Architecture et OS	Chap 02	Processus

2.1 Dompter le métal et l'électricité

Qu'est-ce qu'un ordinateur ?

Un **ordinateur** est une machine électronique capable de **recevoir des informations**, **traiter ces informations**, **mémoriser des données** et **produire des résultats**.

Pour fonctionner, un ordinateur est constitué de plusieurs éléments matériels :

— un **processeur (CPU)** qui exécute les programmes, fait des calculs et manipule les données ;

— de la **mémoire** qui stocke les données en cours d'utilisation ;

— des **périphériques** permettant d'interagir avec l'utilisateur ou avec d'autres machines (clavier, écran, souris, disque dur, carte réseau...).

— un **stockage** (disque dur, SSD, clé USB...) qui conserve durablement les programmes et les fichiers ;

On peut donc voir un ordinateur comme une machine qui exécute en permanence des **programmes informatiques** afin de rendre des services à l'utilisateur.

Un joyeux bazar

Sur un ordinateur, de nombreux programmes peuvent fonctionner en "simultané", un navigateur web, un logiciel de traitement de texte, un lecteur de musique, un jeu vidéo etc ....

On peut alors se poser la question suivante :

Comment tous ces programmes peuvent-ils fonctionner ensemble sans se gêner ni provoquer de dysfonctionnements ?

Par exemple :

— comment partager le **temps du processeur** entre plusieurs programmes ?

— comment éviter qu'un programme **modifie les données d'un autre** ?

— comment gérer l'accès aux **périphériques** (écran, clavier, disque...) ?

Pour répondre à ces problèmes, les ordinateurs utilisent un programme particulier :

le système d'exploitation.

Définition 1 — Système d'exploitation

Un **système d'exploitation** est un ensemble de programmes qui permet d'utiliser un ordinateur. Il joue le rôle d'**intermédiaire** entre :

- l'utilisateur ;
- les applications (navigateurs, jeux, logiciels...) ;
- le matériel (processeur, mémoire, périphériques...).

Son objectif est d'assurer un fonctionnement **cohérent et efficace** de l'ensemble.

Exemple 1 — Systèmes d'exploitation

Parmi les systèmes d'exploitation les plus connus, on peut citer : **Windows**, **GNU/Linux**, **macOS**, **Android** ou encore **iOS**.

Organisation générale

Le système d'exploitation coordonne notamment :

- l'**exécution des programmes** ;
- le **partage du processeur** entre différentes tâches ;
- l'**utilisation de la mémoire** ;
- la **gestion des fichiers** ;
- le **pilotage des périphériques** comme l'écran ou le clavier.

Sans lui, chaque programme devrait dialoguer directement avec le matériel, ce qui rendrait le fonctionnement d'un ordinateur très complexe et peu fiable.

