{(r)*}_p = \int_0^\infty x^r d\ W^*_p(x)$ peut être obtenu explicitement en utilisant la formule de Faa Di Bruno[22] . En particulier :

$$W^{*(1)}_p = W^1_p [1+\mu_p \Delta^{(1)}_p] = \frac{W^1_p}{(1-\mu_p b_p)} \qquad (9)$$

$$W^{*(2)}_p = W^2_p [1+\mu_p \Delta^{(1)}_p]^2 + W^{(1)}_p \mu_p \Delta^{(2)}_p = \frac{W^2_p}{(1-\mu_p b_p)^2} + \frac{W^1_p \mu_p b^{(2)}_p}{(1-\mu_p b_p)^2} \qquad (10)$$

REMARQUE

Si on note $\Psi_p(s) = \int_0^\infty e^{-sx} \, d\, T_p(x)$ la transformée L-S du temps d'attente dans le système; alors la distribution stationnaire du temps d'attente dans le système du client de priorité p est donnée par :

pour la politique 1 [sans préemption]

$$\int_0^\infty e^{-sx} d\, T_p^*(x) = \Omega_p^*(s)\, \Psi_p(s)$$

Pour la politique 2 [avec preemption]

$$\int_0^\infty e^{-sx} d\, T_p^*(x) = \Omega_p^*(s)\, \Psi_p(s+\mu_p(1-\delta_p(s))]$$

III.2.1.6 DETERMINATION DE $W_p(x)$

a) SERVICE AVEC PREEMPTION

THEOREME [Takacs 63] [14]

Si $\lambda_p a_p < 1$ alors $\{\eta_n(p)\}$ a une distribution stationnaire unique de transformée de L.S donnée par la formule de Pollaczek-Khintchine:

$$\Omega_p(s) = \frac{(1-\lambda_p a_p)}{\left\{1-\lambda_p \left(\frac{1-\alpha_p(s)}{s}\right)\right\}}$$

Si $\lambda_p a_p \geq 1$ alors $\{\eta_n(p)\}$ n'a pas de distribution stationnaire.

PREUVE :

La présence des clients de valeur de priorité supérieure à p n'influe pas sur la loi de probabilité de $\{\eta_n(p)\}$, donc pour trouver la distribution stationnaire de $\{\eta_n(p)\}$, on peut considérer que les clients de valeur de priorité > p sont immédiatement éjectés du système dés leurs arrivées. Dans ce cas d'après le théorème de [Lindley 52] si $\lambda_p a_p < 1$ alors $\{\eta_n(p)\}$ possède bien une distribution limite $W_p(x)$ indépendamment de l'état initial, et de plus $\{\eta_n(p)\}$ possède une distribution stationnaire qui coïncide avec $W_p(x)$.

La transformée de L.S de $W_p(x)$ est donnée par la formule de Pollaczek-Khintchine.

Si $\lambda_p a_p \geq 1$ alors $\forall x \quad \lim_{n\to\infty} P\{\eta_n(p) \leq x\} = 0$

Indépendamment de l'état initial, et $\{\eta_n(p)\}$ n'a pas de distribution stationnaire.

b) SERVICE SANS PREEMPTION

Dans le système modifié décrit dans la section 3 pour $p \leq y < \infty$, notons $\xi_n(y)$ « le nombre de clients de valeur de priorité ≤ y présent dans le système immédiatement après le nième départ ».Nous allons nous intéresser à la distribution stationnaire de $\{\xi_n(p)\}$. Si on connaît cette distribution celle de $W_p(x)$ peut facilement être obtenue.

Posons $\chi_n(p) = \begin{cases} 1 & \text{si le nieme depart de priorité} \\ 0 & \text{si non} \end{cases}$

Pour le processus stationnaire:

$$U_p(z) = E[Z^{\xi_n(p)}]$$

Et $$G_p(z) = P\{\chi_n(p) = 1\}.E[\ Z^{\xi_n(p)} / \chi_n(p) = 1\ ]$$

donc $$E[\ Z^{\xi_n(p)} / \chi_n(p) = 1\ ] = \frac{G_p(z)}{G_p(1)}$$

$G_p(z)$ et $G_p(1)$ sont facilement calculables.

Si $|Z| \leq 1$ alors $$\frac{G_p(z)}{G_p(1)} = \Omega_p\{\lambda_p(1-z)\}\ \alpha_p\{\lambda_p(1-z)\} \qquad (11)$$

Le nombre de clients de priorité ≤ p présents dans le système immédiatement après le départ du client de priorité ≤ p, est égal au nombre de clients de priorité ≤ p qui sont arrivés durant la durée d'attente et de service du client partis.

Si on pose $Z = 1 - s/\lambda$ dans (11) on obtient :

$$\Omega_p(s) = \frac{G(1-(\frac{s}{\lambda_p}))}{G_p(1)\ \alpha_p(s)} \qquad (12)$$

Pour $R(s) \geq 0$ et $|s| \leq \lambda_p$, par prolongement analytique $\Omega_p(s)$ est déterminée d'une façon unique, aussi pour $R(s) \geq 0$. Le problème se réduit à la recherche de $U_p(z)$ et $G_p(z)$dans le cas stationnaire.

Il est aisé de démontrer que le processus $\{\xi_n(y)\}_{n \geq 1}$ est un processus markovien (chaine incluse),[Takacs 62] en adoptant le même procédé que dans les files d'attentes aux arrivées Poissonièrne et au service général. On peut alors déduire que si $\lambda_y\ a_y < 1$ $\{\xi_n(y)\}_{n \geq 1}$ possède une distribution limite indépendante de la distribution initiale, et que $\{\xi_n(y)\}$ possède une distribution stationnaire qui coïncide avec la distribution limite, par contre si $\lambda_y\ a_y \geq 1$ alors:

$$P\left\{\lim_{n\to\infty}\xi_n(y)=\infty\right\}=1 \quad \text{et} \quad \{\xi_n(y)\} \text{ n a pas de distribution stationnaire}$$

Si $\lambda.a<1$ alors pour le processus stationnaire: $P\left\{\xi_n(\infty)=0\right\}=1-\lambda.a$

Si $\lambda.a\geq 1$ alors $P\left\{\xi_n(y)=\infty\right\}=1$

Evidemment $\{\eta_n(p)\}$ a une limite unique et une distribution stationnaire si et seulement si $\{\xi_n(p)\}$ en possède.

THEOREME

Si $q = \propto$ i.e. $\lambda_y a_y < 1 \ \forall$ y alors $\{\eta_n(p)\}$ possède une distribution stationnaire unique $W_p(x)$ de transformée de L.S :

$$\Omega_p(s)=\frac{(1-\lambda a+\mu_p[\frac{(1-\nu_p(s)}{s}])}{(1-\lambda_p\frac{[1-\alpha_p(s)]}{s})}$$

PREUVE

pour le cas stationnaire posons :

$U_p(z)=E[Z^{\xi_n(p)}]$ et $P_0=P\left\{\xi_n(\infty)=0\right\}=1-\lambda a.$

$U_p(z)$ satisfait alors la relation:

$$U_p(z)=\frac{\alpha_p(\lambda_p(1-z))\left\{U_p(z)-U_p(0)\right\}}{z+(\frac{\lambda_p}{\lambda})\ P_0}+\nu_p(\lambda_p(1-z)(U_p(0)-(\frac{\lambda_p}{\lambda})\ P_0)$$

Pour calculer $U_p(z)=E[Z^{\xi_{n+1}(p)}]$. Nous prenons en considération que le nième+1 service consiste à servir un client de priorité ≤ p ou > p. Le premier cas a lieu s'il y a un client de priofité ≤ p juste après le départ du nième client, ou la file est vide juste après le départ du nième client et le premier client arrivant est de priorité ≤ p.

Le second cas a lieu si juste après le départ du nième , il n'y a pas de client de priorité ≤ p présent dans le système, ou après nième départ la file est vide et le premier client qui arrive est de priorité > p.

La distribution stationnaire du temps de service est donnée plus haut par $C_p(x)$.

Pour $P_0 = 1 - \lambda a$ et $U(1)=1$ on a:

$U_p(0)=1-\lambda_p a$ Ceci détermine alors complètement U(z), qui donne aussi:

$$G_p(z) = \alpha_p(\lambda_p(1-z)[\frac{(U_p(z)-U_p(0)}{z}+(\frac{\lambda_p}{\lambda})P_o]$$

donc $G_p(1) = (\frac{\lambda_p}{\lambda})$

Finalement $\Omega_p(s)$ est donné par (12) qui prouve le théorème.

III.2.7 EXEMPLES

Illustrons ce qu'on vient de voir par deux exemples :

1) Soit m classes de clients indexées par p=1..m.

Les clients arrivent selon un processus de Poisson de paramètre λ, soit w_p « la proba que le client entrant dans le système est de priorité p ».

Soit $H_p(x)$ la distribution du temps de service associé au client de la classe d'ordre p.

Dans ce cas:

$$P(y) = \sum_{i \le y} w_i \quad \text{et} \quad \text{pour } p = 1,2,\ldots n.$$

$$\lambda_p = \lambda(w_1 + w_2 + \ldots + w_p)$$

$$\lambda_p A_p(x) = \lambda \sum_{i=1}^{p} w_i H_i(x)$$

$$\lambda_p a_p^r(x) = \lambda \sum_{i=1}^{p} w_i \alpha_i^{(r)}$$

De la meme facon:

$\mu_p = \lambda_{p-1}$

$B_p(x) = A_{p-1}(x)$ et $b_p^{(r)} = a_{p-1}^{(r)}$.

Cas 1 : Si $\lambda_p a_p < 1$ alors:

$$W_p^{*(1)} = \frac{\lambda_p a_p^{(2)}}{2(1-\lambda_p a_p)(1-\lambda_{p-1} a_{p-1})}$$

Formules connues sous le nom de formule de R.G.Miller [23].

Cas 2/ Si $\lambda_m a_m < 1$ alors:

$$W_p^{*(1)} = \frac{\lambda_m a_m^{(2)}}{2(1-\lambda_p a_p)(1-\lambda_{p-1} a_{p-1})}$$

qui concorde avec la Formule de Cobham [24].

EXEMPLE 2

Les clients arrivent suivant un processus de Poisson de densité λ. Leurs services sont des variables aléatoires i.i.d suivant la distribution H(x) .

Supposons que les clients de service le plus court sont prioritaires i.e. Nous supposons que la priorité est déterminée par la durée du temps de service. Dans ce cas nous obtenons:

$P(y) = H(y)$

$\lambda_p = \lambda H(p)$

$$H_p(x) = \begin{cases} 1 & \text{si} \quad x \le p \\ 0 & \text{si} \quad x > p \end{cases}$$

$$\lambda_p a_p^{(r)} = \lambda \int_0^p x^r \, d\,H(x)$$

cas 1: si $\lambda_p a_p < 1$ et $x = p$ est un point de continuité de H(x) alors:

$$W_p^{*(1)} = \frac{\lambda \int_0^p x^2 \, d\,H(x)}{2[1 - \lambda \int_0^p x \, d\,H(x)]^2}$$

Cas 2 . Si $\forall y \quad \lambda_y a_y < 1$ et H(x) est continue au point $x = y$ alors d'apres (9):

$$W_p^{*(1)} = \frac{\lambda \int_0^\infty x^2 \, d\,H(x)}{2[1 - \lambda \int_0^p x \, d\,H(x)]^2}$$ Formule de Phipps

III.3. OUTILS DE RESOLUTION POUR D'AUTRES MODELES DE PRIORITE

Dans cette section, nous inversons la convention en supposant qu'il y ait N classes dans le système et qu' un client appartenant à la classe p est plus prioritaire que celui de la classe p-1.

Nous considérons que les arrivées sont Poissoniennes de taux λ_p clients par unité de temps relative à la classe d'ordre p, ainsi que chaque client de cette classe reçoit un temps de service suivant la distribution $B_p(x)$ avec une moyenne de $\overline{x_p}$.

$$\lambda = \sum_{i=1}^{N} \lambda_i$$

On a : $$\overline{X} = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda} \overline{X}_i$$

$$\rho_i = \lambda_i \overline{X}_i$$

$$\rho = \lambda \overline{x} = \sum_{i=1}^{N} \rho_i$$

ρ représente la fraction de temps d'occupation du serveur.

ρ_i représente la fraction de temps d'occupation du serveur par la classe i.

Pour que le système soit stable il faut que : $\rho < 1$.

III.3.1 CALCUL DES TEMPS MOYENS D'ATTENTE

Soit T = « temps moyen d'attente du client dans le système »

W = "temps moyen d'attente du client la file".

On a évidemment: $T = W + \overline{x}$.

Notons Wp=E(d'attente dans la file du client de la classe p »

Tp=E(temps moyen d'attente du client dans le système).

Soit un client particulier appartenant à la classe p (supposons que ce client soit marqué).

Wp se voit décomposé en trois parties :

- La première concerne le temps moyen résiduel [en cas de service non préemptif] du client en cours.
- La deuxième concerne le temps d'attente moyen dans la file des clients trouvés dans le système et qui appartiennent à des classes d'ordre supérieur ou égal à p.

- La troisième concorde avec le temps moyen d'attente dans la file des clients qui arrivent pendant l'attente du client de type p (marqué) et qui appartiennent à des classes d'ordre strictement supérieur.

Notons:

W0= « Le temps résiduel du client en court » d'après la théorie du renouvellement le temps moyen résiduel es donné par:

$\frac{\overline{x}_i^2}{2\overline{x}_i}$ temps moyen residuel et comme:

ρ_i = proba" d' occupation du serveur par le client i" nous avons:

$$W_0 = \sum_{i=1}^{N} \lambda_i \frac{\overline{x}_i^2}{2}$$ formule de Littel

Notons :

Nij "nombre de clients de type i trouvés par notre client particulier et qui seront servis avant lui"

Mij= "nombre de client de type i qui arrivent dans le système pendant son attente et qui lui sont prioritaires".

Nous affirmons donc en utilisant la formule de Littel:

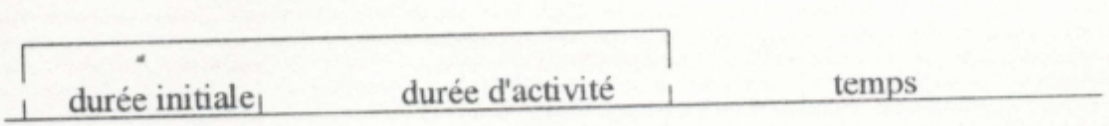
$$W_P = W_0 + \sum_{i=1}^{N} \overline{x}_i (\overline{N}_{ip} + \overline{M}_{ip}) \qquad p = 1 \text{ à } N$$

Donc pour tout modèle avec priorité la démarche à suivre pour calculer les performances temporelles du système consiste en deux étapes.

-Calcul des quantités $\overline{Nip}$ et $\overline{Mip}$

-Résoudre le système d'équations ci dessus.

III.3.2 CYCLE D'ATTENTE, PERIODE D'OCCUPATION GENERALISEE, ET DISTRIBUTION DU TEMPS D'ATTENTE [Gaver 62, Kleinrock 62]



Y0 Yb

Y0 représente « le temps de service à compléter de certains clients temps résiduel +le temps d'attente avant le service ».

Yb représente « le temps de service de tous les clients dans le système avant que celui-ci ne soit vide à partir du service de notre client marqué».

Nous avons : $Yc = Y0 + Yb$

Dans ce schéma nous considérons un client sans se préoccuper de sa priorité. Ce client arrive au système avec un taux égal à λ, ordinaire (en ne prenant pas compte de sa priorité) arrive avec un taux λ, notons qu'après la fin de Y0 (durée initiale achevée) un certain nombre de clients ordinaires attendent leurs services en générant chacun sa propre durée d'activité .

Notons par $B^*(s)$ et $G^*(s)$, les transformées de Laplace des distributions du temps de service et la période d'activité. Nous savons qu'il y a une relation étroite entre ces deux quantités (vue que l'arrivée est un processus sans mémoire) [Kleinrock 75].

$$G^*(s) = B^*(s + \lambda - \lambda G^*(s))$$

Où la période d'activité représente la somme de variables aléatoires i.i.d en B(x).

Calculons maintenant les transformés de Laplace de Y0, Yb et Yc notées respectivement $G_0^*(s)$, $G_b^*(s)$ et $G_c^*(s)$.

Supposons connue $G_0^*(s)$ (car cela est toujours possible à calculer en utilisant la théorie du renouvellement), nous nous intéressons donc au seul calcul de $G_b^*(s)$ et $G_c^*(s)$ en sachant $G_0^*(s)$ et $B^*(s)$.

Commençons par calculer $G_b^*(s) = E(e^{-sY_b})$.

Soit No le nombre de clients ordinaires arrivés pendant Yo, nous avons à l'aide des probabilités totales, et du fait que les n sub-périodes d'activités engendrées durant l'intervalle de durée Y_0 sont indépendantes et identiquement distribuées avec la période d'activité.

$$E(e^{-sY_b} / Y_0 = y, N_0 = n) = (G^*(s))^n$$

$$E(e^{-sY_b} / Y_0 = y) = \sum_{n=0}^{\infty} \frac{(\lambda y)^n}{n!} e^{-\lambda y} (G^*(s))^n$$

$$= e^{(-\lambda y + \lambda G^*(s) y)} = e^{-(\lambda - \lambda G^*(s))y}$$

Finalement : $E(e^{-sY_b}) = \int_{y=0}^{\infty} (\sum_{n=0}^{\infty} \frac{(\lambda y)^n}{n!} e^{-\lambda y})(G^*(s))^n \, d(G_0(y))$

C'est la transformée de L.S de G_0 au point $(\lambda - \lambda G^*(s))$.

Donc $E(e^{-sY_b}) = G_0^*(\lambda - \lambda G^*(s))$.

Suivant le même procédé nous calculons $G_c^*(s) = E[e^{-Y_c s}]$

Nous trouvons: $G_c^*(s) = G_0^*(s + \lambda - \lambda G^*(s))$

A partir de ces équations on peut connaître (par dérivation)le temps moyen d'attente dans le système ainsi que les conditions d'ergodicité du système.

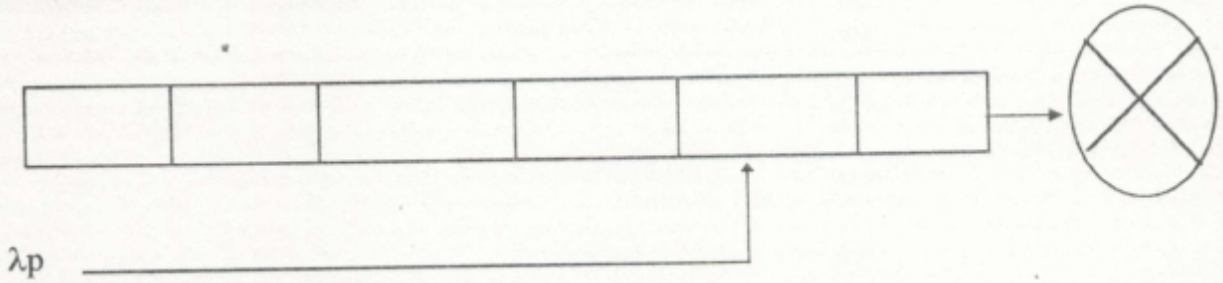
Notons que cet outil est assez puissant pour évaluer les paramètres temporels d'un système spécialement quand il s'agit d'un modèle de priorité en considérant un client particulier d'une classe particulière et en supposant que tout client arrivant durant l'attente de ce client et qui est moins prioritaire que lui est éjecté du système.

Par contre le client plus prioritaire génère sa propre période d'activité identiquement distribuée que toute autre période d'activité de client plus prioritaire que le client marqué.

III.4. MODELE DE PRIORITE [H.O.L]

Considérons maintenant un modèle de priorité assez répandu dans la pratique et qui présente des similitudes avec le modèle étudié dans ce mémoire. Dans ce modèle nous imposons une priorité extérieure qui va structurer les clients entrants suivant le schéma suivant.

Ce système à été étudié pour la première fois par [Cobham] en 1954 qui concerne les priorités fixes dans le temps . Schématiquement le système fonctionne comme suit:



La valeur de la priorité reste dans ce cas constante et elle est de la forme :

$$q_p(t) = q_p$$

Utilisons la méthode des temps d'attente pour trouver le temps moyen d'attente dans la file d'un client du type p : Wp.

Nous avons bien entendu

$\overline{N}_{ip} = 0$ pour $i = 1..p-1.$

$\overline{M}_{ip} = 0$ pour $i = 1..p.$

Tous les clients des classes supérieures ou égales à celle de notre client seront servis avant celui-ci, et d'après la formule de Littel nous avons:

$\overline{N}_{ip} = \lambda_i W_i$ pour $i = p..N$

et parallelement

$\overline{M}_{ip} = \lambda_i W_p$ pour $i = p+1..N$

donc

$$W_p = W_0 + \sum_{i=p}^{N} \overline{x}_i \lambda_i W_i + \sum_{i=p+1}^{N} \overline{x}_i \lambda_i W_p \qquad p = 1..N$$

$$W_p = \frac{w_0 + \sum_{i=p+1}^{N} \rho_i \quad W_i}{1 - \sum_{i=p}^{N} \rho_i} \qquad p = 1...N$$

ce système peut ê tre resolu reccursivement en posant:

$\sigma p = \sum_{i=p}^{N} \rho_i$ on trouve alors:

$$W_p = \frac{W_0}{(1-\sigma_p)(1-\sigma_{p+1})} \qquad p = 1...N$$

A l'aide de la méthode citée à la section III.3, il est possible de trouver la distribution du temps d'attente de chaque classe. Notons par $W_p^{*(s)}$ la transformée de Laplace du temps d'attente dans la file de la classe d'ordre p.

Nous obtenons :[Conway 67, Kesten 57]

$$W_p^*(s) = \frac{(1-\rho)[s+\lambda_H-\lambda_H G_H^*(s)]+\lambda_L[1-B_L^*(s+\lambda_H-\lambda_H G_H^*(s))]}{s-\lambda_p+\lambda_p B_p^*[s+\lambda_H+\lambda_H G_H^*(s)]}$$

Avec:

$$\lambda_H = \sum_{i=p+1}^{N} \lambda_i \qquad \lambda_L = \sum_{i=1}^{p-1} \lambda_i$$

$$B_H^*(s) = \sum_{i=p+1}^{N} \frac{\lambda_i}{\lambda_H} B_i^*(s) \qquad B_L^*(s) = \sum_{i=1}^{p-1} \frac{\lambda_i}{\lambda_L} B_i^*(s)$$

$$G_H^*(s) = B_H^*(s+\lambda_H-\lambda_H G_H^*(s)).$$

A l'aide de cette formule nous retrouvons la formule P-K (Pollazeck-Ketchine) en mettant N=1.

$$\lambda_H = \lambda_L = B_H^* = B_L^* = G_H^* = 0 \quad \text{et} \quad \lambda_p = \lambda$$

$W_1^*(s)$ se réduit à la transformation de Laplace du temps d'attente du système (FIFO).

Une application de ce modèle est le S-J-F étudié par (Phipps 56), nous supposons la durée d'un job entrant dans le système égale à x, celui-ci ira se placer derrière les jobs qui sont plus courts que lui (ou égal). Nous supposons que le service suit une loi B(x) indépendante de la durée du service.

Le modèle regroupe dans une classe les jobs ayant un temps de service $x<\tilde{x}<x+dx$ et la densité du service associée à cette classe est simplement b(x)d(x), et ceci est une quantité infinitésimale quand B(x) est continue .

Calculons maintenant le temps moyen d'attente dans la file W(x) pour un job dont le service se trouve entre (x, x+dx).

La priorité p est une fonction décroissante de $\tilde{x}$.

A la limite nous écrivons:

$$\sum_{i=1}^{N} \rho_i \rightarrow \int_{y=0}^{x^+} \rho(y)\,dy$$

avec: $\rho(x) = \lambda(x)\,x$

$\lambda(x) = \lambda b(x)$

Ceci est dû au fait que le temps moyen de service de chaque job est exactement égal à x unités de temps et que le taux moyen d'arrivée de chaque job est:

$\lambda\, dB(x)/dx = \lambda\, b(x)$, ce qui nous mène par analogie au système d'équations de la section précédente à écrire:

$$W(x) = \frac{W_0}{[1 - \lambda \int_0^{x^-} yb(y)dy][1 - \lambda \int_0^{x^+} yb(y)dy]}$$

Et comme B(x) est considéré comme continu en x alors:

$(1 - \lambda \int_0^{x^-} yb(y)dy) = (1 - \lambda \int_0^{x^+} yb(y)dy)$ le dé nomin ateur aura la forme:

$$(1 - \lambda \int_0^{x} yb(y)dy)^2$$

Dans le cas discret la solution s'applique seulement aux groupes de priorités ayant un temps moyen d'attente fini, cette expression nous donne le temps moyen d'attente de la politique S-J-F pour un client dont la durée de service est égale à x. Notons au passage que pour un client de durée de service très longue on a:

$\lim_{x\to\infty} W(x) = \frac{W_0}{(1-\rho)^2}$ en revanche pour une duré ede service tré sfaible

Nous avons $\lim_{x\to 0} W(x) = W_0$

III.5. PRIORITES DEPENDANTES DU TEMPS OU PRIORITES DYNAMIQUES

Bien que difficile à étudier, les modèles de files d'attente avec priorités dynamiques sont très importants, et s'imposent dans beaucoup d'applications pratiques, notamment en informatique

MODELE 1

Le modèle présenté ci après considère un ensemble de N paramètres $0 \le b1 \le b2 \le b3 \ldots \le b_N$ Ces paramètres sont en général fixés par le concepteur du modèle suivant l'importance des clients aux quels ils sont attribués .

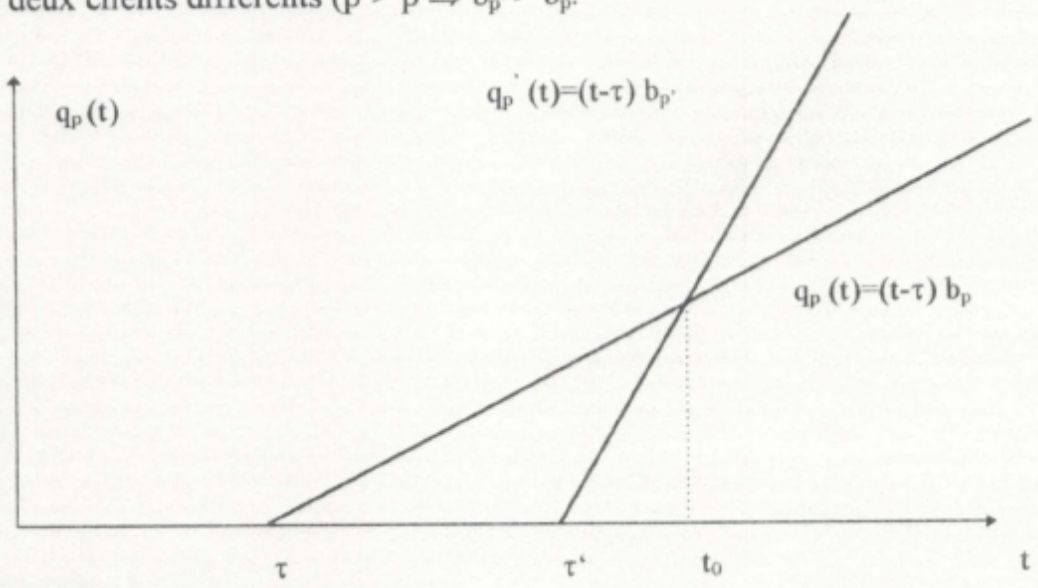
Considérons un certain client (job) qui arrive à l'instant τ et au quel on attribue à tout instant t la priorité linéairement dépendante du temps suivante:

$$q_p(t) = (t - \tau)\, b_p$$

Où t varie entre τ et la fin de service du client considéré. Le système non-préemptif suivant se comporte comme suit:

Le serveur prend en charge le client de plus forte priorité $q_p(t)$ à tout instant t .

Le schéma suivant nous met en évidence l'interaction entre les fonctions de priorités de deux clients différents ($p'> p \Rightarrow b_{p'} > b_p$.



