**CHAPITRE 3 : L’ORDONNANCEMENT DES PROCESSUS**

1. **Introduction**

Un ordinateur possède forcément plusieurs processus en concurrence pour l’obtention du temps processeur, cette situation se produit lorsque 2 ou plusieurs processus sont en état prêt simultanément. L’Ordonnanceur (planificateur, scheduler) est la partie (un programme) du système d’exploitation responsable de régler les états des processus (Prêt, Actif,…etc.) et de gérer les transitions entre ces états ; c’est l’allocateur du processeur aux différent processus, il alloue le processeur au processus en tête de file des Prêts

1. **Objectifs d’un Ordonnanceur**

Les objectifs d’un Ordonnanceur sont :

- Maximiser l’utilisation du processeur

- Présenter un temps de réponse acceptable

- Respecter l’équité entre les processus selon le critère d’ordonnancement utilisé.

1. **Critères d’ordonnancement :**

L’objectif d’un algorithme d’ordonnancement Consiste à identifier le processus qui conduira à la meilleure performance possible du système. Certes, il s’agit là d’une évaluation subjective dans laquelle entrent en compte différents critères à l’importance relative variable. La politique d’ordonnancement détermine l’importance de chaque critère. Un certain nombre d’algorithmes ont fait leur preuve dans la mise en œuvre d’une politique d’ordonnancement. La liste qui suit passe en revue des critères d’ordonnancement fréquemment utilisés.

* **Utilisation de l’UC :** Pourcentage de temps pendant lequel l’UC exécute un processus. L’importance de ce critère varie généralement en fonction du degré de partage du système.
* **Utilisation répartie :** Pourcentage du temps pendant lequel est utilisé l’ensemble des ressources (outre l’UC, mémoire, périphérique d’E/S…)
* **Débit :** Nombre de processus pouvant être exécutés par le système sur une période de temps donnée.
* **Temps de rotation :** durée moyenne qu’il faut pour qu’un processus s’exécute. Le temps de rotation d’un processus comprend tout le temps que celui-ci passe dans le système. Il est inversement proportionnel au débit.
* **Temps d’attente :** durée moyenne qu’un processus passe à attendre. Mesurer la performance par le temps de rotation présente un inconvénient : Le temps de production du processus accroît le temps de rotation ; Le temps d’attente représente donc une mesure plus précise de la performance.
* **Temps de réponse :** Temps moyen qu’il faut au système pour commencer à répondre aux entrées de l’utilisateur.
* **Equité :** degré auquel tous les processus reçoivent une chance égale de s’exécuter.
* **Priorités :** attribue un traitement préférentiel aux processus dont le niveau de priorité est supérieur.

1. **Types d’ordonnancement :**

Il existe 2 types d’ordonnancement

**a. Ordonnancement préemptif :** Avec réquisition où l’Ordonnanceur peut interrompre un processus en cours d’exécution si un nouveau processus de priorité plus élevée est inséré dans la file des Prêts.

**b. Ordonnancement coopératif :** Ordonnancement jusqu’à achèvement : le processus élu garde le contrôle jusqu’à épuisement du temps qui lui a été alloué même si des processus plus prioritaires ont atteint la liste des Prêts

1. **Les algorithmes d’ordonnancement :**
2. **L’algorithme FIFO (First In First Out) :**

L'ordonnancement est fait dans l'ordre d'arrivée en gérant une file unique des processus sans priorité ni réquisition : chaque processus s’exécute jusqu’à son terme ; le processus élu est celui qui est en tête de liste des Prêts : le premier arrivé. Cet algorithme est facile à implanter, mais il est loin d'optimiser le temps de traitement moyen.

1. **L’algorithme SJF (Shortest Job First) :**

L'ordonnancement par ordre inverse du temps d'exécution (supposé connu à l’avance) : lorsque plusieurs travaux d'égale importance se trouvent dans une file, l'Ordonnanceur élit le plus court d'abord (les travaux les plus cours étant en tête de la file des prêts). Cet algorithme possède l’inconvénient de la nécessité de connaissance du temps de service à priori et le risque de privation des tâches les plus longues. Afin de résoudre ces problèmes on pourra attribuer aux travaux une priorité croissante avec leur temps de séjour dans la file (temps d’attente). A temps d’attente égale, le travail le plus court est prioritaire. Toutefois, cette solution nécessite le calcul des priorités périodiquement et un réarrangement de la FA.

