Chapter 1

Processus et ordonnancement

Un ordinateur est un ensemble de dispositifs mécaniques, électroniques et logiciels capable de réceptionner, de traiter et d'émettre de l'information. Nous avons vu que les travaux d'Alan Turing puis de Von Neumann et Grace Hopper ont permis de créer les premiers ordinateurs entre 1940 et 1950. Encore aujourd'hui, les ordinateurs suivent l'architecture de Von Neumann.

Figure 1.1: Grace Hopper devant un UNIVAC I, John Von Neumann et Alan turing vers 1950.

Ce modèle décompose l'ordinateur en 4 parties distinctes :

1. l'unité arithmétique et logique (UAL) ou unité de traitement, qui effectue les opérations de base : opérations arithmétiques et opérations sur les entiers en binaire ;
2. l'unité de contrôle, qui est chargée du séquençage des opérations ; elle récupère en mémoire la prochaine instruction et les données qu'elle envoie à l'UAL.
3. la mémoire, qui contient à la fois les données et le programme qui indique à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise en :
   - mémoire vive ou volatile : RAM (Random-access Memory) programmes et données en cours. Les données peuvent y être lues, déplacées ou effacées comme on le souhaite.
   - La mémoire non volatile : ROM (Readonly Memory) non modifiable qui contient des informations dont l'ordinateur a besoin pour fonctionner.
4. les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.

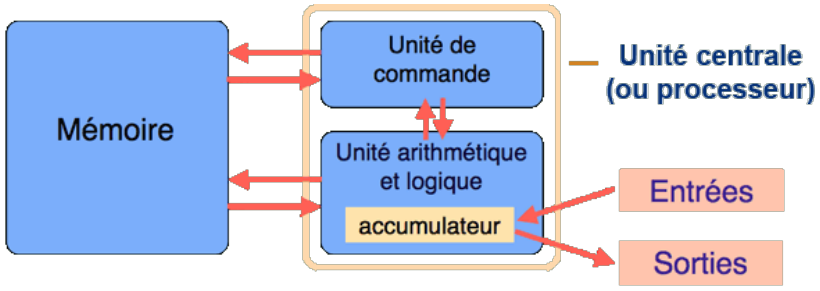


Figure 1.2: Architecture de Von Neumann (du site interstices.info)

1.1 Système d'exploitation

1.1.1 Rôle de l'OS

Nous avons vu que le système d'exploitation est responsable de la gestion des ressources matérielles. Le SE présente à chaque processus un environnement d'exécution, dans lequel il peut tourner sans se soucier des autres processus. Dans le jargon, on dit qu'il *virtualise* les ressources, car chaque processus croît avoir l'ensemble des ressources réelles pour lui alors qu'il n'a accès qu'à une petite partie.

Le SE se situe donc entre les utilisateurs et le matériel. C'est un ensemble de programme qui permet de gérer et partager les ressources. en voici les principales fonctions:

- gestion des fichiers;
- gestion de la mémoire;
- gestion des périphériques;
- traitement des entrées/sorties;
- sécurisation du sytème.

L'exécution des programmes est prise en charge par les processus créés par le SE Le mécanisme d'ordonnancement du système d'exploitation partage l'utilisation du ou des processeurs entre l'ensemble de ces processus où le choix du processus élu repose sur un algorithme d'ordonnancement. L'ordonnanceur ou *scheduler* est le module du système d'exploitation qui organise l'execution des processus à tour de rôle.

Nous détaillons ces notions dans ce chapitre.

1.1.2 Processus

Un processus est l'objet dynamique associé à un programme. Un programme est statique : c'est une suite d'instructions à exécuter associées à des données, le processus est une instance en mémoire de ce programme en train de s'exécuter. Il a donc une durée de vie limité. Il peut bien sûr y avoir plusieurs processus associés au même programme, par exemple on peut lancer simultanément plusieurs interpréteurs python.

On peut dire qu'**un processus est composé d'un programme et d'un contexte.**

Ce contexte est un ensemble d'informations: adresse des données, mémoires, exécutables etc. A la création d'un processus, on associe différents paramètres:

- un PID ou numéro d'identification (Process identification).
- un PPID (Parent Process Identification), numéro du processus père.
- un UID, identifiant de l'utilisateur ayant démarré le processus.
- un GUID, identifiant du groupe de l'utilisateur ayant démarré le processus.

Sur les systèmes Linux, les processus `init` et `kthreadd` ont respectivement pour PID 1 et 2.

1.2 La cuisine de l'ordonnancement

Utilisons une image pour comprendre la logique du SE et des processus. Imaginons que nous *virtualisions* une cuisine pour que deux chefs cuisinent leur plat avec les ressources de cette cuisine, sans que l'un puisse connaître ni modifier ce que l'autre prépare. La cuisine représente les ressources matérielles, les cuisiniers représentent les processus.