Nous distinguons trois cas:

$-\tau \leq t<\tau'$ $\Rightarrow$ le prochain client pris en charge est p (car il est le seul à attendre)

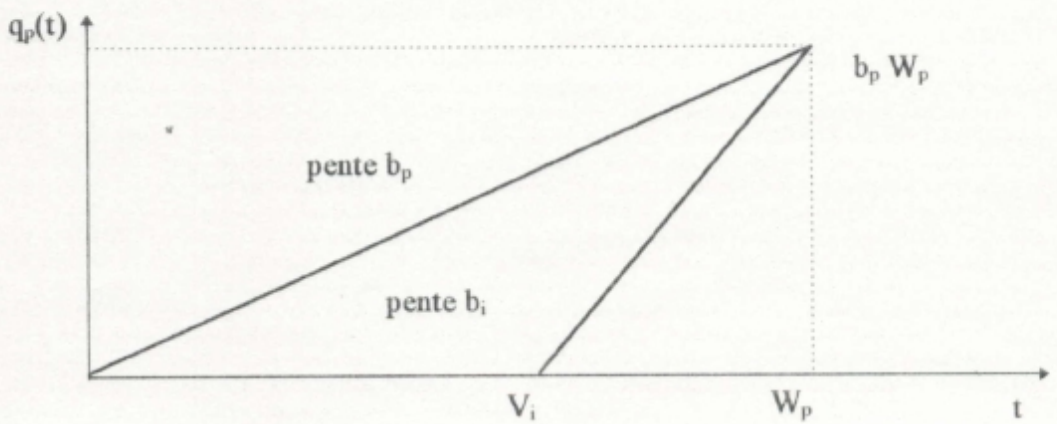
$-\tau' \leq t< t0$ $\Rightarrow$ le prochain client sera toujours p il est de plus forte priorité.

$-t_0 \leq t$ $\Rightarrow$ Le prochain client sera p' (car il est le plus prioritaire).

Simplifions l'étude en considérant le service suivant une loi exponentielle. En utilisant la méthode des temps moyens.

Procédons au calcul de $\overline{N}_{ip}$ et $\overline{M}_{ip}$

Commençons d'abord par calculer $\overline{M_{ip}}$. Soit la figure suivante:



Il est clair que:

$\overline{M}_{ip} = 0$ pour $i \leq p$ car $(b_i \leq b_p)$.

Avant que notre client ne puisse bénéficier du service, il doit attendre Wp unités de temps; donc atteindre au plus la priorité de bpWp. Les seuls clients pouvant influencer sur son attente sont ceux qui lui sont prioritaires et arrivent avant l'écoulement de Wp et atteignent la priorité de bpWp avant lui. Ce sont les clients qui arrivent dans l'intervalle [0,Vi].

Calcul de Vi.

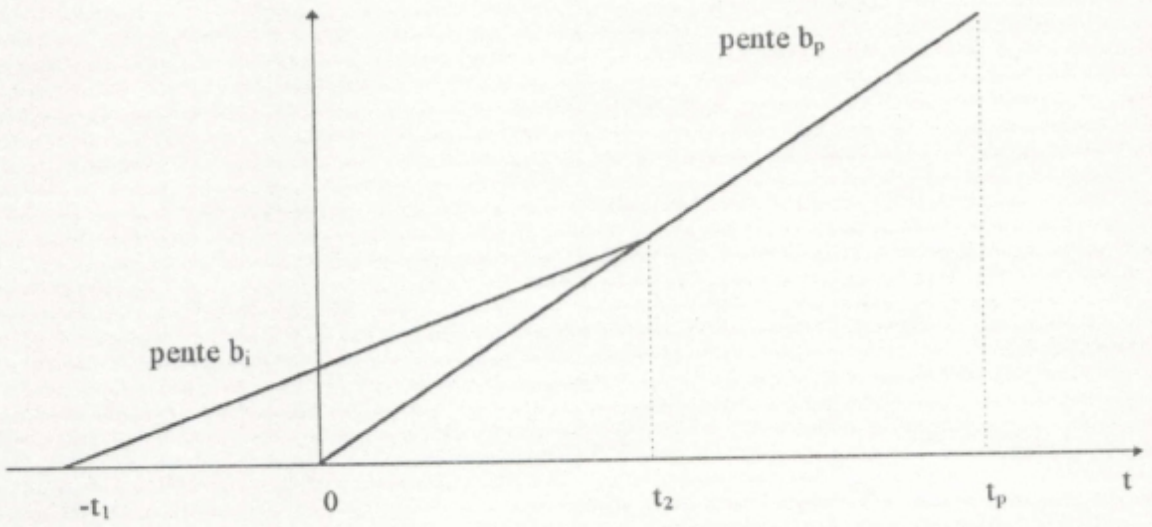
$$b_p W_p = b_i (W_p - V_i) \Rightarrow V_i = W_p (1 - \frac{b_p}{b_i})$$

Le flux des arrivées de la classe i est supposé égal à λi, donc:

$$\overline{M}_{ip} = \lambda_i V_i = \lambda_i W_p (1 - \frac{b_p}{b_i}) \qquad i > p$$

CALCUL DE $\overline{N_{ip}}$

$$\overline{N}_{ip} = \lambda_i W_p (\frac{b_i}{b_p})$$



INTERPRETATION

Supposons que notre client particulier arrive à t=0, il passera au total tp unités de temps dans la file et atteindra la priorité bp tp . Nous nous intéresserons cette fois-ci aux clients déjà présents dans le système à l'arrivée de notre client possédant des paramètres $\leq$ p , et qui seront servis avant lui.

Supposons qu' un de ces clients soit arrivé au temps t= -t1, par définition de $\overline{N_{ij}}$ nous devons calculer le nombre de clients de type i qui sont arrivés avant t= 0, et qui sont encore présents à ce moment et obtiennent le service avant le client particulier.

Il est clair d'après la figure ci-dessus que ces clients doivent arriver après -t1 (t1 >0) et attendre Wi = W(t1) qui doit vérifier:

t1<W(t1)<t1+t2 [t2 = Vi correspondant à $\overline{M_{ip}}$].

W_i (t1) ne doit pas dépasser t1+t2 car si non, le job de type i sera de priorité inférieure à celle de notre client particulier. Trouvons t2:

$$b_p t_2 = b_i (t_1 + t_2) \Rightarrow t_2 = (\frac{b_i}{b_p - b_i})\, t_1$$

ou encore: $t_2 + t_2 = (\frac{b_p}{b_p + b_i})\, t_1$

Donc le nombre moyen $\overline{N}_{ip}$ est:

$$\overline{N}_{ip} = \int_0^{\infty} \lambda_i P\left[t < W_i(t) < \frac{b_p}{(b_p - b_i)} t \right] dt$$

Cette é quation se ramène à:

$$\overline{N}_{ip} = \lambda_i \int_0^{\infty} \left[1 - P(W_i(t) < t\right] dt - \lambda_i \int_0^{\infty} \left[1 - P(W_i(t) < (\frac{b_p}{b_p - b_i}) t \right] dt$$

En faisant le changement $y = \frac{b_p}{(b_p - b_i)} t$, et en utilisant:

$$E[W_i]] = \int_0^{+\infty} (1 - P(W_i) \le x)\, dx = W_i \text{ nous obtenons:}$$

$$N_{ip} = \lambda_i W_i - \lambda_i (1 - \frac{b_i}{b_p}) W_i$$

$$= \lambda_i W_i \frac{b_i}{b_p} \qquad \text{pour } i \le p$$

Pour $i \ge p$. Il est clair que:

$$\overline{N}_{ip} = \lambda_i W_i$$

Car notre client ne peut rattraper la priorité de ceux qui étaient présents dans le système avant lui et qui lui sont prioritaires.

D'après l'équation donnée à la section III.3.1

Nous pouvons ma int enant calculer W_p " temps d'attente moyen dans la file"

$$W_p = \frac{W_0 + \sum_{i=p}^{N} \rho_i W_i + \sum_{i=1}^{p-1} \rho_i W_i (\frac{b_i}{b_p})}{(1 - \sum_{i=p+1}^{N} \rho_i (1 - \frac{b_p}{b_i}))} \qquad p = 1,2 \ldots N.$$

MODELE 2.

Considérons un système d'attente à un serveur, où les clients sont répartis entre deux classes de priorités notées 0 et 1 (la classe 1 est supposée plus prioritaire que la classe 0).

Les processus des arrivées aux deux classes sont des processus de Poisson de taux λ_0 et λ_1 respectivement, et les distributions de service suivent des lois aléatoires $B_0(x)$ et $B_1(x)$ respectivement.

Notons $X_i(t)$ « la période d'activité générée par tous les clients de la classe i évaluée à partir de l'instant t ».

La discipline de service tient compte de paramètres temporels variables, d'où le nom de priorités dynamiques, cette politique de service est décrite comme suit:

Nous associons le nombre u_0 aux clients de la classe 0 et u_1 à ceux de la classe 1, tel que si un client de la classe i arrive au temps t, il est assigné du nombre $t+u_i$. Nous convenons que

$$-\infty \leq u_1 \leq u_0 \leq +\infty.$$

Le serveur servira à un instant t le client du système ayant la plus petite valeur $t + u_i$.

Posons : A_t l'ensemble des dates d'arrivées des clients présents dans le système au temps t, et considérons l'expression :

$\text{Min}\{ t' + u_i\} = \min\{\min(t'_0 + u_0), \min(t'_1 + u_1)\}$: Min sur $t' \in A_t$ et $i = 0,1$ (1)

Où t'_i représente la date d'arrivée du client de la classe i.

Posons C_i la date au plus tôt au temps t des arrivées à la classe i parmi les clients présents à cette date, et t''_i sa durée écoulée jusqu'à cet instant. Nous pouvons alors écrire l'expression (1) comme étant:

$$\min\{(t''_0 + u_0), \min(t''_1 + u_1)\} \qquad (2)$$

Donc dans chaque classe la discipline est FIFO et la classe 1 sera plus prioritaire que 0 si :

$$t''_1 + u_1 < t''_0 + u_0$$

Que nous pouvons écrire comme : $t''_1 < t''_0 + u$ Où $u = u_0 - u_1 \geq 0$.

D'après (2) nous avons :

Si $u_0 = 0$ la politique de service est FIFO.

Si $u_0 = \infty$ la politique de service est basée sur des priorités statiques.

Supposons $W_i(t)$ soit le temps virtuel d'attente du client de la classe i. Nous définissons le processus $W_t(t')$ et la variable aléatoire $T_t(w)$ comme :

$T_t = \text{Inf}\ \{ t' : W_t(t') = 0 \}$.

Théorème [30]

Le temps virtuel d'attente $W_0(t)$ est donné par :

$W_0(t) = \text{Min}\ \{ T_t(w),\ u + W_t(u) \}$.

Où $w = W(t)$ le temps résiduel total d'attente au temps t, et $W_t(t')$ est donné par :

$$W_t(t') = W_t(0) + X_1(t+t') - X_1(t) - t' + \int_0^{t'} \chi_t(s)\, ds. \quad (3)$$

avec : $W_t(o) = w$. et $\chi_t = 1$ si $W(t) = 0$,0 si non.

Preuve:

Considérons le client cl_0 de la classe 0 arrivant à la date t, alors cl_0 va attendre que tous les clients des deux classes présents à cet instant t soient servis ainsi que tous les clients de la classe 1 arrivant durant l'intervalle de temps] t,t+u].

Bien sûr $W_0(t) \geq W(t)$.

. Si $T_t(w) \leq u$, alors cl0 commencera à être servi au temps $t + T_t(w)$, alors $W_0(t) = T_t(w)$.

. Si $T_t(w) > u$, alors cl_0 aura à attendre u + le temps résiduel(en prenant en compte les arrivées de la classe 1)à partir du temps t+u. Dans ce deuxième cas :

$$W_0(t) = u + W_t(u)$$

Maintenant si $T_t(w) > u$ nous avons d'après (3)

$W_t(t') > 0$ $(0 \leq t' \leq u)$ et :

$u + W_t(u) = w + X_1(t+u) - X_1(t)$.

$\leq w + X_1(t + T_t(w)) - X_1(t) = T_t(w)$

En combinant tous ces résultats nous obtenons le résultat du théorème.

Théorème [30].

Nous avons :

$W_0(t) = \text{Min}\ \{ T_t(w),\ u + W_1(t+u) \}$

Preuve:

Considérons un client cl_1 de la classe 1 arrivant à la date t+u, alors cl_1 aura à attendre :

Tous les clients des deux classes qui sont présents depuis le temps t.

Tous les clients de la classe 1 arrivant durant l'intervalle]t, t+u].

. Si $T_t(w) > u$ alors :

$W_1(t+u) = W_t(w)$ et d'après le théorème précédant:

$W_0(t) = u + W_t(u)$ ce qui donne:

$W_0(t) = u + W_1(t+u)$.

. Si $T_t(w) \leq u$ alors :

$W_1(t+u) \geq W_t(w)$ ce qui donne :

$W_0(t) = T_t(w) \leq w \leq w + W_1(t+u)$

Ce qui achève la démonstration.

CHAPITRE IV

IV.1 INTRODUCTION

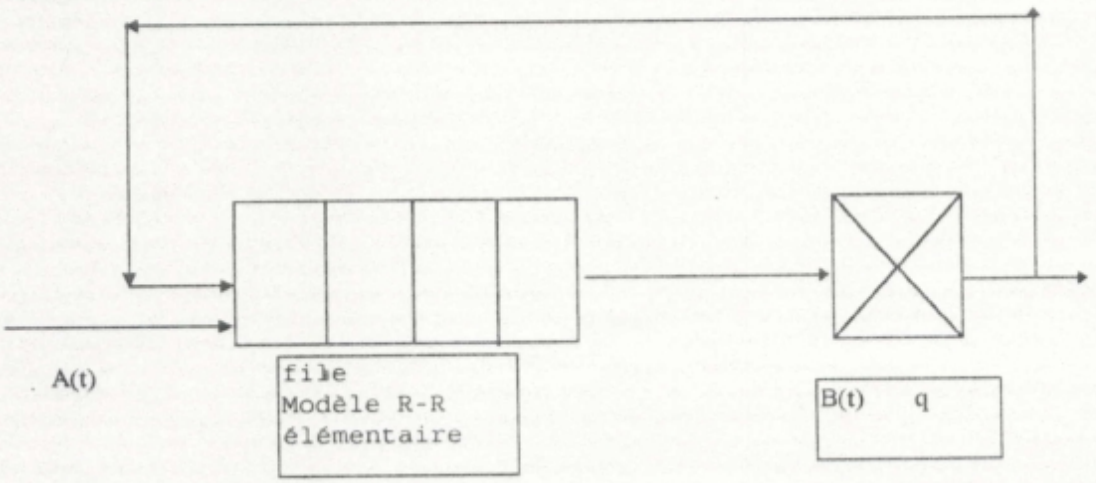
Après l'apparition des stratégies d'ordonnancement destinées pour les systèmes Batch telles que S-J-F ou L-J-F (Shortest-job-first ou Longest-job_first], une autre stratégie tout à fait différente de par son esprit philosophique a vu le jour, celle ci ne nécessite pas la connaissance des durées des jobs à l'avance (au préalable) pour son implémentation, ce qui est le cas dans beaucoup d'applications.

Cette stratégie est appelée modèle ROUND-ROBIN ou R-R-model.

IV.2. PRESENTATION DE LA STRATEGIE[R-R -MODELE] [20]

Les systèmes de file d'attente, où la discipline de service est le R-R modèle ont été apparentées d'abord aux systèmes à temps partagé. Ces systèmes sont particuliers de par le fait que les arrivées constituent un processus stochastique, telles que les distributions des temps inter-arrivées sont identiques suivant la loi A(t) et indépendantes les unes des autres. Les temps de service des unités(jobs, client) obéissent à la même loi de distribution notée B(t). Les unités sont servies suivant la politique FIFO, avec un laps de temps fixé appelé quantum. Si l'unité servie termine son service au bout du quantum alloué elle est automatiquement éjectée du système, par contre si son service n'est pas achevé et requiert plus de temps que le quantum, elle est alors replacée à la fin de la file.

SCHEMATIQUEMENT

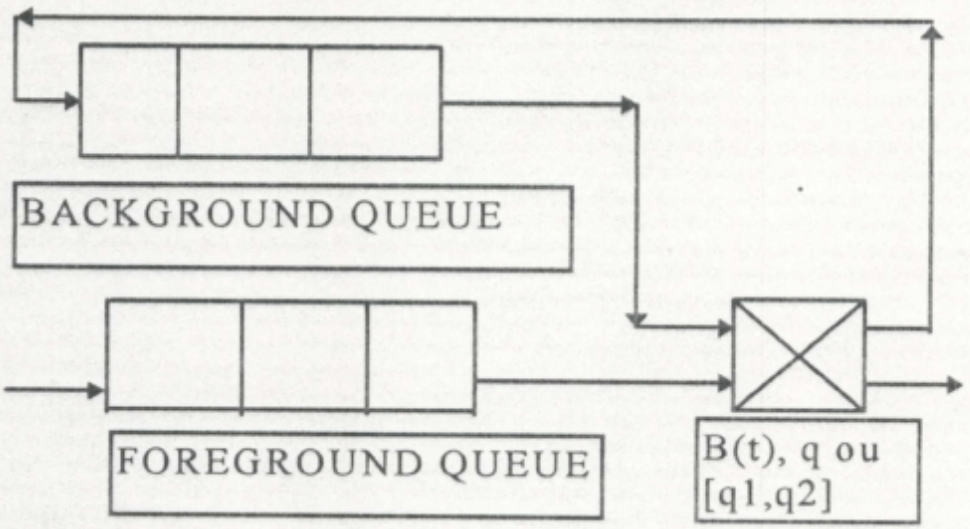


Bien sûr, le service de l'unité replacée reprendra là où il a été interrompu.

IV. 2.1 UNE DERIVEE DU R-R -MODELE LE FB-MODELE

Le modèle « Foreground-Background »F-B [EG-Coffman] est une dérivée du modèle R-R. Sa particularité réside dans le fait que l'unité demandant plus de temps que le quantum alloué lors du premier passage en service ne va pas rejoindre la file des arrivées dite [Foreground queue], mais ira se placer à la fin d'une autre file appelée [Background queue]. Cette file ne sera servie que si la première est complètement vide. A noter aussi que dans les applications, le quantum attribué aux deux files peut ne pas être le même (celui associé au Background queue est plus grand) .

Ce modèle se schématise comme suit:



IV.3 ANALYSE ET DESCRIPTION DE LA STRATEGIE

Vu comme un modèle de priorité, le modèle R-R présenté se distingue par le fait que les priorités ne sont pas connues au préalable, mais basées sur le dynamisme du système. Cette distinction est fondamentale car l'utilisation des résultats de la théorie sur les files d'attente avec priorité s'avère très réduite.

Pour pouvoir utiliser un outil très puissant des processus stochastiques, nous allons d'abord faire remarquer que pratiquement un système informatique ne peut être contrôlé qu'à des instants discrets (points observables, voir chap. I).

L'étude de l'évolution du système se fera aux moments où le programme de contrôle émettra sa décision d'ordonnancement; en outre, nous montrerons que la chaine induite à ces instants est markovienne.

IV.3.1 L'ANALYSE.

L'objectif de cette analyse est l'étude des effets du « SWAPPING (commutation) »et de «L'OVERHEAD » (pénalisation) sur le nombre moyen de jobs dans le système, ainsi que le temps moyen d'attente.

Une des premières études faites sur ce modèle est sans doute celle menée par [Kleinrock 67], en supposant que les lois de distribution du service et des inter-arrivées étaient discrètes suivant la loi géométrique. Dans cette analyse, nous passons aux lois exponentielles de paramètres respectifs μ et λ; sachant évidement que les distributions géométriques et exponentielles sont toutes deux des lois à processus sans mémoires.

Le processus qui nous intéresse dans ce modèle est le nombre de jobs dans le système à un temps donné, notons le par $\xi(t)$.

Par définition du modèle R-R le processus est sans aucune ambiguïté un processus non-markovien. En considérant « la chaîne incluse » au processus original à des instants particuliers t_k, k=1.2.., et en analysant le processus $\xi(t_k)$, nous pouvons alors utiliser les propriétés de Markov.

Nous choisissons les instants t_k comme étant les instants de fin d'un job ou l'expiration d'un quantum [Krishnamoorthi, Woods].

Durant les périodes d'activités les distributions inter-occurrences sont données par:

$$F(x) = \Pr(t_{k+1} - t_k < x) = \begin{cases} 0 & \text{si} \quad x < \tau \\ 1 - e^{-\mu(x-\tau)} & \text{si} \quad \tau \le x \le q + \tau \\ 1 & \text{si} \quad x > q + \tau \end{cases}$$

Où q est la durée du quantum et τ le temps du « swap ». Notons par m_s la moyenne de cette distribution, nous avons :

$$m_s = \int_{-\infty}^{+\infty} t\left(\frac{d\,F(t)}{dt}\right) dt = \frac{1}{\mu}(1 - e^{-\mu q}) + \tau \qquad (1).$$

Du fait de l'absence de mémoire de la distribution exponentielle, un job attendant dans la file a la même distribution exponentielle avec une moyenne de $1/\mu$ unité de temps pour accéder au service sans influence du nombre de quantum déjà reçu par ce job jusqu'à cet instant.

Ceci appuyé par le fait que le processus des arrivées est Poissonnien, prouve bien que les événement $\xi_k(t)$ constituent une chaîne de Markov homogène.

Posons P_{ij} = probabilité « qu'il y'ait j jobs dans le système au temps t_{k+1} sachant qu'il y en avait i à l'instant t_k ».

$$P_{ij} = P\left(\xi(t_{k+1}) = j \,/\, \xi(t_k) = i\right) \qquad k = 1,2.....$$

Pour $j < i-1$ nous avons $P_{ij} = 0$.

Pour $j = i-1 \geq 0$ nous avons P_{ij} = P(0 arrivée en $q+\tau$, le job en cours s'achève en $t \leq q$) .

Pour $j > i-1$, nous avons P_{ij} = P(j-1 arrivées en $q+\tau$, le service en cours ne s'achève pas) + P(j-i+1 arrivées en $q+\tau$, le programme en cours s'achève en $t \leq q$).

Finalement :

$$P_{0j} = P_{1j} \qquad \forall j$$

Cette dernière relation découle de l'observation, en effet, pour passer de 0 job à j jobs, nous devons toujours servir la première arrivée d'au moins un quantum.

Analytiquement ces expressions s'écrivent comme suit:

$$\begin{cases} P_{ij} = 0 & \text{pour } j < i-1 \\ P_{ij} = \int_0^q P\left[0 \,/\, t+\tau\right] \mu e^{-\mu t} dt & \text{pour } j = i-1 \geq 0 \\ P_{ij} = P\left(j-1 \,/\, q+\tau\right) e^{-\mu q} + \int_0^q P\left[j-i+1 \,/\, t+\tau\right] \mu e^{-\mu t} dt & \text{pour } j \geq i \geq 1 \\ P_{0j} = P_{1j} & \forall j \end{cases}$$

Où $P\left(n \,/\, t\right)$ = « proba qu'il y'ait n arrivées pendant l'intervalle de durée t, suivant un processus de Poisson de taux λ» .

La condition d'existence de la distribution stationnaire des probabilités $\{\pi_j\}_{j\geq 0}$ (pour le nombre de jobs dans le système) associés à la chaîne de Markov $\xi(t_k)$ est que le taux moyen des arrivées n'excède pas le taux maximum de sortie des jobs. Sous cette condition nous avons :

$$\pi_k = \sum_{i=0}^{\infty} \pi_i P_{ik}$$

Calculons alors la fonction generatrice T(z):

$$T(z) = \sum_{k=0}^{\infty} z^k \sum_{i=0}^{\infty} \pi_i P_{ik}$$

$$= \pi_0 \sum_{k=0}^{\infty} P_{0k} z^k + \sum_{i=1}^{\infty} \pi_i \left[P_{i,i-1} z^{i-1} + \sum_{k=i}^{\infty} P_{i,k} z^k \right]$$

Posons:

$$P(z) = \sum_{n=0}^{\infty} z^n \int_0^q P\left(n/t+\tau\right) \mu e^{-\mu t}\, dt$$

$$Q(z) = \sum_{n=0}^{\infty} z^n\, e^{-\mu q}\, P\left(n/q+\tau\right)$$

En utilisant les valeurs P_{ik} et les changements P(z) et Q(z) nous obtenons:

$$T(z) = \frac{1-z}{1-z\beta(z)}$$

Avec : $\beta(z) = \left[P(z) + z\,Q(z)\right]^{-1}$

Pour déterminer π_0, remarquons que T(1)= 1

En utilisant la formule de L'Hopital et le fait que $\beta(1)=1$, nous obtenons :

$\pi_0 = 1+\beta'(1)$.

Calculons P(z) et Q(z) pour déterminer P'(z) et Q'(z) donc $\beta'(z)$ et en fin $\beta'(1)$.

$$P(z) = \int_0^q \sum_{k=0}^{\infty} z^k \exp(-\lambda(t+\tau)) \frac{(\lambda(t+\tau))^k}{k!} \mu e^{-\mu t}\, dt$$

En interchangeant les signes $\sum$ et $\int$, nous obtenons:

$$P(z) = \frac{\exp([-\lambda\tau(1-z)]}{1+\rho(1-z)} \left[1 - \exp\left[-\mu q(1+\rho(1-z))\right]\right] \quad ; \quad \rho = \frac{\lambda}{\mu}.$$

D' où

$P'(1) = \rho(1 + \mu\tau)(1 - \exp(-\mu q) - \rho\mu q \exp(-\mu q)$

Pour $Q(z)$ nous avons

$$Q(z) = \exp(-\mu q) \sum_{k=0}^{\infty} P\left(k/q+\tau\right) z^k = \exp[-\lambda\tau(1-z)] \exp[-\mu q(1 + \rho(1 - z)$$

Donc $Q'(1) = \rho(\mu q + \mu\tau)\exp(-\mu q)$

Finalement, nous déduisons :

$\beta'(1)=[\rho'(1)+Q(1)+Q'(1)]$.

Nous concluons que T(z)peut se mettre sous la forme:

$$T(z) = \frac{(1-z)\{P(z) + zQ(z)\}}{P(z) + z(Q(z) - 1)} \pi_0$$

avec $\pi_0 = 1 + \beta'(1) = 1 - \rho(1 - \exp(-\mu q))$

De ce fait nous pouvons calculer le nombre moyen n de jobs dans le système, comme étant:

$$\bar{n} = \lim_{z \to 1} T'(z) = \frac{(1-\pi_0) - \frac{1}{2}\beta''(1)}{\pi_0}$$

Pour déterminer $\beta''(1)$, nous devons calculer $P''(z)$ et $Q''(z)$ ce qui de la même manière que P(z) et Q(z) nous amène au résultat suivant:

$$\beta''(1) = 2\lambda q \delta + 2\delta^2 (1-\rho)^2 + 2\rho\delta(\rho - \lambda\rho) - (\lambda\tau + \delta)\left[\lambda\tau + 2\rho(1-\delta)\right]$$

Où $\delta = \exp(-\mu q)$.

Quand $\tau \to 0$ et $q \to \infty$ nous avons :

$\beta''(1) = 0$, $\pi_0 = \rho$ et $\bar{n} = \frac{\rho}{1-\rho}$ qui constituent les resultats du système d' Erlang

IV.4. TEMPS D'ATTENTE CONDITIONNEL POUR LE MODELE R-R

A partir des résultats de la section précédente, nous sommes en mesure de calculer la moyenne du temps d'attente dans la file conditionné par le service non achevé déjà requis.

Nous considérons que le temps d'attente des arrivants au système R-R est en équilibre i.e.[acceptant un état stationnaire], en outre nous supposons que les temps d'arrivées coïncident avec les points générés par la chaîne de Markov $\xi(t_k)$.

IV.4.1 PROPOSITION

Notons le temps de service requis par un job par S, notons encore par W_s la moyenne conditionnelle d'attente dans la file par ce job. Soit m un entier telle que l'inégalité suivante soit satisfaite $0 \le mq-S < q$ nous avons alors :

$$W_s = \frac{m_s}{1-\alpha}\left(m\lambda(q+\tau) + \left\{ \bar{n} - \frac{\lambda(q+\tau)}{1-\alpha} \right\} (1-\alpha^m) \right)$$

Où $\begin{cases} \alpha = \lambda m_s + \delta \\ \delta = \exp(-\mu q) \\ \bar{n} = \text{nombre moyen de job dans le système (dé ja calculé)} \end{cases}$

m_s est donné par la formule (1) de la section IV.3.1

IV.4.2 PREUVE

Pour pouvoir montrer ce résultat, nous allons utiliser une méthode similaire à celle donnée par [Kleinrock 67] Pour le modèle R-R en temps discret.

