Chapitre 1

Gestion des processus et ressources par un système d'exploitation

Un **système d'exploitation** (en anglais OS pour Operating System) est un ensemble de programmes qui dirige l'utilisation des ressources d'un ordinateur par des logiciels applicatifs. Il reçoit des demandes d'utilisation des ressources de l'ordinateur – stockage des mémoires (accès à la mémoire vive, aux disques durs) ; calcul du processeur central ; communication vers des périphériques ; via le réseau – de la part des logiciels applicatifs.

1.1 Processus

1.1.1 Processus

Un **processus** est l'objet dynamique associé à un programme. Plus précisément, on peut le définir comme :

- un ensemble d'instructions à exécuter chargées depuis la mémoire (la plupart du temps vive mais cela peut aussi être la morte) ;
- un espace d'adressage en mémoire vive pouvant stocker les données de travail, etc ;
- des ressources permettant des entrées et sorties de données.

Un processus a un début mais pas forcément de fin. Le temps de séjour d'un processus au sein du système est le temps écoulé entre la soumission et l'achèvement du processus ; il diffère du temps d'exécution en ceci qu'il peut compter le temps d'attente avant l'exécution :

$$\text{temps de séjour} = \text{temps d'attente} + \text{temps d'exécution}.$$

1.1.2 États d'un processus

Un processus passe par plusieurs états qui sont décidés par l'**ordonnanceur**, programme du système d'exploitation choisissant l'ordre des processus à exécuter par un processeur. De façon générale, on retrouve les états suivants sur la plupart des systèmes d'exploitation.

Initialisé : C'est le premier état d'un processus. Il attend que l'ordonnanceur le place dans l'état *prêt*, ce qui peut prendre plus ou moins longtemps. Les processus initialisés (ou crées) ne sont pas forcément placés dans la file d'attente du processeur, notamment sur les OS temps réel où il y a un risque de saturation empêchant alors la machine de réagir dans les délais voulus.

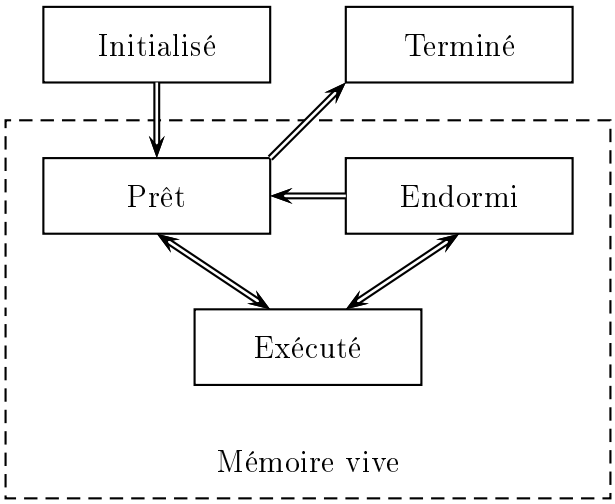
Prêt (en attente) : Le processus a été chargé dans la mémoire centrale et attend l'exécution par le processeur. Dans une machine équipée d'un seul processeur, comme celui-ci ne peut traiter les processus que un par un, il peut y en avoir beaucoup en attente. Les processus disponibles sont rangés dans une file qui est gérée par l'ordonnanceur.

Exécuté (ou élu) : Le processus est choisi (élu) par l'ordonnanceur pour être exécuté parmi les processus **prêts** ou **endormis**. Un processus en cours d'exécution peut-être **endormi** si nécessaire.

Endormi (ou bloqué) : Le processus a été bloqué ou attend un événement (disponibilité d'une ressource nécessaire, fin d'une opération, etc). Soit il retourne parmi la file des processus *prêts* lorsqu'il l'est à nouveau, soit son *exécution* est reprise.

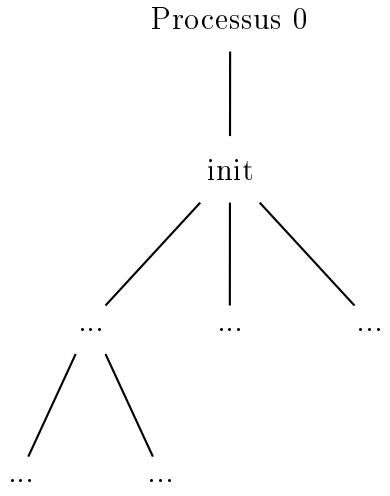
Terminé : Soit le programme a fourni un résultat, soit son arrêt a été forcé.

