Chapitre 7

L'ordonnancement des processus

La sélection dans le temps des processus pouvant accèder à une ressource est un problème dit d'ordonnancement. Nous présentons ici :

- le cas général
- les besoins et les problèmes

et nous décrirons des solutions que l'on trouve sous UNIX pour différents problèmes d'ordonnancement.

Les algorithmes d'ordonnancement réalisent la sélection parmi les processus actifs de celui qui va obtenir l'utilisation d'une ressource, que ce soit l'unité centrale, ou bien un périphérique d'entrée-sortie.

Pour l'unité centrale notre but est de maximiser débit et taux utile de l'unité centrale :

le débit est le nombre moyen de processus exécutés en un temps donné.

le taux utile est la proportion de temps réellement utilisée pour exécuter des processus utilisateurs.

Un exemple :
Soient 2 processus A et B de même comportement 30 périodes de deux seconde :
1 seconde d'activité
1 seconde d'inactivité

```
AIAIAIAIAIAIAIAIAIAIAIAIAIAIAIAIAIAIAI
```

Si l'on exécute les deux processus consécutivement on obtient un débit de 1 processus par minute, et un taux utile de 50%. Si l'on entrelace les périodes actives et inactives des deux processus on obtient un débit de 2 processus par minute et un taux d'utilisation de 100%.

Pour une autre ressource d'autres critères seront utilisés.

7.1 Le partage de l'unité centrale

Ce partage doit être fait non seulement entre les processus utilisateurs mais aussi entre les différentes tâches du système, scheduler, entrées-sorties, gestion des interruptions, etc.

Nous demandons de plus à l'algorithme d'ordonnancement de nous assurer **l'exclusion mutuelle** et **l'absence de famine**, qui sont les points-clefs de la plupart des problèmes d'ordonnancement. L'invention d'un algorithme d'ordonnancement se base en générale sur des remarques statistique sur le comportement des processus :

- Le couple UC/ES (cpu/io), les processus ont tendance à basculer constamment entre des phases d'entrées-sorties et des phases de calcul sur l'unité centrale.
- Les processus consommant de longues périodes d'U.C. sont proportionnellement rares.

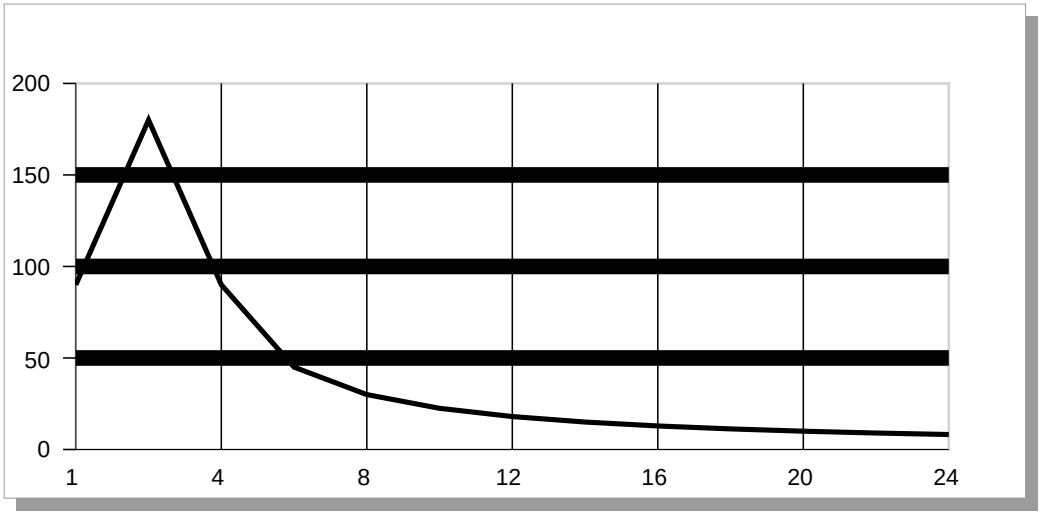

FIG. 7.1 – Histogramme de répartition de la durée de la période d'utilisation de l'unité centrale

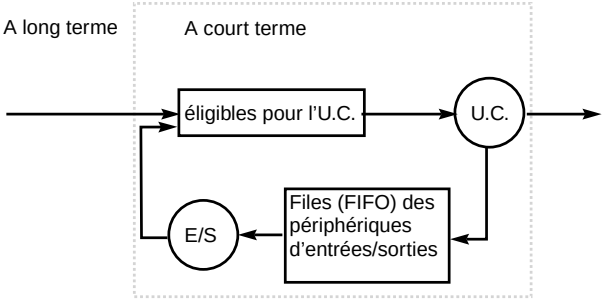

FIG. 7.2 – Stratégie globale d'ordonnancement.