Donner l'accès: La première chose à faire pour le SE est de planifier qui a accès à la cuisine et quand. C'est l'**ordonnancement (scheduling)** : on alloue un temps limité à chaque cuisinier après lequel on lui demandera de sortir pour revenir plus tard.

Changer le contexte: Puis il faut organiser le changement de contexte (*context-switch*) : lorsque l'on change de cuisinier, on nettoie toute la cuisine et on remet en place les affaires du précédent cuisinier, sans laisser de traces.

Verrouiller: Enfin, comme les placards stockent les affaires des deux cuisiniers, il faut empêcher que l'un ouvre et modifie un placard dont l'autre se sert. Pour ce faire, le SE bloque par défaut l'accès aux placards. Lorsqu'un cuisinier essaie d'ouvrir un placard, le SE le fait sortir de la cuisine, et vérifie que le placard n'a pas été alloué à un autre cuisinier avant de le lui attribuer.

Si l'ordonnancement est trop permissif avec les processus, un processus malveillant peut prendre tout le temps et empêcher un autre processus de s'exécuter.

S'il y a une erreur dans le changement de contexte, un processus malveillant peut avoir accès à des informations sur un autre processus.

Enfin, s'il y a un problème dans la gestion de la mémoire (les placards), le SE pourrait autoriser par erreur l'accès d'un processus à une zone de mémoire qu'il n'était pas censé voir. Dans le cas des processus, on peut définir leur état par la mémoire à laquelle ils ont accès et leurs registres.

1.3 Différents processus

Au cours du temps, l'augmentation continue de la vitesse des processeurs et de la taille des mémoires, accompagnée par la baisse de leur coût, a facilité la commutation des processus. Celle-ci était handicapante sur les premières machines et des architectures apparurent où les services fournis par le système étaient délégués à des processus créés pour cela.

On aboutit donc à des systèmes ayant un plus grand nombre de petits processus, parfois appelés **démons** quand il s'agit des processus du système. L'énorme différence de rapidité entre processeur et vitesse de réaction de l'utilisateur est mise à profit pour simuler le parallélisme de déroulement des programmes en partageant les processeurs entre les processus — c'est ce qu'on appelle le pseudo-parallélisme.

Les processus peuvent être regroupés en deux catégories:

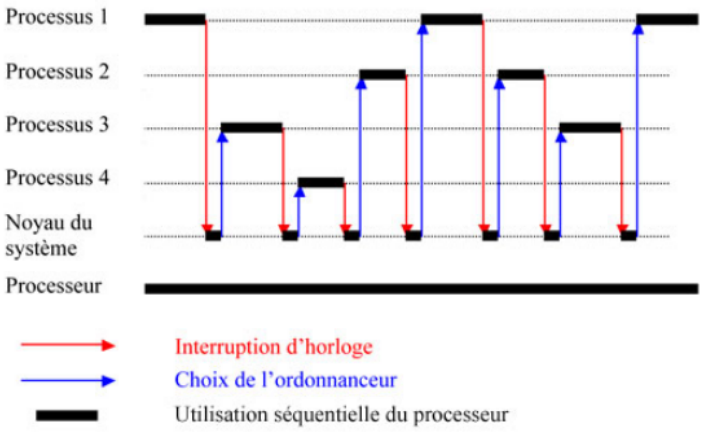
Processus utilisateurs: ce sont les applications lancé par l'utilsateur comme un navigateur ou l'explorateur de fichier.

Processus système: les processus de l'OS dont certains fonctionnent en permanence: ce sont les *services* sous Windows ou *daemon* sous Linux.

1.3.1 Interruption et ordonnanceur

Ce partage est rythmé par un top d'horloge qui déclenche l'**interruption** du processus en cours et l'attribution du processeur à un autre processus: on change de cuisinier.

Cette attribution est régie par l'**ordonnanceur**, programme du noyau du système. La figure ci-dessous montre un exemple de partage d'un processeur (on suppose ici qu'il n'y en a qu'un) entre quatre processus déclenchés simultanément qui l'utilisent l'un après l'autre, par tranches séquentielles successives.



Partage d'un processeur entre 4 processus

]

Comme on peut le voir ci-dessus, l'attribution du processeur par le noyau consomme aussi du temps de processeur: il faut éviter trop de commutations de contexte.