Soit « un programme t particulier marqué ».

Soit y_i =« temps passé dans la file par le job t lors du $i^{ème}$ passage en service ».

W_s correspondra clairement à $E\left[\sum_{i=1}^{m} y_i\right]$.

Définissons N_i comme étant le nombre moyen de jobs bénéficiant du service avant notre job t lors du $i^{ème}$ passage (juste au début de l'insertion à la file).

Développons alors l'expression de N_i .

$N_1 = \bar{n}$ par définition de $\bar{n}$, pour $i > 1$ N_i va représenter les jobs de N_{i-1} qui nécessitent plus de q unités de temps lors du ième-1 passage (qu'on appellera les jobs retournés) et le nombre moyen des nouveaux arrivés occurants durant la durée $y_{i-1} + q + \tau$. Nous avons alors la formule récurrente:

$$\begin{cases} N_i = \delta\, N_{i-1} + \lambda(\overline{y_{i-1}} + q + \tau), & i > 1 \\ N_1 = \bar{n} \end{cases}$$

Substituons $\overline{y_{i-1}} = m_s N_{i-1}$ nous obtenons

$$N_i = N_{i-1}\left[\delta + m_s * \lambda\right] + \lambda\left[q + \tau\right]$$
$$= \alpha\, N_{i-1} + \lambda(q + \tau)$$

En utilisant la condition $N_1 = \bar{n}$ nous obtenons

$$N_i = \alpha^{i-1}\,\bar{n} + \lambda(q + \tau)\sum_{k=0}^{i-2}\alpha^k \quad \text{pour} \quad i > 2.$$

Comme $E[y_i] = m_s N_i$, nous retrouvons la relation de la proposition.

CHAPITRE V

V. ORDONNANCEMENT EQUITABLE PAR PRIORITES BORNEES

V.1. INTRODUCTION

Après avoir passé en revue les différents thèmes abordés dans les chapitres précédents, nous faisons la jonction en introduisant et en présentant une nouvelle stratégie d'ordonnancement par priorités dynamiques bornées d'un système informatique multiprogrammé monoprocesseur [1].

L'ordonnancement des processus d'un système multiprogrammé, réalise l'attribution du processeur aux processus candidats (prêts), ce problème a suscité de nombreux travaux dont [Dietel 84], [Peterson 85], [Krakowiak 87], décrivent les plus connus.

Comme pour les files d'attente avec priorités, on repartit les stratégies d'ordonnancement utilisées en deux familles. Les ordonnancements non-préemptifs qui sont caractérisés par le fait qu'un processus détenant le processeur ne le libère qu'à sa terminaison ou volontairement lors d'une opération de synchronisation (E/S). Dans cette famille, les stratégies adoptées nécessitent souvent la connaissance ultérieure des temps de service demandés par les processus en concours à activer, et cette connaissance est pratiquement impossible pour beaucoup d 'applications .

Différentes méthodes proposées qui se basent principalement sur les théories statistiques qui permettent des estimations assez convenables de ces temps de service, celles ci conduisent à des solutions diverses qui dans certaines applications ne sont pas satisfaisantes.

Une de ces stratégies introduite par [Brinch-Hansen 71] [2], dite [HIGEST-RESPONSE-RATIO-NEXT]ou [H-R-N] établit une relation d'ordre total entre les processus prétendants au processeur, cette relation est définie en associant à chaque processus un réel appelé sa priorité, qui est une fonction du temps de service demandé t_s, et du temps d'attente t_a pendant lequel le processus a attendu ce service.

Cette priorité est donnée par la formule suivante:

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

Le numérateur de cette fraction représente le temps de réponse du système au service demandé par le processus. Comme signalé plus haut la difficulté d'implémentation de cette méthode consiste en l'estimation de t_s.

La deuxième famille d'ordonnancement, est l'ordonnancement préemptif, ce dernier consiste à attribuer à tout instant le processeur au processus le plus prioritaire, cette priorité peut être fixe ou évoluer dynamiquement en fonction de son histoire passée ou d'un ensemble de paramètres fixés par le concepteur.

Une autre famille intermédiaire a fait son apparition au début des années 70, qui regroupe les algorithmes qui implémentent des stratégies mixtes. Ainsi par exemple, l'organisation en files multiniveaux [MULTILEVELS-QUEUES]réalise une partition de l'ensemble des processus attendant le service du processeur.

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 classes (files) de processus et non sur les processus eux même [Dietel 84], l'algorithme de choix de l'ordonnanceur (SCHEDULER) élit les processus de la file non vide la plus prioritaire selon la stratégie « ROUND-ROBIN» .

Dans cette étude, l'ordonnancement original proposé par [C.Haro et C. Proust 92] est un ordonnancement préemptif par priorités bornées dans un système multiprogrammé monoprocesseur.

La stratégie (politique) proposée prend en compte des paramètres temporels définis dans le passé récent de chaque processus; En cela, elle est similaire à H.R.N dont elle dérive le mode de calcul des priorités, mais elle définit des classes de processus en bornant leurs priorités, ce qui permet de la doter des caractéristiques d'une file multiniveaux tout en étant plus simple à implémenter.

V.2 ORDONNANCEMENT PAR PRIORITE BORNEE[1]

V.2.1 PRESENTATION

Soit $\Psi = \{ \tau_i \ / \ 1 \leq i \leq L \}$ l'ensemble des L processus en compétition pour l'acquisition du processeur. La structure de contrôle de l'allocation du processeur aux processus est une liste unique appelée QUEUE D'EXPLOITATION [chap. 1],

Cette structure ordonne totalement les processus prêts à s'exécuter en fonction de leurs priorités. A tout instant, c'est le processus le plus prioritaire qui s'alloue le processeur, c'est donc par définition une stratégie d'allocation péemptive, lorsque plusieurs processus se trouvent avoir la même et plus forte priorité, la stratégie adoptée est le R-R, chacun de ces processus reçoit alors à tour de rôle un quantum de temps processeur t_q.

La priorité p_i (t), à tout instant de tout processus de Ψ évolue entre deux réels positifs m_i et M_i.

D'où $\forall\ 1 \le i \le L \qquad m_i \le p_i(t) \le M_i \qquad [1]$

Notons au passage qu'un processus de priorité fixe est associé à des bornes égales m_i et M_i.

V.2.2. DEFINITION 1.

On appelle ordonnancement par priorités bornées ou O.P.B. la stratégie d'ordonnancement préemptif par priorités obéissants aux conditions [1].

V.2.3. EXEMPLES DE STRATEGIES REALISABLES PAR O.P.B.

Nous allons voir comment projeter sur O.P.B. les propriétés de quelques stratégies remarquables:

1°) Si nous posons : $m^- = \inf_{1 \le i \le L} m_i$ et $M^+ = \sup_{1 \le i \le L} M_i$

O.P.B. sera un simple ordonnancement par priorités dynamiques si nous posons :

$\forall i\ 1 \le i \le L\ \ m_i = m^-$, $M_i = M^+$ car alors toutes les priorités appartiendraient à la même classe $[m^-, M^+]$ et leurs évolutions incarnent bien le système dynamique.

2°) Si nous posons : $\forall i\ 1 \le i \le L\ \ m_i = M_i = P$.

L'ordonnancement serait du type R-R, car alors tous les processus auront la même priorité fixe.

3°) Si nous posons : $\forall i\ 1 \le i \le L\ \ m_i = M_i = p_i(0)$, et $p_i(0)$ non obligatoirement égale à $p_j(0)$, alors chaque processus dispose de sa propre priorité qui reste fixe dans le temps et l'ordonnancement deviendrait par priorités statiques.

4°) Si nous définissons une relation R sur ΨxΨ de la façon suivante:

$\forall i \; 1 \le i \le L$

$$R(i,j) \Leftrightarrow \begin{cases} m_i = m_j \\ M_i = M_j \end{cases}$$

$\forall j \; 1 \le j \le L$

Deux processus sont en relation si leurs priorités évoluent dans le même intervalle de priorités. R serait alors une relation d 'équivalence sur Ψ d'ensemble quotient de card r= card (Ψ/R). Les processus d'une même classe sont ordonnancés suivant leurs priorités. A chaque classe est associé un intervalle de priorités $I_k = [m_k, M_k]$

$1 \le k \le r$.

O.P.B implémentera dans ces conditions l'ordonnancement en files multiniveaux, lors que $I_k \cap I_j = \varnothing \quad \forall \; k \neq j \quad \forall j \; 1 \le j \le L, \forall k \; 1 \le k \le L$

Chaque classe de (Ψ/R) modélise alors une file. Pour pouvoir assimiler cet ordonnancement à un ordonnancement en file multinivaux, il suffit de forcer une priorité fixe à tous les processus d'une même classe i.e.

$$\forall k \quad 1 \le k \le r \quad m_k = M_k .$$

et d'attribuer un quantum de temps aux classes inversement proportionnel à la valeur de cette priorité fixe.

5°) Si à chaque passage du processus en service nous modifions sa valeur de priorité dans son descripteur de processus, de telle façon que celui-ci appartienne à la file (classe) supérieure, l'ordonnancement deviendrait alors un ordonnancement en files multiniveaux avec rétroaction [MULTILEVEL-FEEDBACK] [Dietel 84] [Krakowiak 87].

V. 2.4 REMARQUE

Dans ce qui a précédé, nous remarquons que O.P.B implémente différentes stratégies par une simple modification ou adaptation des bornes associées au processus et au besoin, en forçant sa priorité à prendre une valeur particulière, mais ceci n'influence pas les gestionnaires des files, et les algorithmes gérant le dispatcher restent indépendants de la stratégie projetée sur O.P.B.

V.2.5. FONCTION DE PRIORITE

Les processus de Ψ interagissent entre eux et avec leur environnement. De ce fait ils subissent des contraintes que nous allons modéliser à l'aide de deux paramètres notés t_i^a (t) qui représente le temps d'attente du processus τ_i pour avoir le processeur, et t_i (t) le temps d'utilisation effective du processeur. Ces deux paramètres sont calculés à partir d 'une origine de date fixe.

V.2.5.1. DEFINITION DE LA FONCTION DE PRIORITE

1°) Définition 1

Un processus τ_i i= 1...L quelconque de Ψ commence à s'exécuter avec la priorité p_i (0). A une date t > 0 quelconque il s'exécute avec la priorité p_i (t) donnée par:

$$p_i(t) = \frac{M_i\, t_i^a(t) + m_i t_i(t)}{t}$$

D'après la définition des paramètres t_i (t) et t_i^a (t) nous avons : $t_i(t) + t_i^a(t) = t$

$\forall\, t > 0$ et $\forall\, i$

p_i (t) représente la moyenne des bornes pondérée par les temps d'attente et d'exécution du processus i mesurés dans]0 , t]. La priorité ainsi calculée est d'autant plus forte que le processus a attendu plus longtemps pour s'exécuter.

En posant $\forall i$ i=1...L $\Delta P = M_i - m_i$ nous vérifions que:

$\forall\, i$, i=1...L

$$p_i(t) = M_i - \Delta P_i \frac{t_i(t)}{t} \qquad (2) \forall\ t > 0$$

A la date t= 0, τ_i reçoit une priorité initiale p_i (0) établie dans son descripteur de processus. Nous ne précisons rien sur sa valeur si ce n'est que $p_i(0) \in [m_i, M_i]$.

2°) Définition 2

On appelle ordonnancement par priorités bornées moyennes ou O.P.B.M. l'ordonnancement O.P.B où la priorité des processus est calculée suivant (2).

V.3 L'EQUITE

Dans les systèmes classiques des files d'attente, les performances étudiées sont généralement les temps moyens d'attente dans la file ou dans le système, le nombre moyen de clients dans la file ou dans le système etc ... .

Dans cette étude, une performance particulière dite « équité »est principalement étudiée. De la nature de cette performance dépendra l'efficacité de la stratégie.

Un ordonnancement est dit équitable pour un ensemble de n processus

$$E=\{\tau_i, 1\leq i \leq n\}$$

si le taux d'utilisation du processeur est identique pour tous les processus de E.

Dans notre cas, les L processus de ψ sont répartis en r classes disjointes de priorité. Nous nous intéressons à une classe particulière que nous notons E contenant n processus. Cette classe sera caractérisée par ses bornes m_i et M_i que nous notons m et M respectivement.

Nous faisons l'hypothèse qu'aucun processus de E ne possède de priorité fixe, donc $M\neq m$, soit donc $\Delta P=M-m$. En réorganisant au besoin la classe E nous pouvons écrire: $E=\{\tau_i, 1\leq i \leq n\}$.

V.3.1. TAUX D'OCCUPATION DE LA CLASSE

Le taux d'occupation de l'unité centrale (processeur) par les n processus de E ou par abus de langage par la classe E, est noté ρ(t) défini par:

$$\forall t>0 \quad \sum_{i=1}^{n} t_i(t) = t\rho(t) \quad (1)$$

Comme E est une classe propre de ψ/R, le processeur n'exécute pas uniquement que les processus de E et ceux là ne restent pas indéfiniment inactifs donc $0 < \rho(t) < 1$.

V.3.2. TAUX D'OCCUPATION DU PROCESSUS DE LA CLASSE

Le taux d'occupation de l'unité centrale (processeur) par un processus quelconque de la classe particulière E, est le réel $\rho_i(t)$ défini par:

$$\forall i \ 1\leq i \leq n \qquad \forall t>0 \quad \rho_i(t) = \frac{t_i(t)}{t}$$

Nous dirons que l'ordonnancement est équitable pour E si chaque processus de E reçoit en moyenne la même fraction du temps d'activité du processeur que tout autre processus de E .

Nous dirons aussi que l'ordonnancement est asymptotiquement équitable pour E si à la limite ($t\rightarrow\infty$) l'ordonnancement est équitable pour E.

V.4 CONJECTURES DE HARO-PROUST

Après avoir établi ces définitions, Haro-Proust émettent deux conjectures. La première concerne le taux d'occupation d'une classe propre de E. Les auteurs font alors l'hypothèse que l'ensemble des n processus de la classe E, utilise toujours la même fraction de temps d'unité centrale, autrement dit que $\rho(t)$ est constant. Les auteurs argumentent leur conjecture en faisant remarquer que cette propriété du taux d'occupation s'avère exacte et démontrée pour des systèmes à processus périodiques de période fixe.

La deuxième conjecture émise est que la probabilité qu'un processus i de la classe E dispose du processeur pour s'exécuter est d'autant plus élevée que sa priorité est forte. Haro et Proust font alors l'hypothèse que cette probabilité est égale à :

$$\forall i \ 1 \le i \le n \qquad \forall t > 0 \ \ q_i(t) = \frac{p_i(t)}{\sum_{i=1}^{n} p_i(t)} \rho(t)$$

Les auteurs poursuivent par dire que pendant un intervalle de temps de durée infiniment faible dt, à la date t, le processus i s'exécute pendant une durée moyenne donnée par : $\dfrac{p_i(t)}{\sum_{i=1}^{n} p_i(t)} \rho(t)\,dt \quad \forall i \ , \forall t>0.$

Ce qui mène à dire qu'à la date $t > 0$, le processus i s'est exécuté en moyenne pendant $t_i(t)$ et qu'à la date t+dt il s'est exécuté pendant $t_i(t+dt)$ exprimé comme:

$$\forall i \ 1 \le i \le n \ , \forall t > 0 \ \ t_i(t+dt) = t_i(t) + q_i(t)\,dt$$

$$= t_i(t) + \frac{p_i(t)}{\sum_{i=1}^{n} p_i(t)} \rho(t)\,dt \ .$$

V.4.1. DEMONSTRATION DE LA PREMIERE CONJECTURE DE Haro-Proust.

En acceptant la deuxième conjecture, nous allons montrer que le processeur sert la classe E durant une fraction de temps unité centrale toujours constante i.e. que $\rho(t)$ est constant.

V.4.1.2. DEMONSTRATION

Soit donné $t_i(t+dt) = t_i(t) + \frac{p_i(t)}{\sum_{i=1}^{n} p_i(t)} \rho(t)\, dt \qquad \forall t$

En sommant ces équations sur les processus de E nous obtenons:

$$\sum_{i=1}^{n} t_i(t+dt) = \sum_{i=1}^{n} t_i(t) + \frac{\sum_{i=1}^{n} p_i(t)}{\sum_{i=1}^{n} p_i(t)} \rho(t)\, dt \qquad \forall t.$$

Par dé finition nous ecrivons:

$\{t+dt\}\ \rho(t+dt) = \{t\}\ \rho(t) + \{dt\}\rho(t) \qquad \forall t$

$\{t+dt\}\ \rho(t+dt) = \{t+dt\}\ \rho(t) \qquad \forall t$

D' où $\rho(t+dt) = \rho(t) \qquad \forall t.$

Donc $\rho(t)$ est bien constant, et par suite la stratégie établie ne sera bâtie que sur une seule conjecture. De ce fait nous montrons que:

V.4.1.3. LEMME 1

La somme des priorités des n processus de E à tout instant $t > 0$ est constante, non nulle et donnée par :

$$\forall t > 0 \quad \sum_{i=1}^{n} p_i(t) = P = nM - \rho \Delta P$$

DEMONSTRATION

$$\forall t > 0 \quad \sum_{i=1}^{n} p_i(t) = \sum_{i=1}^{n} (M - \Delta P \frac{t_i(t)}{t})$$

$$= nM - \Delta P \frac{\sum_{i=1}^{n} t_i(t)}{t}$$

$$= nM - \Delta P \frac{\rho \times t}{t}$$

$$= nM - \rho \Delta P$$

P n'est pas nulle .

En effet, supposons le contraire.

P = 0, mais puisque $\forall i,\ 1 \le i \le n$, $\forall t > 0\ \ p_i(t) \ge 0$, alors il est nécessaire que :

$m\, t_i(t) = M\, t_i^a(t) = 0$.

Mais comme M≠m alors $M \neq 0$ ceci exige que $t_i^a(t) = 0$ qui entraîne :

$t_i(t) = t$ et $m = 0$.

L'égalité $t_i(t) = t$ est inconcevable dés que le nombre de processus de la classe E dépasse l'unité, car il faudrait que chaque processus de E reçoive le processeur pendant toute la durée t.

La supposition faite est donc fausse et P n'est donc jamais nulle.

V.4.1.2 LEMME 2

Si nous posons: $\mu = \rho \dfrac{\Delta P}{P}$ alors:

$0 < \rho < 1$ et $n > 1 \Rightarrow 0 < \mu < 1$

Démonstration

La classe E n'est pas un ensemble de processus de priorité fixe par hypothèse, nous avons donc : $\Delta P \neq 0$.

$$1 = \frac{\rho \Delta P}{2\rho \Delta P - \rho \Delta P} > \frac{\rho \Delta P}{nM - \rho \Delta P} \quad \text{dé sque} \quad 2\rho \Delta P < nM$$

Donc : $\mu < 1$ dés que $2\rho \Delta P < nM$

Ou $2\rho (M-m) < nM$

Nous pouvons écrire par majoration:

$$2\rho (M-m) < 2\rho M$$

Mais $2\rho M < 2nM$

dés que $2\rho < n$

Et ceci est vrai si $n > 1$. Donc $\mu < 1$.

d'autre part $\Delta P > 0$ et $nM > \rho \Delta P$ d'où $\mu > 0$.

ce qui achève la démonstration.

V.5. EQUITE DE O.P.B.M

V.5.1. PROPOSITION

Soit E une classe propre de ψ/R de processus de durée illimitée(c'est le cas, par exemple, des taches permanentes périodiques ou non). O.P.B.M est asymptotiquement équitable pour E lorsque les processus de cette classe utilisent une fraction constante d'unité centrale.

Démonstration

D'après ce qui précède:

$$t_i(t+dt) = t_i(t) + \frac{\rho}{P} p_i(t)\, dt$$

$$\forall i\ 1 \le i \le n\ ,\ \forall t > 0 \qquad \text{D'où} \quad \frac{t_i(t+dt) - t_i(t)}{t} = \frac{\rho}{P}\left(M - \Delta P \frac{t_i(t)}{t}\right)$$

qui peut etre ecrite comme:

$$\frac{d(t_i(t)}{t} = \frac{\rho M}{P} - \mu \frac{t_i(t)}{t} \qquad (3)$$

Cette équation est une équation différentielle du premier ordre qui se résoud par séparation de variables en posant $t_i(t) = \rho_i(t)\, t$ (par définition de $\rho_i(t)$).

Pour simplifier l'écriture nous écrirons $t_i(t) = \rho_i\, t$.

Il vient alors $dt_i(t) = \rho_i\, dt + t\, d\rho_i$

$$\frac{dt_i(t)}{dt} = \rho_i + t \frac{d\rho_i}{dt}$$

En remplaç ant dans l'equation (3) nous avons:

$$\frac{\rho M}{P} - \mu \rho_i = \rho_i + t \frac{d\rho_i}{dt}$$

$$\text{qui donne} \quad \frac{dt}{t} = \frac{d\rho_i}{\frac{\rho M}{P} - (1+\mu)\rho_i}$$

Puis que $\mu = \rho\Delta P/P$ et $P = nM - \rho\Delta P$, il vient que:

$$1 + \mu = \frac{nM}{P}$$

On aboutit alors à l'équation:

$$\frac{dt}{t} = \frac{P}{M\rho - n\rho_i} \frac{d\rho_i}{}$$

ou

$$\frac{M}{P}\frac{dt}{t} = \frac{d\rho_i}{\rho - n\rho_i}$$

qui s' integre comme:

$$\frac{M}{P} Log t = -\frac{1}{n} Log|\rho - n\rho_i| + C_1 \Leftrightarrow \frac{-nM}{P} Log t = Log|\rho - n\rho_i| + C_2$$

D'où $\forall i \ 1 \le i \le n \ \forall t > 0 \ t^{-\frac{nM}{P}} = K|\rho - n\rho_i|$

On deduit alors puisque $K \neq 0$ et $\frac{nM}{P} > 0$ que:

$$\forall i \ 1 \le i \le n \quad \lim_{t\to\infty} \rho_i = \frac{\rho}{n}$$

Par conséquent O.P.B.M est asymptotiquement équitable car à l'infini, chaque processus reçoit en moyenne la même fraction de temps unité centrale que tout autre processus de sa classe.

On vérifie aussi que :

$$\lim_{t\to\infty} p_i(t) = \frac{P}{n}$$

Car
$$\begin{aligned}\lim_{t\to\infty} p_i(t) &= M - \Delta P \lim_{t\to\infty} \frac{t_i}{t} \\ &= M - \Delta P \lim_{t\to\infty} \rho_i \\ &= M - \Delta P \frac{\rho}{n} \\ &= \frac{nM - \Delta P\, \rho}{n} \\ &= \frac{P}{n}\end{aligned}$$

V.6. IMPLEMENTATION DE O.P.B.M [3]

La mesure du temps et le recalcul des priorités ne peuvent être continus dans le temps pour un système réel, on propose alors de discrétiser le temps en périodes de longueur T_s. A chaque début de période il y a entrée en service du processus qu'on notera τ' (l'ordonnanceur ou SCHEDULER) qui possède une priorité fixe plus élevée que n'importe quel processus de ψ. Donc, ce processus n'appartient à aucune classe de ψ (en particulier E). Ce processus est périodique, donc toutes les T_s unités de temps, il réévalue la priorité de tous les processus de ψ et réorganise en conséquence la queue d'exploitation et les différentes classes impliquées.

Le DISPATCHER quant à lui assigne toutes les T_s unités de temps le processeur au processus le plus prioritaire.

Donc lors de la j éme période d'ordonnancement (ou entre (j-1) T_s et j T_s), un processus τ_i de la classe E considérée possède la priorité p_i (j-1) = C_i qui sera fixe tout au long de cette période, et ainsi de suite pour les périodes supérieures.

On envisage alors une implémentation de O.P.B.M. qui ne tient compte que du passé récent d'un processus pour recalculer sa priorité, plus précisément, elle ne tiendra compte que du comportement du processus lors de la dernière période d'ordonnancement. Haro-Proust affirment alors que le processus τ_i s'exécute avec la priorité $p_i(j) = C_i$ pour une durée $t_i(j)$ et il attend $t_i^a(j)$ ce qui les amène à écrire d'après ce qui précède:

$\forall i \quad 1 \le i \le n \qquad t_i(j) + t_i^a(j) = T_s > 0$

Donc à l'instant $t = j\,T_s$:

$\forall i \quad 1 \le i \le n$

$$\forall j > 0 \qquad p_i(j) = M - \frac{\Delta P}{T_s} t_i(j-1)$$

Les auteurs poursuivent en représentant le taux d'occupation ρ_i (j-1) de l'unité centrale par le processus τ_i durant la période j-1 par :

$$\rho_i(j-1) = \frac{t_i(j-1)}{T_s}$$

Ce ci ne peut être vrai pour les raisons suivantes:

Durant la période T_s toutes les priorités sont fixées à C_i et ce $\forall i \quad 1 \le i \le n$ et tous les processus les gardent jusqu'à la prochaine entrée en service de l'ordonnanceur. Si nous indexons par α le processus de la classe E élu à l'acquisition du processeur, alors ce dernier ne le lâchera que s'il y a fin de période. Nous pouvons écrire:

$$t_i^a(j-1) = \begin{cases} T_s & \text{si} \quad i \neq \alpha \\ 0 & \text{si} \quad i = \alpha \end{cases}$$

$$t_i(j-1) = \begin{cases} T_s & \text{si} \quad i = \alpha \\ 0 & \text{si} \quad i \neq \alpha \end{cases}$$

Et la prochaine priorité se verra écrire pour la prochaine période de numéro j:

$$p_i(j) = \begin{cases} m & si \quad i = \alpha \\ M & si \quad i \neq \alpha \end{cases}$$

Donc le taux d'occupation ainsi défini par Haro-Proust sera:

$$\rho_i(j-1) = \begin{cases} 1 & si \quad i = \alpha \\ 0 & si \quad i \neq \alpha \end{cases}$$

Ce qui n'est pas concevable car par définition $0 < \rho_i < 1$.

L'erreur vient du fait qu' après avoir subdivisé le temps T en N périodes de longueur T_s, les priorités et les temps d'exécution des processus de la période j dépendent de ceux de j-1 certes, mais le temps de leur évaluation est porté à $j*T_s$ et non calculés pour la période T_s seulement comme il est fait mention .

Donc les formules de calcul des priorités et des taux d'exécution deviennent:

$\forall i \quad 1 \leq i \leq n$

$\forall j > 0$

$$p_i(j) = M - \frac{\Delta P}{jT_s} t_i(j-1)$$

$$\rho_i(j-1) = \frac{t_i(j-1)}{jT_s}$$

Avec $t_i(j) = t_i(j-1) + \delta_{ij} T_s$

$\delta_{ij} = 1$ Si i est élu pour la période j.

$\delta_{ij} = 0$ Si non.

V. 7. JUSTIFICATION THEORIQUE DE L'IMPLEMENTATION

V.7.1 EQUITE

Le rapport $\frac{t_i(j-1)}{jT_s}$ représente le taux d'occupation ρ_i (j-1) de l'unité centrale par le processus i jusqu'à la date $j\,T_s$.

Le taux d'occupation de l'unité centrale par la classe E à cette date est alors ρ défini par:

$\forall j > 0$

$$\sum_{i=1}^{n} t_i(j-1) = \rho\, jT_s = \sum_{i=1}^{n} \rho_i(j-1)$$

Si nous avons :

$$\forall i \ 1 \le i \le n \qquad \frac{t_i(j-1)}{jT_s} = \frac{\rho}{n} \qquad \forall \ j > 0$$

Alors l'ordonnancement est dit équitable. Il est dit asymptotiquement équitable si à la limite nous avons : $\lim_{j \to \infty} \frac{t_i(j-1)}{jT_s} = \frac{\rho}{n}$

On fait alors l'hypothèse que jusqu'à une période quelconque d'ordre j, un processus i de E s'exécute un temps moyen $t_i(j-1)$ donné par :

$$\forall i \ 1 \le i \le n \qquad t_i(j-1) = \frac{p_i(j-1)}{P} \rho \, jT_s \qquad \forall \ j > 0$$

Ceci nous permet d'énoncer le lemme suivant.

