Numérique et science informatique
Classe de Terminale

Lycée hoche

année scolaire 2025-2026

Contents

1 Introduction

1.1 Processus

Le système UNIX est multi-tâches: Il donne l'impression d'exécuter plusieurs traitements simultanément.(On peut par exemple à la fois consulter ses mails , exécuter un programme python sous *Pyzo* tout en écoutant de la musique sur le web).
Le processeur de la machine ne peut pourtant exécuter qu'une tâche (un *processus*) à la fois :on parle de *processus actif*.

Graphe non orienté

Définition 1 *Un processus est l'objet dynamique associé à un programme:*

- *Le programme 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 peut y avoir plusieurs processus associés à un même programme:Plusieurs processus permettent à un ordinateur d'effectuer plusieurs tâches à la fois. Ils se partagent les ressources physiques.*
- *L'ensemble de la mémoire associée à un processus est appelé son contexte.*

Un processus possède plusieurs caractéristiques:

- **CMD** : le programme qu'il exécute .
- **PID**: un numéro d'identification que lui affecte le système.
- **PPID**: numéro d'identification du père du processus
- **UID**: l'utilisateur pour lequel il fonctionne .
- **TTY**: le terminal ou la fenêtre du processus .
- **CPU**: une consommation CPU .
- **MEM**: une consommation mémoire .
- **TIME**: une durée de traitement .
- **STIME**: une heure de lancement.
- **C** : un facteur de priorité .

Pour observer ses caractéristiques on peut utiliser la commande ps:

```
rene@debian9:~$ ps
PID TTY          TIME CMD
3827 pts/0    00:00:00 bash
3832 pts/0    00:00:00 gedit
3842 pts/0    00:00:00 ps
```

Et avec l'option -l

```
dartagnan@dartagnan005:~$ ps -l
F S   UID    PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   3144    3061  0  80   0 -  3324 do_wai pts/0    00:00:00 bash
0 R  1000   4360    3144  0  80   0 -  3514 -      pts/0    00:00:00 ps
```

Comme on le voit dans le tableau précédent , le processus créé par la commande **ps** a un PID égal à 4360.
Qui est le père du processus créé par la commande *ps*?

Options de la commande *ps*: Dans **Terminal** , tapez la commande *man ps* pour voir les options de la commande *ps*.

1.2 Commandes UNIX de gestion des processus

- La commande *ps* vue plus haut:
  Liste des processus avec des options (voir exemples plus haut)
- La commande ***pstree*** affiche une arborescence des processus(pères,fis,..)
- Destruction d'un processus :

  ```
  utilisateur$ kill -9 n° processus
  ```

  ```
  utilisateur$ kill -9 521
  ```

- Lancement en arrière-plan d'un processus :
  nom_processus &

  ```
  utilisateur$ firefox &
  ```

- Affichage des processus en background :

  ```
  utilisateur$: jobs
  ```

- Changement de la priorité d'un processus : nice -valeur commande

  ```
  utilisateur$ : nice -5 gcc programme.c
  ```

- On peut lancer plusieurs processus à la suite à l'aide de **;**

  ```
  utilisateur$ ps -l; cat > alpha
  ```

- On peut également lancer plusieurs processus en parallèle à l'aide de |.
  Par exemple pour afficher toutes les lignes commençant par la lettre "o":