7.1.1 Famine

Notre première tâche est d'affecter une ressource (l'UC par exemple) à un unique processus à la fois (exclusion mutuelle) et s'assurer de l'absence de famine.
famine : un processus peut se voir refuser l'accès à une ressource pendant un temps indéterminé, il est dit alors que le processus est en famine.

Un système qui ne crée pas de cas de famine : fournira toujours la ressource demandée par un processus, au bout d'un temps fini.

Si on prend le cas des périphériques (tels que les disques) l'ordonnancement peut se faire de façon simple avec par exemple une file d'attente (FIFO).

Pour l'unité centrale on va devoir utiliser des structures de données plus complexes car nous allons avoir besoin de gérer des priorités. C'est par exemple, autoriser l'existence de processus qui évitent la file d'attente. La structure de données utilisée peut parfaitement être une file, une liste, un arbre ou un tas, ceci en fonction de l'élément-clef de notre algorithme de sélection (âge, priorité simple, priorité à plusieurs niveaux, etc).

Cette structure de données doit nous permettre d'accéder à tous les processus prêts (éligibles).

7.1.2 Stratégie globale

On peut représenter l'ordonnancement global avec le schéma 7.2

Les ordonnancements à court terme doivent être très rapides, en effet le processus élu ne va utiliser l'unité centrale que pendant un très court laps de temps (10 milli-secondes par exemple).

Si on utilise trop de temps (1 milli-seconde) pour sélectionner cet élu, le taux utile décroît très rapidement (ici on perd 9% du temps d'unité centrale).

Par contre l'ordonnancement à long terme peut être plus long car il a lieu moins souvent (toutes les secondes par exemple). La conception de l'ordonnanceur à long terme est faite dans l'optique d'obtenir un ordonnanceur à court terme rapide.

7.1.3 Critères de performance

Les critères de performance des algorithmes d'ordonnancement
- Taux d'utilisation de l'unité centrale
- Débit
- Temps réel d'exécution
- Temps d'attente
- Temps de réponse

Ces cinq critères sont plus ou moins mutuellement exclusifs.
Les comparaisons des différents algorithmes se fait donc sur une sélection de ces critères.

7.2 Ordonnancement sans préemption.

- FCFS : First Come First served
  Facile à écrire et à comprendre, peu efficace ...
- SJF : Shortest Job First
  le plus petit en premier.
  Optimal pour le temps d'attente moyen ...
- A priorité :
  L'utilisateur donne des priorités aux différents processus et ils sont activés en fonction de cette priorité.

  problème $\longrightarrow$ famine possible des processus peu prioritaires

  Solution $\longrightarrow$ faire augmenter la priorité avec le temps d'attente :
  plus un processus attend, plus sa priorité augmente ainsi au bout d'un certain temps le processus devient nécessairement le plus prioritaire.
  re-problème $\longrightarrow$ si le processus en question (le très vieux très gros) est exécuté alors que de nombreux utilisateurs sont en mode interactif chute catastrophique du temps de réponse et du débit

  solution $\longrightarrow$ préemption.

La **préemption** est la possibilité qu'a le système de reprendre une ressource à un processus sans que celui-ci ait libéré cette ressource.
Ceci est impossible sur bon nombre de ressources. Lesquelles ?

7.3 Les algorithmes préemptifs

FCFS ne peut être préemptif ...
SJF peut être préemptif : si un processus plus court que le processus actif arrive dans la queue, le processus actif est préempté.

Dans des systèmes interactifs en temps partagé un des critères est le temps de réponse, c'est à dire que chaque utilisateur dispose de l'unité centrale régulièrement. Heureusement, les processus interactifs utilisent l'UC pendant de très courts intervalles à chaque fois.