On retrouve les états mentionnés ci-dessus dans le diagramme ci-contre. Les flèches expriment les transitions d'états autorisées.



1.1.3 Création d'un processus

Un processus peut être déclenché par l'utilisateur par l'intermédiaire d'un logiciel applicatif ou d'un périphérique mais aussi par un autre processus. La création d'un processus par un autre donne naissance à une arborescence de processus. Sous Linux, un processus 0 est crée au démarrage, lequel donne naissance à un second processus généralement appelé « init » dont découle les autres processus ; le processus 0 s'apparente à la racine du système de gestion de fichiers « / ».
Chaque processus possède un identifiant numérique auto-incrémenté **PID** (**P**rocess **Id**entification). Le processus 0 reçoit le numéro 0, init reçoit le 1, etc. Chaque processu – excepté le processus 0 – reçoit également une PPID (**P**arent **PID**) qui permet de connaître le PID du processus parent. Par exemple, le PPID de init est le PID du processus 0 : 0.



1.1.4 Arrêt d'un processus

On distingue quatre cas d'arrêt d'un processus.

1. Volontaire et normal. Par exemple un programme s'exécutant et s'achevant comme prévu par le programmeur.
2. Volontaire pour cause d'erreur. Par exemple un programme débouchant sur une boucle infinie devant être arrêtée volontairement par le programmeur.
3. Involontaire pour cause d'erreur. Par exemple une erreur de programmation rendant l'exécution de celui-ci impossible.
4. Involontairement interrompu par un autre processus.

1.2 Ordonnancement des processus

1.2.1 Ordonnanceur

Les systèmes de type Unix sont multi-tâches : on a l'impression que plusieurs processus peuvent s'exécuter en même temps. Un processeur ne peut pourtant exécuter qu'un seul processus à la fois, qu'on appelle processus actif. L'ordonnanceur change régulièrement de processus actif en procédant à un changement d'état. Il est lui-même un processus et est confronté à deux problématiques :

1. le choix du processus à exécuter ;
2. le temps de processeur à allouer au processus choisi.

Sachant que ses objectifs sont :

— permettre l'accès de chaque processus au processeur ;
— minimiser le temps réponse ;
— utiliser pleinement le processeur ;
— prendre en compte les priorités ;
— utiliser les ressources de façon équilibrée ;
— être prédictible.

Ces objectifs sont parfois complémentaires et parfois antagonistes. Il est impossible d'optimiser simultanément tous ces critères.

1.2.2 Types de systèmes d'exploitation

On distingue deux types de systèmes d'exploitation multi-tâches (i.e. donnant l'impression de pouvoir exécuter plusieurs tâches simultanément).

Coopératif : si le système d'exploitation alloue aux processus des ressources dont ils sont libres de disposer. Il y a dans ces systèmes un risque accru de blocage.

Préemptif : si le système d'exploitation peut stopper à tout moment un processus en cours afin d'en élire un autre. Dans un tel système, on peut lancer plusieurs applications simultanément, passer de l'une à l'autre. Toutefois, le noyau garde toujours le contrôle et a la possibilité de fermer des processus qui monopolisent les ressources du système, réduisant ainsi les risques de blocages du système.

1.2.3 Ordonnanceur non préemptif

Dans un système non préemptif, l'ordonnanceur ne peut pas réquisitionner des ressources au détriment d'un processus afin d'en prioriser un autre. Les ressources allouées à un processus le sont donc jusqu'à ce que celui-ci se termine ou soit bloqué.

On distingue deux principales stratégies afin d'ordonnancer les processus dans un tel cas :

— premier arrivé, premier servi (**F**irst **C**ome **F**irst **S**erved **FCFS**) qui est le principe de la fil d'attente ;

— le plus rapide d'abord (**S**hort **J**ob **F**irst **SJF**) où on traite les tâches par ordre croissant de temps nécessaire.