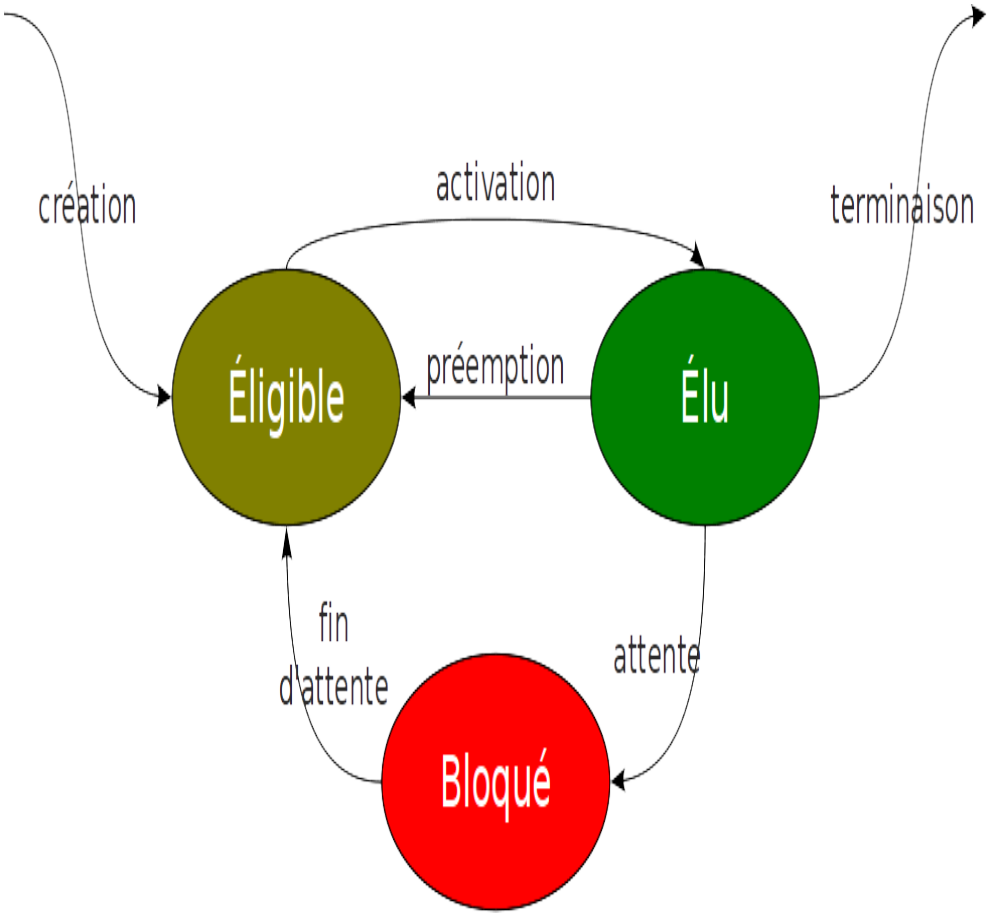
  ```
  utilisateur$  cat alpha | grep ^o
  ```

1.3 Les états d'un processus

On peut définir pour chaque processus un état parmi plusieurs .
On a mis entre parenthèses la lettre qui apparaît dans la colonne **STAT** ou **S** lors de l'affichage donné par la commande **ls -l**:

1. Etat **actif ou élu** (**R**): Le processeur exécute le processus.
   Comme nous l'avons dit plus haut, Il y a deux sortes d'exécution d'un processus:
   - Exécution de programmes initiés par l'utilisateur :c'est le **mode utilisateur**
   - Exécution d'appels système par le noyau (lecture ou écriture dans la mémoire par exemple).C'est le **mode noyau**.
2. Etat **éligible** ou **prêt** (**R**): le processus attend d'être exécuté par le processeur.
   Remarque: Sous Linux **R** désigne les états "lu" ou "éligible".
3. Etat **en attente** ou **endormi** ou **bloqué** (**S** (moins de 20s) ou **I** (plus de 20s)):Le processus attend un évènement.
4. Etat **mort** ou **zombi** (**Z**) :le processus s'est terminé ou a été interrompu.
5. Etat **suspendu** (**T**): Le processus est suspendu et est en attente d'un *signal* de reprise.