7.3.1 Round Robin (tourniquet)

Cet algorithme est spécialement adapté aux systèmes en temps partagé.
On définit un **quantum de temps** (time quantum) d'utilisation de l'unité centrale.
La file d'attente des processus éligibles est vue comme une queue circulaire (fifo circulaire).
Tout nouveau processus est placé à la fin de la liste.

De deux choses l'une, soit le processus actif rend l'Unité Centrale avant la fin de sa tranche de temps (pour cause d'entrée/sortie) soit il est préempté, et dans les deux cas placé en fin de liste.

Un processus obtiendra le processeur au bout de (n -1)*q secondes au plus (n nombre de processus et q longueur du quantum de temps), la famine est donc assurément évitée.

Remarquons que si le quantum de temps est trop grand, round-robin devient équivalent à FCFS. De l'autre coté si le quantum de temps est très court, nous avons théoriquement un processeur n fois moins rapide pour chaque processus (n nombre de processus).

Malheureusement si le quantum de temps est court, le nombre de changements de contexte dûs à la préemption grandit, d'où une diminution du taux utile, d'où un processeur virtuel très lent.

Une règle empirique est d'utiliser un quantum de temps tel que 80 pourcent des processus interrompent naturellement leur utilisation de l'unité centrale avant l'expiration du quantum de temps.

7.3.2 Les algorithmes à queues multiples

Nous supposons que nous avons un moyen de différencier facilement les processus en plusieurs classes de priorité différentes (c'est le cas sous UNIX où nous allons différencier les tâches système, comme le swappeur, des autres tâches).

Pour sélectionner un processus, le scheduler parcourt successivement les queues dans l'ordre décroissant des priorités.

Un exemple de queues organisées en fonction du contenu des processus :
- les processus systèmes
- les processus interactifs
- les processus édition
- les processus gros calcul
- les processus des étudiants

pour qu'un processus étudiant soit exécuté il faut que toutes les autres files d'attente soient vides ...

Une autre possibilité est de partager les quantums de temps sur les différentes queues.

Il est aussi possible de réaliser différents algorithmes de scheduling sur les différentes queues :
- Round Robin sur les processus interactifs
- FCFS sur les gros calculs en tâche de fond.

7.4 Multi-level-feedback round robin Queues

Le système d'ordonnancement des processus sous UNIX (BSD 4.3 et system V4) utilise plusieurs files d'attente qui vont matérialiser des niveaux de priorité différents et à l'intérieur de ces différents niveaux de priorité, un système de tourniquet.

7.4.1 Les niveaux de priorité

Le scheduler parcourt les listes une par une de haut en bas jusqu'à trouver une liste contenant un processus éligible. Ainsi tant qu'il y a des processus de catégorie supérieure à exécuter les autres processus sont en attente de l'unité centrale.

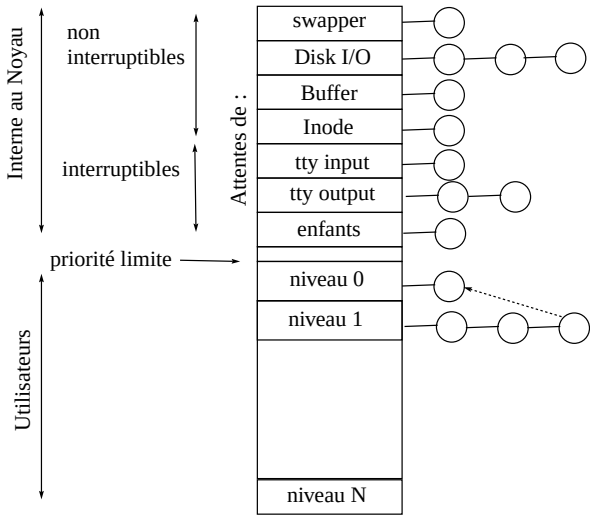

FIG. 7.3 – Les queues multiples en tourniquet

Dans les listes internes au noyau, de simples files d'attente sont utilisées avec la possibilité de doubler les processus endormis de la même liste (en effet seul le processus réveillé par la fin de son entrée/sortie est éligible).