V.7.1.2. LEMME

$$\forall i \ 1 \le i \le n \qquad p_i(j) = (-\mu)^j \, p_i(0) + \frac{P}{n}\left[1 - (-\mu)^j\right]$$

$\forall \ j > 0$

Si $p_i(0) = M$ ce résultat s'énoncera comme:

$$\forall i \ 1 \le i \le n \qquad p_i(j) = \frac{P}{n}\left[1 - (-\mu)^{j+1}\right] \qquad \forall \ j > 0$$

V.7.1.2.1. DEMONSTRATION

Par récurrence sur l'ordre de la période, i s'exécute pendant une durée moyenne égale à $t_i(0) = (p_i(0)/P) \, \rho \, T_s$, avec la priorité $p_i(0)$ durant la première période. L'ordonnanceur calcule à la fin de la première période.

$$\begin{aligned} p_i(1) &= M - \frac{\Delta P}{T_s} t_i(0) \\ &= M - \frac{\Delta P}{T_s} \frac{p_i(0)}{P} \rho T_s \\ &= M - \mu \, p_i(0) \end{aligned}$$

$$p_i(2) = M - \frac{\Delta P}{2T_s} t_i(1)$$

$$= M - \frac{\Delta P}{2T_s} \frac{p_i(1)}{P} \rho 2T_s$$

$$= M - \mu p_i(1)$$

$$= M - \mu[M - \mu p_i(0)]$$

$$= (-\mu)^2 p_i(0) + (-\mu)^1 M + (-\mu)^0 M$$

Supposons que cette proprié té soit vraie jusqu'au rang (j - 1) i.e.

$$p_i(j-1) = (-\mu)^{j-1} p_i(0) + M \sum_{k=0}^{j-2} (-\mu)^k$$

Montrons que cette propriété reste vraie pour le rang j .

Le processus i s'exécute jusqu'à la période j-1 avec la priorité p_i (j-1) et pendant une durée moyenne de t_i (j-1)= p_i (j-1)/P ρ j T_s.

Donc:

$$p_i(j) = M - \frac{\Delta P}{jT_s} t_i(j-1)$$

$$= M - \frac{\Delta P}{P} \frac{p_i(j-1)}{jT_s} \rho jT_s$$

$$= M - \mu p_i(j-1)$$

$$= M - \mu \left[(-\mu)^{j-1} p_i(0) + M \sum_{k=0}^{j-2} (-\mu)^k \right]$$

$$= \left[M + (-\mu)^j p_i(0) + M \sum_{k=0}^{j-2} (-\mu)^{k+1} \right]$$

$$= (-\mu)^j p_i(0) + M[1 + \sum_{k=1}^{j-1} (-\mu)^k]$$

$\mu > 0$ donc $(-\mu)^\circ = 1$ d'où

$\forall i \ 1 \le i \le n \qquad \forall j > 0.$

Nous avons:

$$p_i(j) = (-\mu)^j\, p_i(0) + M\sum_{k=0}^{j-1}(-\mu)^k$$

Comme :

$$M\sum_{k=0}^{j-1}(-\mu)^k = M\frac{1-(-\mu)^j}{1+\mu} \qquad \text{(Suite géométrique de raison } \mu)$$

Et comme $\mu = \rho\frac{\Delta P}{P}$ alors $\frac{M}{1+\mu} = \frac{P}{n}$

D'où $p_i(j) = (-\mu)^j p_i(0) + \frac{P}{n}[1-(-\mu)^j] \quad \forall i\ 1 \le i \le n\ ,\forall\ j > 0.$

Lors que $p_i(0) = M$ nous vérifions aisément que:

$$p_i(j) = \frac{P}{n}\left[1-(-\mu)^{j+1}\right]$$

V.7.1.3 DEMONSTRATION DE L'EQUITE ASYMPTOTIQUE

A l'aide de cette priorité calculée discrètement dans le temps (à des temps discrets), nous pouvons d'après ce qui précède dire que le processus i de la classe E s'exécute pendant un temps $t_i(j)$ durant la période d'ordre j donné par:

$$\begin{cases} t_i(j) = \dfrac{p_i(j)}{P}\rho(j+1)T_s = \dfrac{\rho T_s(j+1)}{n}\left[1-(-\mu)^j\right] + \dfrac{\rho T_s}{P}(-\mu)^j\, p_i(0) & \text{si } p_i(0) \neq M \\ t_i(j) = \dfrac{\rho T_s(j+1)}{n}\left[1-(-\mu)^j\right] & \text{si } p_i(0) = M \end{cases}$$

Comme $0 < \mu < 1$ alors

$\forall\ i\ 1 \le i \le n$

$$\lim_{j\to\infty} p_i(j) = \frac{P}{n}$$

$$\lim_{j\to\infty} \frac{t_i(j)}{(j+1)T_s} = \lim_{j\to\infty} \rho_i(j) = \frac{\rho}{n}.$$

V.8. REMARQUE SUR $q_i(t)$

Nous pouvons penser vu la signification de la quantité $q_i(t)$ comme l'ont laissé paraître les auteurs Haro et Proust que $q_i(t)$ peut valoir la quantité $t_i(t)/t$, et donc $\rho_i(t)$. La somme sur i de ces quantités ($q_i(t)$ et $t_i(t)/t$)donnerait le même résultat à savoir ρ. En réalité ceci n'est pas juste pour la raison suivante:

Supposons que :

$$q_i(t) = \frac{p_i(t)}{P}\rho = \frac{t_i(t)}{t} = \frac{\text{temps utile}}{\text{temps total}}$$

Nous aurons donc:

$$\frac{m\,t_i(t) + M\,t_i^a(t)}{\sum_{i=1}^{n} m\,t_i(t) + M\,t_i^a(t)} \cdot \frac{\sum_{i=1}^{n} t_i(t)}{t} = \frac{t_i(t)}{t}$$

$$\frac{m\,t_i(t) + M\,t_i^a(t)}{\sum_{i=1}^{n}[m\,t_i(t) + M\,t_i^a(t)]} = \frac{t_i(t)}{\sum_{i=1}^{n} t_i(t)}$$

$$m\,t_i(t)\sum_{i=1}^{n} t_i(t) + M\,t_i^a(t)\sum_{i=1}^{n} t_i(t) = m\,t_i(t)\sum_{i=1}^{n} t_i(t) + M\,t_i(t)\sum_{i=1}^{n} t_i^a(t)$$

D'où

$$t_i^a(t)\sum_{i=1}^{n} t_i(t) = t_i(t)\sum_{i=1}^{n} t_i^a(t)$$

Mais comme $t_i^a(t) = t - t_i(t)$ nous pouvons é crire:

$$(t - t_i(t))\sum_{i=1}^{n} t_i(t) = t_i(t)\sum_{i=1}^{n}(t - t_i(t))$$

D'où $$t\sum_{i=1}^{n} t_i(t) - t_i(t)\sum_{i=1}^{n} t_i(t) = n\,t\,t_i(t) - t_i(t)\sum_{i=1}^{n} t_i(t) \qquad \forall t>0\ ,\ \forall i$$

Finalement nous avons:

$$\sum_{i=1}^{n} t_i(t) = n\,t_i(t)$$

Ce ci n'est pas vrai en général, il suffit de prendre :

$$t_i(t) = \min t_i(t) \qquad i=1..n$$

Donc : $q_i(t) \neq t_i(t)/t$

V.8.1 ANALOGIE AVEC LES PROCESSUS ALTERNES DE RENOUVELLEMENT

L'intuition de supposer que $q_i(t)$ est égale à $t_i(t)/t$ n'est pas sans fondement, car si nous considérons le processus $X_i(t)$ définit par:

$$X_i(t) = \begin{cases} 1 & \text{si le processus utilise le processeur à la date t} \\ 0 & \text{si non} \end{cases}$$

$q_i(t)$ représentera alors la probabilité $P(X_i(t)=1)$.

V.8.1.1 DISPONIBILITE

V.8.1.2 DEFINITION 1 [17]

On appelle disponibilité $D_i(t)$ à la date t, la probabilité pour qu'un élément i soit en vie (en fonctionnement) à cette date.

On a : $D_i(t) = P(X_i(t)=1) = E(X_i(t))$.

V.8.1.3 DEFINITION] 2

On appelle disponibilité limite ou simplement disponibilité de l'élément i, la valeur si elle existe.

$$D_i = \lim_{t\to\infty} D_i(t)$$

V.8.1.3 DEFINITION] 3

On appelle disponibilité moyenne au cours de la période [0, T], la quantité:

$$d(T) = \frac{1}{T}\int_0^T D(T)\,dt$$

La disponibilité moyenne limite (si elle existe) vaut : $d = \lim_{T\to\infty} d(T)$

Après avoir introduit ces notions, nous pouvons voir qu'il y a une analogie entre $q_i(t)$ et $D_i(t)$ et que contrairement à ce qui est utilisé dans le rapport Haro-Proust, il faudrait utiliser la disponibilité moyenne et non la disponibilité tout court pour pouvoir affirmer que $q_i(t) = t_i(t)/t$ car par définition de d(T) [17], elle représente la proportion de temps (au cours de [0,T]) pendant le quel l'élément fonctionne.

De ce fait en posant $q_i(t) = t_i(t)/t$ nous aboutissons à l'équation $t_i(t) = \frac{\sum_{i=1}^{n} t_i(t)}{n}$

$\forall\ t>0, \forall i$ qui signifie que l'équité est réalisée quelque soit t indépendamment de i, mais en pratique ce n'est qu'un résultat asymptotique. De cette dernière remarque il nous faut préciser que $q_i(t)$ ne peut être évaluée instantanément mais son interprétation en terme de disponibilité n'aura de sens qu'en régime permanent.

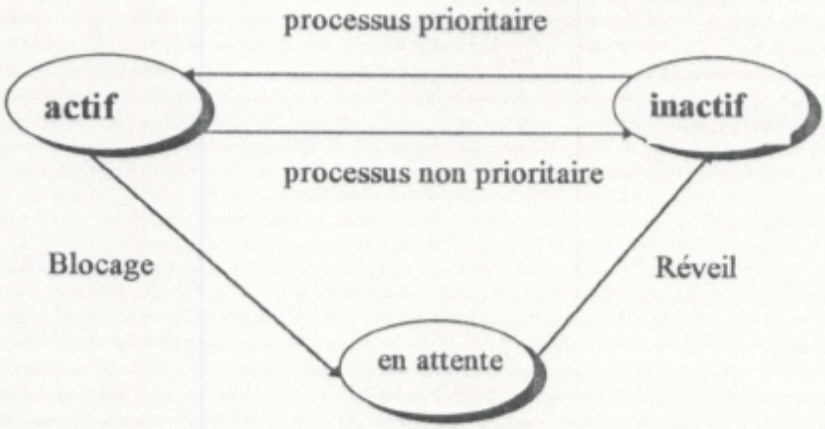
V.9 COMPORTEMENT DU SYSTEME REGI PAR O.P.B.M

V.9.1 ETATS POSSIBLES DES PROCESSUS

Le système étudié par Haro-Proust est un système contenant L processus qui sont supposés tous présents à l'instant t = 0 et qui le resteront indéfiniment.

Un processus d'une classe donnée, par exemple la classe E qu'on a étudiée plus haut, évolue dans trois états possibles.

V.9.1.1 DIAGRAMME D'ETAT DES PROCESSUS



-L'état actif est caractérisé par le fait que le processus détient le processeur.

-L'état inactif est caractérisé par le fait que le processus ne détient pas le processeur mais attend seulement sa libération.

-L'état en attente est caractérisé par le fait que le processus ne peut accéder au processeur même s'il est le plus prioritaire, car il attend un événement extérieur pour pouvoir continuer son traitement par exemple une entrée-sortie.

V.9.2 ETAT DES CLASSES DE PROCESSUS

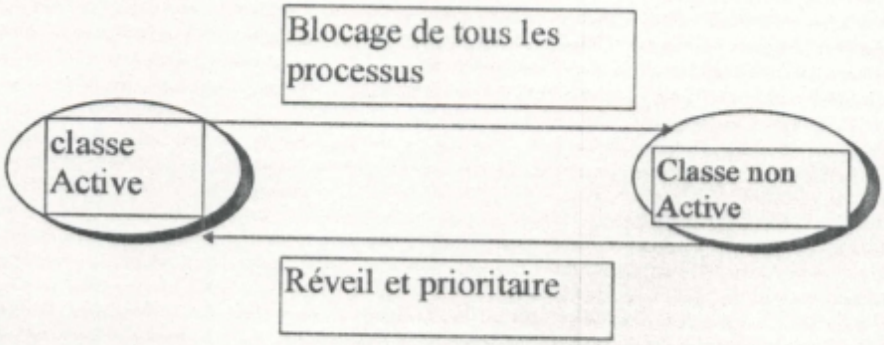
Les classes sont ordonnées de 1 à k par priorités décroissantes c-a-d la classe d'ordre i est plus prioritaire que celle d'ordre i+1.

De ce qu'on vient de voir on peut conclure que les classes des processus basculent entre deux états possibles.

1- Etat Actif: la classe est la plus prioritaire, qui sous-entend que toutes les classes d'ordre inférieur sont vides, ou plus précisément leurs processus se trouvent à l'état d'attente.

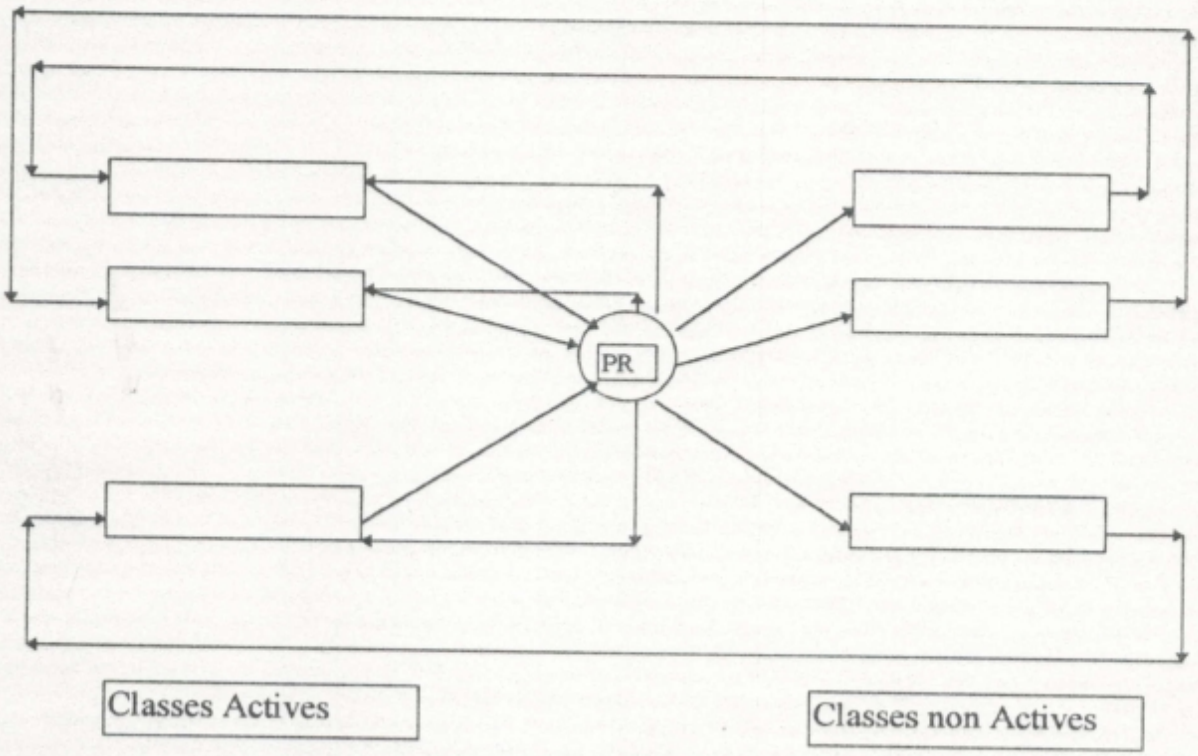
2- Etat inactif: La classe est vide, ou non prioritaire c-a-d une des classes d'ordre inférieur n'est pas vide.

V.9.2.1 DIAGRAMME D'ETAT D'UNE CLASSE.



V.9.3 MODELISATION

Nous pouvons modéliser notre système en un modèle de file d'attente que nous présentons ci-dessous.



Ce système est donc un système fermé; mais si nous l'étudions uniquement avec ses classes actives il ne peut être considéré comme tel, car à un instant t quelconque le nombre de processus dans le système n'est pas fixe.

V.9.4 PRIORITE ET TEMPS D'EXECUTION D'UN PROCESSUS

Dans la section V..2.5 nous avons établit une relation entre la priorité P_i (t) d'un processus et son temps d'exécution t_i (t) qui se traduit par:

$\forall$ i , i=1...L

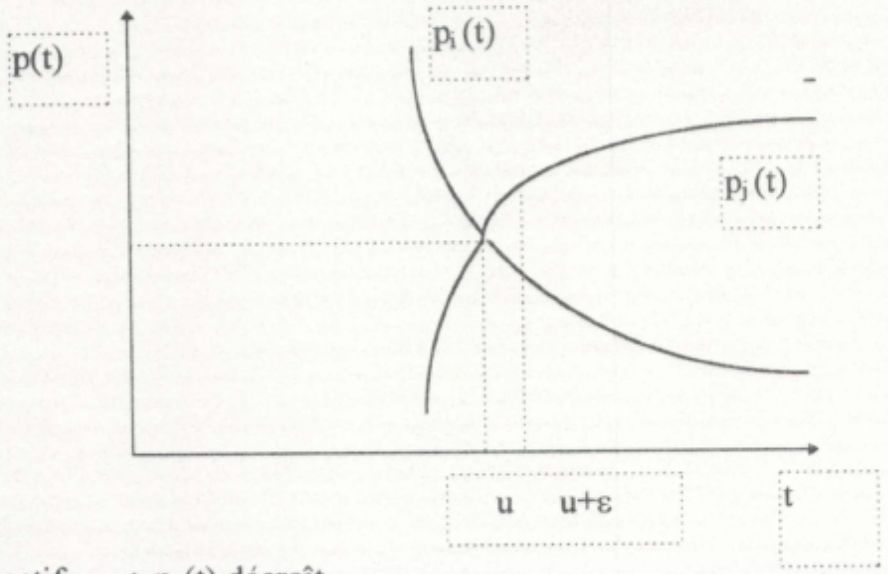
$$p_i(t) = M_i - \Delta P_i \frac{t_i(t)}{t} \qquad \forall\ t > 0$$

On peut conclure alors que:

$$t_i(t) = t\left(\frac{M - p_i(t)}{\Delta P_i}\right)$$

V.9.5 INCOMPATIBILITE-AMBIGUITE OU CONTRADICTION

Dans [3] l'auteur signale une contradiction dans la définition même de l'O.P.B.M en faisant remarquer que si deux processus τ_i , τ_j sont l'un actif (τ_i), l'autre (τ_j) , inactif alors leurs priorités évoluent comme le montre la figure suivante:



Pour $t < u$

τ_i actif $\Rightarrow p_i(t)$ décroît

τ_j inactif $\Rightarrow p_j(t)$ croit.

Pour $t = u$

$p_i(t) = p_j(t) = q$

Pour $t > u$ Comment le système va-t-il évoluer.

Si τ_j est activé alors à l'instant $u+\varepsilon$ nous avons:

$t_j(t)$ croit $\Rightarrow p_j(t)$ décroît $\Rightarrow p_j(u+\varepsilon) < p_j(u) = q$

τ_i est inactif $\Rightarrow p_i(t)$ croit $\Rightarrow p_i(u+\varepsilon) > p_i(u) = q$

Donc: $p_j(u+\varepsilon) < p_i(u+\varepsilon)$.

Même constatation si τ_i est activé.

τ_i est actif alors à l'instant $u+\varepsilon$ nous avons:

$t_i(t)$ croit $\Rightarrow p_i(t)$ décroît $\Rightarrow p_i(u+\varepsilon) < p_i(u) = q$

τ_j est inactif $\Rightarrow p_j(t)$ croit $\Rightarrow p_j(u+\varepsilon) > p_j(u)$

Donc: $p_i(u+\varepsilon) < p_j(u+\varepsilon)$.

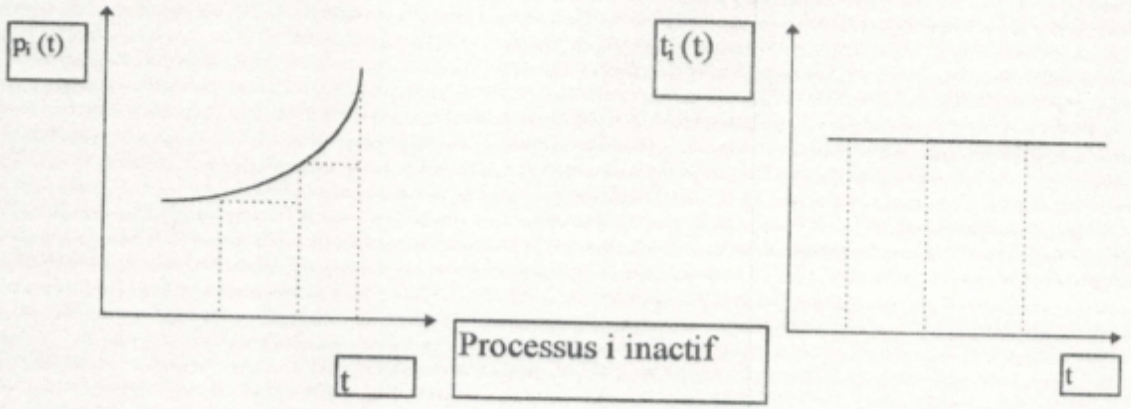
Contradiction avec le fait qu'à tout instant c'est le processus le plus prioritaire qui détient le processeur.

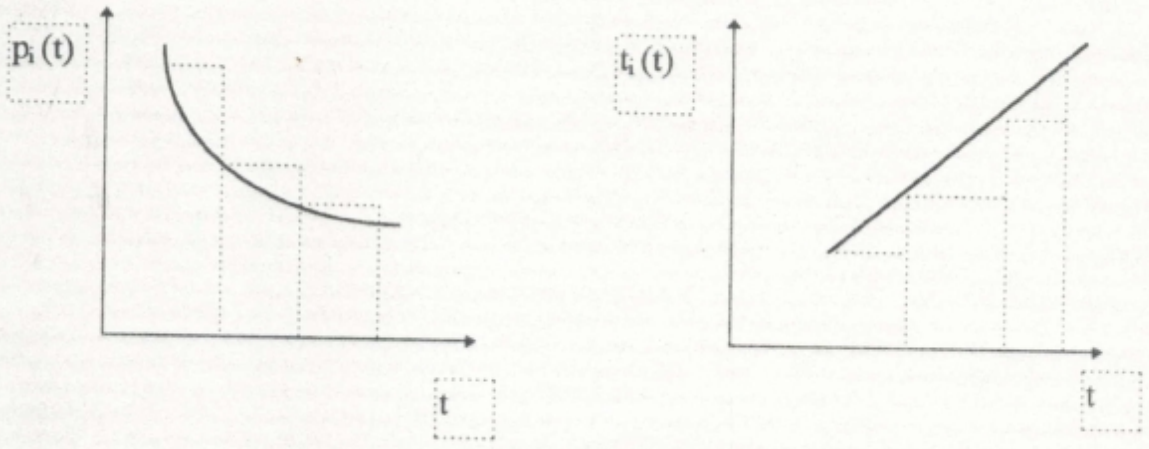
V.9.6 PRECISION DE LA DEFINITION D'O.P.B.M

Il faut préciser à la section V.2 qu'à tout instant ***OBSERVABLE*** c'est le processus le plus prioritaire qui détient le processeur, de ce fait $t_i(t)$ et $p_i(t)$ évoluent par sauts.

Entre deux sauts consécutifs $t_i(t)$ évolue lineairement suivant une droite d'équation $y = AX$ et $p_i(t)$ Décrit un arc d'hyperbole.

SCHEMATIQUEMENT.





Processus i actif

En prenant en compte ces éléments la contradiction signalée dans [3] est levée car:

Soit un temps t_0 qu'on supposera observable.

Si à ce temps $p_j(t_0) = p_i(t_0)$ alors deux sous cas se présentent.

Si τ_j est élu il le sera pour un laps de temps q, puisque par définition de O.P.B.M la politique est R-R. Il en découle

$p_j(t_0) = p_j \Rightarrow p_j(t_0+q) < p_j(t_0) \downarrow$

$p_i(t_0) = p_i \Rightarrow p_i(t_0+q) > p_i(t_0) \uparrow$

$t_j(t_0) = t_j \Rightarrow t_j(t_0+q) = t_j + q \uparrow$

$t_i(t_0) = t_i \Rightarrow t_i(t_0+q) = t_i + 0$ constant

Si i est élu le même raisonnement est applicable avec permutation d'indice.

Donc à aucun moment le SCHEDULER ne se trouve dans une situation d'ambiguïté de choix du prochain processus à élire.

Conformément à ce qu'on vient de voir nous pouvons affirmer que les quantités $t_i(t)$ et $p_i(t)$ varient suivant l'algorithme ci-dessous:

V.9.6.1 ALGORITHME DE CALCUL DES $t_i(t)$ ET DES $P_i(t)$.

$t_i(0) = 0$

$P_i(0) = P_0$

Répéter

si est élu

Alors

$t_i(t_a) = t_i(t_{a-1}) + t_x$

$P_i(t_a)$ = calculée suivant V.6

Si non

$t_i(t_a) = t_i(t_{a-1})$

$P_i(t_a)$ = calculée suivant V.6

a: représente le $a^{ième}$ point observable.

t_x: représente le temps durant le quel i est actif.

V.9.7 ANALOGIE DE O.P.B.M AVEC UN SYSTEME COURANT

Pour mieux comprendre le fonctionnement de notre système nous allons établir une analogie avec le système simple suivant:

Imaginons un infirmier spécialisé travaillant dans une grande clinique. Cet infirmier a la charge d'un nombre L toujours fixe de malades de différents âges variant entre 20 et 90 ans et qui demandent incessamment son assistance, ce dernier décide alors de

répartir ses malades en classes, chaque classe est caractérisée par le regroupement de patients ayant un âge compris entre deux valeurs fixes, par exemple 3 classes, la première regroupent les malades de 90 à 60 ans la deuxième les malades de 40 à 59 ans, la troisième enfin ceux de 20 à 39 ans.

Notre infirmier décide alors de toujours venir en aide à la classe de plus forte valeur d'âge. Les malades d'une classe donnée sont dotés d'une préférence initiale estimée par l'infirmier suivant le cas des malades qu'elle regroupe.

Les malades commencent par solliciter les soins de l'infirmier en appuyant sur une sonnerie munie d'une horloge marquant la durée de leurs appels qui fait varier sa préférence et que l'infirmier peut consulter à la fin d'une durée de soin apporté à un malade.

L'infirmier commence par celui qui a la plus grande valeur en préférence, au bout d'un certain temps il quitte ce malade, vérifie toutes les valeurs de préférences des malades et prendra en charge celui qui est doté de la valeur supérieure en préférence (qui correspondra dans ce cas à celui qui a le plus longtemps appelé), ainsi de suite.

Il faut signaler qu'un malade pris en charge perd en valeur de préférence.

Il arrive parfois qu'un malade ait besoin pour son traitement d'un médicament qu'on doit lui apporter de l'extérieur, l'infirmier n'ira s'occuper de ce malade que lorsque son médicament est disponible même s'il possède la plus forte valeur de préférence.

De ce fait, il arrive que tous les malades d'une même classe se trouvent dans ce cas, l'infirmier ira alors s'occuper des malades de la classe suivante, ainsi de suite.

Dans le cas où tous les malades de la clinique sont en manque de leurs médicaments, le soignant reste alors inactif demandant à Dieu pitié pour ses malades.