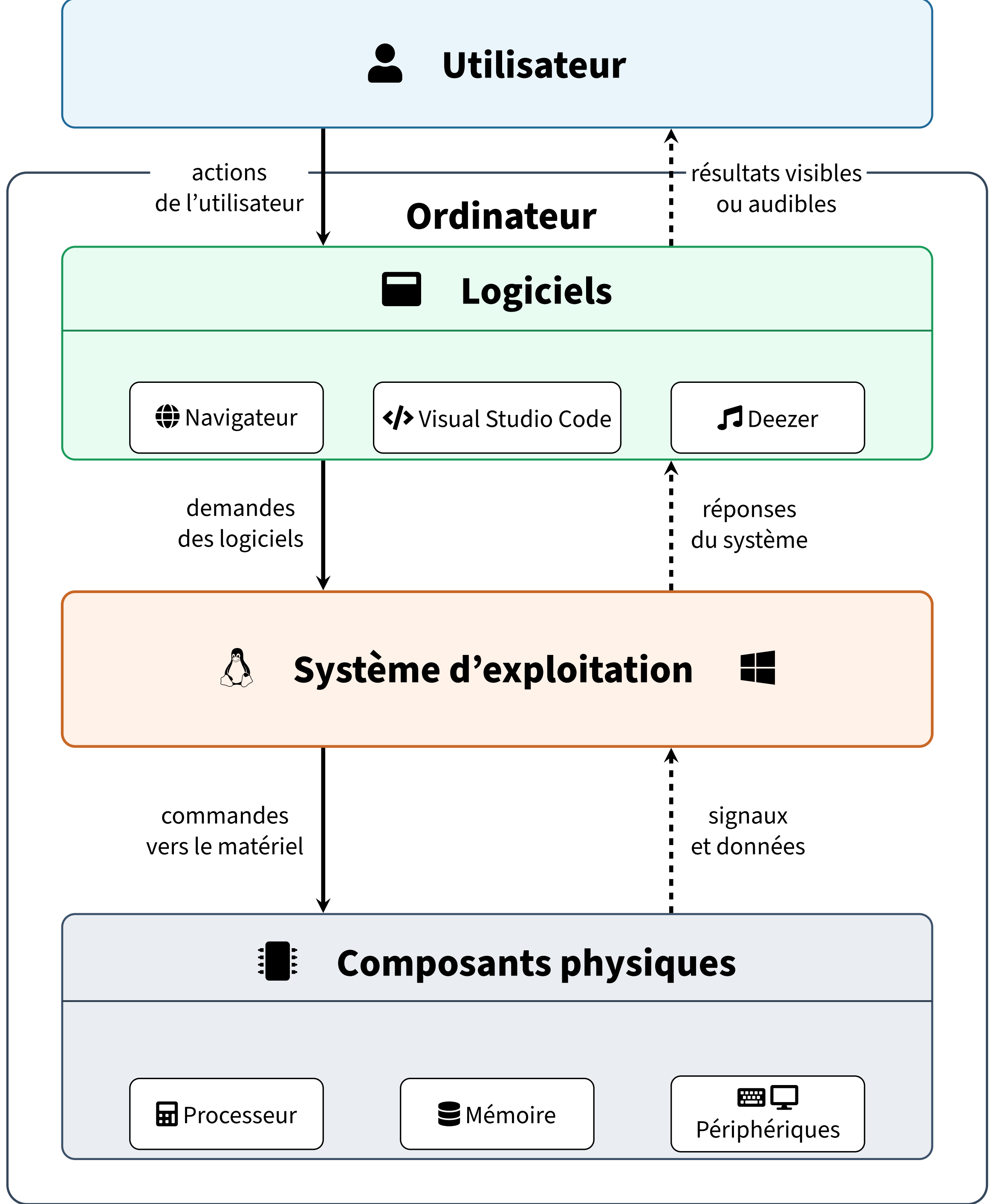


FIGURE 1 – Schéma global d'interaction utilisateur / ordinateur

2.2 Les programmes en action : les processus

2.2.1 Généralités sur les processus

Du programme au processus

Un logiciel installé sur un ordinateur est initialement un **programme**, c'est-à-dire un ensemble d'instructions stockées sous forme de fichier.
Lorsqu'un utilisateur lance ce programme, le système d'exploitation le charge en mémoire et organise son exécution.
On parle alors d'un **processus** : il s'agit du programme **en cours d'exécution**, accompagné des ressources nécessaires à son fonctionnement.

Définition 2 — Processus

Un **processus** est une instance d'un programme en cours d'exécution.
Il est caractérisé notamment par :

- un espace mémoire qui lui est propre,
- un état d'exécution,
- des ressources associées (fichiers ouverts, périphériques...) ;
- un identifiant unique

Exemple 2 — Programme et processus

Le fichier `python.exe` est un **programme**.
Lorsque l'on exécute un script Python, le système crée un **processus** chargé d'exécuter les instructions du programme.
Si plusieurs scripts Python sont lancés simultanément, plusieurs processus distincts sont créés.

C'est comme en cuisine

Un **programme**, c'est comme une **recette de gâteau** puisqu'il s'agit d'une suite d'instructions écrites sur une feuille. Tant que personne ne la suit, il ne se passe rien.

Un **cuisinier qui prépare réellement le gâteau** correspond à un **processus** : il applique les instructions, utilise des ingrédients, réquisitionne des ustensiles et avance étape par étape.

Dans une même cuisine, plusieurs cuisiniers peuvent utiliser **la même recette** :

- l'un peut avoir déjà terminé son gâteau,
- un autre peut être en train de mélanger la pâte,
- un autre peut devoir attendre que la farine ou une casserole se libère.

Chaque cuisinier représente alors un **processus différent**, même si tous exécutent le **même programme**.

Plusieurs processus en même temps

Sur un ordinateur moderne, de nombreux processus existent simultanément :

— les applications visibles par l'utilisateur ;

— des services du système d'exploitation ;

— des programmes exécutés en arrière-plan.

L'utilisateur a souvent l'impression que toutes ces tâches s'exécutent en même temps.
En réalité, le processeur exécute les instructions **une par une**, mais il change très (très) rapidement de processus.

Définition 3 — Multitâche

Un système d'exploitation est dit **multitâche** lorsqu'il est capable de gérer l'exécution concurrente de plusieurs processus en partageant le temps du processeur entre eux.

Remarque 1. Sur un ordinateur possédant plusieurs cœurs de processeur, plusieurs instructions peuvent réellement être exécutées en parallèle.