(Image :`Interstices.infos` *CC-BY-NC-ND 4.0*)

1.3.2 Unix

Les systèmes de type UNIX (1969) sont multi-tâches : on a l'impression que plusieurs processus peuvent s'exécuter en même temps (plus que le nombre de processeurs en cas de système multi-processeurs). Chaque processus est identifié par un entier, son PID (process identifier). Cela permet de communiquer avec ce processus pour interrompre son cycle normal. Par exemple quand l'utilisateur se déconnecte de son compte, un signal est envoyé à chacune de ses applications pour qu'elle se ferme. L'envoi d'un signal peut se faire en ligne de commande grâce à la commande `kill` suivie du PID. On peut voir la liste des processus qui tournent sur le système avec la commande `ps` :

```
romain@Hal:/home/romain$ ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 225944  9372 ?        Ss   19:42   0:01 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    19:42   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   19:42   0:00 [rcu_gp]
..........
romain   17396  2.0  0.0  24140  5096 pts/1    S    22:18   0:00 bash
root     17406  0.0  0.0      0     0 ?        I    22:18   0:00 [kworker/u8:0]
romain   17894  4.7  0.8 1027620 82724 pts/1   Sl   00:18   0:00 vlc
romain   17408  0.0  0.0  40992  3728 pts/1    R+   22:18   0:00 ps aux
```

`ps -aux` classe les processus par ordre de PID.

`ps -aef` affiche en plus le PPID: identificateur du processus père.

`kill` permet de tuer un processus. Avec l'option `-9`, on peut forcer l'arrêt d'un processus bloqué.
La commande `kill 17894` tue le processus 17894 (*vlc* d'après la capture ci-dessus).

1.4 Etat d'un processus

L'état d'un processus dépend du fait qu'il est en attente d'une ressource, ou encore qu'il ait été élu par l'algorithme d'odonnancement.

1.4.1 Différents états

Pendant sa durée de vie, un processus peut-être:

bloqué: En attente d'un évènement ou de données. La ressource peut-être indisponible temporairement.

prêt ou passif: C'est l'état quand il est créé. Les données sont disponibles, il attend le processeur pour devenir **actif**.

actif ou élu: Il est en cours d'exécution et peut terminer, repasser à l'état **bloqué** (ressource indisponible) ou à l'état **prêt** (désactivé par l'OS).

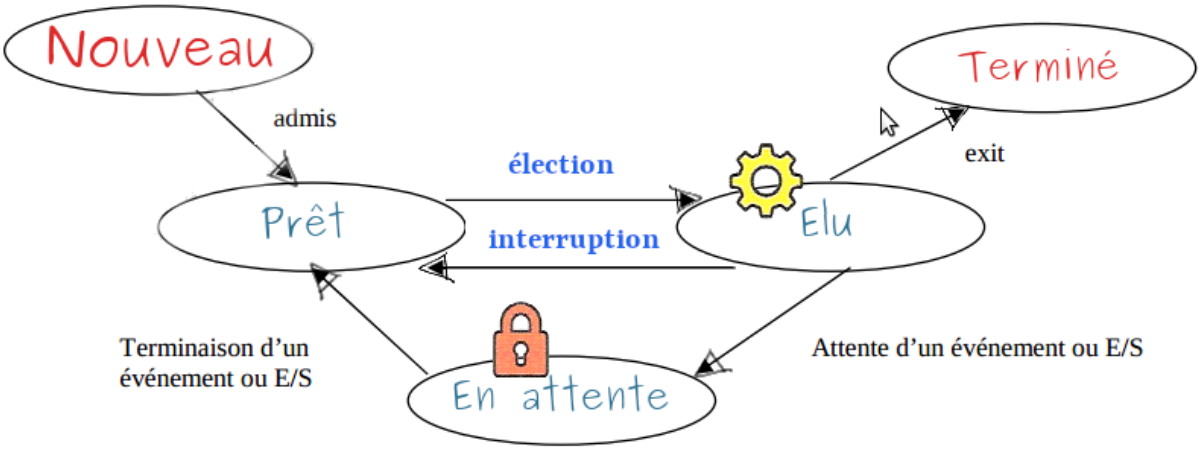


Figure 1.3: L'interruption d'un processus en cours d'exécution est appelée **préemption**.