SCHEMATIQUEMENT

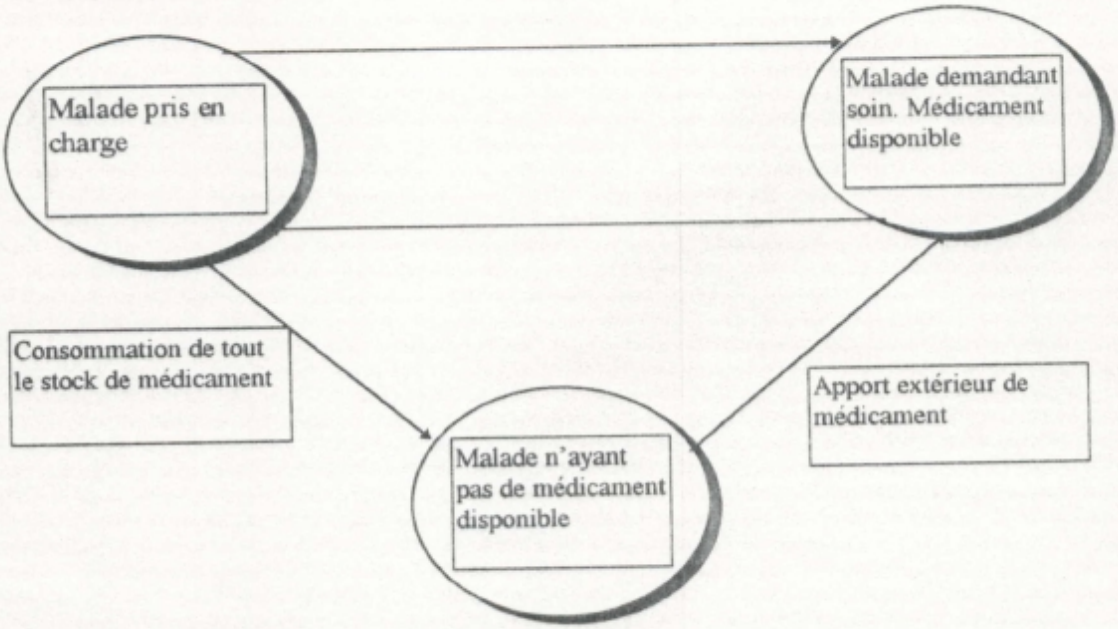


DIAGRAMME DE CHANGEMENT D'ETAT D'UN MALADE DE LA CLINIQUE

V.10. MODELE DE SIMULATION DE O.P.B.M

V.10.1 VARIABLES DE SIMULATION

Chaque classe contiendra n(classe) de processus que nous représentons par un vecteur de taille n(classe), ce vecteur sera accompagné d'un autre vecteur que nous désignons par indicateur (rang) qui contiendra deux valeurs possibles, 1 si le processus est activable 0 s'il est en attente. Un nombre dit nombre processus(classe) qui lui variera entre 0 et n(classe) et qui indiquera le nombre de processus activables dans la classe. Enfin deux vecteurs P et T, l'un contenant les valeurs des priorités, l'autre les valeurs de $\rho_i(t)$ de chaque processus de la classe.

V.10.2 PROCEDURES DE LA SIMULATION DE L' O.P.B.M

V.10.2.1 PROCEDURES REVEIL ET BLOCAGE

A/PROCEDURE REVEIL.

Cette procédure simulera les passages des processus de l'état attente à l'état activable.

Nous supposerons que ce passage se fera d'une manière aléatoire suivant une loi exponentielle. Ceci n'est pas inconcevable car le passage de l'état attente à l'état activable ne dépend que du dernier passage, on peut donc le supposer markovien.

Il faut aussi remarquer que cette procédure concerne toutes les classes ayant un nombre de processus(classe) inférieur à n(classe).

B/ PROCEDURE BLOCAGE

Cette procédure quant à elle concerne le dernier processus servi par le processeur, ce blocage se fait aussi d'une manière aléatoire suivant une loi générale.

V.10.2.2 PROCEDURE CHOIX-CLASS.

Cette procédure désigne la classe à activer c-a-d la classe de la quelle se fera le choix du prochain processus à élire. Là aussi il faut signaler que cette classe ne doit pas être vide et que toutes les classes d'ordre supérieur le soient.

V.10.2.3 PROCEDURE CALCUL

Cette procédure est constituée de deux sous procédures:

A/ PROCEDURE CALCUL ACTIF.

Cette sous procédure utilisera une fonction qu'on appellera SUP-INDICE

A.1 FONCTION SUP-INDICE

Cette fonction désignera le processus le plus prioritaire de la classe choisie par CHOIX-CLASSE qui se verra attribué le processeur.

LA PROCEDURE CALCUL ACTIF calculera alors les nouvelles quantités P et ρ de cette classe.

V.10.2.4 PROCEDURE CALCUL NON ACTIF.

Cette procédure se chargera de faire les calculs des priorités et des taux d'exécution des autres classes qui n'ont bien sûr pas bénéficié des services du processeur lors du dernier passage.

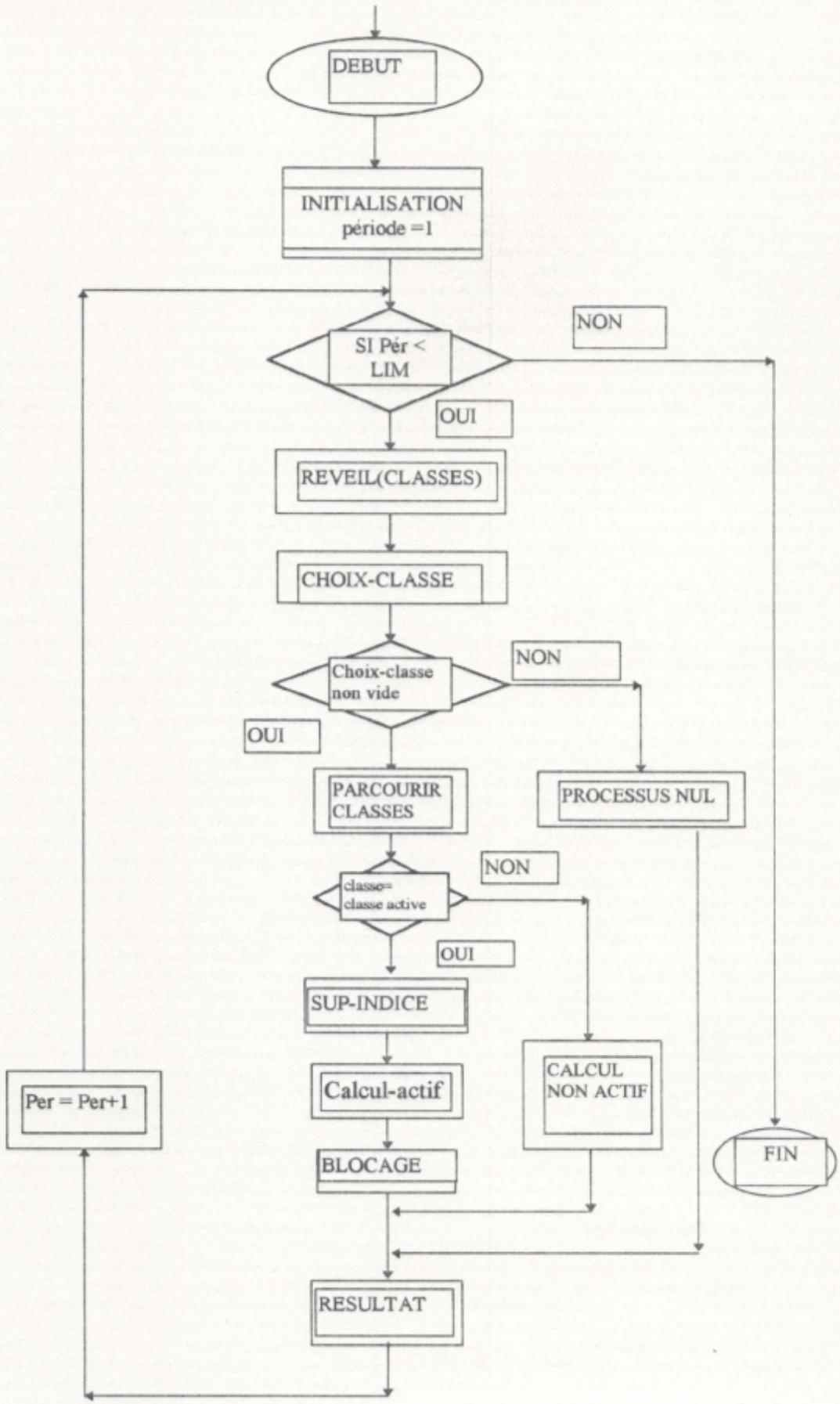
V.10.2.5 PROCEDURE PROCESSUS NUL

Comme nous l'avons fait remarquer, il peut arriver que tous les processus soient bloqués, le processeur ne reste pas réellement inactif mais prendra en charge un processus de priorité nulle, jusqu'à ce que un processus de n'importe quelle classe se réveille et sera alors élu. Cette période consacrée au processus nul représentera la période d'inoccupation du processeur.

V.10.2.6 PROCEDURE RESULTAT

Cette procédure se charge de calculer les taux d'occupation des processus ainsi que ceux des classes correspondantes et de les afficher.

V.10.3 ORGANIGRAMME DE SIMULATION DE O.P.B.M

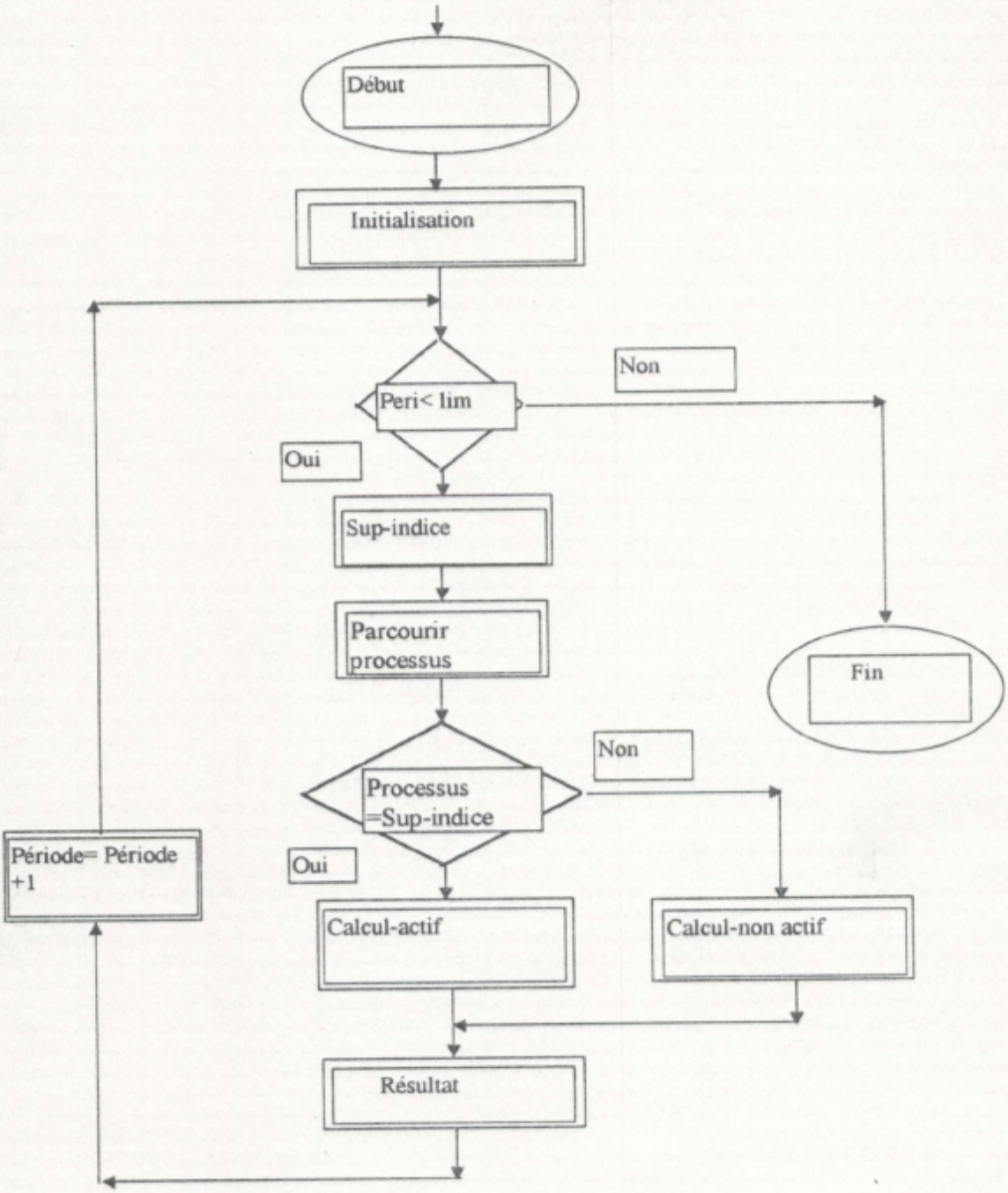


V.11 APPLICATIONS

V.11.1 CAS D'UNE SEULE CLASSE TOUJOURS ACTIVE.

Ce cas est toujours possible à réaliser, il suffit de regrouper tous les processus dans une même classe et d'obliger le processeur à toujours prendre en charge un des processus de la classe.

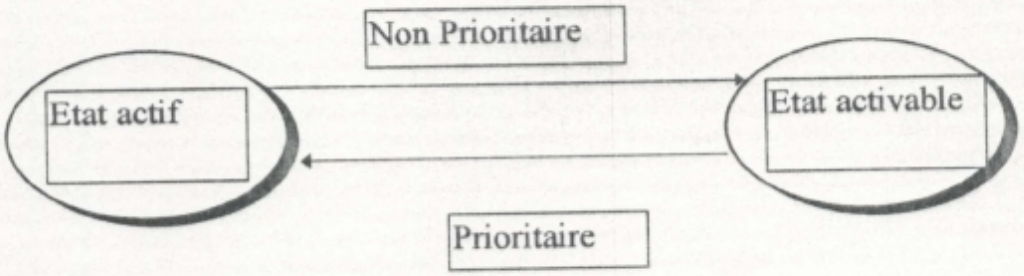
Dans ce cas l'organigramme de simulation de O.P.B.M sera amputé des procédures REVEIL, BLOCAGE et PROCESSUS NUL et est représenté ci dessous.



Cette application peut être envisagée dans les systèmes de surveillance continue.

Théoriquement, nous devons avoir après un temps assez grand tous les ρ_i (t) égaux à une constante de valeur 1/n où n représente le nombre de processus de la classe, car le taux d'occupation ρ de la classe est inévitablement égal à l'unité.

Le diagramme de transition d'état de ce cas particulier est le suivant:



V.11.1.1 APPLICATION 1

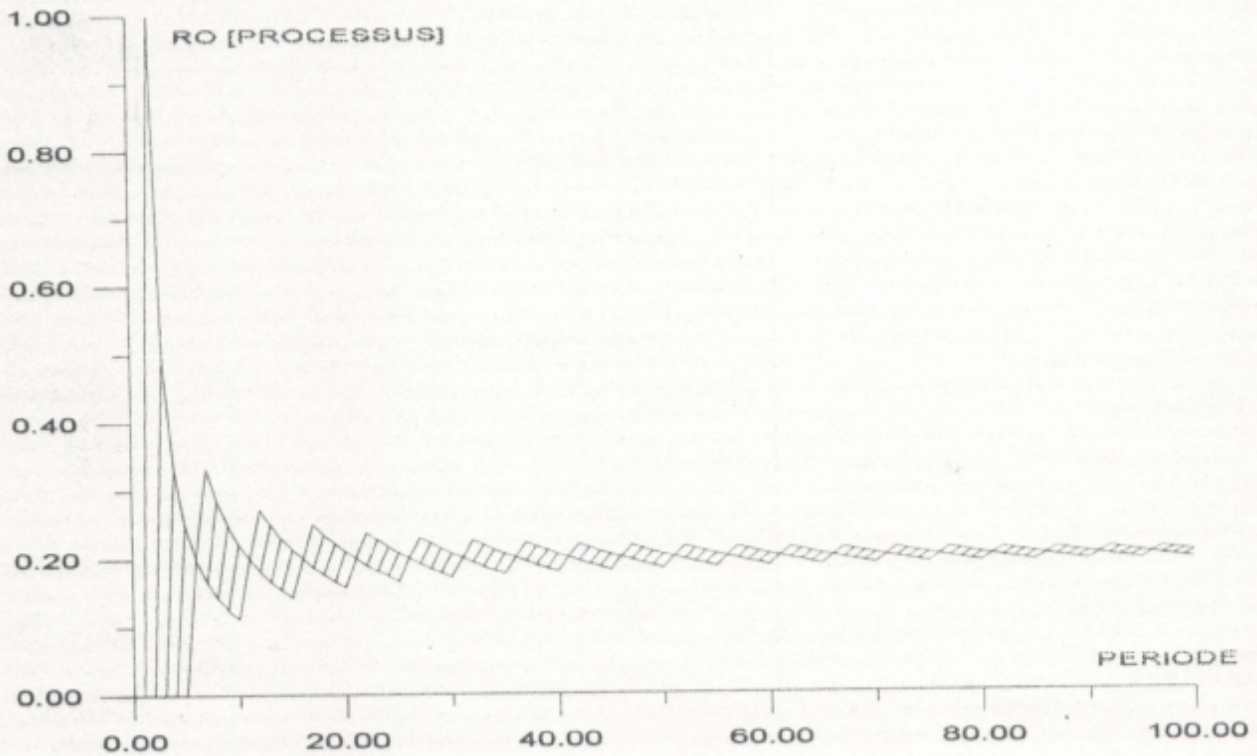
Soit une classe contenant 5 processus de valeur de priorité bornées dans le temps par:

M = 20 et m = 5.

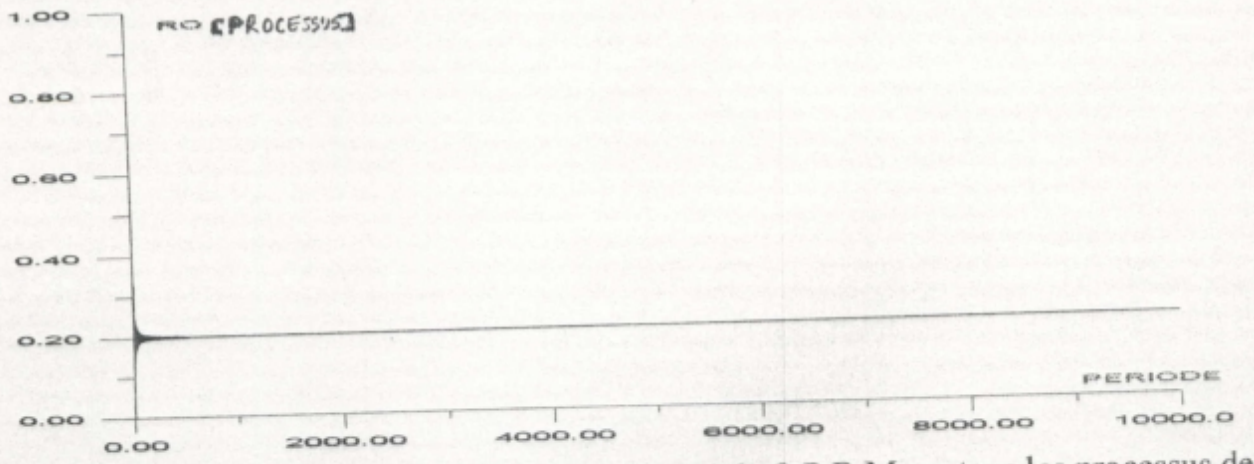
Les priorités initiales des processus sont:

$P_1 = 17$ $P_2 = 12$ $P_3 = 6$ $P_4 = 9$ $P_5 = 14$

Prenons $T = 10000\ T_s$.

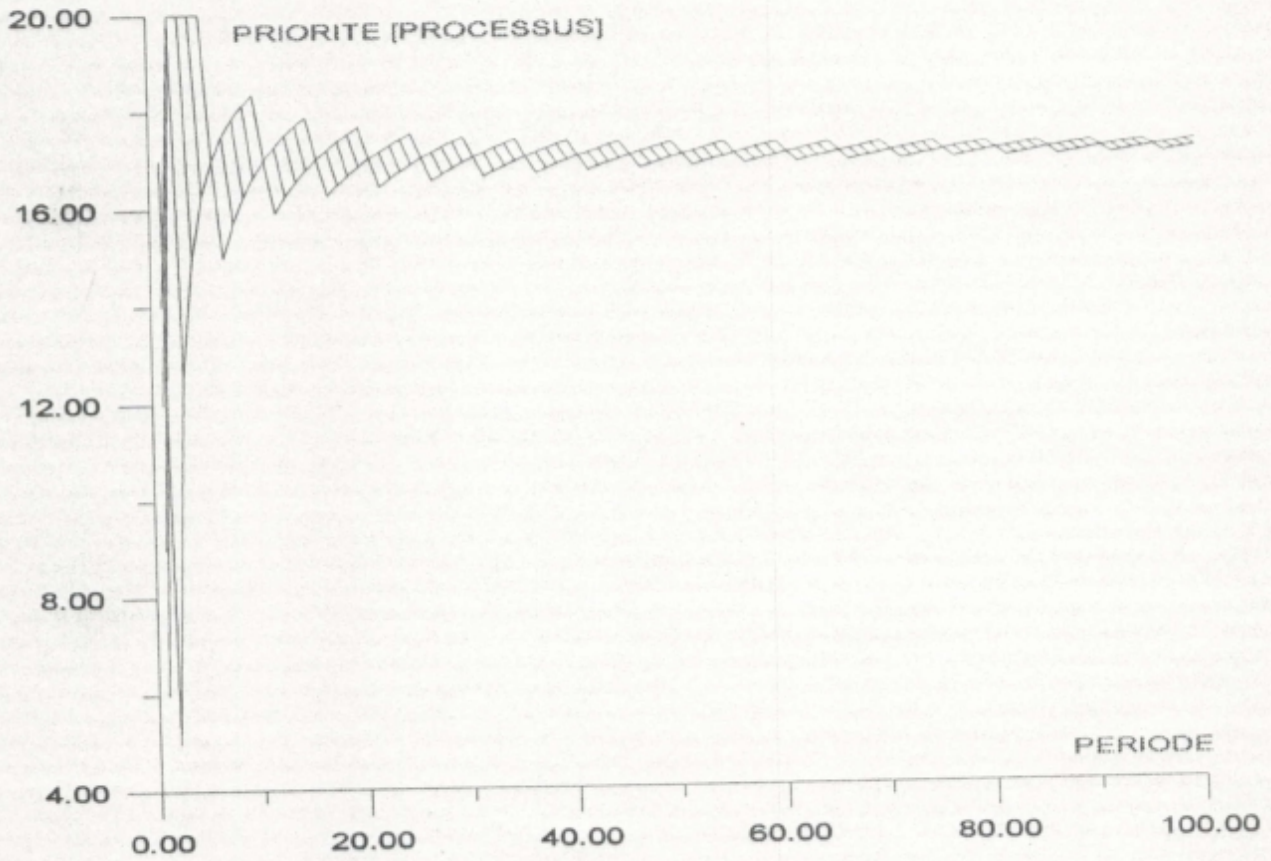


EVOLUTION DES ρ DES PROCESSUS LORS DES 100 PREMIERES PERIODES.

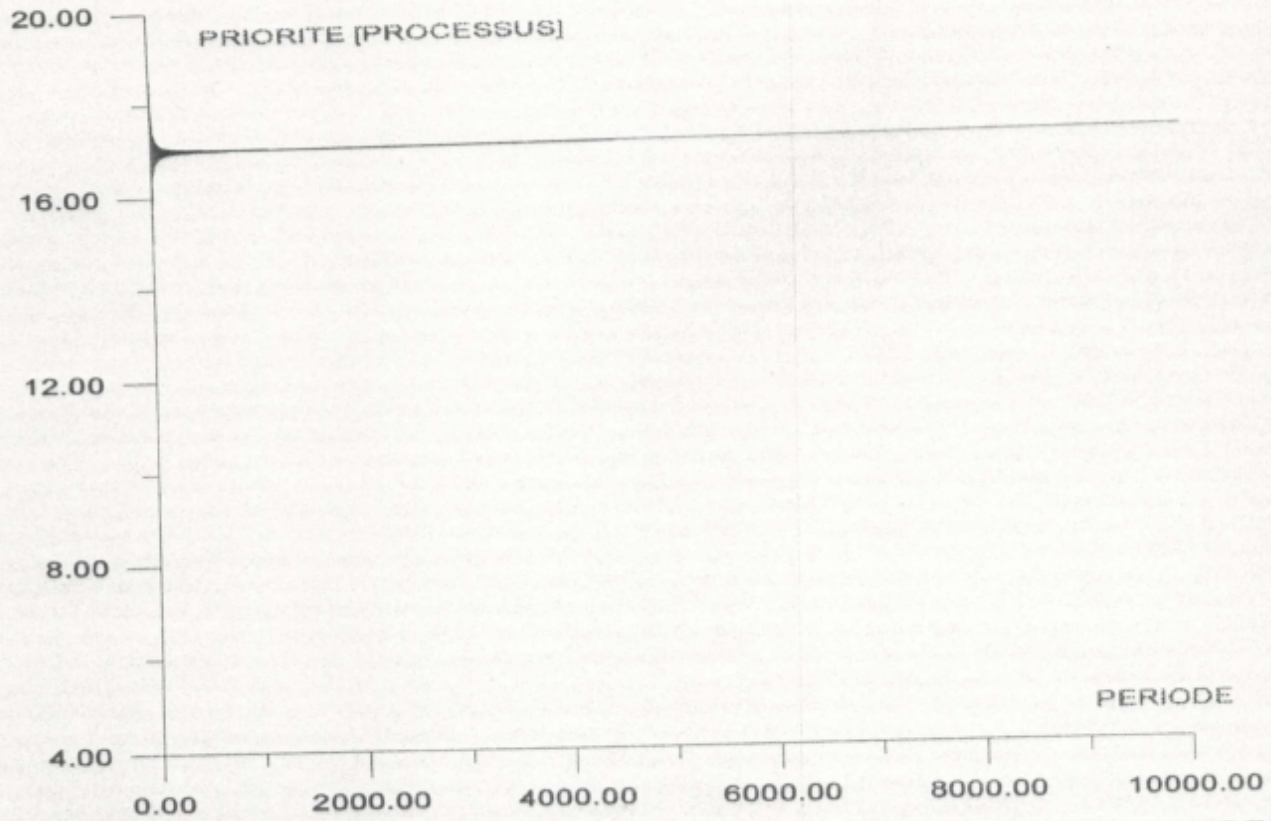


Nous observons clairement sur ce graphe l'équité de O.P.B.M car tous les processus de la classe utilisent asymptotiquement le même taux d'utilisation du processeur à savoir 1/ n, où n représente le nombre de processus de la classe(dans ce cas 0.2 du taux total).

Le graphe ci après nous met en évidence la variation des priorités des processus en fonction du nombre de périodes (de 0 à la 100 ième période).



Ces priorités tendent à se stabiliser autour d'une valeur constante rendant tous les processus homogènes et confirmant ainsi le phénomène de l'équité.



EVOLUTION DES PRIORITES DES N PROCESSUS SUR UN HORIZON DE 10000 PERIODES.