2.2.2 Etats d'un processus

Définition 4 — États d'un processus

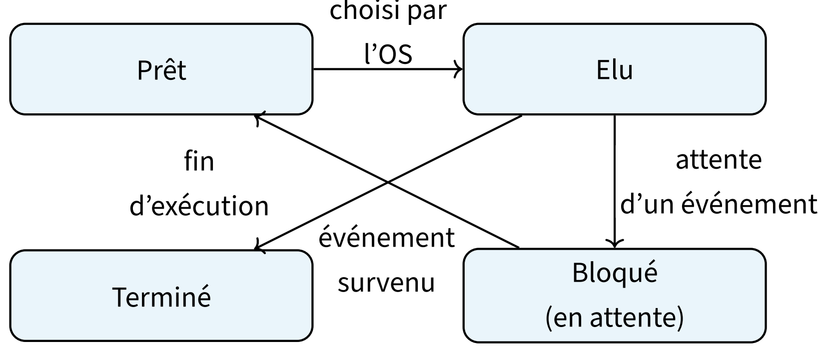
Au cours de son exécution, un processus passe par différents **états**.
Le système d'exploitation gère en permanence les transitions entre ces états afin d'assurer le bon fonctionnement global de l'ordinateur.
On distingue généralement les états suivants :

— **Prêt** : le processus est prêt à s'exécuter mais attend que le processeur soit disponible ;

— **Elu** : le processeur exécute actuellement ses instructions ;

— **Bloqué (ou en attente)** : le processus attend un événement extérieur (par exemple la fin d'une lecture sur le disque ou une saisie clavier) ;

— **Terminé** : l'exécution du processus est achevée.



Exercice 1 — Programme ou processus

Pour chaque situation, indiquer s'il s'agit d'un **programme** ou d'un **processus**.

1. Le fichier `firefox.exe` présent sur le disque.
2. Une fenêtre du navigateur actuellement ouverte.
3. Un fichier `python`.
4. Un script python en cours d'exécution.

Exercice 2 — États d'un processus

Associer chaque situation à l'état correspondant : **prêt**, **élu**, **bloqué**, **terminé**.

1. Un programme attend que l'utilisateur saisisse du texte.
2. Un processus utilise actuellement le processeur.
3. Un programme vient de finir son calcul.
4. Un processus observe de loin son voisin utiliser le processeur.

2.2.3 Création de processus

Création d'un processus

Un processus est généralement **créé** lorsqu'un utilisateur ou un programme demande l'exécution d'un autre programme.
Le système d'exploitation prépare alors tout ce qui est nécessaire à son exécution : espace mémoire, ressources, état initial et identifiant.

Un processus peut également **créer lui-même un nouveau processus** le processus initial est appelé **processus père**, et le nouveau processus créé est appelé **processus fils**.

Exemple 3 — Création

Lorsque l'on ouvre un terminal et que l'on lance la commande :

```
python mon_script.py
```

le terminal demande au système d'exploitation de créer un **nouveau processus** chargé d'exécuter l'interpréteur Python.
Ce nouveau processus fonctionne alors de manière indépendante du terminal, même si celui-ci reste ouvert.

Processus et threads

Un processus peut être vu comme une **unité d'exécution indépendante** possédant ses propres ressources.
Cependant, un même processus peut contenir plusieurs unités d'exécution plus légères, appelées **threads**.

Les threads d'un même processus :

- partagent le même espace mémoire ;
- peuvent s'exécuter de manière concurrente ;
- permettent d'effectuer plusieurs tâches en parallèle à l'intérieur d'un même programme.

Exemple 4 — Threads

Par exemple, dans un navigateur web :

- un thread peut afficher une page ;
- un autre peut charger des images ;
- un autre peut exécuter du code JavaScript.

2.2.4 Identifier un processus

Définition 5 — Identifiant de processus — PID

Chaque processus possède un **identifiant unique**, appelé **PID** (pour *Process Identifier*).
Il s'agit d'un nombre entier attribué par le système d'exploitation lors de la création du processus.
Le système maintient un compteur interne qui permet d'attribuer un nouveau PID à chaque processus créé.

Définition 6 — PPID

Chaque processus possède également un **PPID** (*Parent Process Identifier*), qui correspond au PID du processus qui l'a créé.

Hiérarchie

On obtient donc une **hiérarchie de processus** :

- certains processus sont créés directement par le système d'exploitation au démarrage ;
- d'autres sont créés par des programmes déjà en cours d'exécution ;
- un processus peut lui-même créer plusieurs processus fils.

