1 Processus et multiprogrammation

Définition 1 *Multiprogrammation*

Hypothèse importante : On considère un ordinateur d'architecture Von Neumann avec un unique processeur.
Les systèmes d'exploitations modernes permettent la **multiprogrammation**, c'est-à-dire qu'on peut lancer en même temps l'exécution de plusieurs programmes.
On parle de systèmes à **temps partagé**.
Un programme en cours d'exécution est un **processus**.
Un ordinateur monoprocesseur ne peut exécuter qu'un seul processus à la fois. Les processus progressent en parallèle, mais ils sont exécutés séquentiellement, à tour de rôle, sur le processeur. On parle de **pseudo-parallélisme**.

Définition 2 *Processus*

Un **processus** est une **instance d'exécution** d'un programme.
On peut par exemple exécuter en même temps plusieurs processus d'un même programme Python (voir exemple ci-dessous).
Chaque processus est repéré par un identifiant nommé `PID`.

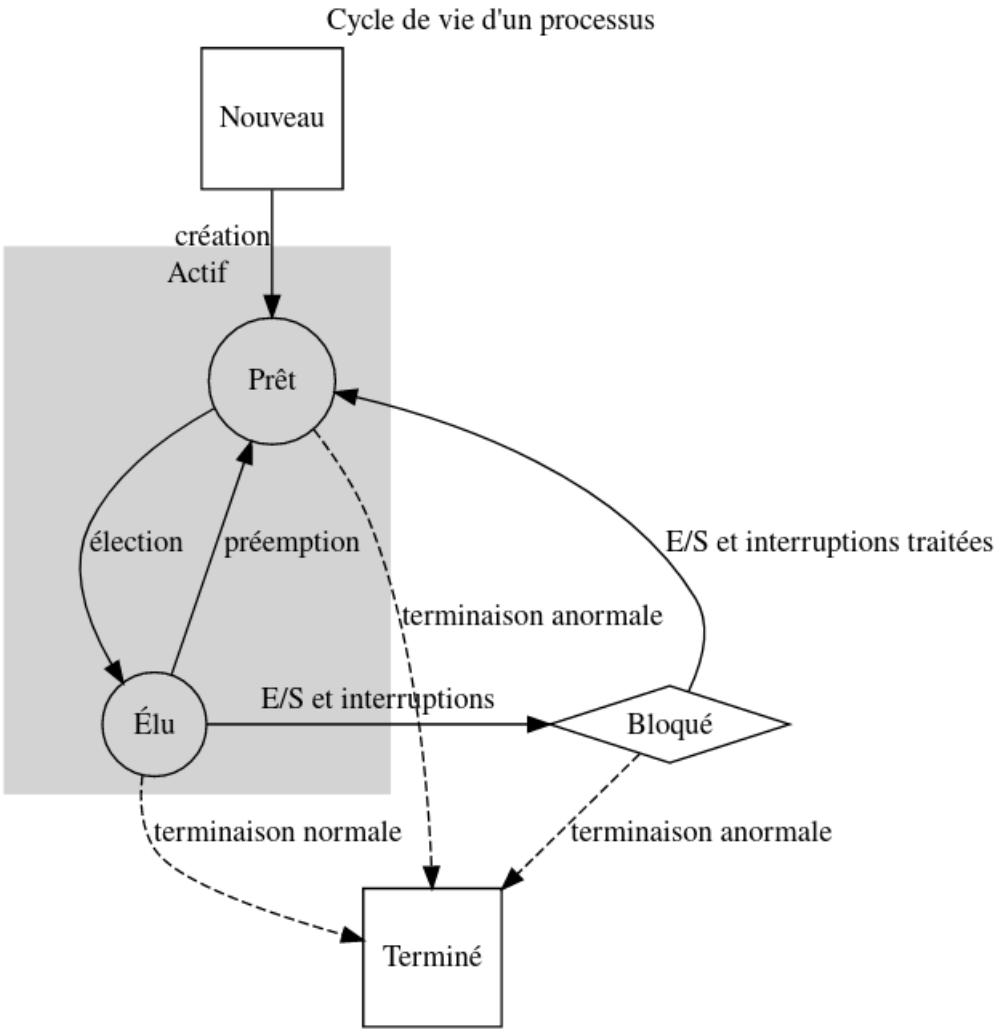
```
fjunier@fjunier:~$ echo "while True:pass" > programme.py
fjunier@fjunier:~$ python3 programme.py &
[1] 4981
fjunier@fjunier:~$ python3 programme.py &
[2] 4985
fjunier@fjunier:~$ ps
    PID TTY          TIME CMD
   4651 pts/1    00:00:00 bash
   4981 pts/1    00:00:04 python3
   4985 pts/1    00:00:02 python3
   4992 pts/1    00:00:00 ps
fjunier@fjunier:~$ kill 4981
fjunier@fjunier:~$ ps
    PID TTY          TIME CMD
   4651 pts/1    00:00:00 bash
   4985 pts/1    00:00:17 python3
   5014 pts/1    00:00:00 ps
[1]- Complété                 python3 programme.py
fjunier@fjunier:~$ kill 4985
fjunier@fjunier:~$ ps
    PID TTY          TIME CMD
   4651 pts/1    00:00:00 bash
   5033 pts/1    00:00:00 ps
[2]+ Complété                 python3 programme.py
```