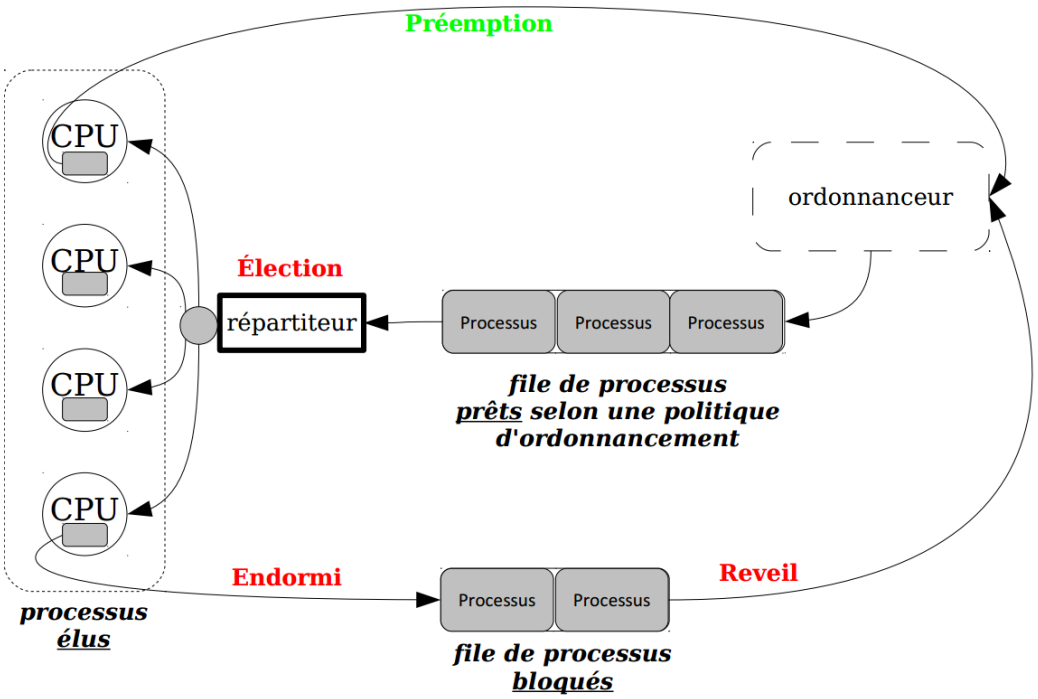
1.5 Fonctionnement

1.5.1 Ordonnancement

La politique d'ordonnancement mise en oeuvre par le système d'exploitation peut donc ou non autoriser l'opération de réquisition du processeur : celui-ci peut être retiré au processus élu alors que celui-ci dispose de toutes les ressources nécessaires à la poursuite de son exécution. **Cette réquisition porte le nom de préemption**. Un processus préempté quitte l'état *élu* pour passer dans l'état *prêt*.

L'OS décide quel processus est actif de manière efficace et équitable. Il joue le rôle d'**ordonnanceur** (scheduler) en gérant l'accès concurrent aux ressources. Il range pour cela par ordre de priorité une liste des processus.

Dans un cadre multiprocesseur, le **répartiteur** attribue un processeur au processus élu. Il dispose pour cela d'une table des processus, d'une file d'attente des processus prêts et d'une file d'attente des processus bloqués. En pratique , on peut avoir plusieurs files de processus en attente de ressources.



Ordonnancement préemptif

Le processeur peut être retiré à un processus qui met trop de temps à se terminer ; on n'a donc pas besoin d'attendre que le processus achève son exécution avant de lui retirer le processeur.

Les ordinateurs ont une horloge électronique qui génère périodiquement une interruption. A 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.

1.5.2 Politiques d'ordonnancement:

Les critères les plus souvent utilisés pour comparer et évaluer les performances des algorithmes de *scheduling* du processeur sont :

A-Critères d'ordonnancement

Utilisation du processeur: Un bon algorithme de *scheduling* sera celui qui maintiendra le processeur aussi occupé que possible.

Capacité de traitement : C'est la quantité de processus terminés par unité de temps.

Temps de restitution : C'est le temps s'écoulant entre la soumission du travail et sa terminaison.

Temps d'attente : C'est le temps passé à attendre dans la file d'attente des processus prêts.

Temps de réponse : C'est le temps passé dans la file d'attente des processus prêts avant la première exécution.

B-Exemples d'ordonnancement

Premier arrivé, premier servi (FCFS): simple mais souvent inadapté. Il n'y a pas de préemption, un processus élu s'exécute jusqu'à ce qu'il soit terminé ou bloqué de lui-même.

Plus court d'abord (SJF): efficace mais il faut connaître à l'avance le temps d'exécution du processus. C'est un algorithme dit *temps réel*.

Priorité: le système alloue un niveau de priorité aux processus. Cette politique se décline en deux versions selon si la réquisition est autorisée ou non. Si la réquisition est admise, alors le processus couramment élu est préempté dès qu'un processus plus prioritaire, donc possédant une priorité plus forte devient prêt. Pour empêcher les processus de priorité élevée de s'exécuter indéfiniment, l'ordonnanceur diminue régulièrement la priorité du processus en cours d'exécution.

Processus de même priorité Ils sont regroupés dans une file du type FIFO avec autant de files qu'il y a de niveaux de priorité. L'ordonnanceur choisit le processus le plus prioritaire qui se trouve en tête de file. En général, les processus de même priorité sont ordonnancés selon l'algorithme du tourniquet.