Pour les processus utilisateurs, la même règle est utilisée mais avec préemption et la règle du tourniquet.

C'est à dire, on calcul une priorité de base qui est utilisée pour placer le processus dans la bonne file d'attente.

Un processus qui utilise l'unité centrale voit augmenter sa priorité.
Un processus qui libère l'unité centrale pour demander une entrée/sortie ne voit pas sa priorité changer.
Un processus qui utilise tout sont quantum de temps est préempté et placé dans une nouvelle file d'attente.

Attention : plus la priorité est grande moins le processus est prioritaire.

7.4.2 Evolution de la priorité

Regardons la priorité et l'évolution de la priorité d'un processus utilisateur au cours du temps. Les fonctions suivantes sont utilisées dans une implémentation BSD.

Pour calculer la priorité d'un processus utilisateur, le scheduler utilise l'équation suivante qui est calculée tous les 4 clicks horloge (valeur pratique empirique) :

$$P_{\text{usrpri}} = \text{PUSER} + \frac{P_{\text{cpu}}}{4} + 2 \times P_{\text{nice}}$$

cette valeur est tronquée à l'intervalle PUSER..127. En fonction de cette valeur le processus est placé dans une des listes correspondant à son niveau courant de priorité.

Ceci nous donne une priorité qui diminue linéairement en fonction de l'utilisation de l'unité centrale (il advient donc un moment où le processus devient le processus le plus prioritaire !).

P_{nice} est une valeur spécifiée par le programmeur grâce à l'appel système `nice`. Elle varie entre -20 et +20 et seul le super utilisateur peut spécifier une valeur négative.

P_{cpu} donne une estimation du temps passé par un processus sur l'unité centrale. A chaque click d'horloge, la variable `p_cpu` du processus actif est incrémentée. Ce qui permet de matérialiser la

consommation d'unité central du processus. Pour que cette valeur ne devienne pas trop pénalisante sur le long terme (comme pour un shell) elle est atténuée toute les secondes grâce à la formule suivante :

$$P_{\text{cpu}} = \frac{2 \times \text{load}}{2 \times \text{load} + 1} \times P_{\text{cpu}} + P_{\text{nice}}$$

la valeur de `load` (la charge) est calculée sur une moyenne du nombre de processus actifs pendant une minute.

Pour ne pas utiliser trop de ressources, les processus qui sont en sommeil (sleep) voient leur P_{cpu} recalculé uniquement à la fin de leur période de sommeil grâce à la formule :

$$P_{\text{cpu}} = \left(\frac{2 \times \text{load}}{2 \times \text{load} + 1} \right)^{\text{sleep\_time}} \times P_{\text{cpu}}$$

la variable `sleep_time` étant initialisée à zéro puis incrémentée une fois par seconde.

7.4.3 Les classes de priorité

La priorité des processus en mode système dépend de l'action à réaliser.

PSWAP 0 priorité en cours de swap
PINOD 10 priorité en attendant une lecture d'information sur le système de fichiers
PRIBIO 20 priorité en attente d'une lecture/écriture sur disque
PZERO 25 priorité limite
PWAIT 30 priorité d'attente de base
PLOCK 35 priorité d'attente sur un verrou
PSLEP 40 priorité d'attente d'un évènement
PUSER 50 priorité de base pour les processus en mode utilisateur

Le choix de l'ordre de ces priorités est très important, en effet un mauvais choix peut entraîner une diminution importante des performances du système.

Il vaut mieux que les processus en attente d'un disque soient plus prioritaires que les processus en attente d'un buffer, car les premiers risquent fort de libérer un buffer après leur accès disque (de plus il est possible que ce soit exactement le buffer attendu par le deuxième processus). Si la priorité était inverse, il deviendrait possible d'avoir un interblocage ou une attente très longue si le système est bloqué par ailleurs.

De la même façons, le swappeur doit être le plus prioritaire et non interruptible $\longrightarrow$ Si un processus est plus prioritaire que le swappeur et qu'il doit être swappé en mémoire ...
En Demand-Paging le swappeur est aussi le processus qui réalise les chargements de page, ce processus doit être le plus prioritaire.