Premier arrivé, premier servi

Considérons cinq processus avec les temps d'arrivage et d'exécution donnés dans les tableaux ci-dessous. L'unité de temps est le temps CPU, i.e. celui du processeur.

Processus	Temps d'exécution	Temps d'arrivée
A	3	0
B	6	1
C	4	4
D	2	6
E	1	7

La pile des processus avec leurs temps d'arrivée, d'attente et d'exécution peut se représenter dans le tableau ci-dessous. Les cases grisées correspondent à l'exécution des processus. On peut constater que la plupart des processus passent une grande partie de leur temps en attente.

Temps CPU															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A	A													
	B	B	B	B	B	B	B	B							
				C	C	C	C	C	C	C	C	C			
						D	D	D	D	D	D	D	D	D	
							E	E	E	E	E	E	E	E	E

On obtient alors les temps d'attente et de séjour ci-contre. Ce qui donne donc comme temps d'attente moyen 4,4 unités de temps CPU et comme temps de séjour moyen 7,6 unités de temps. Le tout exécuté en 16 unités de temps, ce qui fait en moyenne 3,2 unités de temps par processus.

Processus	Attente	Séjour
A	0	3
B	2	8
C	5	9
D	7	9
E	8	9

Le plus rapide d'abord

Remarquons tout d'abord que pour obtenir un ordonnancement optimal, il faut que tous les processus soient disponibles simultanément. En effet, dans un système non préemptif, si un processus plus rapide arrive après qu'un plus long ait commencé, il devra attendre que ce

dernier ait terminé avant de pouvoir s'exécuter.

Reprenons nos processus précédents. Étant donné l'ordre d'arrivée des processus, il n'y aucun changement pour A et B qui passent en premiers. Pour C, D et E, il y a un changement dans la mesure où ils arrivent tous pendant l'exécution de B ; à la fin de cette dernière, l'ordonnanceur aura donc le choix entre ces trois processus ; il les choisira par ordre croissant de temps d'exécution. Cela donne le tableau d'avancement ci-dessous.

Temps CPU															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A	A													
	B	B	B	B	B	B	B	B							
				C	C	C	C	C	C	C	C	C	C	C	C
						D	D	D	D	D	D				
							E	E	E						

On obtient alors les temps d'attente et de séjour ci-contre. Ce qui donne donc comme temps d'attente moyen 3,2 unités de temps CPU et comme temps de séjour moyen 7,6 unités de temps. Le tout toujours exécuté en 16 unités de temps, ce qui fait en moyenne 6,4 unités de temps par processus.

Processus	Attente	Séjour
A	0	3
B	2	8
C	8	12
D	4	6
E	2	3

On a donc dans ce cas un avantage pour le plus court d'abord par rapport au premier arrivé premier servi. Toutefois, les ordonnanceurs non préemptifs ne sont généralement pas intéressants pour les systèmes multi-utilisateurs car les temps de réponse ne sont pas toujours acceptables.

1.2.4 Ordonnanceur préemptif

Dans un schéma d'ordonnanceur préemptif, ou avec réquisition, pour s'assurer qu'aucun processus ne s'exécute pendant trop de temps, les ordinateurs ont une horloge électronique qui génère périodiquement une interruption. À chaque interruption d'horloge, le système d'exploitation reprend la main et décide si le processus courant doit poursuivre son exécution ou s'il doit être suspendu pour laisser place à un autre.

S'il décide de suspendre son exécution au profit d'un autre, il doit d'abord sauvegarder l'état des registres du processeur avant de charger dans les registres les données du nouveau processus à lancer. C'est qu'on appelle la commutation de contexte ou le changement de contexte. Cette sauvegarde est nécessaire pour pouvoir poursuivre ultérieurement l'exécution du processus suspendu. Le processeur passe donc d'un processus à un autre en exécutant chaque processus pendant quelques dizaines ou centaines de millisecondes. Le temps d'allocation du processeur au processus est appelé quantum.