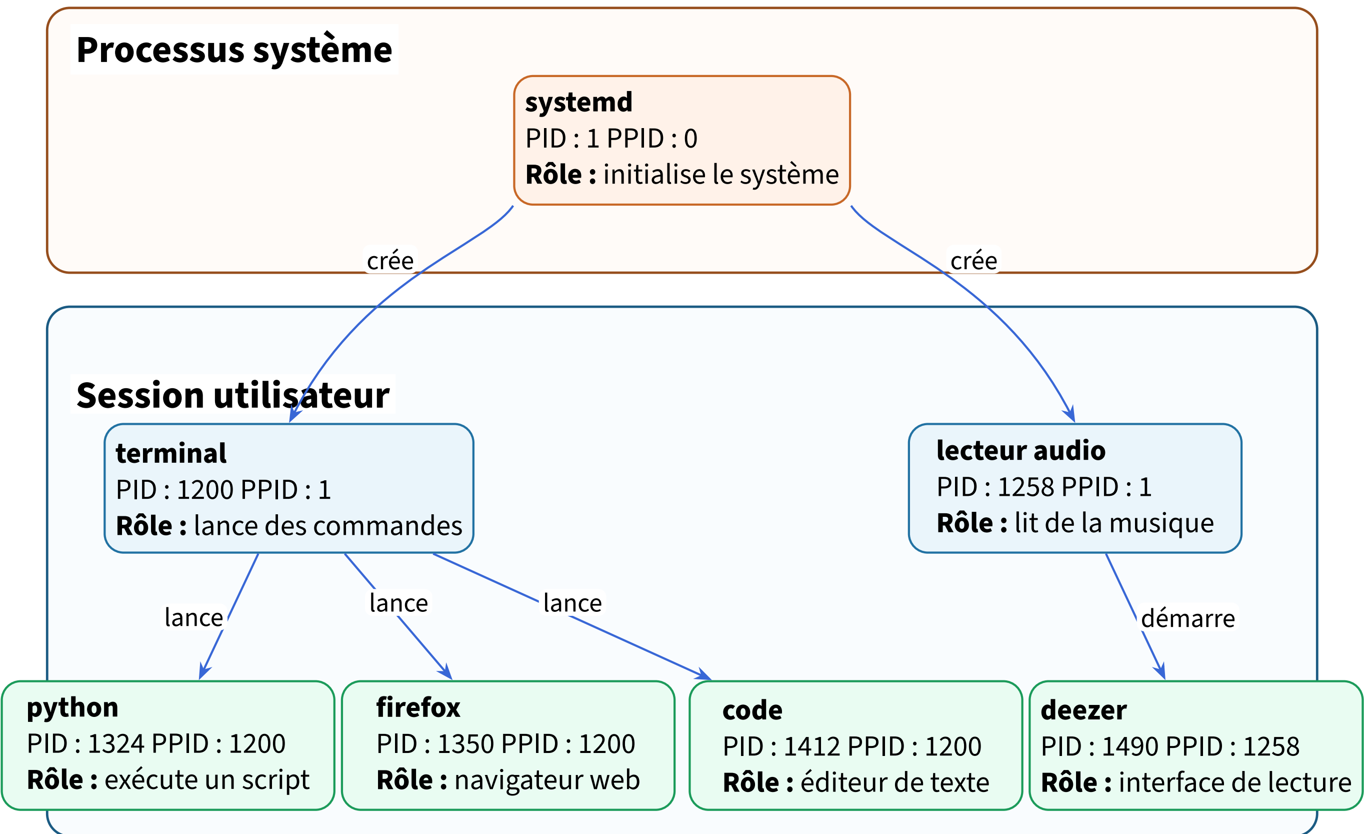


FIGURE 2 – Exemple d'une hiérarchie de processus

2.2.5 Fin de vie

Fin de vie d'un processus

Un processus n'est pas destiné à s'exécuter indéfiniment. Son exécution peut s'arrêter pour plusieurs raisons.
On distingue généralement plusieurs cas d'arrêt et on associe un **code de fin à chacun**

Fin normale (code 0)
Le processus termine l'exécution de son programme. Il libère alors les ressources qu'il utilisait (mémoire, fichiers ouverts, périphériques...).

Arrêt demandé par l'utilisateur
L'utilisateur peut décider de fermer une application (par exemple en cliquant sur une croix ou en utilisant un gestionnaire des tâches).
Le système d'exploitation demande alors au processus de s'arrêter.

Arrêt provoqué par une erreur
Un processus peut être interrompu en raison d'une erreur (par exemple une instruction invalide ou un accès mémoire interdit).

Arrêt forcé par le système d'exploitation
Le système peut décider d'arrêter un processus afin de préserver la stabilité ou la sécurité de l'ordinateur (par exemple si le processus consomme trop de ressources).

2.3 Ordonnancement des processus

2.3.1 Principe général

Pourquoi ordonnancer ?

Sur un ordinateur, plusieurs processus peuvent être prêts à s'exécuter en même temps.
Le processeur ne pouvant exécuter qu'un seul processus à la fois, le système d'exploitation doit décider **quel processus utilise le processeur à chaque instant**.
Cette décision s'appelle **l'ordonnancement**.

Définition 7 — Caractéristiques d'un processus

Dans toute la suite, chaque processus sera caractérisé par :

— son **temps d'arrivée** : instant où il devient prêt ;

— sa **durée d'exécution** : temps total nécessaire pour terminer.

Définition 8 — Instant de fin d'un processus

L'**instant de fin** d'un processus est le moment où ce processus termine complètement son exécution.

Définition 9 — Temps de séjour

Le **temps de séjour** est la durée entre l'arrivée du processus et la fin de son exécution.