Tourniquet (Round Robbin): L'algorithme du tourniquet, circulaire ou *round robin* mémorise dans une file du type FIFO (First In First Out) la liste des processus prêts, c'est à dire en attente d'exécution.

Un quantum de temps entre 20 et 50 *ms* selon les OS est alloué à chaque processus chacun leur tour. Si le temps alloué est insuffisant, il est mis en bout de file en état *prêt*. Si le processus se bloque ou se termine avant la fin de son quantum, le processeur est immédiatement alloué au processus en tête de file.

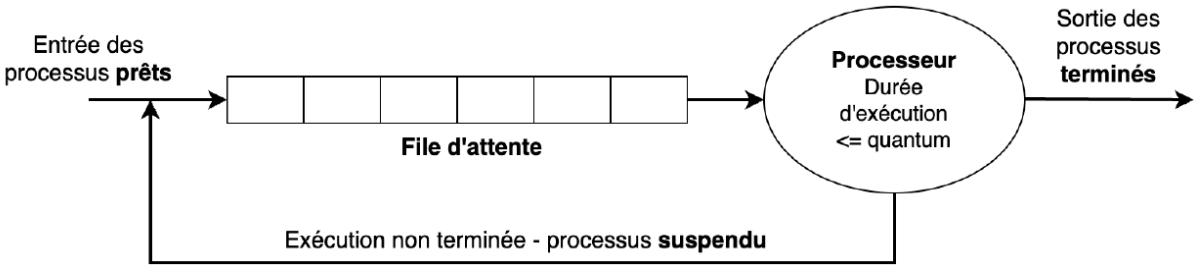


Figure 1.4: Ordonnancement avec l'algorithme du tourniquet

Hybride: hybride entre tourniquet et priorité (Unix).

Exemple: Dans le cas de d'un ordonnancement *premier arrivé, premier servi*, on donne les trois processus P1, P2, P3 suivants, tous soumis à l'instant 0:

Nom du processus	Durée d'exécution	Ordre de soumission
P1	16	1
P2	4	2
P3	6	3

On peut alors donner le chronogramme ou diagramme de Gantt corrspondant à cette situation.
Diagramme de Gantt: Représentation graphique de l'exécution de plusieurs processus affectés à plusieurs ressources au cours du temps.

On lit les temps d'attente respectifs de chaque processus: 16, 20, 26. Le temps moyen d'attente est donc de $\frac{0+16+20}{3} = 12$

Sauvegarde du contexte

Le contexte sauvegardé lorsqu'un processus est désactivé par l'OS contient également:

- L'état du processus: PID, priorité et UID;
- temps de CPU, fichiers utilisés
- adresse de l'instruction suivante qui doit être exécutée par le processus etc.

Ce contexte est sauvegardé dans un bloc de contrôle de processus ou PCB (*process control block*). Commuter le processeur sur un autre processus nécessite d'effectuer cette sauvegarde et de charger l'état sauvegardé par le nouveau processus. Cette tâche est connue sous le nom de commutation de contexte.

Thread

Un processus léger ou **thread** est une séquence d'instructions traitée par le processeur. Celui-ci ne peut traiter qu'un seul thread à la fois et chaque thread possède son propre contexte. Cependant, les thread forment des groupes se partageant le même espace mémoire.

1.6 Gestion des ressources et interblocage

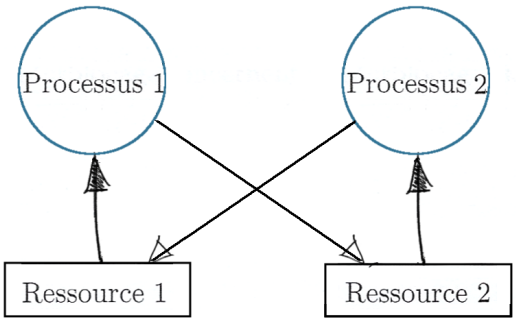
Un processus dispose de différentes ressources: RAM, disques et clé USB, périphériques etc. L'utilisation d'une ressource passe par 3 étapes:

1. demande d'accès, mise en attente si la ressource n'est pas disponible;
2. utilisation proprement dite;
3. libération de la ressource, qui peut être réaffectée.

Interblocage

Il peut arriver que deux ressources s'attendent mutuellement : par exemple quand un premier processus P1 mobilise une ressource R1 et attend une ressource R2 utilisée par le second processus P2. Le blocage est définitif et on parle d'**interblocage** ou **deadlock**.

L'OS doit être capable de détecter ces situations : en contrôlant l'affectation des ressources de manière régulière ou en observant l'utilisation du processus. Il peut alors arrêter le processus concerné ou lui retirer la ressource pour l'affecter à un autre processus.