Etats d'un processus



(d'après https://perso.liris.cnrs.fr/pierre-antoine.champin/enseignement/se/_images/ps_states.png)

2 l'ordonnanceur

Dans un système multi-utilisateurs à temps partagé, plusieurs processus peuvent être présents en mémoire centrale en attente d'exécution. Si plusieurs processus sont prêts, le système d'exploitation doit gérer l'allocation du processeur aux différents processus à exécuter. C'est l'ordonnanceur qui s'acquitte de cette tâche.

2.1 Mode noyau - mode utilisateur

Le cycle de vie d'un processus est compliqué.En particulier,quand il utilise le processeur,un processus peut être en mode **noyau** ou en mode **utilisateur**:

- Le mode *noyau* correspond aux moments où le processus délègue au noyau du système son action quand celle-ci est critique:Par exemple quand on demande l'ouverture d'un fichier en lecture , le système doit vérifier les droits, c'est donc à lui de faire cette ouverture.
- Quand le processus fait des actions non critiques (par exemple une opération arithmétique)il reste en mode utilisateur.
- Un processus en mode *noyau* ne peut être interrompu , ceci afin de s'assurer que toutes les opérations visant à garantir l'intégrité des données du système ont bien été appliquées.

2.2 Rôle de l'ordonnanceur

- **L'ordonnanceur est un processus** qui joue un rôle important. Il porte le **numéro 0**.
- L'ordonnancement consiste à choisir le processus à exécuter à un instant t et à déterminer le temps durant lequel le processeur lui sera alloué .
- Il sélectionne à intervalles réguliers (de l'ordre du millième de seconde) le processus auquel affecter les ressources du processeur de l'unité centrale.

2.3 Contraintes et ordonnancement

Pour le moment nous n'avons pas vu les dépendances entre tâches :Une tâche qui ne peut pas démarrer avant qu'une autre termine.

Par exemple producteur-consommateur : certaines tâches consomment une ressource produite par d'autres et doivent donc attendre.
Plus précisément : quand deux processus dialoguent sur une socket,celui qui lit doit attendre que l'autre ait écrit quelque chose.Deux tâches ne doivent pas s'exécuter en même temps (exclusion mutuelle).

Par exemple, deux écritures dans un fichier ne peuvent pas avoir lieu en même temps.Le système doit proposer des moyens de bloquer des tâches.

Dès que l'on fait de la programmation multitâches, il faut tenir compte des contraintes.Une tâche peut être ralentie car elle ne parvient pas à obtenir une ressource.

Exemples:

- Exemple de mars pathfinder dont le système était réinitialisé suite à une inversion de priorité.

- Exemple de votre ordinateur fortement ralenti lors d'une copie de disque.

3 Types d'ordonnancement

L'ordonnanceur agit de plusieurs manières[1]:

1. Ordonnancement **non préemptif**(ou sans réquisition):

   L'ordonnanceur alloue le processeur au processus jusqu'à ce qu'il se termine ou qu'il se bloque (en attente d'un événement). Il n'y a pas de réquisition.En général:

   - Ordonnancement selon l'ordre d'arrivée : premier arrivé, premier servi (**FIFO**)
   - Ordonnancement selon la durée de calcul : travail le plus court d'abord (Shortest Job First ou **SJF**) (nécessité de connaître leur durée)
   - **A priorité** - les tâches ont des importances, on choisit la plus importante ("la plus prioritaire") :

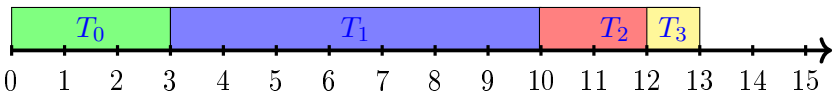
   Remarque : Dans tous les cas précédents , une tâche longue qui a accès au processeur le bloque pendant toute sa durée.

   Exemple: On considère 4 processus correspondant à 4 tâches T_0, T_1, T_2 et T_3 .On connaît leur durée , temps d'arrivée et leur priorité.

