Numérique et science informatique
Classe de Terminale

Lycée hoche

année scolaire 2021-2022

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*.

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.(on peut par exemple lancer simultanément plusieurs interpréteurs python)

L'ensemble de la mémoire associée à un processus est appelé son *contexte*.

On peut voir la liste des processus qui tournent sur le système avec 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
```

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 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 l'ordonnanceur

L'ordonnanceur est un processus qui joue un rôle important.
Il porte le **numéro 0**.
Il sélectionne à intervalles réguliers (de l'ordre du millième de seconde)le processus auquel affecter les ressources du processeur de l'unité centrale.

2.1 Les états d'un processus

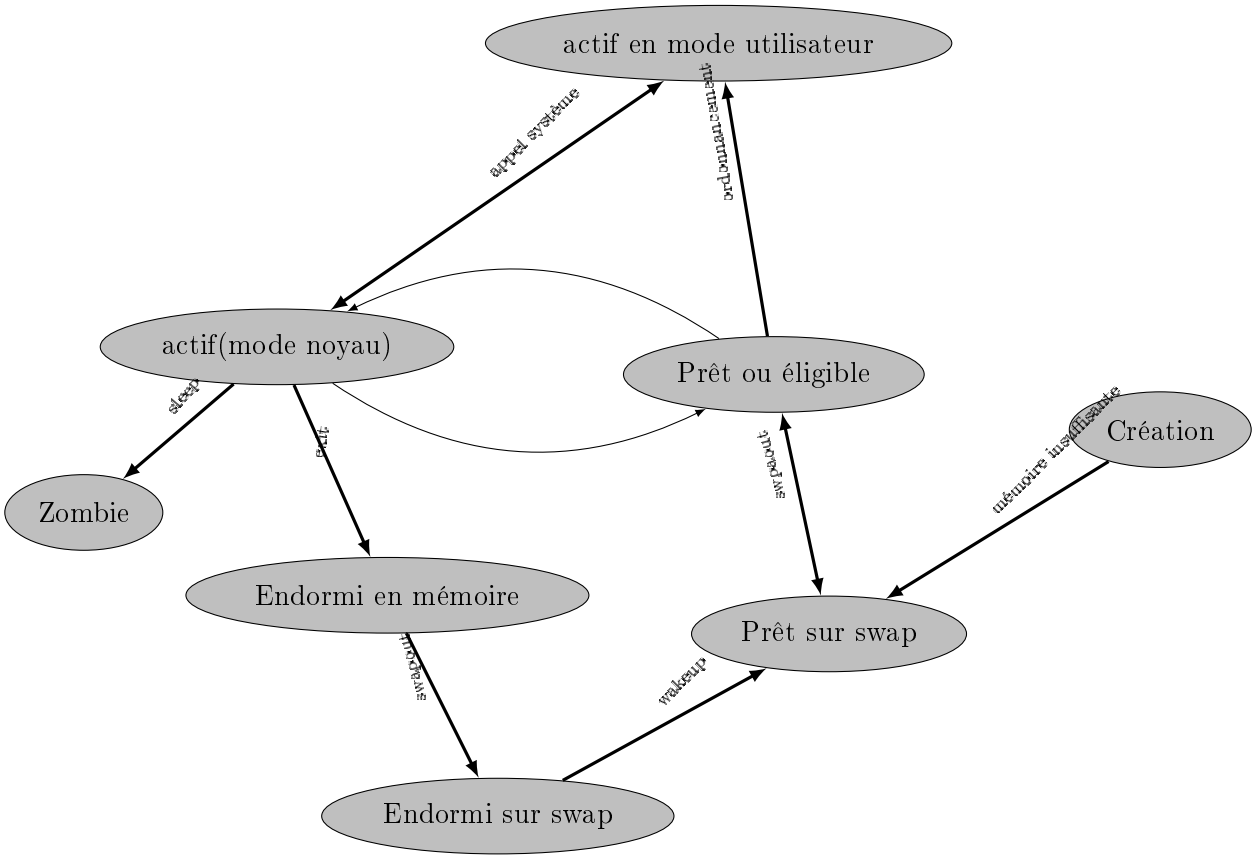
On peut définir pour chaque processus un état parmi plusieurs:

1. Etat **actif**: Le processeur exécute le processus.
   Il y a deux sortes d'exécution d'un processus:
   - Exécution de programmes initiés par l'utilisateur.
   - Exécution d'appels système par le noyau (lecture ou écriture dans la mémoire par exemple)

2. Etat **éligible**ou **prêt**: le processus attend d'être exécuté par le processeur.
3. Etat **en attente** ou **endormi**:Le processus attend un évènement.
4. Etat **mort** ou **zombi** :le processus s'est terminé ou a été interrompu.
5. Etat **suspendu** : Le processus est suspendu et est en attente d'un *signal* de reprise.

L'état d'un processus est donné par la colonne STAT lorsqu'on utilise la commande ps -l

- **T** : Suspendu.
- **S** : endormi (sleeping) depuis moins de 20 s.
- **I** : endormi (idle) depuis plus de 20 s.
- **Z** : Terminé (zombi)
- **W** : placé sur le disque (swap).



(d'après DIU-EIL-UCBL)

3 Contraintes et ordonnancement-Interblocage

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.

3.1 Dead-lock ou interblocage

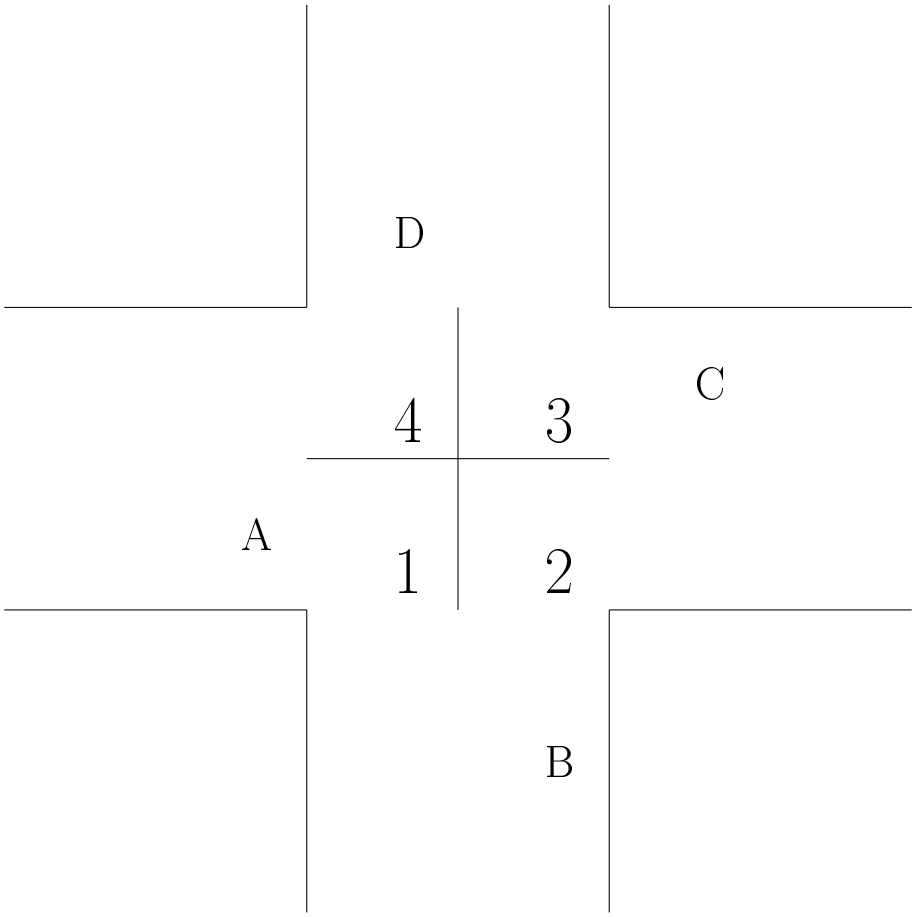
Des tâches peuvent se bloquer irrémédiablement lorsqu'elles demandent les même ressources : dead-lock
Si les deux voitures avancent, chacune bloque l'autre et ne peut plus la débloquer.
Un dead-lock apparait Lorsque:

- **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. P_1 attend une ressource détenue par P_2 qui à son tour attend une ressource détenue par P_3 etc... qui attend une ressource détenue par P_1 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.

4 Notions définies par le système(compléments)