Graphe d'allocation des ressources

Remarquez l'orientation de ce graphe : Les relations symbolisées par les arêtes sont orientées. Un processus pointe vers une ressource qu'il attend. Dans le cas contraire, il a obtenu la ressource. Si on trouve un **cycle** dans ce graphe, alors il y a risque d'interblocage.

Note: Il existe quatre conditions à la création d'un interblocage dites conditions de *Coffman*:

1. Exclusion mutuelle: au moins une ressource du système doit être en accès exclusif.
2. Rétention et attente: un processus détient une ressource et demande une autre ressource détenue par un autre processus.
3. Non préemption: Une ressource ne peut être rendue que par la ressource qui la détient: celle-ci ne peut être préemptée par un autre processus.
4. Attente circulaire : il doit exister une chaîne d'au moins deux processus dont chacun attend une ressource détenue par le memebre suivant de la chaîne.

1.7 Exercices

Exercice 1 Tracer les chronogrammes d'exécution pour les processus P1, P2, P3 dans les cas suivants où la priorité la plus élevée correspond à la valeur la plus petite. On considère qu'ils sont tous soumis à l'instant 0.

1. Avec priorité:

Nom du processus	Durée d'exécution	Priorité
P1	16	3
P2	4	1
P3	6	2

2. On donne ici les dates de soumission.
   (a) Priorité sans préemption.
   (b) Priorité avec préemption

Nom du processus	Date de soumission	Durée d'exécution	Priorité
P1	6	16	3
P2	4	4	1
P3	0	6	2

3. Avec un tourniquet et un quantum de temps de 2 unités.

Nom du processus	Durée d'exécution	Ordre de soumission
P1	16	1
P2	4	2
P3	6	3

Exercice 2

1. On soumet au système quatre processus P1, P2, P3 et P4 dont les durées d'exécution sont données par le tableau suivant :

Processus	Durée d'exécution
P1	6
P2	8
P3	7
P4	3

(a) Donner le diagramme de *Gantt* de l'algorithme du travail le plus court d'abord, puis du premier arrivé premier sorti.

(b) Donner les temps moyen d'attente des processus.

2. On dispose de 5 processus ayant des priorités différentes, comme le montre ce tableau :

Processus	Durée d'exécution	Priorité
P1	10	2
P2	1	4
P3	2	2
P4	1	1
P5	5	3

Donner le diagramme de *Gantt* pour l'algorithme de travail avec priorité.

3. On dispose de 3 processus P1, P2 et P3 ayant comme durée d'exécution respectivement 24, 3 et 3 ms. En utilisant un algorithme *Round Robin*, avec un quantum de 4 ms, donner le diagramme de Gantt.

Note: La performance de l'algorithme *Round Robin* dépend largement de la taille du quantum. Si le quantum est très grand , la politique *Round Robin* serait similaire à celle du FCFS. Si le quantum est très petit, la méthode *Round Robin* permettrait un partage du processeur : chacun des utilisateurs aurait l'impression de disposer de son propre processeur.

Cependant le quantum doit être choisi de sorte à ne pas surcharger le système par de fréquentes commutations de contexte.

Exercice 3 On considère les processus suivants, définis par leur durée d'exécution estimée, leur date d'arrivée et leur priorité initiale. Les plus grandes valeurs de priorité indiquent les processus les plus prioritaires.

Proc.	Date	Durée	Priorité
P1	0	8	2
P2	0	1	1
P3	2	2	1
P4	4	6	4
P5	6	1	3

1. Dessinez un diagramme de Gantt correspondant au résultat d'un ordonnancement FIFO non préemptif et indiquez le temps d'attente moyen.
2. Dessinez le diagramme de Gantt correspondant au résultat d'un ordonnancement par priorité préemptif et indiquez le temps d'attente moyen.
3. Dessinez le diagramme de Gantt correspondant au résultat d'un ordonnancement plus court d'abord préemptif et indiquez le temps d'attente moyen.
4. Dessinez le diagramme de Gantt correspondant au résultat d'un ordonnancement *round-robin* préemptif, sans priorité, avec un quantum de temps fixé à 2 et indiquez le temps d'attente moyen.
5. Quel est le meilleur algorithme suivant le critére du temps d'attente moyen ? Qui minimise le temps d'attente maximum ?

Exercice 4 Tracer pour chaque scénario le schéma d'allocation des ressources puis préciser pour quel(s) scénario on atteint la situation d'interblocage (*extrait d'un sujet de BAC 2022*).