Temps de séjour = Instant de fin - Temps d'arrivée

Définition 10 — Temps d'attente

Le **temps d'attente** est la durée pendant laquelle un processus est prêt mais n'utilise pas le processeur.

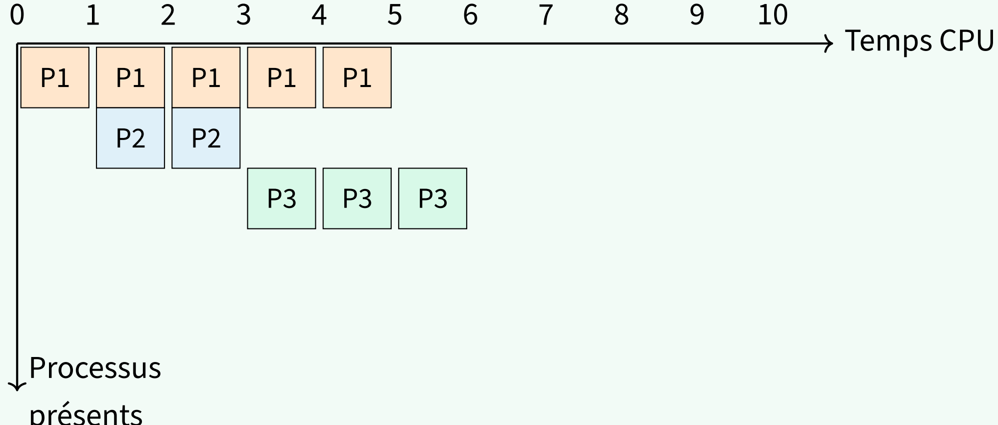
Temps d'attente = Temps de séjour - Durée d'exécution

Exemple 5 — Un ordonnancement possible

On considère les trois processus suivants :

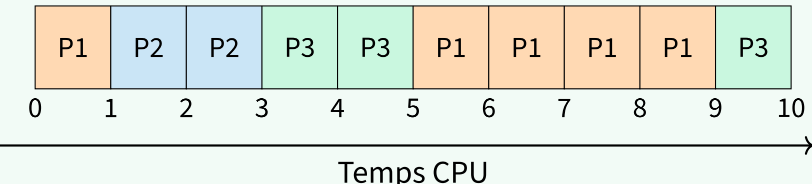
Processus	Temps d'arrivée	Durée
P1	0	5
P2	1	2
P3	3	3

1) Représentation des arrivées et des durées



2) Ordonnancement choisi

On suppose l'ordonnancement suivant :



3) Analyse des temps

Processus	Arrivée	Durée	Fin	Séjour	Attente
P1	0	5	9	$9-0=9$	$9-5=4$
P2	1	2	3	$3-1=2$	$2-2=0$
P3	3	3	10	$10-3=7$	$7-3=4$

$$\text{Temps d'attente moyen} = \frac{4+0+4}{3} = \frac{8}{3} \simeq 2{,}67$$

2.3.2 Ordonnancements non préemptifs

Principe

Dans un **ordonnancement non préemptif**, un processus qui commence son exécution **garde le processeur jusqu'à la fin**.
Le système d'exploitation ne peut pas l'interrompre pour donner la main à un autre processus.
Un nouveau choix d'ordonnancement n'est effectué que lorsqu'un processus se termine.

Ordonnancement FCFS (First Come First Served)

Définition 11 — Principe de FCFS

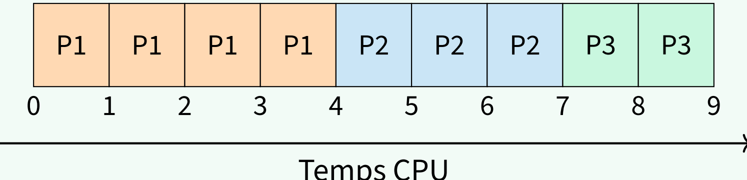
La stratégie **FCFS** (premier arrivé, premier servi) consiste à exécuter les processus **dans l'ordre de leur arrivée**.
Le premier processus arrivé obtient le processeur et l'utilise jusqu'à sa terminaison.

Exemple 6 — FCFS

On considère les processus suivants :

Processus	Temps d'arrivée	Durée
P1	0	4
P2	1	3
P3	2	2

Ordre d'exécution : **P1 → P2 → P3**



Analyse

Processus	Arrivée	Durée	Fin	Séjour	Attente
P1	0	4	4	4	0
P2	1	3	7	6	3
P3	2	2	9	7	5

$$\text{Attente moyenne} = \frac{0+3+5}{3} = 2{,}67$$

Ordonnancement SJF (Shortest Job First)

Définition 12 — Principe de SJF

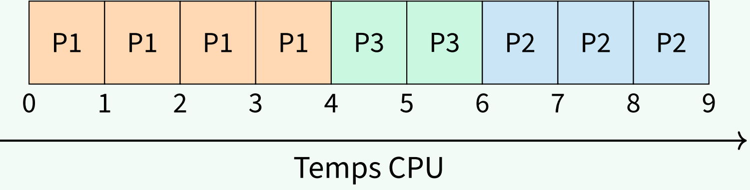
La stratégie **SJF** consiste à exécuter en priorité le processus dont la **durée d'exécution est la plus courte**.
Lorsque le processeur se libère, on choisit donc parmi les processus prêts celui qui nécessite le moins de temps CPU.