V.11.2. CAS DE PLUSIEURS CLASSES (CAS GENERAL)

Considérons 3 classes de processus réparties comme suite:

****

Classe 1: Borne supérieure $M_1 = 30$

Borne inférieure $m_1 = 21$

Nombre de processus $n(1) = 6$

Priorités initiales $P_{11}(0) = 27$

$P_{12}(0) = 22$

$P_{13}(0) = 28$

$P_{14}(0) = 25$

$P_{15}(0) = 26$

$P_{16}(0) = 24$

Taux de sortie $\mu_1 = 0.95$

Taux d'entrée $\lambda_1 = 0.1$

****

Classe 2: Borne supérieure $M_2 = 20$

Borne inférieure $m_2 = 11$

Nombre de processus $n(2) = 6$

Priorités initiales $P_{21}(0) = 17$

$P_{22}(0) = 14$

$P_{23}(0) = 12$

$P_{24}(0) = 11$

$P_{25}(0) = 13$

$P_{26}(0) = 15$

Taux de sortie $\mu_2 = 0.75$

Taux d'entrée $\lambda_2 = 0.3$

****

Classe 3: Borne supérieure $M_3 = 10$

Borne inférieure $m_3 = 1$

Nombre de processus $n(3) = 5$

Priorités initiales

$P_{31}(0) = 2$

$P_{32}(0) = 7$

$P_{33}(0) = 5$

$P_{34}(0) = 8$

$P_{35}(0) = 4$

Taux de sortie $\mu_3 = 0.65$

Taux d'entrée $\lambda_3 = 0.4$

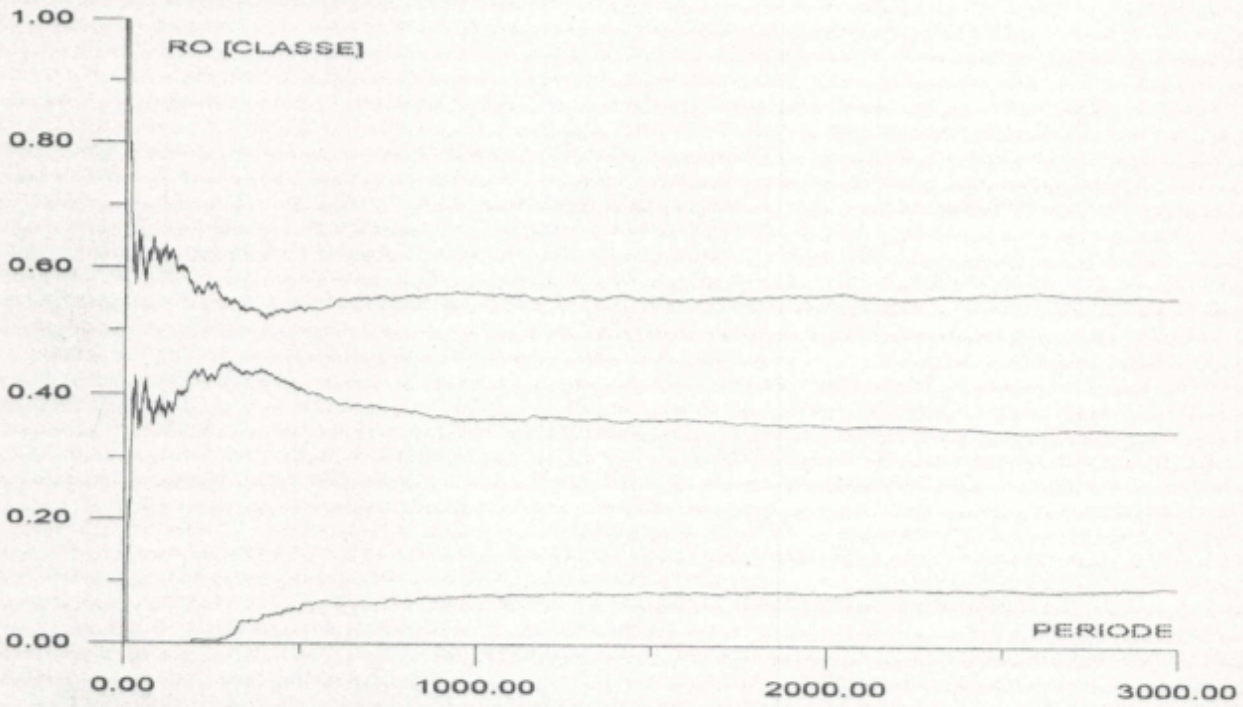
$T = 60000\ T_s$

Ces données vont être conservées pour l'ensemble des applications ci dessous.

V.11.2.1 APPLICATION 1

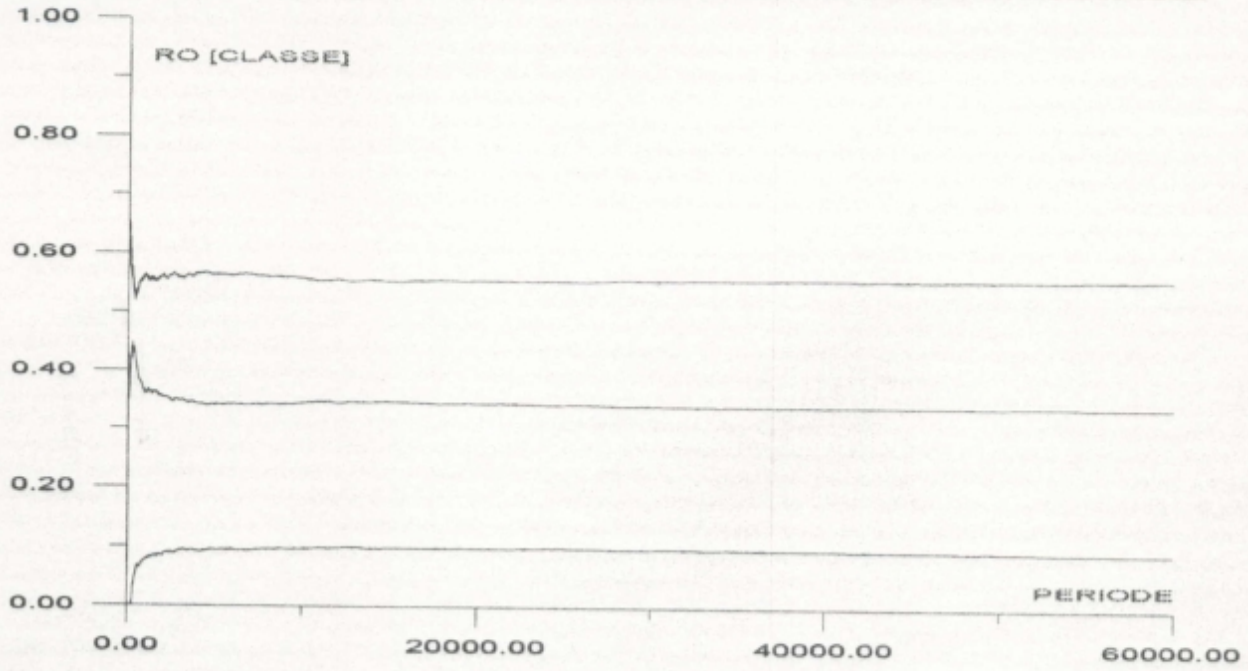
Loi gérant le blocage est la loi exponentielle.

RESULTATS ET INTERPRETATION



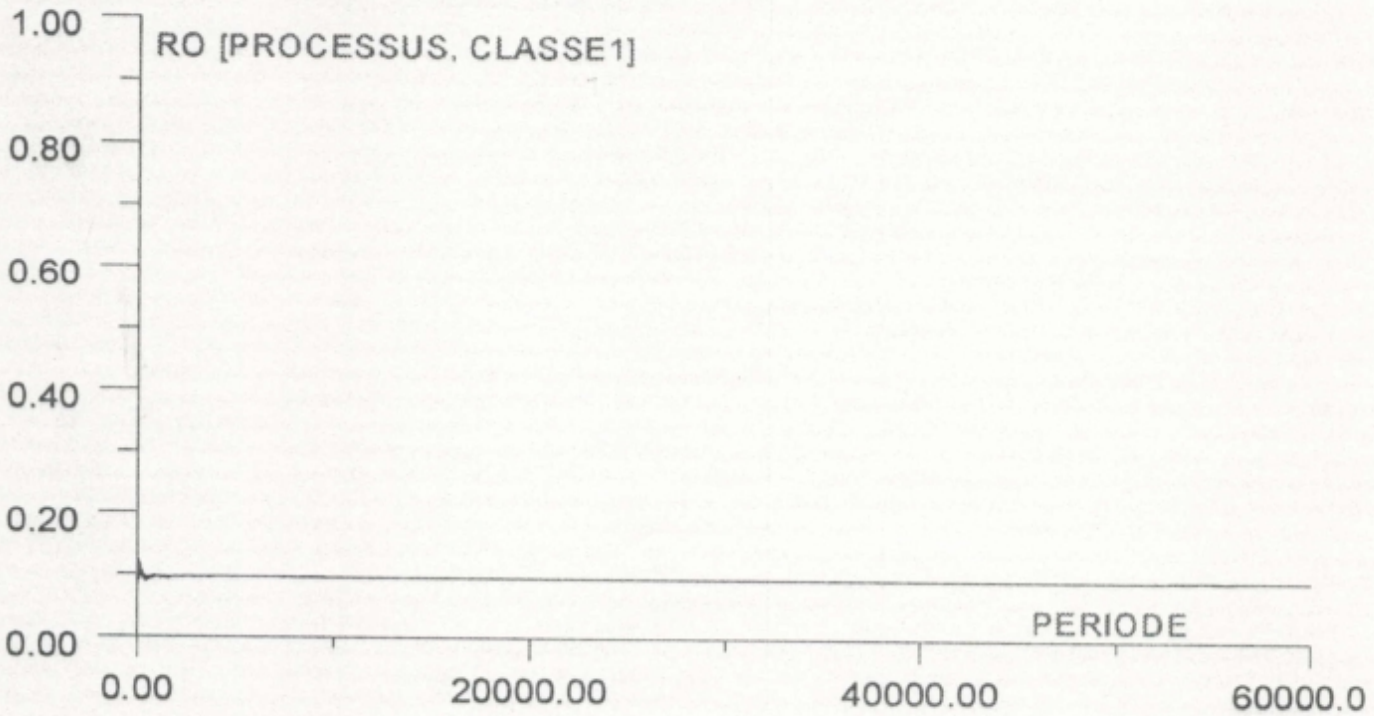
EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES CLASSES LORS DES 3000 PREMIERES PERIODES .

Sur le graphe ci dessus, nous remarquons la flexibilité de la loi exponentielle du fait que tous les ρ des classes se stabilisent très rapidement autour de leurs valeurs constantes confirmant ainsi la première hypothèse de Haro-Proust sur la constance de ρ.



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES PROCESSUS DES TROIS CLASSES SUR UN HORIZON DE 60000 PERIODES.

Le graphe ci dessous nous met en évidence l'évolution des taux d'utilisation du processeur par les six processus de la classe 1. Nous remarquons que l'équité est satisfaite ce qui consolide les résultats théoriques découlant de la deuxième conjecture de Haro-Proust.



V.11.2.2 APPLICATION 2

Loi gérant le blocage est la loi de WEIBULL.

La densité de cette loi comporte trois paramètres, δ qui décrit la durée de vie minimale (ici qui représentera la durée minimale de blocage du processus), θ-δ qui représentera l'échelle et β représentera la forme de la loi.

Sa densité de probabilité s'écrit comme suit:

$$f(x)=\frac{\beta}{\theta-\delta}\left(\frac{x-\delta}{\theta-\delta}\right)^{\beta-1}\exp\left\{\left(\frac{x-\delta}{\theta-\delta}\right)^{\beta}\right\} \qquad x \geq \delta > 0.$$

En posant $\mu=(\theta-\delta)^{\beta}$ et y = x-δ on se ramène à une loi de Weibull à deux paramètres de fonction de répartition :

$$F(x)=1-\exp((-\mu x)^{\beta})$$

En général deux cas sont à distinguer.

V.11.2.2 APPLICATION 2.1

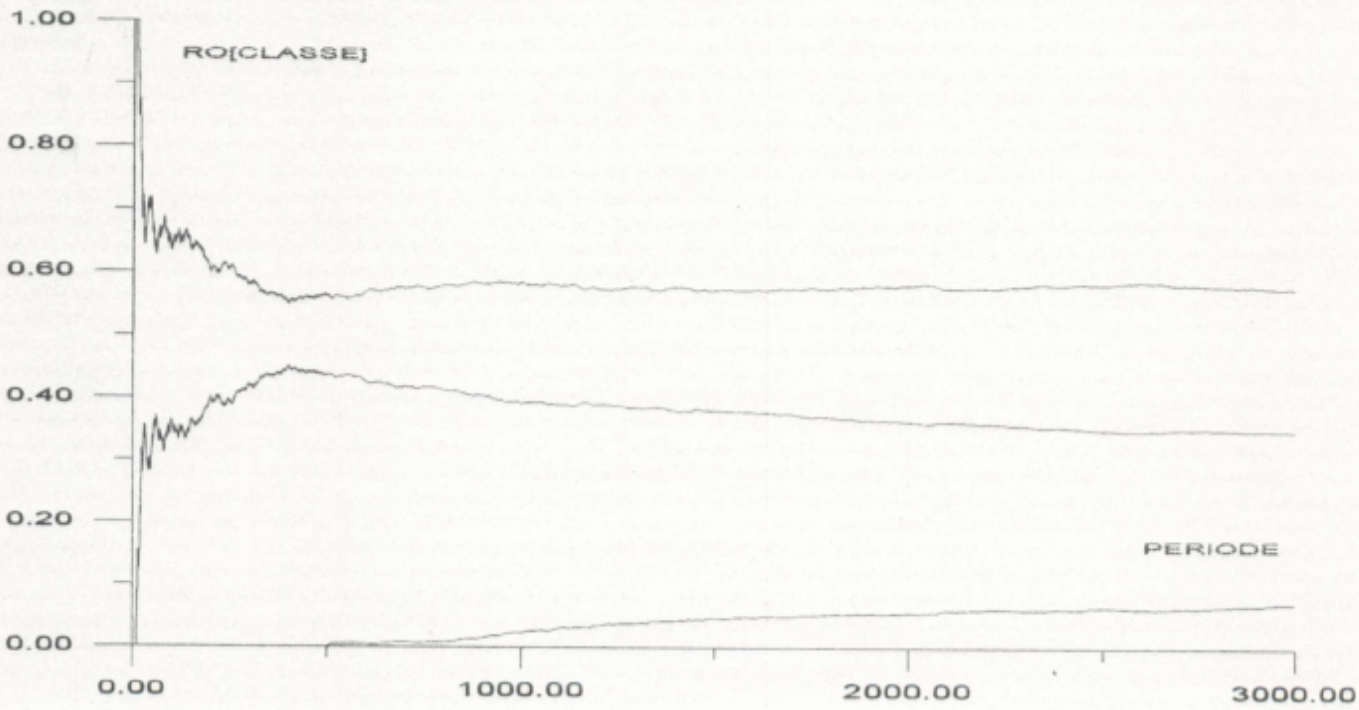
Cas où $\beta < 1$.

$\beta = 0.6$ pour la première classe.

$\beta = 0.5$ pour la deuxième classe.

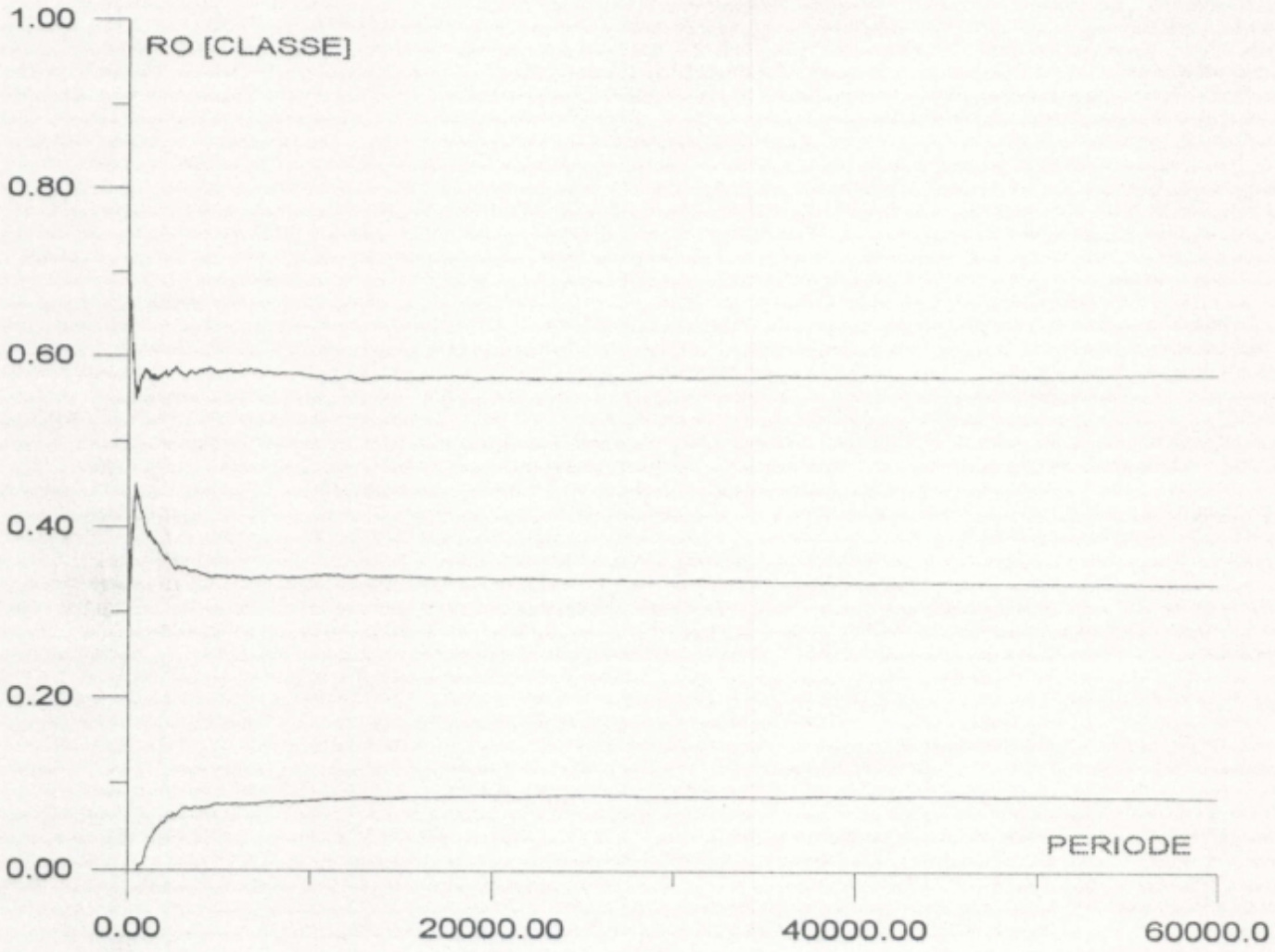
$\beta = 0.8$ pour la troisième classe.

RESULTATS ET INTERPRETATION



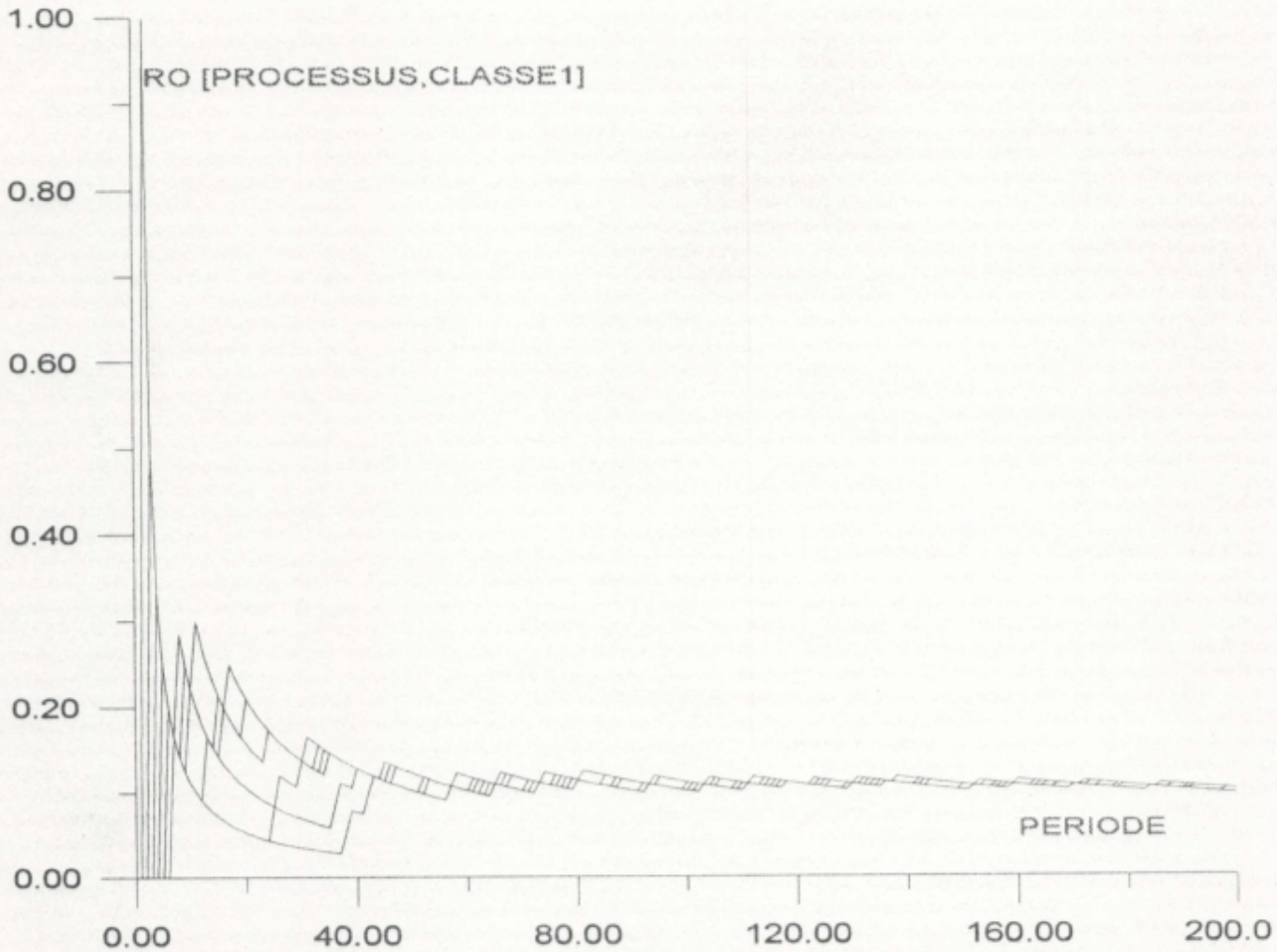
Le graphe ci dessus met en évidence l'évolution des taux d'utilisation du processeur par l'ensemble des processus de chaque classe lors des 3000 premières périodes de simulation, nous pouvons voir que les deux premières classes s'activent très rapidement par rapport à la troisième qui est la moins prioritaire, ceci est dû au fait que le blocage de l'ensemble des processus des deux premières classes prend du temps pour s'effectuer.

Le graphe ci dessous représente l'évolution de ces mêmes taux sur un horizon de 60000 périodes.

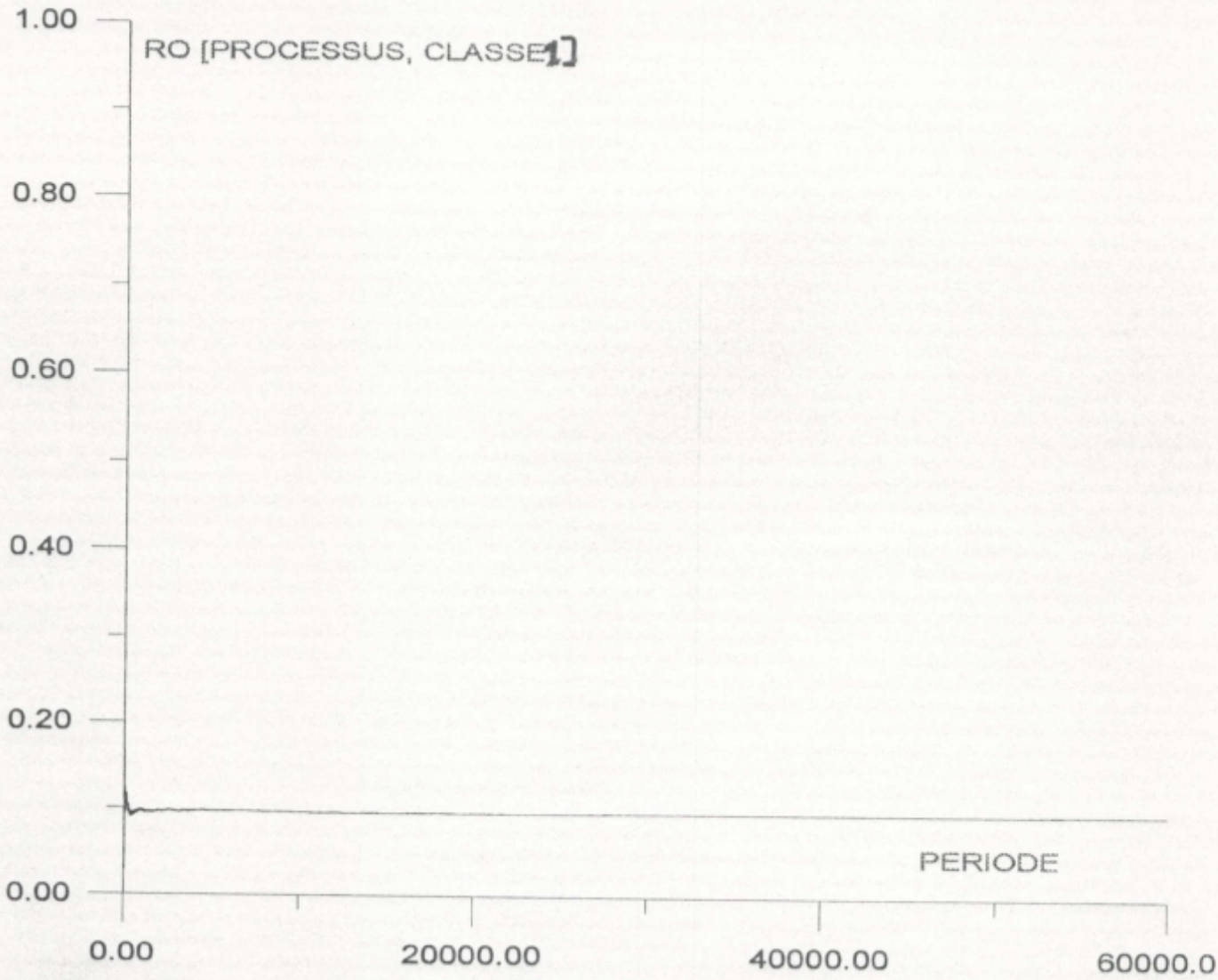


Sur le graphe ci dessus, on distingue clairement la stabilisation de ρ [classe] autour d'une constante c'est ce qui confirme la première conjecture de HARO-PROUST, sauf qu'il faut signaler que ce ci n'est observable non pas quelque soit t mais après que le système se soit stabilisé, c'est à dire que toutes les classes se soient activées, cette période de stabilisation peut être assimilée au régime transitoire du système.

Les graphes ci après nous mettent en évidence le comportement des processus de la première classe en fonction des périodes, le premier nous montre l'évolution de ces taux sur 200 périodes, le second sur 60000 périodes.



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES PROCESSUS DE LA PREMIERE CLASSE LORS DES 200 PREMIERES PERIODES.



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES PROCESSUS DE LA PREMIERE CLASSE DURANT LES 60000 PERIODES.

Ce deuxième graphe nous valide les résultats théoriques obtenus à propos de l'équité de O.P.B.M car nous voyons clairement que les processus de la classe 1 utilisent le processeur d'une manière asymptotiquement équitable et ceci pratiquement à partir de la 10000 ième période.