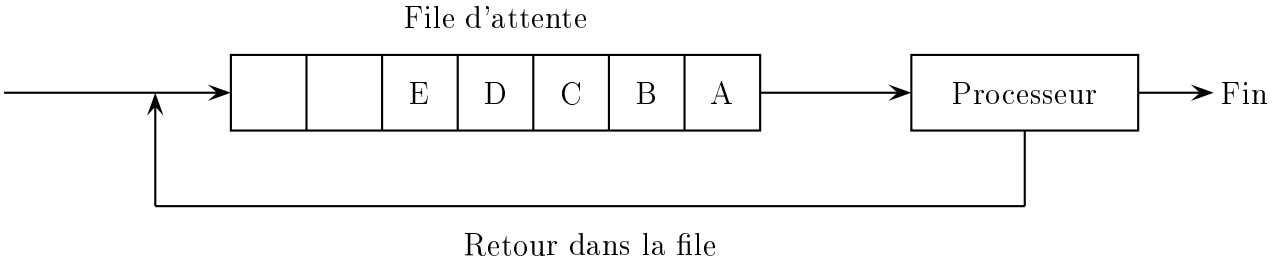
Cette commutation entre processus doit être rapide : un temps nettement inférieur au quantum. Le processeur, à un instant donné, n'exécute réellement qu'un seul processus, mais pendant une seconde, le processeur peut exécuter plusieurs processus et donne ainsi l'impression de parallélisme (pseudo-parallélisme).

Cela soulève plusieurs problématiques.

1. Le choix de la valeur du quantum.
2. Le choix du processus à exécuter dans chacune des situations suivantes :
   (a) Le processus en cours est bloqué / endormi.
   (b) Le processus en cours retourne à l'état prêt à la fin du quantum.
   (c) Un processus passe de l'état bloqué / endormi à l'état prêt.
   (d) Le processus en cours se termine.

Le tourniquet circulaire

L'algorithme du tourniquet circulaire (**R**ound **R**obin **RR**) est la version préemptive du premier arrivé, premier servi. C'est un algorithme ancien, fiable, simple et très utilisé.



Choix du processus : Il alloue le processeur au processus en tête de file, pendant un quantum de temps.

- Si le processus se bloque ou se termine avant la fin de son quantum, le processeur est immédiatement alloué à un autre processus : celui en tête de file.
- Si le processus ne se termine pas au bout de son quantum, son exécution est suspendue. Le processeur est alors alloué à un autre processus : à nouveau celui en tête de file. Le processus suspendu est inséré en queue de file. Les processus qui arrivent ou qui passent de l'état *bloqué* à l'état *prêt* sont insérés en queue de file.

Choix de la valeur du quantum : Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur. Un quantum trop élevé augmente le temps de réponse des courtes commandes en mode interactif. Un quantum entre 20 et 50 ms est souvent un compromis raisonnable. Le quantum était de 1 s dans les premières versions d'UNIX.

Exemple : considérons deux processus dont les temps d'arrivée et d'exécution sont donnés dans le tableau ci-dessous.

Processus	Temps d'exécution	Temps d'arrivée
A	12	0
B	4	1

Quantum de 10 : on obtient le tableau d'avancement suivant.

Temps CPU															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	B	B	B	B	B	B	B	B	B	B	B	B	B		

On obtient alors les temps d'attente et de séjour ci-contre. Ce qui donne donc comme temps d'attente moyen 6,5 unités de temps CPU et comme temps de séjour moyen 14,5 unités de temps.

Processus	Attente	Séjour
A	4	16
B	9	13

Quantum de 3 : on obtient le tableau d'avancement suivant.

Temps CPU															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	B	B	B	B	B	B	B	B	B						

On obtient alors les temps d'attente et de séjour ci-contre. Ce qui donne donc comme temps d'attente moyen 4,5 unités de temps CPU et comme temps de séjour moyen 10 unités de temps.

Processus	Attente	Séjour
A	4	16
B	5	9

Le plus petit temps de séjour

L'ordonnancement du plus petit temps de séjour (**S**hortest **R**emaining **T**ime **SRT**) est la version préemptive de l'algorithme SJF. Un processus arrive dans la file de processus, l'ordonnanceur compare la valeur espérée pour ce processus contre la valeur du processus actuellement en exécution. Si le temps du nouveau processus est plus petit, il rentre en exécution immédiatement.

