Chapitre 2 **SCHEDULING**

1. ASPECTS GENERAUX DES PROCESSUS

Un processus est un programme qui s'exécute, ainsi que ses données, sa pile, son compteur ordinal, son pointeur de pile et les autres contenus de registres nécessaires à son exécution. Un processus fournit l'image de l'état d'avancement de l'exécution d'un programme.

1.1 Simultanéité, ressources

On appelle **simultanéité** l'activation de plusieurs processus au même moment.

Si le nombre de processeurs est au moins égal au nombre de processus, on parle de simultanéité totale ou vraie, sinon de pseudo-simultanéité.

pseudo-simultanéité : La simultanéité est obtenue par commutation temporelle d'un processus à l'autre sur le processeur. Si les basculements sont suffisamment fréquents, l'utilisateur a l'illusion d'une simultanéité totale.

Avec un SE qui gère le temps partagé (pseudo-parallélisme par commutation de temps), conceptuellement chaque processus dispose de son propre processeur virtuel. Ainsi, concrètement il n'existe qu'un seul compteur ordinal dont le contenu est renouvelé à chaque commutation. Conceptuellement, tout se passe comme si chaque processus disposait de son propre compteur ordinal.

De façon simplifiée, on peut imaginer un SE dans lequel les processus pourraient être dans trois états:

- **élu** : en cours d'exécution. Un processus élu peut être arrêté, même s'il peut poursuivre son exécution, si le SE décide d'allouer le processeur à un autre processus

- **bloqué** : il attend un événement extérieur pour pouvoir continuer (par exemple une ressource; lorsque la ressource est disponible, il passe à l'état "prêt")

- **prêt** : suspendu provisoirement pour permettre l'exécution d'un autre processus

1.2 Implémentation des processus

Pour mettre en œuvre le modèle des processus, le SE gère une **table des processus** dont chaque entrée correspond à un processus. Chaque ligne peut comporter des informations sur un processus : son état, son compteur ordinal, son pointeur de pile, son allocation mémoire, l'état de ses fichiers ouverts, et d'autres paramètres.

1.3 Mécanismes de commutation

- par interruptions
- par trap ou déroutement sur erreur : il s'agit d'une extension du mécanisme des interruptions. En cas de détection d'erreur interne au processus (ex : division par 0, erreur d'adressage), le contrôle est passé au SE en cas d'erreur mettant en cause son intégrité (ex : erreur d'adressage) ou à une fonction de traitement de l'erreur du processus courant.
- appel au superviseur (noyau du SE) : dans un SE multi-utilisateur multi-programmé, toutes les interruptions sont contrôlées par le superviseur. Ainsi, en cas d'interruption :

  - on sauve le mot d'état et le contexte du processus en cours et on passe en mode superviseur (ou noyau ou système)

- le traitement de l'interruption est réalisé soit par le superviseur lui-même (horloge, coupure), soit par un processus spécifique (pilote de périphérique d'E/S), soit par le processus interrompu (erreur interne)
- on élit un nouveau processus à exécuter (peut-être celui qui avait été interrompu)

2. ORDONNANCEMENT DES PROCESSUS

L'ordonnanceur (scheduler) définit l'ordre dans lequel les processus prêts utilisent l'UC (en acquièrent la ressource) et la durée d'utilisation, en utilisant un algorithme d'ordonnancement. Un bon algorithme d'ordonnancement doit posséder les qualités suivantes :

- équitabilité : chaque processus reçoit sa part du temps processeur
- efficacité : le processeur doit travailler à 100 % du temps
- temps de réponse : à minimiser en mode interactif
- temps d'exécution : minimiser l'attente des travaux en traitement par lots (batch)
- rendement : maximiser le nombre de travaux effectués par unité de temps

L'ensemble de ces objectifs est contradictoire, par exemple le 3ème et le 4ème objectif.

2.1 Ordonnancement circulaire ou tourniquet (round robin)

Il s'agit d'un algorithme ancien, simple et fiable. Le processeur gère une liste circulaire de processus. Chaque processus dispose d'un quantum de temps pendant lequel il est autorisé à s'exécuter. Si le processus actif se bloque ou s'achève avant la fin de son quantum, le processeur est immédiatement alloué à un autre processus. Si le quantum s'achève avant la fin du processus, le processeur est alloué au processus suivant dans la liste et le processus précédent se trouve ainsi en queue de liste.