V.11.2.2 APPLICATION 2.2

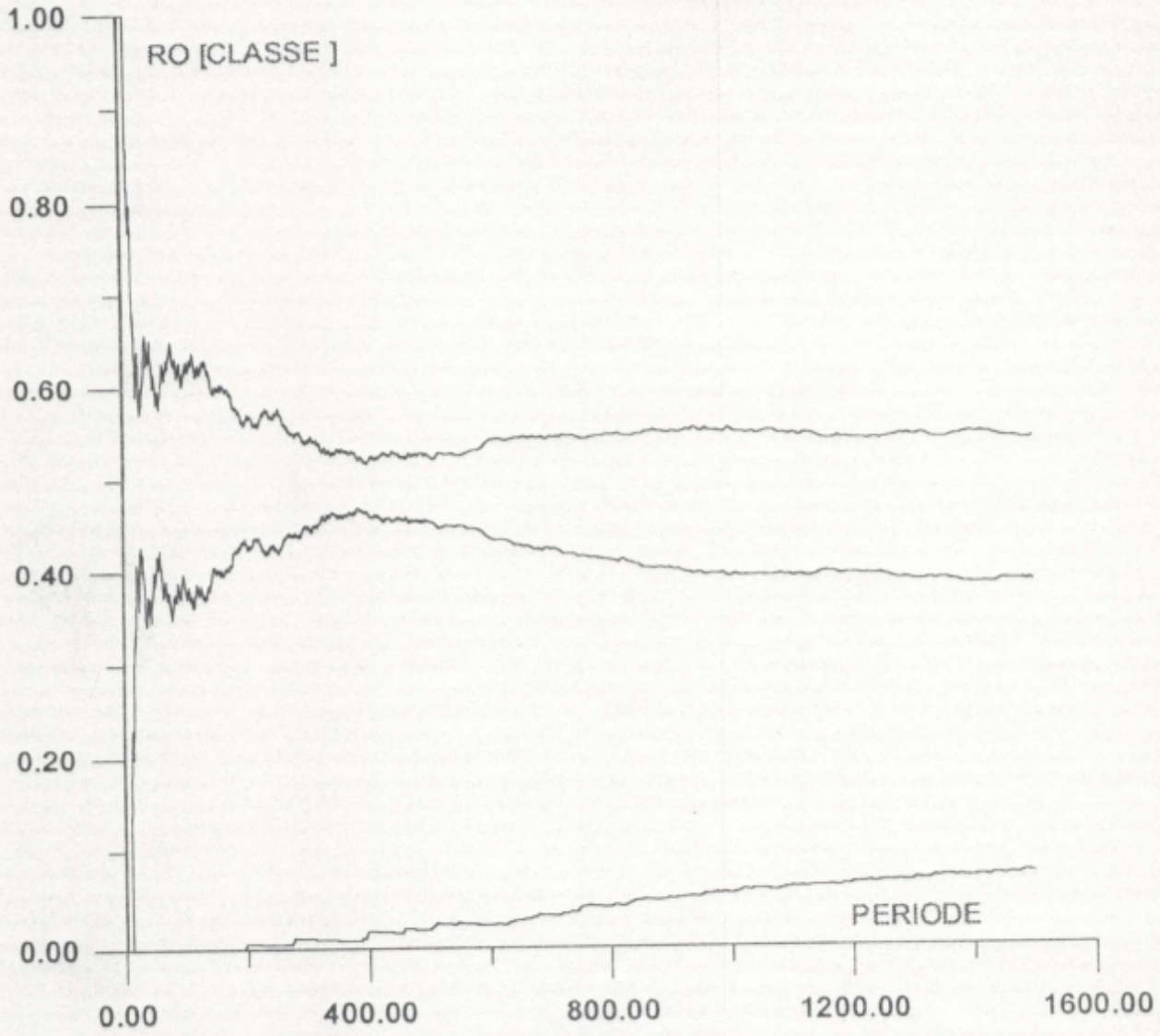
Cas où $\beta > 1$.

$\beta = 1.5$ pour la première classe.

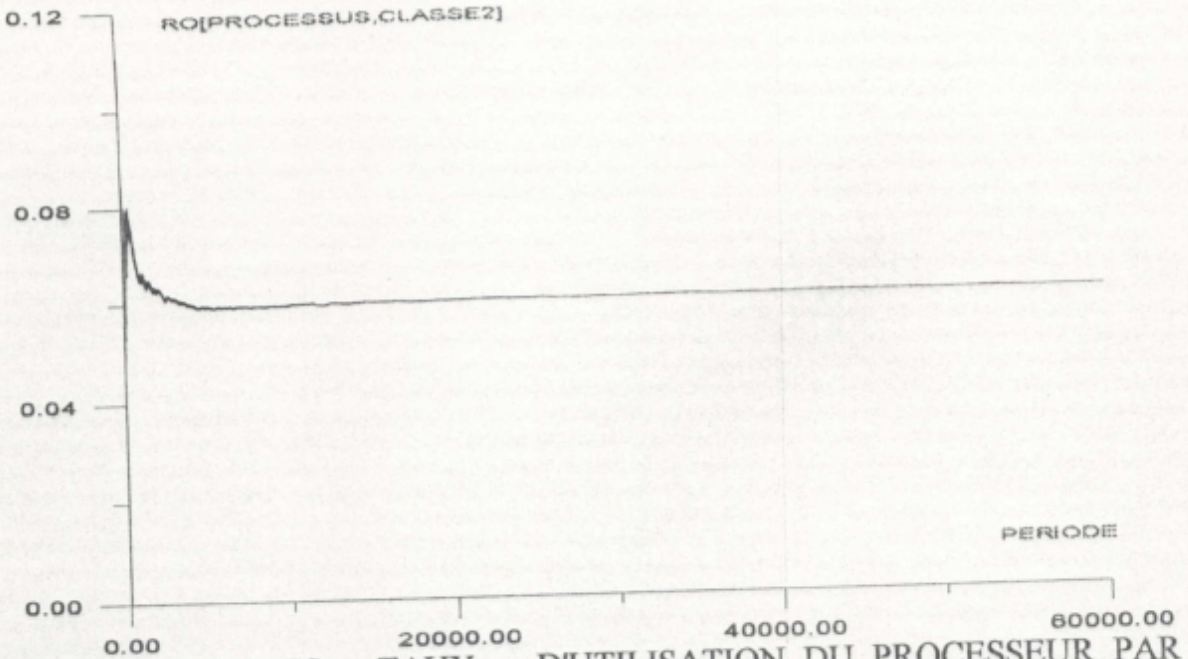
$\beta = 1.4$ pour la deuxième classe.

$\beta = 1.6$ pour la troisième classe.

RESULTATS ET INTERPRETATION



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES CLASSES LORS DES 1500 PREMIERES PERIODES .



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES PROCESSUS DE LA DEUXIEME CLASSE DURANT LES 60000 PERIODES.

V.11.2.3 APPLICATION 3

La Loi gérant le blocage est la loi d'Erlang:

Sa fonction de densité de probabilité est donnée par:

$$f(x) = \frac{\mu(\mu x)^{k-1}}{k!} \exp(-\mu x) \qquad \mu > 0 \text{ , k entier , } x \geq 0.$$

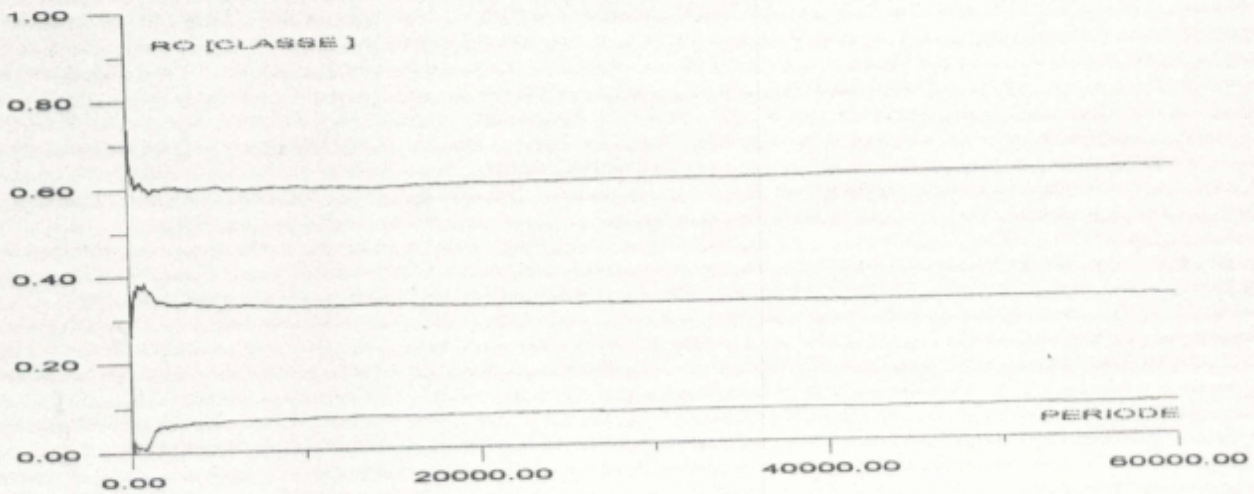
La valeur de k pour la première classe est de : 3

La valeur de k pour la deuxième classe est de : 2

La valeur de k pour la troisième classe est de : 3

RESULTATS ET INTERPRETATION

EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES TROIS CLASSES

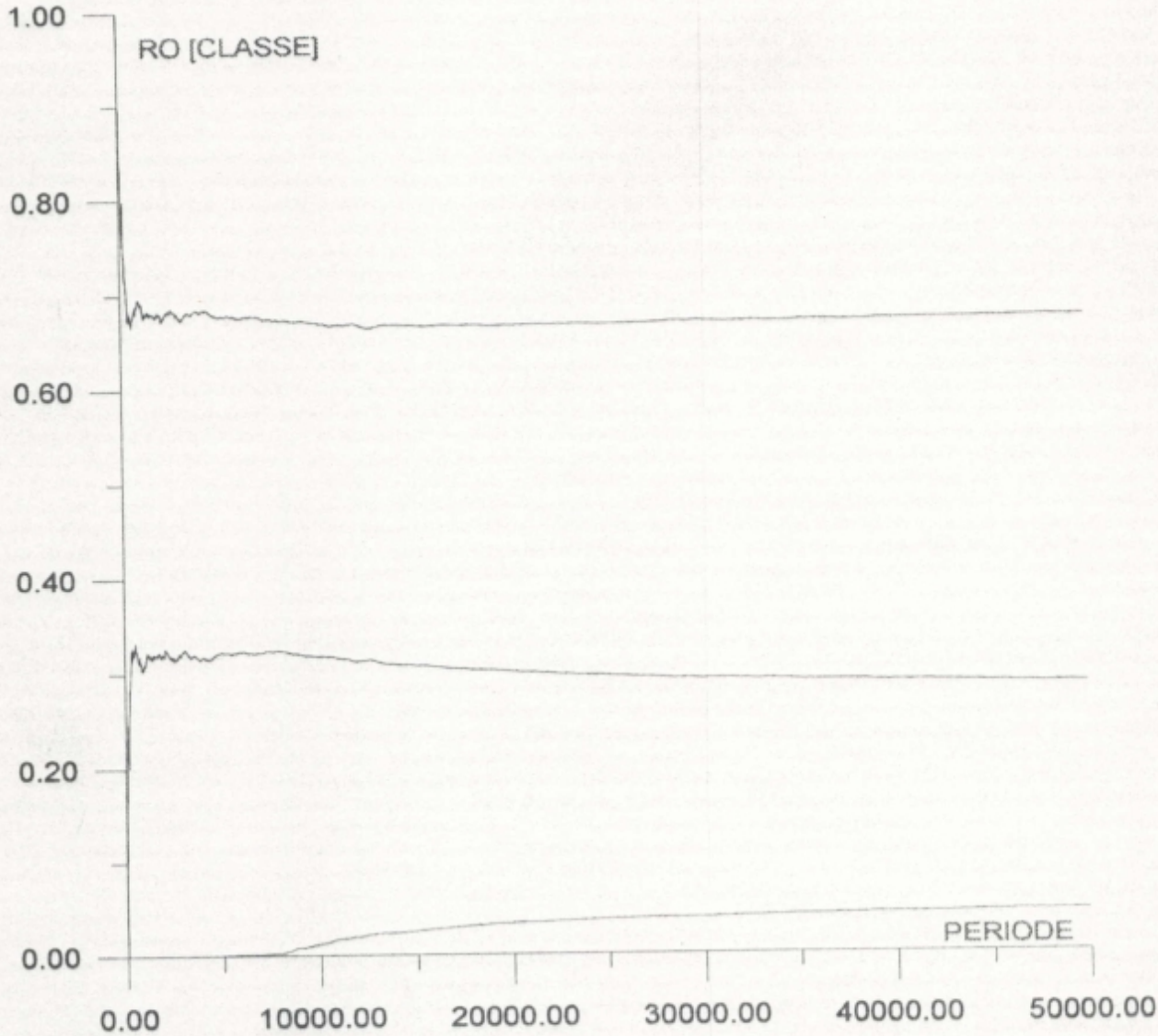


V.11.2.4 APPLICATION 4

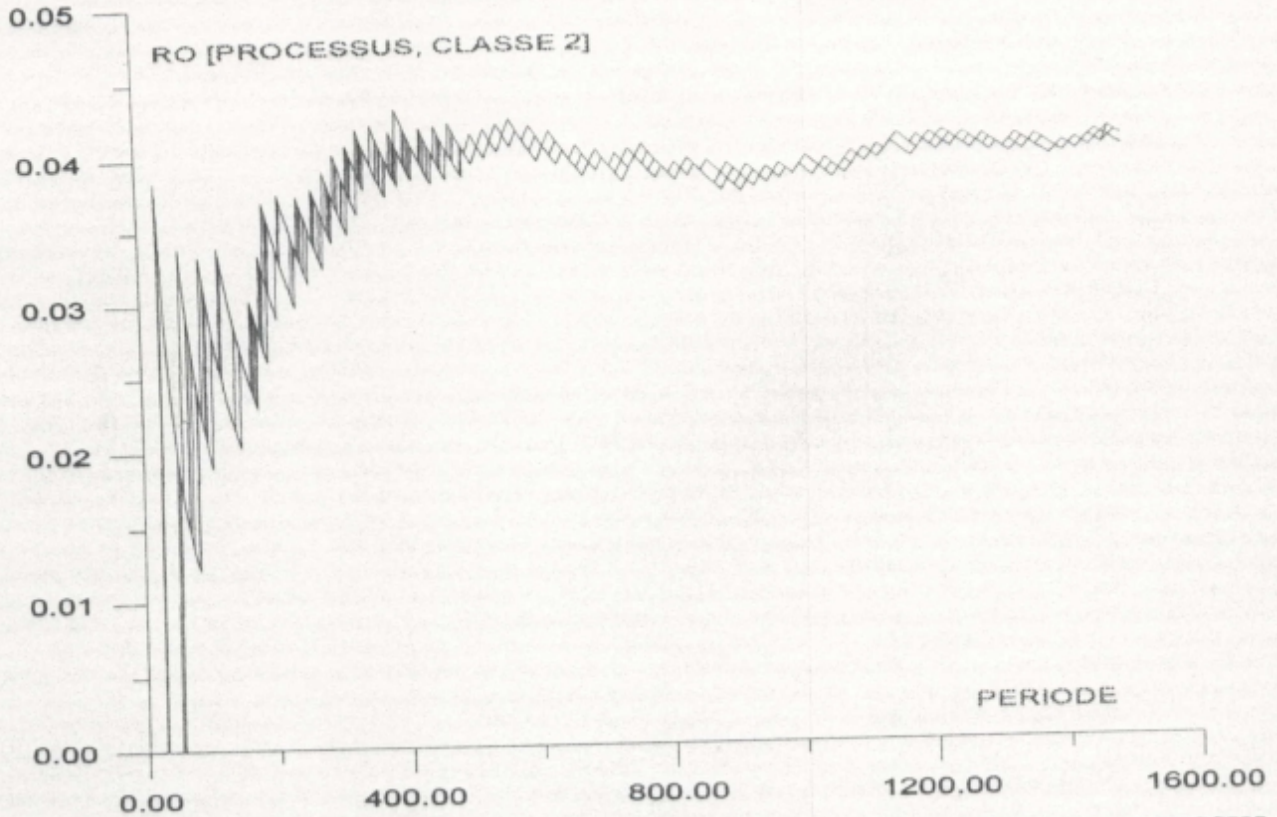
La loi gérant le blocage est la loi de Rayleigh.

Sa fonction de densité de probabilité est donnée par:

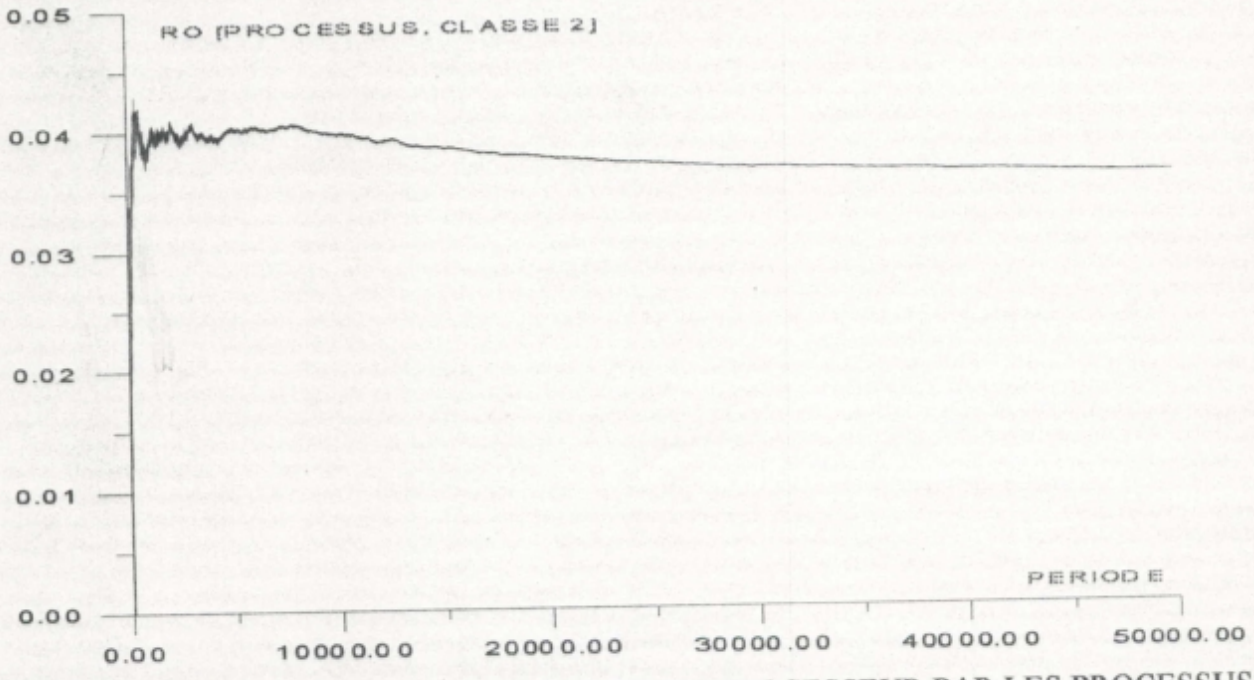
$$f(x)=\mu x \exp(-\mu \frac{x^2}{2})$$



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES TROIS CLASSES SUR UN HORIZON DE 50000 PERIODES.



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES PROCESSUS DE LA DEUXIEME CLASSE DURANT LES 1500 PREMIERES PERIODES.



EVOLUTION DES TAUX D'UTILISATION DU PROCESSEUR PAR LES PROCESSUS DE LA DEUXIEME CLASSE DURANT LES 50000 PERIODES.

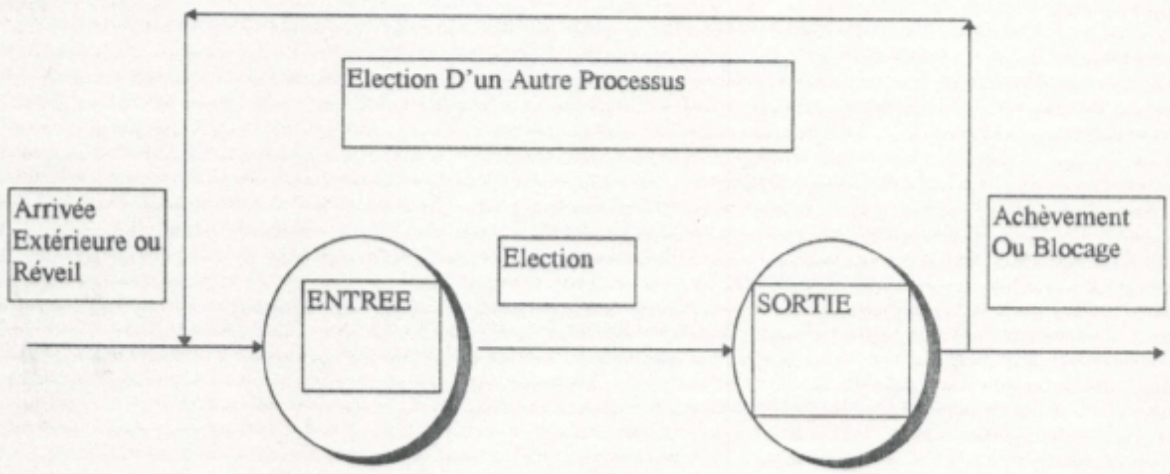
INTERPRETATION

Les mêmes affirmations que les applications précédentes peuvent être formulées.

V.12 O.P.B.M APPLIQUEE A UN SYSTEME OUVERT

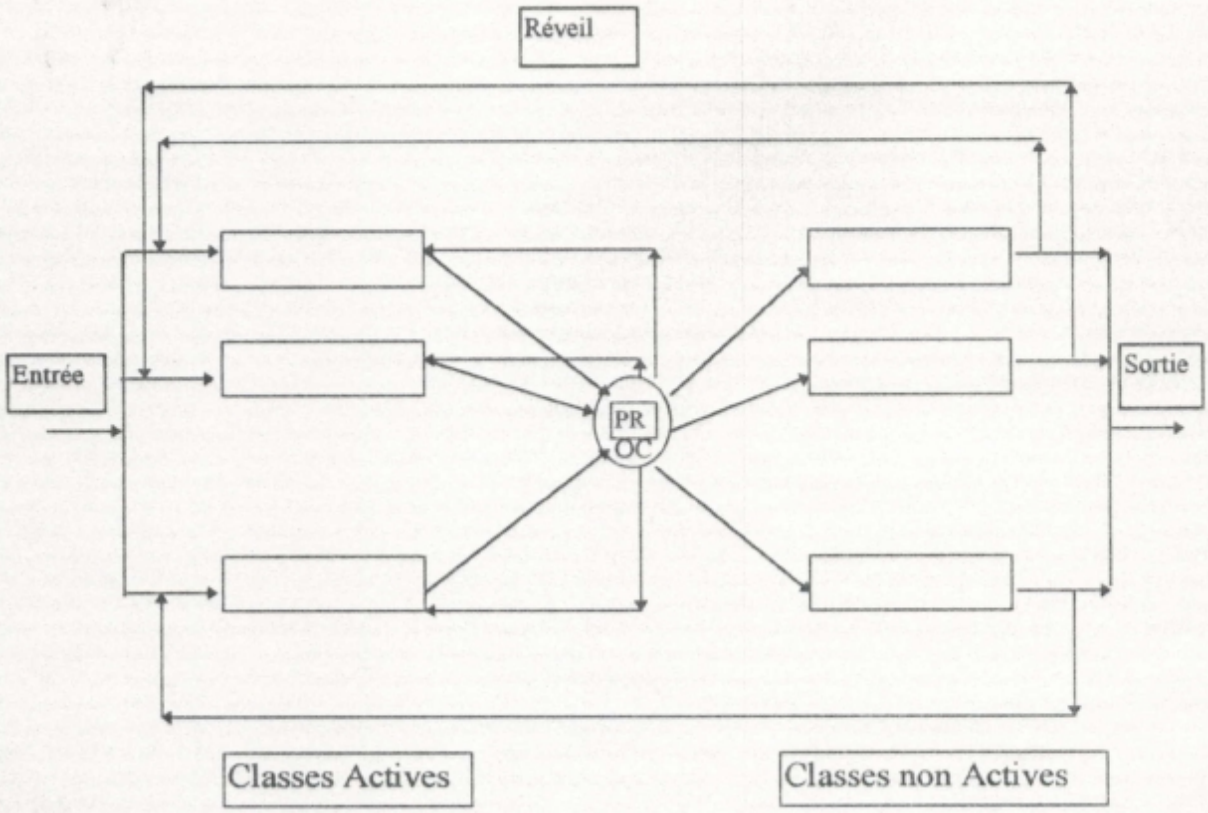
Dans cette section nous nous intéressons à un modèle plus général et de plus vaste utilisation. Ce modèle ne considère plus des processus de durée de vie infinie, mais ses processus sont dotés d'une vie limitée dans le temps; sa similitude avec le modèle précédant se situe dans le fait que le nombre de processus arrivant et sortant du système est borné par un entier égal à L, c-a-d que le concepteur du modèle doit savoir au préalable les taches pour les quelles est destiné son système, les recense en processus à qui il assigne des priorités initiales qu'il regroupe en classes disjointes contenant chacune un nombre de processus qu'elle ne peut dépasser quelque soit t. Par analogie avec la clinique décrite plus haut, le système se comporte comme si les malades peuvent quitter la clinique s'ils se trouvent en bonne forme mais reviennent à chaque fois qu'ils nécessitent des soins, ils appartiendront toujours à la même classe dotés d'une priorité initiale située dans l'intervalle de priorités de la classe. Nous pouvons voir vu la nature du système que l'état bloqué peut être confondu avec l'état sortie et l'état Début à celui d'entrée. Le diagramme de transition se voit alors décrire de la manière suivante:

V.12.1 DIAGRAMME DE TRANSITION DES ETATS



V.12.2 MODELE DE FILE D'ATTENTE

Le modèle en file d'attente peut alors être représenté comme suit:



Pour ce modèle, nous considérons un système à N classes, numérotées de 1 à N par ordre décroissant de priorités contenant chacune un nombre fixe de processus qui à tout moment peuvent être oui ou non présents dans le système. Adoptons une autre démarche pour étudier ce modèle. Nous supposons non posée la deuxième conjecture, car en fait, nous avons vu que la première n'a pas lieu d'être si la deuxième est levée.

Nous essayons de calculer en tenant compte de la nature du système, la quantité q_i (t) "probabilité que le processus d'une certaine classe E de rang j détienne le processeur à l'instant t ".

Cette approche propose de calculer q_i (t) par la méthode des événements en exhibant une équation différentielle d'inconnue q_i (t) . Soit donc l'événement "le processus i de la classe E de rang j perd le processeur au temps t+dt ".

La probabilité de cet événement n'est rien d'autre que $1-q_i\ (t+dt)$ .Cet événement se réalise si l'un des événements exhaustifs suivants se réalise.

A " Au moins une arrivée d'un processus d'une classe de priorité supérieure(de rang inférieur) survient durant la période [t, t + dt]" .

B " Dépassement de priorité d'un processus de la même classe E au temps t+dt ".

C "Au moins une arrivée de la même classe de priorité initiale supérieure à $p_i(t+dt)$ survient durant la période [t, t + dt] ".

D " fin de service ou blocage durant la période [t, t + dt] ".

L'événement C exprime le fait que durant la période [t, t + dt]une arrivée extérieure d'un processus i' appartenant à la même classe E a une priorité initiale $p_{i'}(0) > p_i(t+dt)$.

On a: $1-q_i\ (t+dt)$= proba (que le processus i détienne le processeur au temps t)*proba (A+B+C+D+les intersections possibles de ces événements)+ prob(le processus i n'avait pas le processeur au temps t).

L'événement "intersections possibles" est d'un ordre supérieur à dt, donc négligeable devant celui ci.

La seule supposition que nous ferons est sur la nature du système des arrivées des processus que nous supposerons Poissoniènnes et ceci est justifiable vu que l'interaction homme-machine est un phénomène naturel. Explicitons les probabilités de ces événements:

Proba (A) = $\sum_{k=1}^{j-1} \lambda_k dt$ « la somme de toutes les probabilités d'arrivées d'un processus d'une classe de rang inférieur à j »

$$Proba(B) = \sum_{k=1}^{n_j(t)-1} P\left(p_i(t+dt) < p_k(t+dt) \Big/ p_i(t) > p_k(t) \right)$$

= somme des proba des priorites depassant notre processus au temps $t+dt$ sachant que celui ci dé tenait le processeur au temps t.

$$= \sum_{k=1}^{n_j(t)-1} P\left(\frac{Mt_i^a(t+dt) - mt_i(t+dt)}{t} < \frac{Mt_k^a(t+dt) - mt_k(t+dt)}{t} \Big/ p_k(t) < p_i(t) \right)$$

$$= \sum_{k=1}^{n_j(t)-1} P\left(Mt_i^a(t) - m[t_i(t)+dt] < M[t_k^a(t)+dt] - mt_k(t) \right)$$

car k ne s'est pas execute pendant dt et i n'a pas attendu durant dt.

$$= \sum_{k=1}^{n_j(t)-1} P\left(p_i(t) - p_k(t) < (M-m)\frac{dt}{t} \right)$$

$$Proba(C) = \sum_{i=1}^{N(j)-n_j(t)} \lambda_j \left(\frac{M - p_i(t)}{M-m} \right) dt$$

Qui représente la somme des probabilités des arrivées de la même classe pendant dt ayant une priorité initiale supérieure à $p_i(t)$ i.e. comprise entre M et $p_i(t)$.