Scénario 1
P1 acquiert R1
P2 acquiert R2
P3 attend R1
P2 libère R2
P2 attend R1
P1 libère R1

Scénario 2
P1 acquiert R1
P2 acquiert R3
P3 acquiert R2
P1 attend R2
P2 libère R3
P3 attend R1

Scénario 3
P1 acquiert R1
P2 acquiert R2
P3 attend R2
P1 attend R2
P2 libère R2
P3 acquiert R2

Exercice 5 Sept processus P_i sont dans la situation suivante par rapport aux ressources R_i:

- P1 a obtenu R1 et demande R2;
- P2 demande R3 et n'a obtenu aucune ressource;
- P3 demande R2 et n'a obtenu aucune ressource;
- P4 a obtenu R2 et R4 et demande R3;
- P5 a obtenu R3 et demande R5;
- P6 a obtenu R6 et demande R2;
- P7 a obtenu R5 et demande R2;

Construire un graphe d'allocation des ressources et vérifier s'il y a interblocage.

Exercice 5 Adapté d'un sujet de BAC Cet exercice traite du thème architecture matérielle, et plus particulièrement des processus et leur ordonnancement.

1. Avec la commande `ps -aef` on obtient l'affichage suivant :

```
PID  PPID  C STIME TTY          TIME CMD

8600     2  0 17:38 ?        00:00:00 [kworker/u2:0-fl]
8859     2  0 17:40 ?        00:00:00 [kworker/0:1-eve]
8866     2  0 17:40 ?        00:00:00 [kworker/0:10-ev]
8867     2  0 17:40 ?        00:00:00 [kworker/0:11-ev]
8887  6217  0 17:40 pts/0    00:00:00 bash
9562     2  0 17:45 ?        00:00:00 [kworker/u2:1-ev]
9594     2  0 17:45 ?        00:00:00 [kworker/0:0-eve]
9617  8887 21 17:46 pts/0    00:00:06 /usr/lib/firefox/firefox
9657  9617 17 17:46 pts/0    00:00:04 /usr/lib/firefox/firefox -contentproc -childID
9697  9617  4 17:46 pts/0    00:00:01 /usr/lib/firefox/firefox -contentproc -childID
9750  9617  3 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox -contentproc -childID
9794  9617 11 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox -contentproc -childID
9795  9794  0 17:46 pts/0    00:00:00 /usr/lib/firefox/firefox
9802  7441  0 17:46 pts/2    00:00:00 ps -aef
```

Figure 1.5: Capture d'un terminal

   (a) Donner sous forme d'un arbre de `PID` la hiérarchie des processus liés à *Firefox*.
   (b) Indiquer la commande qui a lancé le premier processus de firefox.

(c) La commande `kill` permet de supprimer un processus à l'aide de son `PID`. Indiquer la commande qui permettra de supprimer tous les processus liés à *Firefox* et uniquement cela.

2. (a) Recopier et compléter le schéma ci-dessous avec les termes suivants concernant l'ordonnancement des processus : *Élu, En attente, Prêt, Blocage, Déblocage, Mise en exécution.*

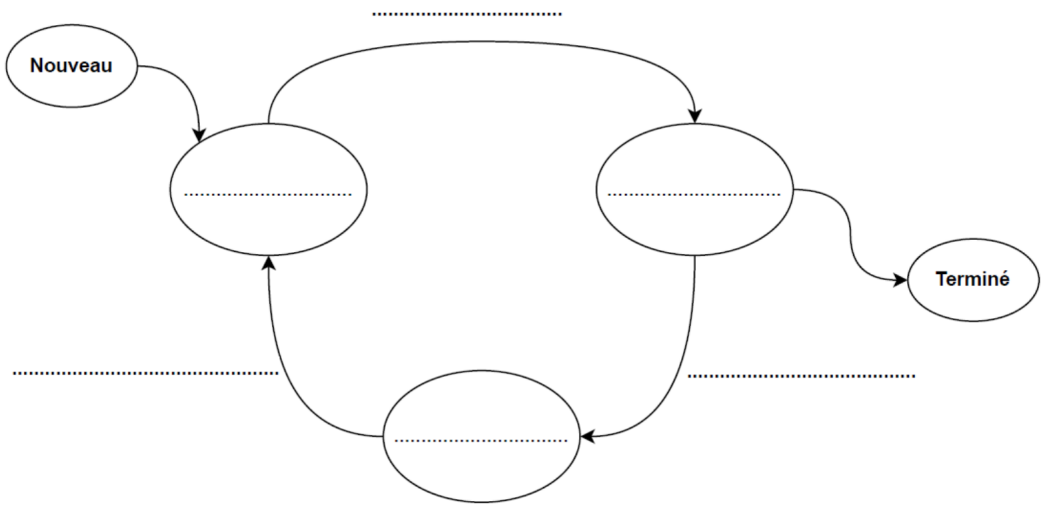


Figure 1.6: Compléter le schéma des états d'un processus

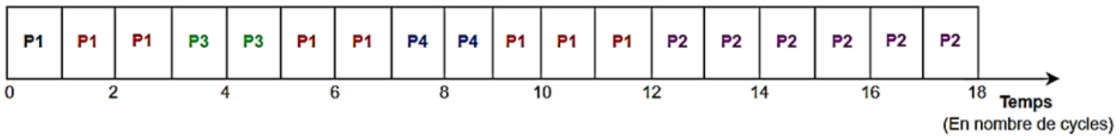
On donne dans le tableau ci-dessous quatre processus qui doivent être exécutés par un processeur. Chaque processus a un instant d'arrivée et une durée, donnés en nombre de cycles du processeur.