Exemple 7 — SJF

On reprend les mêmes processus.

Processus	Temps d'arrivée	Durée
P1	0	4
P2	1	3
P3	2	2

À l'instant 0 seul P1 est présent → il commence.
À l'instant 4, P2 et P3 sont prêts. On choisit donc **P3** car sa durée est plus courte.
Ordre d'exécution : **P1 → P3 → P2**



Temps CPU

Analyse

Processus	Arrivée	Durée	Fin	Séjour	Attente
P1	0	4	4	4	0
P2	1	3	9	8	5
P3	2	2	6	4	2

$$\text{Attente moyenne} = \frac{0+5+2}{3} = 2{,}33$$

Exercice 3 — Ordonnancements non préemptifs

On considère les processus suivants :

Processus	Temps d'arrivée	Durée
P1	0	6
P2	2	2
P3	3	4
P4	5	3

1. Représenter l'ordonnancement obtenu avec la stratégie **FCFS**.
2. Calculer pour chaque processus :
   — son instant de fin,
   — son temps de séjour,
   — son temps d'attente.
3. Calculer le **temps d'attente moyen**.
4. Refaire les mêmes questions avec la stratégie **SJF**.
5. Comparer les deux stratégies : laquelle semble la plus efficace dans cet exemple ?

2.3.3 Ordonnancements préemptifs

Principe

Dans un **ordonnancement préemptif**, le système d'exploitation peut **interrompre un processus en cours d'exécution** afin de donner le processeur à un autre processus.

Contrairement aux stratégies non préemptives, un processus qui a commencé à s'exécuter n'est donc pas certain de conserver le processeur jusqu'à la fin.

Cela permet par exemple :

— de réagir rapidement à l'arrivée d'un processus court ;
— d'éviter qu'un processus long monopolise le processeur ;
— d'améliorer la réactivité globale du système.

Définition 13 — Préemption

La **préemption** est le mécanisme par lequel le système d'exploitation **retire le processeur à un processus en cours d'exécution** avant qu'il ait terminé.
Le processus interrompu repasse alors dans l'état **prêt** et pourra reprendre plus tard.

Ordonnancement SRT (Shortest Remaining Time)

Définition 14 — Principe de SRT

La stratégie **SRT** (*Shortest Remaining Time*) est la version préemptive de SJF.

À chaque instant, on choisit parmi les processus prêts celui dont la **durée restante d'exécution** est la plus courte.
Si un nouveau processus arrive avec une durée restante plus petite que celle du processus en cours, alors le système interrompt le processus courant et exécute le nouveau.

Exemple 8 — SRT

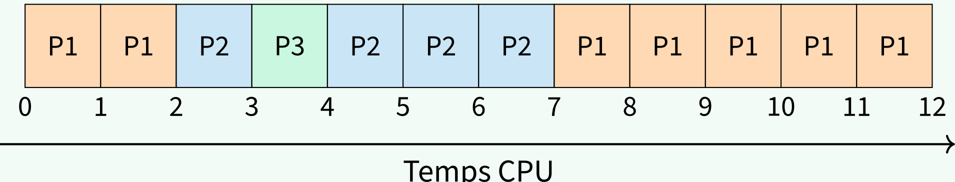
On considère les processus suivants :

Processus	Temps d'arrivée	Durée
P1	0	7
P2	2	4
P3	3	1

Analyse pas à pas

- À l'instant 0, seul **P1** est présent : il commence.
- À l'instant 2, **P2** arrive. P1 a encore **5** unités à exécuter, P2 en demande **4**. On choisit donc **P2**, qui **interrompt P1**.
- À l'instant 3, **P3** arrive. P2 a encore **3** unités à exécuter, tandis que P3 ne demande que **1**. On choisit donc **P3**, qui **interrompt P2**.
- Une fois P3 terminé, il reste P1 (5 unités) et P2 (3 unités). On choisit donc **P2**, puis enfin **P1**.

Ordonnancement obtenu : P1 $\rightarrow$ P2 $\rightarrow$ P3 $\rightarrow$ P2 $\rightarrow$ P1



Analyse des temps

Processus	Arrivée	Durée	Fin	Séjour	Attente
P1	0	7	12	$12-0=12$	$12-7=5$
P2	2	4	7	$7-2=5$	$5-4=1$
P3	3	1	4	$4-3=1$	$1-1=0$

$$\text{Temps d'attente moyen} = \frac{5+1+0}{3} = 2$$

Remarque 2. La stratégie SRT favorise souvent les processus courts, ce qui peut réduire le temps d'attente moyen.