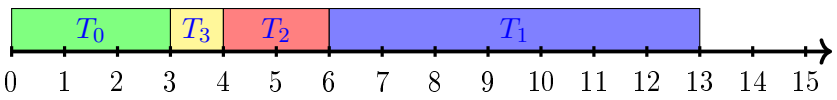
	T_0	T_1	T_2	T_3
Durée	3	7	2	1
Arrivée	0	1	2	2,5
Priorité	1	5	3	6

   On donne l'ordre d'exécution suivant la méthode utilisée :

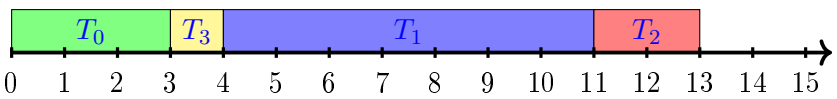
   FIFO

   SJF

   Priorités

2. Ordonnancement **préemptif**:

   Un système d'exploitation multitâche est préemptif lorsque celui-ci peut arrêter (réquisition) à tout moment n'importe quelle application pour passer la main à la suivante.

   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.

[1] La rédaction de cette partie a été largement inspirée par le cours de Audrey Quedet,Université de Nantes 2010(Miage)

Quelques algorithmes d'ordonnancement avec préemption:

- Ordonnancement selon la durée de calcul restante : temps restant le plus court d'abord Shortest Remaining Time (**SRT**)

  Un processus arrive dans la file de processus, l'ordonnanceur compare la valeur espérée (d'exécution) pour ce processus à la valeur du processus actuellement en exécution. Si le temps du nouveau processus est plus petit, il rentre en exécution immédiatement.
- Ordonnancement sans notion de priorité : temps-partagé avec politique du tourniquet (Round-Robin ou **RR**) (voir un exemple sur mathgreen.fr)
- Ordonnancement à priorités (statiques ou dynamiques) : la tâche la plus prioritaire obtient le processeur

3.1 Accès aux ressources

Plusieurs processus ont parfois (souvent ?) accès aux mêmes ressources.

Une **ressource** désigne toute entité dont a besoin un processus pour s'exécuter :

- Ressource matérielle (processeur, périphérique, etc)
- Ressource logicielle (variable).

Par ailleurs , il existe 3 phases pour l'exploitation d'une ressource par un processus :

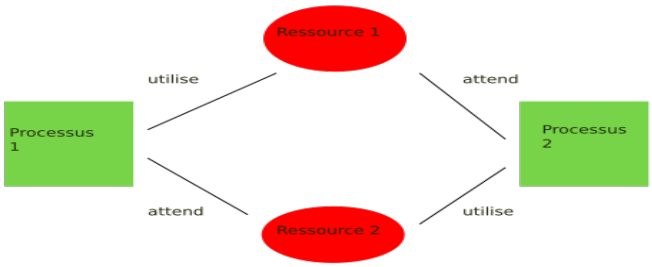
- Sollicitation de la ressource.
- Utilisation de la ressource.
- Libération de la ressource.

Une ressource est dite critique lorsque des accès concurrents à cette ressource peuvent mener à un état incohérent.

On parle aussi de situation de compétition (*race condition*) pour décrire une situation dont l'issue dépend de l'ordre dans lequel les opérations sont effectuées

3.2 dead-lock ou interblocage

Des tâches peuvent se bloquer irrémédiablement lorsqu'elles demandent les même ressources : dead-lock



Un dead-lock apparaît :

- Lorsque des tâches réservent plusieurs ressources et que l'ordre de réservation permet un cycle.
- Lorsque des tâches utilisent une opération bloquante et attendent que les autres les débloquent.