Processus	P1	P2	P3	P4
Instant d'arrivée	0	2	3	7
Durée	8	6	2	2

Les processus sont placés dans une file d'attente en fonction de leur instant d'arrivée. On se propose d'ordonnancer ces quatre processus avec la méthode suivante :

- Parmi les processus présents en liste d'attente, l'ordonnanceur choisit celui dont la durée restante est la plus courte ;
- Le processeur exécute un cycle de ce processus puis l'ordonnanceur désigne de nouveau le processus dont la durée restante est la plus courte ;
- En cas d'égalité de temps restant entre plusieurs processus, celui choisi sera celui dont l'instant d'arrivée est le plus ancien ;
- Tout ceci jusqu'à épuisement des processus en liste d'attente.

On donne en exemple ci-dessous, l'ordonnancement des quatre processus de l'exemple précédent suivant l'algorithme ci-dessus.

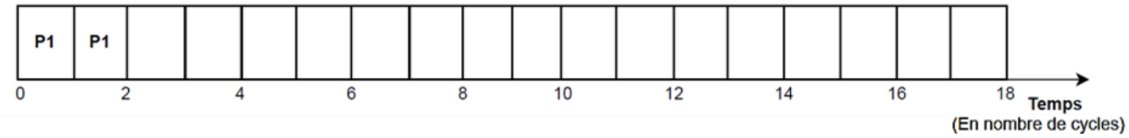


On définit le temps d'exécution d'un processus comme la différence entre son instant de terminaison et son instant d'arrivée.

(b) Calculer la moyenne des temps d'exécution des quatre processus. On se propose de modifier l'ordonnancement des processus. L'algorithme reste identique à celui présenté

précédemment mais au lieu d'exécuter un seul cycle, le processeur exécutera à chaque fois deux cycles du processus choisi. En cas d'égalité de temps restant, l'ordonnanceur départagera toujours en fonction de l'instant d'arrivée.

(c) Recopier et compléter le schéma ci-dessous donnant le nouvel ordonnancement des quatre processus.



(d) Calculer la nouvelle moyenne des temps d'exécution des quatre processus et indiquer si cet ordonnancement est plus performant que le précédent.

Exercice 6

Cet exercice porte sur la gestion des processus et des ressources par un système d'exploitation.

Les parties A et B peuvent être traitées indépendamment.

A. Ordonnancement des processus

Dans le laboratoire d'analyse médicale d'un hôpital, plusieurs processus peuvent demander l'allocation du processeur simultanément.

Le tableau ci-dessous donne les demandes d'exécution de quatre processus et indique :

- le temps d'exécution du processus (en unité de temps) ;
- l'instant d'arrivée du processus sur le processeur (en unité de temps) ;
- le numéro de priorité du processus (classé de 1 à 10).

Plus la priorité est grande plus le numéro de priorité est petit.
Ainsi le processus P3, du tableau ci-dessous, est plus prioritaire que le processus P1.
L'ordonnancement est de type préemptif, ce qui signifie qu'à chaque unité de temps, le processeur choisit d'exécuter le processus ayant le plus petit numéro de priorité (un seul processus à la fois). Ceci peut provoquer la suspension d'un autre processus qui reprendra lorsqu'il deviendra le plus prioritaire dans la file d'attente.

Processus	**Temps d'exécution**	**Instant d'arrivée**	**Numéro de priorité**
P1	3	0	4
P2	4	2	2
P3	3	3	1
P4	4	5	3

1. Reproduire le diagramme ci-dessous, sur votre copie, et indiquer dans chacune des cases le processus exécuté par le processeur entre deux unités de temps (il peut y avoir des cases vides).

2. Recopier et compléter les temps de séjour ainsi que les temps d'attente de chacun des processus (toujours en unités de temps).

$$Temps\ de\ séjour\ =\ instant\ de\ terminaison - instant\ d'arrivée$$
$$Temps\ d'attente\ =\ temps\ de\ séjour - temps\ d'exécution$$

Processus	Temps d'exécution	Instant d'arrivée	Temps de séjour	Temps d'attente
P1	3	0	14 – 0 = 14	14 – 3 = 11
P2	4	2		
P3	3	3		
P4	4	5		

3. À quelles conditions le temps d'attente d'un processus peut-il être nul ?