1. **L’algorithme du temps restant le plus court (SRT : Shortest Remaining Time) :**

L’algorithme du temps restant le plus court, est la version préemptive de l’algorithme SJF. Chaque fois qu’un nouveau processus est introduit dans la file des processus à ordonnancer, l’Ordonnanceur compare la valeur estimée du temps de traitement restant à celle du processus en cours d’ordonnancement. Si le temps de traitement du nouveau processus est inférieur, le processus en cours d’ordonnancement est préempte. Tout comme l’algorithme SJF, l’algorithme SRT favorise les travaux courts : les travaux longs en revanche peuvent être victimes de famine.

1. **L’algorithme 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.

1. **L’algorithme HPF(Highest Priority First) :**
2. **Stratégie d’ordonnancement :**

Un système informatique commun peut posséder un processeur unique ou plusieurs processeurs. La gestion des processus s’occupe alors de l’allocation des processeurs aux processus créés.

Une stratégie d’ordonnancement est implémenté réellement moyennant des algorithmes dites algorithmes d’ordonnancement.

L’objectif d’un algorithme d’ordonnancement est d’optimiser une des grandes temporelles suivantes, sachant que l’on intéresse par ailleurs au taux de l’utilisation de l’unité centrale et au débit moyen :

* Le temps de traitement moyen décrit la moyenne des intervalles de temps séparant la soumission et l’accomplissement d’un processus.
* Le temps d’attente moyen est le temps est la moyenne des durées de temps que les processus ont passé dans la file d’attente des processus à l’état prêt.
  + - 1. **Algorithme d’ordonnancement sans réquisition :**
         1. **Ordonnancement FCFS (first come first served)**

On l’appelle aussi FIFO (pour first in first out) ou premier arrivé premier servi. Cet algorithme traite les processus selon l’ordre de leur soumission (date arrivée) sans prendre compte de leur temps d’exécution. L’organisation de la file d’attente des processus prêt est donc tout simplement FIFO.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | … | P4 | P3 | P2 | P1 |  |

Terminé  
 File d’attente « prêt »

* + - * 1. **Ordonnancement SJF (shortest job first)**

SJF, ou la plus courte durée d’exécution, choisit les processus ayant le plus court temps d’exécution sans réellement tenir compte de leur date d’arrivée. Cet algorithme est non implémentable à cause de l’aspect statistique de ses calculs. Néanmoins, il est intéressant en vue d’étudier des performances de l’algorithme d’ordonnancement réellement mis en œuvre par le système

* + - * 1. **Ordonnancement avec tourniquet ou Round Robin**

On procède ici à un recyclage des processus sur le processeur tant que les processus ne sont pas terminés.

Lorsque un processus est élu on lui attribue une tranche de temps fixe appelé **quantum** laquelle il s’exécute Au bout de ce temps on ne poursuit plus l’exécution du processus on lui retire donc le processeur et on le réinsère dans la file des processus prêt. Là il devra attendre sa prochaine élection.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | … | P4 | P3 | P2 | P1 |  |

Terminé

File d’attente « prêt »

Si le quantum n’est pas épuisé, alors retour à la file d’attente

* + - 1. **Application**

Soit la liste des processus suivante :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Processus | P1 | P2 | P3 | P4 |
| Date d’arrivée | 0 | 1 | 2 | 3 |
| Durée d’exécution | 5 | 4 | 2 | 2 |

Pour chaque stratégie d’ordonnancement : FCFS, SJF, Round Robin déterminer les grandeurs suivantes : le temps de traitement moyen TTM et le temps d’attente moyen TAM ,Diagramme de GANTT.