Exemple : toujours avec nos deux processus A et B, on obtient de tableau d'avancement suivant.

Temps CPU															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	B	B	B	B											

On obtient alors les temps d'attente et de séjour ci-contre. Ce qui donne donc comme temps d'attente moyen 2 unités de temps CPU et comme temps de séjour moyen 10 unités de temps.

Processus	Attente	Séjour
A	4	16
B	0	4

On a clairement dans ces exemples un avantage au SRT par rapport au RR.

1.3 Utilisation des ressources par un processus

1.3.1 Attribution des ressources

C'est le système d'exploitation qui alloue les ressources nécessaires aux processus. L'attribution des ressources peut s'effectuer selon trois modalités :

— si l'ordinateur possède plusieurs processeurs, les processus sont répartis équitablement entre ceux-ci ;

— une matrice (ou table) de droits donnant la relation entre ressources et processus ;

— un héritage de droits d'utilisateur : le processus possède les mêmes droits que ceux de l'utilisateur qui l'a déclenché.

1.3.2 Risque d'interblocage dans l'attribution des ressources

Une ressource utilisée par un processus ne peut être libérée que si celui-ci est dans l'état *élu*. Le système doit alors s'assurer qu'un processus n'interfère pas avec les autres : le processus peut en empêcher d'autres d'accéder à une ressource nécessaire. On parle d'**isolation** des processus.

L'**interblocage** (ou deadlock) est une situation pouvant se produire en programmation concurrente. On peut l'obtenir dans les cas suivants.

— Deux processus attendent chacun que l'un libère une ressource nécessaire à l'autre.

— Deux processus attendent chacun le résultat de l'autre processus.

— Un processus s'attend lui-même.

Il s'agit d'une impasse conduisant à un blocage définitif du système. Il s'agit donc d'une erreur informatique à éviter impérativement.

Exemple : considérons deux processus P_1 et P_2 *initialisés* et deux ressources R_1 et R_2 libres.

1. P_1 a besoin de R_1 pour être *prêt*, il en fait donc la demande. P_2 est passé dans l'état *prêt*.
2. P_2 est *élu* et a besoin de R_2 en cours d'exécution ; il l'obtient immédiatement car elle est libre et reprend son exécution. Pendant ce temps, R_1 est attribuée à P_1 et celui-ci est alors *prêt*.
3. P_2 demande maintenant R_1, laquelle est attribuée à P_1 et n'est donc pas libre. P_2 est donc *endormi* le temps que R_1 se libère.
4. C'est maintenant au tour de P_1 d'être *élu* et de s'exécuter.
5. En cours d'exécution, P_1 demande R_2. Or R_2 est attribuée à P_2 qui n'a pu s'achever et donc la libérer. En attendant que R_2 soit libérée P_1 est *bloqué*.
6. Comme P_1 est *bloqué*, il n'a pas pu libérer la ressource R_1 nécessaire au déblocage de P_2. Lequel bloque la libération de R_2 nécessaire au déblocage de P_1. Le système est alors complètement bloqué.

1.4 Exercices

Exercice 1.1. On considère les cinq processus avec leurs temps d'exécution et d'arrivée donnés dans le tableau ci-dessous. L'unité de temps est le temps CPU, i.e. celui du processeur.

Processus	Temps d'exécution	Temps d'arrivée
A	10	0
B	1	1
C	2	2
D	1	3
E	5	4

Pour chacun des algorithmes suivants, donner le tableau d'avancement des processus, les temps d'attente et de séjour de chaque processus et enfin les temps d'attente et de séjour moyen. On prendra si besoin un quantum égal à 3.

1. FCFS.
2. SJF.
3. RR.
4. SRT.

Exercice 1.2. Même questions que l'exercice précédent avec les processus suivants.

Processus	Temps d'exécution	Temps d'arrivée
A	4	0
B	2	0
C	1	3

Exercice 1.3. [Terminal]

1. Si vous êtes sur Linux, ouvrir un terminal. Sinon, un simulateur de terminal Linux est disponible ici.
2. Utiliser la commande `ps -aef`. Que fait-elle ?
3. Utiliser la commande `top`. Que fait-elle ? On pourra quitter à l'aide de `Ctrl` + `Maj` + `C`.