$Proba(D) = \dfrac{b^j(t)}{1 - B^j(t)}$ dt qui signifie la probabilité qu'il y ait fin de service ou blocage du processus i de la classe j entre t et t+dt (Théorème de renouvellement).

$1-q_i(t)$ = probabilité que le processus i n'avait pas le processeur au temps t.

Ce ci donne alors l'équation différentielle suivante:

$$1 - q_i(t+dt) = q_i(t)\left(\sum_{k=1}^{j-1} \lambda_k dt + \sum_{i=1}^{N(j)-n_j(t)} \lambda_j \left(\frac{M - p_i(t)}{M-m} \right) dt + \sum_{k=1}^{n_j(t)-1} P(p_i(t) - p_k(t) \le dt \frac{M-m}{t}) + \frac{b^j(t)}{1 - B^j(t)} dt \right) + (1 - q_i(t)).$$

$n_j(t)$ "nombre aléatoire de processus présents dans le système au temps t et qui appartiennent à la classe E d'ordre j ".

N(j) "nombre fixe de processus initialement présent dans la classe d'ordre j "

Le problème réside dans le calcul de la loi de probabilité $P(p_i(t)-p_k(t)<=dt*(M-m)/t)$.

La démarche se fait comme suit:

Par définition:

$$p_i(t) = \frac{M * t_i^a(t) + m * t_i(t)}{t}$$

La loi $P(p_i(t) - p_k(t) \le x)$ est un produit de convolution negatif.

$$Q = \int_m^M P(p_i(t) \le x + y \,/\, p_k(t) = y)\ P(y \le p_k(t) \le y + dy)$$

$$= \int_m^M H_i(x+y)\ d\,H_i(y)$$

On remplace x par $\frac{M-m}{t}$ dt

Il reste maintenant à calculer $H_i(t)$ qui représente la probabilité $P(p_i(t) \le x)$, or celle ci fait intervenir deux variables aléatoires $t_i^a(t)$ et t_i (t) de fonction de répartition respectives $A_i(t)$ et $B_i(t)$.

La loi $B_i(t)$ est connue en faisant l'analogie avec les processus de renouvellement:

THEOREME [17]

Soit $R_t(z)=P(\gamma(t)> z)$, où $\gamma(t)$ est la durée de vie résiduelle. Alors :

La fonction $R_t(z)$ est la solution de l'équation intégrale .

$$\hat{R}_t(z) = R(t+z) + \int_0^t R(t+z-x)\,h(x)\,dx.$$

R(t) = probabilité qu'il n'y a aucune interruption à la date t.

h(t) = probabilité inconditionnelle d'occurrence d'une interruption au cours de [t, t+dt].

Cette fonction ne peut être obtenue explicitement dans les cas généraux. En pratique, on a beaucoup plus recours au théorèmes limites.

Par analogie avec notre système $B_i(t) = R_t(t)$ qui représentera la probabilité qu'il n'y ait pas d'interruption ni de fin de service jusqu'au temps t.

Mais comme nous avons besoin des fonctions de répartition de $M^* t_i^a(t)$et $m^* t_i$ (t), on fera intervenir les règles de composition des variables aléatoires à savoir:

$Y = H(x)$ alors $f_y(y) = f_X(H^{-1}(y))| H'^{-1}(y) |$.

Dans notre cas $Y = MX$ cas linéaire on obtient donc:

$a_i'(t) = 1/M \; a_i(t/M)$

$b_i'(t) = 1/m \; b_i(t/m)$

$a_i'(t)$ et $a_i(t)$ sont les densités respectives de $A_i(t)$ et $A'_i(t)$.

La loi de $p_i(t)$ est le produit de convolution

$$h_i(t) = \int_0^{\infty} a_i'(t-x)\; b_i'(x)\, dx$$ qui représente la densité de $H_i(t)$.

Il nous reste à déterminer la loi du nombre de processus présents dans le système, et nous aboutissons à une équation différentielle d'inconnue $q_i(t)$ de la forme :

$$q_i(t+dt) - q_i(t) = -q_i(t)\, f(t)\, dt$$

$$\frac{dq_i(t)}{q_i(t)} = -f(t)\, dt$$

Au lieu de résoudre cette équation, nous pourrons injecter la probabilité $q_i(t)$ proposée par Haro-Proust et déduire des conditions sur $t_i^a(t)$ et $t_i(t)$ de telle sorte que cette équation soit satisfaite.

V.12.3 CONDITION DE VALIDITE DE LA QUANTITE $q_i(t)$ DE Haro-Proust.

D'après l'équation différentielle établie dans la section précédente, nous pouvons écrire en injectant $q_i(t)$ proposée intuitivement par Haro-Proust [1], et sachant que

$t_i^a(t) + t_i(t) = t$: $q_i'(t) = -f(t)\, q_i(t)$ [f(t) est l'expression décrite dans la section précédente, donc supposée connue].

Nous pouvons écrire:

$$\frac{\rho}{P}\frac{dp_i(t)}{dt} = -\frac{\rho}{P} p_i(t)\, f(t)$$

D'où :

$$\left[\frac{M\,t_i^a(t)+m\,t_i(t)}{t}\right]' = -f(t)\left[\frac{M\,t_i^a(t)+m\,t_i(t)}{t}\right]$$

$$t\left[M\frac{dt_i^a(t)}{dt}+m\frac{dt_i(t)}{dt}\right]-\left[M\,t_i^a(t)+m\,t_i(t)\right]=-t\ f(t)\left[M\,t_i^a(t)+m\,t_i(t)\right]$$

Mais $\frac{dt_i^a(t)}{dt}+\frac{dt_i(t)}{dt}=1$ Donc:

$$\left[(m-M)\frac{dt_i(t)}{dt}+M\right]=\left(M\,t+(m-M)\,t_i(t)\right)\left[\frac{1-t\,f(t)}{t}\right]$$

Pour pouvoir utiliser les résultats de O.P.B.M pour ce système ouvert, en particulier la propriété concernant son équité, il faut imposer non seulement la discipline .O.P.B.M comme politique de service mais de plus il faudrait que le temps de service obéisse à l'équation différentielle écrite ci-dessus. De ce fait la deuxième conjecture se trouve levée, et l' O.P.B.M modifiée devient théoriquement fondée.

V.13. CONCLUSION

Dans ce chapitre, nous avons mis en évidence la nouvelle stratégie d'ordonnancement O.P.B.M accompagnée de ses deux conjectures.

Nous avons montré qu'en réalité la première conjecture se trouve levée si la deuxième est acceptée. Nous avons apporté des précisions sur la contradiction citée en [3] ainsi que sur l'implémentation de cette stratégie.

Une analogie avec les processus de renouvellement vient conforter l'étude d'O.P.B.M; une analogie similaire est mise en évidence dans un rapport de Rouillon [3] que nous n'avons pu consulter que récemment. Après quoi une modélisation en file d'attente à système fermé à été formulée qui s'est vu suivre par une simulation. Divers expériences ont été menées pour confirmer la validité de cette stratégie, en particulier son équité asymptotique.

Enfin, nous avons généralisé l'utilisation d'O.P.B.M à un système ouvert, en adjoignant une condition que doivent vérifier les temps d'exécution des processus pour pouvoir utiliser les propriétés de cette stratégie.

CONCLUSION

CONCLUSION GENERALE.

Ce travail a présenté l'évolution des systèmes d'exploitation, leurs principales fonctions ainsi que leurs mécanismes de base. Une synthèse sur les systèmes à file d'attente à un serveur a été élaborée, principalement, sur les systèmes avec priorités constantes, dynamiques et à temps partagé.

Ceci a trouvé une suite logique en présentant l'originale stratégie O.P.B.M appliquée à un système multiprogrammé monoprocesseur agissant sur des processus de durée de vie illimitée. Cette stratégie, initialement basée sur deux conjectures s'est révélée ne dépendre que d'une seule. Un modèle de file d'attente étudié par une simulation de la stratégie et divers exemples numériques viennent confirmer les résultats issus de la conjecture.

Quelques remarques et précisions sont apportées clarifiant des contradictions ou des ambiguïtés observées dans les travaux originaux sur cet ordonnancement.

Une direction d'étude portant sur la généralisation de l'utilisation de O.P.B.M à un système ouvert à processus de durée de vie finie a été proposée, menant à une équation différentielle gérant le système et conditionnant les temps d'exécution des processus vient clore le travail.

Néanmoins quelques questions restent à préciser, entre autres, quelles conditions doivent satisfaire les paramètres des lois générant les blocages (sorties) et réveils (entrées) des processus pour que le phénomène de privation ne soit pas observé; et comment doit on choisir les bornes des classes pour maîtriser les taux d'utilisation du processeur relatifs à celles ci.

Enfin, nous espérons par cette étude avoir apporté une pierre à l'édifice de cette stratégie, et d'avoir enrichi par la même occasion ce vaste domaine de l'ordonnancement dans les systèmes d'exploitation.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

1[C.HARO,C.PROUST 92] *Un Ordonnancement Equitable Par Priorités Bornées*, Laboratoire d'Informatique, E3I,Université De Tours, Rapport Interne N° 119, France1992.

2[BRINCH-HANSEN 71] *Short-Time Scheduling In Multiprograming*:Third A.C.M Symposium On Operating Systems, Standford University, pp 103-105, 1971.

3[A.ROUILLON 94] *Etude du comportement de O.P.B.M* Laboratoire d'Informatique, E3I,Université De Tours, Rapport Interne N°158, France, Septembre 94.

4[A.M. LISTER 85] *Principes Fondamentaux Des Systèmes d'Exploitation*, Eyrolles, Paris1985

5[S.KROKOWIAK 87] *Principes Des Systèmes d'Exploitation Des Ordinateurs*, Dunod, Paris 1987.

6[J.PETERSON,A.SILBERSHATZ 83] *Operating Systems Concepts*, Addison-Wesley,1983.

7[A.BELKHIR 94] *Système d'Exploitation: Mécanismes De Base*, O-P-U, Alger 1994.

8[G.PUJOLLE,S-FDIDA 89] *Modèles De Systèmes Et De Réseaux*, T_1 ,T_2 , EYROLLES, Paris 1989.

9[L.KLEINROCK 75] *Queuing Systems: Theory*, Volume I,John Wiley and Sons, 1975.

10[L.KLEINROCK 76] *Queueing systems : computer Applications*, Volume II. John Wiley and Sons, 1976.

11[JM.HELARY,R.PEDRONO 83] *Recherche Opérationnelle,Travaux Dirigés*,Herman-Collection, Paris 1983.

12[RAGUNATHAN RAJKUMAR 91] *Synchronisation In Real Time Systems, A Priority Inheritance Approach*,Kluwer Academic Publischer,BOSTON,1991.

13[C.HARO 91] *Un Exécutif Multitâches Pour Un Cours Sur Les Systèmes d'Exploitation*, Laboratoire d'informatique, E3I UFR Sciences et Techniques, Université de Tours, Rapport interne N°115 , 1991.

14[L.TAKACS 1963] *Priority Queues*

15[L.TAKACS 1964] *Delay Distributions For One Line Whith Poisson Input General Holding Times And Various Orders Of Services*,Bell Systems, Technical Journal,N°42,pp487-503, 1964.

16[L.TAKACS 62] *The Stochastic Low Of Busy Period For A Single Server Queue With Poisson Input*, Journal Math Anal And Appl, N°6, pp33-42, 1962.

17[A.AISSANI 92] *Modèles Stochastiques De La Théorie De Fiabilité*, O.P.U, Alger,1992.

18[A.AISSANI 95] *Préouvrage Sur Les files d'Attentes*, Chapitre 7 Blida, 1995.

19[M.PATEROK,M.ETTL 93] *Sojourn Time And Waniting Time Distribution For M/GI/1 Queues With Preemption-Distance Priorities*, Operations Research vol 42, N° 6, 1993.

20[E.G.COFFMAN,JR]*An Analysis Of Computer Operation Under Running Time Priority Disciplines*. Interactive Systems For Experimental, Applied Mathematics, Princeton,pp 257-266,New Jersy.

21[P.GORDON 65] *Théorie Des Chaînes De Markov Finies Et Ses Applications*,Dunod, Paris 1965.

22[C.H.JORDON 47] *Calculs Of Finite Differences*, Chesley,New York,1947.

23[R.G.MILLER 60] *Priority Queues*, Anna Math stat, N°31, pp86-103, 1960.

24[A.COBHAM 54] *Priority Assignement In Waiting-Line Problems*, Operations Research, N°2, pp70-76, 1954.

25[T.E PHIPPS 56] *Machine Repair As A Priority Waiting-Line problems*, Operations Research,N°4,pp76-86, 1956.

26[B. LEMAIRE 78] *Une Démonstration Directe De La Formule De Pollaczek-Khintchin*,Rairo, Vol 12,N°2, pp 229-231, 1978.

27[R. SHASSBERGER 75] *A Note On optimal Service Selection In A Single Server Queue*, Management Science, Vol 21,N°11,1975.

28[Y. LEVY, YECHAILI 75] *Utilisation Of Idle Time In An M/G/1 Queuing System*, Management Science,vol 22,N°2, pp 22-211,1975.

29[W.WHITT 89] *Planning Single Machine Early/Tardy Problem*, Management Science, vol 35, N°11, pp 177-191, 1989.

30 [T. PRABHU 74] Stochastic storage Processus, *Queues, Insurance Risk and dams,* Springer-Verlag, New York, 1974.

31[FRANK. W. MILLER 93] *The Performance Of A Mixed Priority Real-Time-Scheduling Algorithm*, Operating Systems Review,Vol 27, N°1 pp 5-13 , 1993.

32[JACK.E.SHEMER 67] *Some Mathematical Considerations Of Time-Sharing Scheduling Algorithms*,Journal Of Association For Computing Machinery, Vol 14,N°2,pp 262-272, 1967.

33 [J. Beauquier. B. Berard 90] *Système d'éxploitation: Concepts et Alogrithmes,* MG-Hill , Paris 1990

34 [W.H.PRESS AND CO 86] *Numerical Recipes, The Art Of Scientific Computing,* Cambridge University Press, 1986.

35 [B. LE CUN. B.MANS. C.RONCAIROL 91] *Operations Concurrentes et File de Priorité,* Rapport de Recherche n° 1548, INRIA, France, Novembre 1991.

36 [J.BLAZEWICZ 94] *Scheduling in computer and manufacturing systems*, Springer-Verlag 1994.

37 [S.HILLIER.G.LIEBERMAN 75] *Operation Research, Standford University,* 1975.

38 [A.AISSANI 95] *Cours De Simulation Pour L'ingenieur,* Université de Blida, 1995.

39 [T. HAMDY 75] *Operation Research*, Standford Univerity, M.G. Hill, Usa 1975.

40 [F. BLANC. P. BRANDEIS 88] *Clefs pour Turbo Pascal sur P.C* P.S.I ,Paris 1988.

41 [P. GROGONO 84] *La Programmation En Pascal,* Addisson-Wesley , Paris 1984.

42 [M. TISCHER 90] *La Bible Du Turbo Pascal*, Data Becker, GmbH 1990

43 [C.Haro, C.Proust 92] *Equitable Sceduling By Bounded Priorities,* Proceedings of an International Conference MOAD 92, pp 43-50, Bejaia, Algerie 92.

44 [N.K. Jaiswal 68] *Priority Queues*, Academic Press, New York and London, 1968.

45[L. Zerguini, A. Aissani,C. Proust 96] *Ordonnancement stochastique optimal par classes de priorités bornées*, Rencontres de recherche opérationnelle , 06-08 octobre 1996, Alger.

46 [L. Zerguini 96] *Ordonnancement stochastique optimal par classes de priorités bornées*, Thèse de magister en maths. Appliquées , **Univ. Blida** avril 1996.

ANNEXE

```
uses printer;

Const

 nbrclsmx=9;
 nbrprocesin=10;

 fnam_p: array[1..nbrclsmx] of
string=('cls1_p','cls2_p','cls3_p','cls4_p','cls5_p','cls6_p','cls7_p','cls8_p','cls9_p');
 fnam_r: array[1..nbrclsmx] of
string=('cls1_r','cls2_r','cls3_r','cls4_r','cls5_r','cls6_r','cls7_r','cls8_r','cls9_r');
 fnam_g: array[1..nbrclsmx] of
string=('cls1_g','cls2_g','cls3_g','cls4_g','cls5_g','cls6_g','cls7_g','cls8_g','cls9_g');

 Type
  Matrice1 =array [1..nbrclsmx,1..nbrprocesin] of Real;
  Matrice2= array [1..nbrclsmx,1..nbrprocesin] of Integer;
  Vect1   = array [1..nbrclsmx] of Integer;
  Vect2   = array [1..nbrclsmx] of Real;

  files   = array [1..nbrclsmx] of text;

Var
f_p,f_r,f_g:files;
f_i: text;
pas,iteration:longint;
i,j,k:integer;
Clasactif,clas,nbrclas,loi:Byte ;
                rando1,rando2,Rot,Sup: Real;
        kerl,nbrproces,nbrprocesmax: vect1;
Lambda,mu,borsu,borin,Delta,avantent,maintent,avantsor, maintsor,Roc,
 beta,muerl,muweib,muray       :Vect2;
                  U,P,Ro: Matrice1;
                                                Indica: Matrice2;

Procedure Initialisation;
Var
I,J:integer;
 Begin
        writeln('faites rentrer le nombre de classes');Readln(nbrclas);
   writeln('faites rentrer le pas de simulation');Readln(pas);
   writeln('choisissez la loi qui correspond … votre choix');
        writeln(' 1:Exponnentielle:  2:Erlang:  3:Weibull: 4:Rayleigh ');readln(loi);

   {**************ouverture des fichiers**********************}
   Assign(F_i,'itration');  { Standard output }
        Rewrite(F_i);
```

```
for i:=1 to nbrclas do
                begin
                        Assign(F_r[i],fnam_r[i]);  { Standard output }
                        Rewrite(F_r[i]);
                        Assign(F_p[i],fnam_p[i]);  { Standard output }
                        Rewrite(F_p[i]);
                        Assign(F_g[i],fnam_g[i]);  { Standard output }
                        Rewrite(F_g[i]);
        end;

  {**************ouverture des fichiers************************}

  For I:=1 to nbrclas Do
  Begin
   writeln('Faites rentrer le nombre de processus de la classe ',i);Readln(nbrprocesmax[i]);
   writeln('faites rentrer la borne superieure de la classe ',i);Readln(borsu[i]);
   writeln('faites rentrer la borne inferieure de la classe ',i);Readln(borin[i]);
   Delta[i]:= borsu[i]-borin[i];
   nbrproces[i]:=nbrprocesmax[i];
   writeln('faites rentrer le Taux d" entr,e ... la classe ',i);Readln(lambda[i]);
   case loi of
        1:      begin
                                writeln('LOI EXPONNENTIELLE:faites rentrer le Taux de
sortie de la classe ',i);
                                Readln(mu[i]);
                end;
        2:      begin
                                writeln('LOI D"ERLANG :faites rentrer le Taux de sortie de la
classe',i);Readln(muerl[i]);
                writeln('faites rentrer la valeur de k d"erlang');readln(kerl[i]);
        end;
        3:      begin
                writeln('LOI DE WEIBULL:faites rentrer le Taux de sortie de la classe
',i);Readln(muweib[i]);
                writeln('Faites rentrer la valeur beta d"erlang');readln(beta[i]);
        end;
        4:      begin
                writeln('LOI DE RAYLEIGH:faites rentrer le Taux de sortie de la classe
',i);Readln(muray[i]);
        end;
END;
  End;
   For I:= 1 to nbrclas Do
    For J:= 1 to nbrprocesmax[I] Do
     Begin
        Writeln ('P[',i,',',J,']=');Readln(P[I,J]);
        U[I,J]:=0;
        Indica[i,j]:= 1;
      End;

   End;
```

```
Function SupIndice:Integer;
        Var
        I,ll : Integer;
        Sup  :  Real;
        begin
        Sup := P[clasactif,nbrprocesmax[clasactif]];
       ll:= nbrprocesmax[clasactif];
      For I:= nbrprocesmax[clasactif] downto 1 Do
                If ( P[clasactif,I] > sup ) and ( indica[clasactif,I] <> 0 )
                                        Then
                begin
                        sup:=  P[clasactif,I];
                                ll := I;
                               end;
      supindice:=ll;

   end;

PROCEDURE Calcul;


{******************** Procedure calcactif ******************}

Procedure calcactif;
var
         l:integer;
         Begin
         l:= Supindice;
         For I:= 1 to nbrprocesmax[clasactif] Do
       begin
                  If I= l  Then
             begin
              U[clasactif,I]:=U[clasactif,I]+1;
              P[clasactif,I]:= borsu[clasactif]-(delta[clasactif]*U[clasactif,I]/(iteration-1));
              end
                    else
                    begin
                            U[clasactif,I]:=U[clasactif,I];
                            P[clasactif,I]:= borsu[clasactif]-(delta[clasactif]*U[clasactif,I]/(iteration-
 1));
                     end;
        end;
          end;
 {******************** Procedure calcactif ******************}

 {******************** Procedure calc_non_actif ******************}
```

```
Procedure Calcnonactif;
  Begin
      For I:= 1 to nbrprocesmax[clas] Do
      Begin
            U[clas,I]:=  U[clas,I];
            P[clas,I]:=borsu[clas]-(delta[clas]*U[clas,I]/(iteration-1));
      end;
      end;

Begin {procedure Calc}
      For clas:= 1 to nbrclas Do
      If clas= clasactif
      then  Calcactif
      else  Calcnonactif;
end;

Function choix_class:integer;
var
      l:integer;
      Begin
      l:= 1;
    While (nbrproces[l]<= 0)and (l<=nbrclas) do
            l:=  l+1;
    choix_class:=l;
  end;

Procedure Entree ;
var classe : Integer;
      Begin
            For classe:= 1 to nbrclas Do
            Begin
            Avantent[classe]:= (-1/Lambda[classe])*Ln(Random);
            Maintent[classe]:= (-1/Lambda[classe])*Ln(Random);
                        If ( Maintent[classe]-Avantent[classe]<= Lambda[classe])  and
(nbrproces[classe]<nbrprocesmax[classe])
            then
                  Begin
                        K:=1;
                        While (k<=nbrprocesmax[classe])and(Indica[classe,k]=1) do
                  begin
                              k:= k+1;
                  end;
                        if  k<= nbrprocesmax[classe]
                              then
                                    Begin
                                          Indica[classe,k]:=1;
                                          nbrproces[classe]:= nbrproces[classe]+1;
                                    End;
                  End;
            End;
      End;
```

```
Procedure Sortie;
var
ind,bb:integer;
        Begin
     ind:= supindice;

     case loi of
      1:begin
        Maintsor[clasactif]:= (-1/mu[clasactif])* Ln(Random);
        Avantsor[clasactif]:= (-1/mu[clasactif])* Ln(Random);
        if (Maintsor[clasactif]-Avantsor[clasactif]<= mu[clasactif])
       Then
              Begin
                        nbrproces[clasactif]:=nbrproces[clasactif]-1;
                   Indica[clasactif,ind]:=0;
              End;
       end;

      2:begin
        rando1:=random;
        rando2:=random;
         for bb:= 1 to kerl[clasactif]-1 do
          begin
           rando1:=rando1*random;
            rando2:=rando2*random
          end;
        Maintsor[clasactif]:= (-1/muerl[clasactif])* Ln(Rando1);
        Avantsor[clasactif]:= (-1/muerl[clasactif])* Ln(Rando2);
         if (Maintsor[clasactif]-Avantsor[clasactif]<= muerl[clasactif])
          Then
               Begin
                        nbrproces[clasactif]:=nbrproces[clasactif]-1;
                   Indica[clasactif,ind]:=0;
               End;
       end;

     3:begin
      Maintsor[clasactif]:= (exp(1/beta[clasactif]*Ln(-1/muweib[clasactif]* Ln(Random))));
       Avantsor[clasactif]:=(exp(1/beta[clasactif]*Ln(-1/muweib[clasactif]* Ln(Random))));
       if (Maintsor[clasactif]-Avantsor[clasactif]<= muweib[clasactif])
       Then
              Begin
                        nbrproces[clasactif]:=nbrproces[clasactif]-1;
                   Indica[clasactif,ind]:=0;
              End;
      end;
     4:begin
        Maintsor[clasactif]:= sqrt((-2/muray[clasactif])* Ln(Random));
        Avantsor[clasactif]:= sqrt((-2/muray[clasactif])* Ln(Random));
        if (Maintsor[clasactif]-Avantsor[clasactif]<= mu[clasactif])
```

```
        Then
                Begin
                                nbrproces[clasactif]:=nbrproces[clasactif]-1;
                        Indica[clasactif,ind]:=0;
                End;
        end;
      end;
   End;
Procedure procesnul;
 var
  f,d: Integer;
  Begin
    For f:= 1 to nbrclas do
      For d:= 1 to nbrprocesmax[f] do
       Begin
        U[f,d]:= U[f,d];
        P[f,d]:=Borsu[f]-(Delta[f] *  U[f,d]/(iteration-1));
       end;
  end;
Procedure RESULTAT;
Begin
        if (iteration<501) or (iteration mod 5 =0) then
                Writeln(f_i,iteration);
        writeln('**********************************************************');
    writeln('**********************************************************');
    if iteration=2 then
        writeln('iteration',iteration-1)
    else
        writeln('iteration',iteration);
    for j:= 1 to nbrclas do
        begin

        writeln('**********************************************************');
                Roc[j]:=0;
                for k:= 1 to nbrprocesmax [j] do
                Begin
                        Ro[j,k]:=U[j,k]/(iteration-1);
                        Roc[j]:= Roc[j]+Ro[j,k];
                        writeln ('P[',j,',',k,']=',P[j,k]:5:7,' **   Ro[',j,',',k,']=',Ro[j,k]:5:7);
               if  (iteration<501) or (iteration mod 5 =0) then
                 begin
                        Write(f_p[j],p[j,k],'   ');
                        Write(f_r[j],Ro[j,k],'   ');
                 end;
                end;
                if (iteration<501) or (iteration mod 5 =0) then
                begin
                        Writeln(f_g[j],Roc[j]);
                        Writeln(f_p[j]);
                        Writeln(f_r[j]);
            end;
```

```
        writeln('                                RO[',j,']=',Roc[j]:2:7);
        end;
End;

Procedure PRINT;
Begin
   writeln(lst,'*************************************************************');
   writeln(lst,'*************************************************************');
   if iteration=2 then
   writeln(lst,'iteration',iteration-1)
   else
     writeln(lst,'iteration',iteration);
  for j:= 1 to nbrclas do
  begin
   writeln(lst,'*************************************************************');
    Roc[j]:=0;
     for k:= 1 to nbrprocesmax [j] do
       Begin
        writeln (lst,'P[',j,',',k,']=',P[j,k]:5:7,' **   Ro[',j,',',k,']=',U[j,k]/(iteration-1):5:7);
        Ro[j,k]:=U[j,k]/(iteration-1);
        Roc[j]:= Roc[j]+Ro[j,k];
       end;
        writeln(lst,'                                RO[',j,']=',Roc[j]:2:7);
  end;
End;
{  Programme principal }
Begin
   Initialisation;
   Iteration:= 2;
   resultat;
   Clasactif:= 1;
   Calcul;
   Resultat;
   Sortie;

   For iteration:= 3 to pas-1 do
         Begin
        Entree;
        Clasactif:=Choix_class;
        if clasactif<= nbrclas
              then
              begin
                    calcul;
                    sortie;
              end
              else
        Procesnul;
        Resultat;
        { if(iteration mod 100 =0) utilisez lors des impressions
                      then print;}
```

```
        if iteration = pas then
         BEGIN

writeln('*************************************************************');
          writeln(  'TAPER ENTREE POUR AVOIR DES INFORMATIONS SUR LE
PROCESSEUR');

writeln('*************************************************************');
         end;
       end; {for}

         readln;
         Rot:=0;
                for i:=1 to nbrclas do
                rot:= rot+roc[i];
         writeln('le taux d"occupation du processeur est de ',rot:2:7);
         writeln('*************************************************************');
         writeln('le taux d"inoccupation du processeur est de ',1-rot:2:7);
       readln;
       close(F_i);
         for i:=1 to nbrclas do
                        begin
                                close(F_r[i]);
                                close(F_p[i]);
                                close(F_g[i]);

          end;

 end.
```