1. **Les processus sous unix**
   * + 1. **Contexte d'un processus :**

On distingue des **processus utilisateurs** et des **processus système**. Ces derniers :

* ne sont sous le contrôle d'aucun terminal
* ont comme propriétaire le super utilisateur (processus démons). Ils restent résidents en MC en attente d'une requête
* ils assurent des services généraux accessibles à tous les utilisateurs
* ils peuvent être créés au lancement du système ou à des dates fixées par l'administrateur.

Un processus peut s'exécuter dans deux modes différents :

* mode utilisateur (user mode) : le processus n'accède qu'à son espace d'adressage et n'exécute que des instructions ordinaires du programme chargé.
* mode noyau ou système (kernel mode) : le processus exécute des instructions n'appartenant pas au programme, mais au noyau. Il accède à des données externes à son espace d'adressage. On peut passer en mode noyau soit par une interruption appropriée, soit par la réalisation d'un appel système par le processus lui-même (exemple : requête de lecture au clavier dans le programme)
  + - 1. **Structure d'un processus sous unix**

Le noyau gère une table des processus, ***listable*** par la commande **ps** avec une entrée par processus, et contenant des informations sur celui-ci. Cette entrée est allouée à la création du processus etretirée à son achèvement.

Un processus est identifié par un **pid** (process identifier), entier de 0 à 32767 retourné par le noyau. Il est caractérisé aussi par sa priorité, son propriétaire et son groupe propriétaire, son terminal de rattachement.

* + - 1. **Ordonnancement d'un processus**

L'ordonnanceur ou scheduler (gestionnaire des ressources et d'enchaînement des processus) gère plusieurs processus concurremment et autorise le multitâche par partage de temps. Il attribue des quanta de temps. A l'expiration de son quantum de temps, le processus actif, s'il n'est pas terminé, sera obligé de relâcher le processeur. Un autre processus est alors élu et attribué au processeur. Le processus suspendu reprend son exécution quand son tour arrive à nouveau pour disposer du processeur.

Sous UNIX, l'horloge de l'ordonnanceur délivre 100 tops, ou cycles mineurs, par seconde et un quantum, ou cycle majeur, correspond à plusieurs tops (par exemple 100, soit 1 seconde).

Un processus peut prendre l'un des 9 états décrits dans /usr/include/sys/proc.h), parmi lesquels :

* actif : en mode utilisateur (pas de droit d'accès aux ressources du système) ou en mode système (ou noyau)
* activable ou prêt à exécuter : en mémoire ou en zone de swap. Il est éligible par l'ordonnanceur. Ces deux états sont codés R
* bloqué : en attente d'un événement (horloge, montage d'un disque, résultat d'un autre processus, ...), en mémoire ou en zone de swap. Il ne consomme pas de temps CPU. Il repasse à l'état activable dès que les conditions le permettent. non prêt : en création ou zombie (désigné par Z, le processus est achevé, mais son père n'en a pas encore connaissance). Il ne consomme pas de temps CPU.



Figure 7 : Cycle de vie d’un processus sous unix

* 1 : le processus créé par fork a acquis les ressources nécessaires à son exécution
* 2 : le processus vient d'être élu par l'ordonnanceur
* 3 : le processus revient d'un appel système ou d'une interruption
* 4 : le processus a réalisé un appel système ou une interruption est survenue
* 5 : le processus se met en attente d'un événement (libération de ressource, terminaison de processus par wait). Il ne consomme pas de temps UC
* 6 : l'événement attendu par le processus s'est produit
* 7 : conséquence d'un signal particulier
* 8 : réveil du processus par le signal de continuation
* 9 : le processus s'achève par exit, mais son père n'a pas pris connaissance de sa terminaison. Il ne consomme pas de temps UC et ne mobilise que la ressource table des processus

1. **Conclusion**

Ce chapitre était consacré à la gestion des processus. Après avoir été élu par l’ordonnanceur, un processus se charge en mémoire centrale pour s’exécuter. La mémoire centrale représente le lieu d’exécution des processus. Le prochain chapitre traitera en détail la gestion de la mémoire centrale.