La commutation de processus (overhead) dure un temps non nul pour la mise à jour des tables, la sauvegarde des registres. Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur. Un quantum trop grand augmente le temps de réponse en mode interactif. On utilise souvent un quantum de l'ordre de 100 ms.

Dans ce cas, on utilise une liste circulaire contenant les processus prêts. L'implantation de cet algorithme se réalise de la manière suivante: la liste des processus prêts est organisée en file (FIFO). Un nouveau processus est rattaché à la queue de la file. Le scheduler alloue le processeur au premier processus de la file des processus prêts, initialise l'horloge pour une interruption après un quantum de temps. Il en résultera une des deux situations:

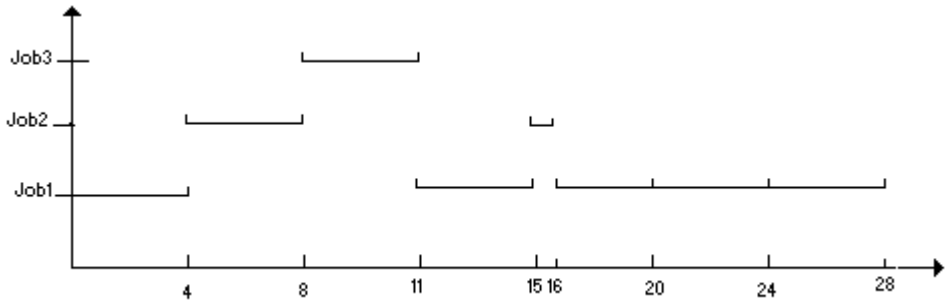
1) Le temps d'exécution est inférieur au quantum: dans ce cas le processus libère volontairement le processeur central à la suite d'une entrée/sortie ou une terminaison. Le prochain processus de la file est élu (le processeur central lui est alloué).

2) Le temps d'exécution CPU est supérieur au quantum: dans ce cas, le processus consommera son quantum de temps et par conséquent, il y aura une interruption horloge. Les registres du processeur sont sauvegardés dans le bloc de contrôle du processus et il est mis à la queue de la file. Ensuite, le scheduler sélectionne un nouveau processus.

Exemple: soient les jobs suivants:

N° Job	temps d'exécution
1	20
2	5
3	3

Si on utilise un quantum de temps égal à 4 unités de temps et un temps de commutation négligeable, alors l'algorithme de scheduling ROUND ROBBIN donnera ce qui suit:

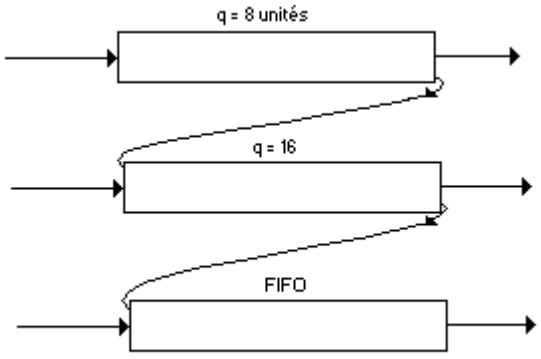


T = (11+16+28)/3 = 55/3

Remarque:
La stratégie ROUND ROBBIN dépend largement du temps de quantum. Supposons que l'on dispose d'un seul processus dans le système dont le temps d'exécution est de 10 unités et un quantum de temps évalué à 12 unités. Dans ce cas, il n'y aura aucune commutation du contexte. Par contre, si le quantum est de 6 unités, on aura une (01) commutation de contexte. Alors que si le quantum est de 1 unité ; on aura 9 commutations de contexte.
En conclusion, la stratégie est identique à la stratégie FIFO quand le quantum est très grand (infini). A l'opposé, si le quantum est très petit, Il apparaît que chaque processus possède son propre processeur. Cette stratégie est appelée "processeur partagé". Cependant, il y aura une surcharge due au changement des contextes.

2.2 La politique à plusieurs niveaux dépendants:
Dans la stratégie précédente, un processus ne change pas de file tout le long de son exécution. Maintenant, un processus peut basculer entre des files. Ceci est réalisé dans le but d'isoler les processus consommateurs de temps d'unité centrale d'une part, et de relancer les processus de faible priorité d'autre part.

Exemple: (les files d'attente rétroactives)
Cette méthode utilise N files d'attente avec des règles suivantes:
- Un processus qui entre dans le système est mis dans la première file.
- Après avoir reçu une tranche de temps, il est mis dans la deuxième file.
- Après chaque tranche reçue, il passe dans la file suivante.
- L'algorithme choisit le premier processus de la première file non vide.