- Principe de localité : le fait qu'une donnée accédée est souvent proche dans l'espace ou le temps d'une données déjà accédée. C'est un principe à considérer lors de la programmation.
- Mémoire : mémoire disponible pour les processus.
- Espace d'adressage : ensemble des adresses que peut utiliser un processus.
- Segment : partie de l'espace d'adressage effectivement utilisé par un processus et ayant un rôle (système, code, données, . . .).
- Droit sur les pages : on attribut un rôle au pages, si la règle n'est pas suivie, cela génère une exception traitée par le système.
- Pages mémoire : découpage de la mémoire en petits morceaux (les pages) qui sont affectés au processus.
- Swapping : Le fait de déplacer une page de la mémoire au disque dur pour libérer de l'espace ou donner des moyens de travail au processus.
- Écroulement : Lorsque le nombre de pages effectivement utilisées par l'ensemble des processus dépasse le nombre de places disponibles, le système va consacrer son temps à swapper.

4.1 Rôle de la mémoire

1. Stocker le code du programme
2. Stocker le code des fonctions partagées.
3. Stocker les variables globales (Tas).
4. Stocker les variables locales (Pile).
5. Une donnée pour pouvoir être traitée doit être dans un registre du processeur.

4.2 Les problèmes posés

Chaque accès à une variable ou une fonction est un accès à la mémoire.
Le compilateur doit faire la correspondance entre un nom qui est dans le code et une adresse demandée par les opérations du processeur.
Mais le programme doit être portable, c'est à dire fonctionner sur des ordinateurs différents et en concurrence avec d'autres programmes.
Le compilateur ne peut donc pas donner l'adresse de la case mémoire où se trouve réellement une variable.
Historiquement, il y a plusieurs façons de régler le problème : -translation d'adresse ; -notions de segment ; -mémoire virtuelle.

4.3 Principe de la mémoire virtuelle

Idée : le processeur travaille avec des adresses mémoires sans rapport avec les adresses physiques.

- **Capacité d'adressage** : l'ensemble des adresses que l'on peut coder,par ex. 4Go sur un processeur 32 bits (0x00000000 - 0xFFFFFFFF), 16Eo pour un 64 bits.

- **Espace d'adressage** : partie utilisable par le processus. L'espace d'adressage est partitionné en segments dépendant du système d'exploitation.

À chaque lecture mémoire, l'adresse physique est calculée. Ce calcul permet de faire des vérifications supplémentaires :

- présence effective de l'adresse en mémoire ;
- droits. Il faut un matériel dédié **la Memomy Management Unit**.