En revanche, un processus long peut être interrompu de nombreuses fois si de nouveaux processus plus courts arrivent régulièrement.

Ordonnancement Round Robin

Définition 15 — Principe de Round Robin

La stratégie **Round Robin** consiste à faire passer les processus **chacun leur tour** sur le processeur.

Chaque processus reçoit le processeur pendant une durée maximale fixée, appelée **quantum**.

Si le processus n'a pas terminé à la fin de son quantum :

— il est interrompu ;

— il retourne dans la file des processus prêts ;

— le processus suivant obtient le processeur.

Justice !

Round Robin cherche avant tout à assurer une forme d'**équité** : aucun processus ne garde trop longtemps le processeur.
C'est une stratégie bien adaptée lorsque l'on souhaite que chaque programme réagisse rapidement.

Exemple 9 — Round Robin

On considère les processus suivants :

Processus	Temps d'arrivée	Durée
P1	0	5
P2	0	3
P3	0	4

On choisit un **quantum de 2 unités de temps**.

Pas à pas

— P1 s'exécute de 0 à 2 : il lui reste 3 unités.

— P2 s'exécute de 2 à 4 : il lui reste 1 unité.

— P3 s'exécute de 4 à 6 : il lui reste 2 unités.

— P1 s'exécute de 6 à 8 : il lui reste 1 unité.

— P2 s'exécute de 8 à 9 : il termine.

— P3 s'exécute de 9 à 11 : il termine.

— P1 s'exécute de 11 à 12 : il termine.

Analyse des temps

Processus	Arrivée	Durée	Fin	Séjour	Attente
P1	0	5	12	$12 - 0 = 12$	$12 - 5 = 7$
P2	0	3	9	$9 - 0 = 9$	$9 - 3 = 6$
P3	0	4	11	$11 - 0 = 11$	$11 - 4 = 7$

$$\text{Temps d'attente moyen} = \frac{7+6+7}{3} = \frac{20}{3} \simeq 6{,}67$$

Remarque 3. Avec **Round Robin**, le choix du **quantum** est important :

— si le quantum est **trop grand**, la stratégie se rapproche de FCFS ;

— si le quantum est **trop petit**, le processeur change très souvent de processus, ce qui peut dégrader les performances.

Exercice 4 — Ordonnancements préemptifs

On considère les processus suivants :

Processus	Temps d'arrivée	Durée
P1	0	6
P2	1	3
P3	4	2

1. Représenter l'ordonnancement obtenu avec la stratégie **SRT**.
2. Calculer pour chaque processus :
   - son instant de fin ;
   - son temps de séjour ;
   - son temps d'attente.
3. Calculer le **temps d'attente moyen**.
4. Refaire les mêmes questions avec la stratégie **Round Robin** en prenant un **quantum de 2**.
5. Comparer les deux stratégies : laquelle semble la plus favorable dans cet exemple ?

2.4 L'interblocage

2.4.1 Quand tout le monde attend

Une situation de blocage mutuel

Dans un système informatique, plusieurs processus peuvent avoir besoin d'accéder à des **ressources** pour continuer leur exécution, une imprimante, un fichier, une zone mémoire etc..

Il peut alors arriver qu'un processus attende une ressource détenue par un autre processus, qui attend lui-même une autre ressource.
Dans certains cas, cette attente peut former un **cercle** dans lequel plus aucun processus ne peut avancer.

Définition 16 — Ressource

Une **ressource** est un élément dont un processus peut avoir besoin pour s'exécuter.
Une ressource peut être :

— **matérielle** : imprimante, disque, processeur, périphérique ;

— **logicielle** : fichier, zone mémoire, verrou, accès à une donnée partagée.

Définition 17 — Interblocage

On parle d'**interblocage** lorsqu'un ensemble de processus reste bloqué de manière permanente, chacun attendant une ressource détenue par un autre processus du même ensemble.

Aucun de ces processus ne peut alors poursuivre son exécution.

Exemple 10 — Avec deux ressources

Considérons deux processus :

— **P1** possède la ressource **R1** et attend **R2** ;

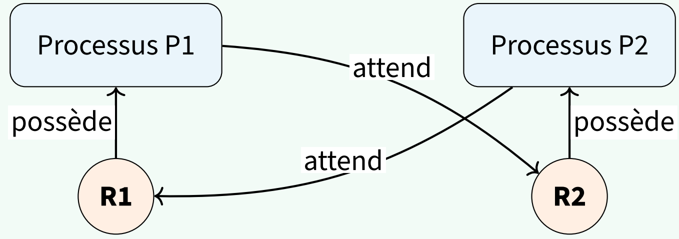
— **P2** possède la ressource **R2** et attend **R1**.