Cette stratégie est définie par les paramètres suivants:
-le nombre de files.
-l'algorithme de scheduling de chaque file.
-une méthode de transition d'un processus d'une file à une autre.
-une méthode qui choisit la file à recevoir un nouveau processus.

Remarque :

Cette stratégie permet de favoriser les petits travaux, sans avoir besoin de savoir à l'avance combien de temps CPU ceux-ci vont utiliser.

Exemple :

1. L'algorithme de sheduling des processus du système UNIX est de la classe des schedulers du type ROUND ROBIN avec plusieurs niveaux dépendants. Le scheduler alloue le processeur à un processus pour un quantum de temps. Il préempte le processus qui a consommé son quantum de temps et, il le met dans l'un des niveaux de priorités (file). La priorité d'un processus est fonction de l'utilisation récente (temps) du processeur central.
   Soient pri la priorité d'un processus et, cpu l'utilisation récente (temps) du processeur central. Après un quantum de temps, de scheduler calcule pour chaque processus la nouvelle valeur cpu en utilisant l'équation cpu = cpu/2. Après cela, il recalcule pri en utilisant l'équation pri = (cpu/2 + k) ; où k est une constante connue par le système.
   Soient trois processus A, B et C dans le système ayant une priorité initiale de 60. La valeur de la constante k est 60, le quantum de temps est fixé à 1 seconde. L'horloge interrompe le processeur central 60 fois à la seconde. A chaque interruption, on incrémente de 1 la valeur de cpu. Après chaque quantum, on recalcule les valeurs de cpu et pri selon le processus décrit dans l'algorithme de scheduling.

temps	A		B		C	
	Pri	Cpu	Pri	Cpu	Pri	Cpu
0	60	0 1 2 ... 60	60	0	60	0
1	75	30	60	0 1 2 ... 60	60	0
2	67	15	75	30	60	0 1 2 ... 60
3	63	7 8 9 ... 67	67	15	75	30
4	76	33	63	7 8 9 ... 67	67	15
5	68	16	76	33	63	7 8 9 ... 67

Remarque :
On remarque que l'algorithme possède les propriétés suivantes :

- Nombre de files fixe (la plus grande priorité-la plus basse)
- Priorité dynamique convergente
- Equité entre processus

L'algorithme utilise une priorité dynamique à un seuil limité, ce qui permet de conclure que le nombre de files est fixe. Pour trouver la valeur maximale de priorité, il suffit de considérer un seul processus prêt et dérouler l'algorithme.

2. Le principe de l'ordonnanceur (scheduler) de partage équitable (UNIX version V) est de diviser la communauté des utilisateurs en un ensemble de groupes de partage équitable. Cependant, le système alloue son temps d'unité centrale proportionnellement à chaque groupe, indifféremment du nombre de processus qu'il y a dans les groupes.

Soient pri la priorité d'un processus et cpu l'utilisation récente (temps) du processeur central et gpe le groupe du processus. L'horloge interrompe le processus en exécution 60 fois pendant son quantum de temps. La procédure de traitement de l'interruption horloge incrémente de 1 la valeur cpu du processus en exécution et le champ gpe de tous les processus du même groupe. Après chaque quantum, le noyau recalcule les champs pri et gpe de chaque processus selon les formules suivantes:

cpu = cpu/nombre de groupes
gpe = gpe/nombre de groupes
pri = priorité de base + cpu/nombre de groupes + gpe/nombre de groupes

temps	A			B			C		
	Pri	Cpu	gpe	Pri	Cpu	gpe	Pri	Cpu	gpe
0	60	0 1 2 ... 60	0 1 2 ... 60	60	0	0 1 2 ... 60	60	0	0
1	90	30	30	75	0	30	60	0 1 2 ... 60	0 1 2 ... 60
2	74	15	15 16 17 ... 75	67	0 1 2 ... 60	15 16 17 ... 75	90	30	30
3	81	7	37	93	30	37	75	15 16 17 ... 75	15 16 17 ... 75
4	70	3 4 5 ... 63	18 19 20 ... 78	76	15	18 19 20 ... 78	96	37	37
5	104	31	39	82	7	39	63	18 19 20 ... 78	18 19 20 ... 78

Remarque :
L'algorithme possède les propriétés suivantes :

- Nombre de files fixe
- Priorité dynamique convergente
- Equité entre utilisateurs