Plus précisément[2] , un dead-lock apparaît lorsque les quatre conditions suivantes sont vérifiées:

[2]voir la ressource vidéo par mathieu moy:https://www.youtube.com/watch?v=l_dnpUkkaSg&list=PL6-YbcqXawf7wl23TTE8Oy6LZV2T4k1Xg&index=2

Interblocage(dead-lock)

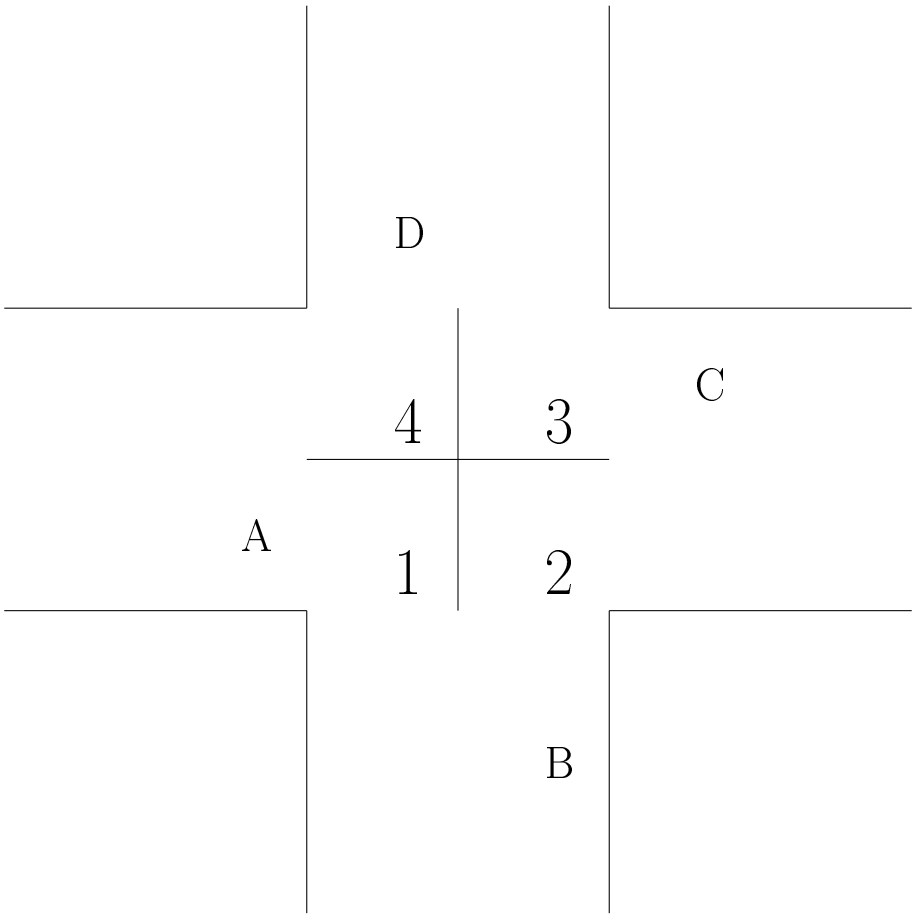
- **Exclusion mutuelle**:Un seul processus au plus possède la ressource.
- **Tenir et attendre**:Un processus en attente ne relâche pas les ressources qu'il détient.
- **Non préemption**: Un ressource attribuée n'est pas reprise.
- **Apparition d'un cycle**:

  Chaque processus attend une ressource détenue par un autre processus. P1 attend une ressource détenue par P2 qui à son tour attend une ressource détenue par P3 etc... qui attend une ressource détenue par P1 ce qui clot la boucle.

Exemple d'un carrefour:

- Processus : Avancée d'une voiture.
- Ressource: partie d'un carrefour.

Processus	Ressources
A	1 et 2
B	2 et 3
C	3 et 4
D	4 et 1



Situation d'interblocage:
A attend que B libère la ressource 2.
B attend que C libère la ressource 3.
C attend que D libère la ressource 4.
D attend que A libère la ressource 1.

Voir la ressource vidéo:

`https://www.youtube.com/watch?v=l_dnpUkkaSg&list=PL6-YbcqXawf7wl23TTE8Oy6LZV2T4k1Xg&index=2`