Alors :

— P1 ne peut pas continuer tant que R2 ne se libère pas ;

— P2 ne peut pas continuer tant que R1 ne se libère pas.

Aucun des deux ne progresse : il y a **interblocage**. On voit un **cycle** sur le graphe suivant.



2.4.2 Représenter un interblocage

Graphe d'allocation des ressources

Pour analyser les situations de blocage entre processus, on peut utiliser une représentation appelée **graphe d'allocation des ressources**.

Dans ce type de graphe :

— chaque **processus** est représenté par un nœud;

— chaque **ressource** est représentée par un nœud;

— une flèche d'un processus vers une ressource signifie que le processus **demande** cette ressource;

— une flèche d'une ressource vers un processus signifie que cette ressource est **attribuée** à ce processus.

Exemple 11 — Exemple avec trois processus

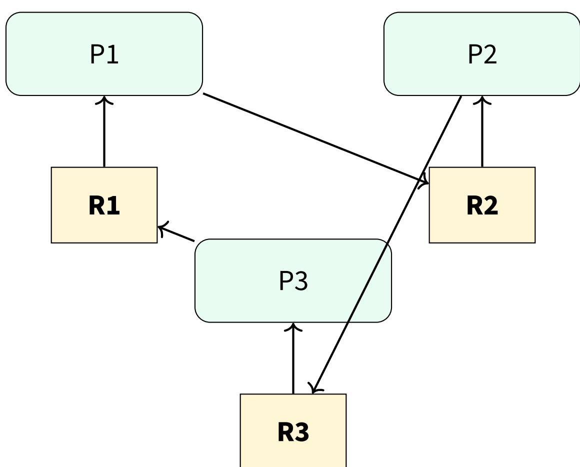
On considère la situation suivante :

— le processus **P1** possède la ressource **R1** et demande la ressource **R2**;

— le processus **P2** possède la ressource **R2** et demande la ressource **R3**;

— le processus **P3** possède la ressource **R3** et demande la ressource **R1**.

On peut représenter cette situation par le graphe suivant :



Lecture du graphe

On obtient donc une **attente circulaire** :

$$P1 \rightarrow P2 \rightarrow P3 \rightarrow P1$$

Aucun processus ne peut libérer sa ressource, car chacun attend une ressource détenue par un autre.
Il y a donc **interblocage**.

2.4.3 Les quatre conditions d'apparition

Propriété 1 (Conditions classiques)**.** Un interblocage ne peut apparaître que si les 4 conditions suivantes sont réunies simultanément

Définition 18 — Exclusion mutuelle

Une ressource est en **exclusion mutuelle** lorsqu'elle ne peut être utilisée que par un seul processus à la fois.

Par exemple, une imprimante ne peut pas imprimer deux documents différents exactement au même instant.

Définition 19 — Possession et attente

La condition de **possession et attente** signifie qu'un processus détient déjà au moins une ressource tout en attendant une autre ressource.

Autrement dit, il garde ce qu'il possède sans le libérer, tout en réclamant autre chose.

Définition 20 — Absence de réquisition

La condition d'**absence de réquisition** signifie qu'une ressource ne peut pas être retirée de force à un processus.

Seul le processus qui la détient peut la libérer volontairement.

Définition 21 — Attente circulaire

Il y a **attente circulaire** lorsqu'on peut former une chaîne de processus telle que chacun attend une ressource détenue par le suivant.

Dans un cas simple :

$$P1 \text{ attend } P2, \quad P2 \text{ attend } P3, \quad P3 \text{ attend } P1$$

2.4.4 Éviter ou résoudre un interblocage

Que peut faire le système ?

Face au risque d'interblocage, plusieurs approches sont possibles :

- **prévenir** l'interblocage grâce à des règles d'attribution ;
- **le détecter** puis intervenir ;
- parfois, dans les systèmes simples, **ignorer le problème** s'il est très rare.

Prévention

Définition 22 — Prévention

La **prévention** consiste à organiser le système de manière à rendre impossible au moins une des quatre conditions de l'interblocage.

Exemple 12 — Exemple de prévention

On peut imposer un **ordre de demande des ressources**.
Par exemple, si tous les processus doivent toujours demander d'abord **R1** puis **R2**, alors la situation suivante devient impossible :

- P1 possède R2 et attend R1 ;
- P2 possède R1 et attend R2.

On casse ainsi la possibilité d'une attente circulaire.

Détection et récupération

Détection

La **détection** consiste à laisser le système fonctionner, puis à rechercher si un interblocage est apparu.

Lorsque le système détecte un interblocage, il doit ensuite **rétablir une situation normale**.
Cela peut se faire par exemple en :

- arrêtant un ou plusieurs processus ;
- libérant certaines ressources ;
- redémarrant une application.

Conséquence pratique

Mettre fin à un processus pour casser un interblocage peut entraîner une perte de travail non sauvegardé.
C'est pourquoi les systèmes essaient généralement d'éviter ce type de situation autant que possible.