Méthode *Gestion des processus en ligne de commande*

Dans une ligne de commandes `Linux` (de la famille `UNIX`), plusieurs commandes permettent d'observer les processus en cours :

- La commande `ps` fournit un instantané figé à un instant t des processus en cours.
- La commande `top` fournit un tableau similaire mais dynamique.
- La commande `kill` permet d'interrompre un processus en lui envoyant un **signal d'interruption**.

2 Cycle de vie d'un processus



3 Ordonnancement

Définition 3 *Ordonnancement*

☞ **Qu'est-ce que l'ordonnancement ?**

Sur un ordinateur fonctionnant en temps partagé, à un instant donné, le nombre de processeurs est le plus souvent très inférieur au nombre de processus en cours. On dit que les processus sont en **concurrence**.

L'ordonnanceur est le programme du système d'exploitation qui choisit l'ordre d'exécution des processus sur le processeur.

L'ordonnancement désigne l'activité de **l'ordonnanceur**.

Le processus choisi est dit **élu**, les processus prêts pour l'exécution sont stockés dans une **file d'attente**.

☞ **Quand ordonnancer ?**

L'ordonnanceur peut élire un nouveau processus extrait de la file d'attente des processus prêts dans différents cas :

- le processus élu se termine ;
- le processus élu est bloqué dans l'accès à une ressource ;
- une interruption matérielle d'entrée/sortie a débloqué un processus qui devient prêt à remplacer le processus élu ;
- une interruption matérielle de signal d'horloge indique la fin de l'unité de temps d'exécution, ou quantum, allouée au processus élu.

Si **l'ordonnanceur** prend en compte les interruptions par signal d'horloge pour faire un nouveau choix de processus élu à la fin de chaque **quantum** de temps, on parle d'**ordonnancement préemptif**.

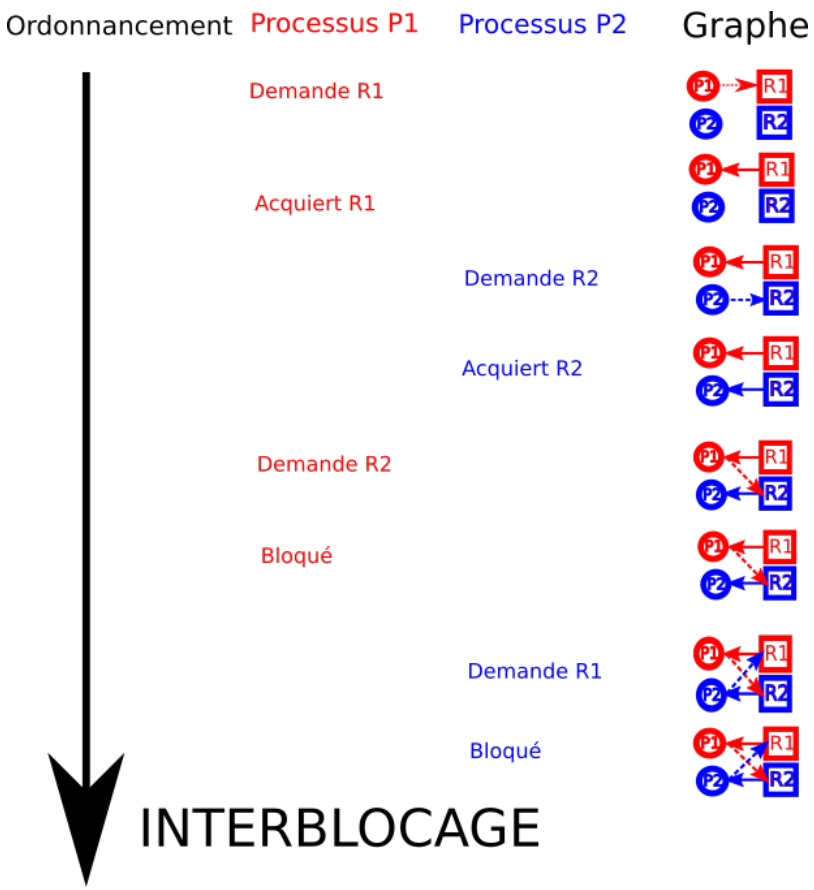
Si **l'ordonnanceur** laisse le processus élu s'exécuter jusqu'à ce qu'il se termine ou se bloque, on parle d'**ordonnancement non préemptif**.

Il existe de nombreux algorithmes d'ordonnancement (premier arrivé/premier servi, plus court d'abord, tourniquet . . .)

4 Interblocage

Définition 4 *Interblocage*

Lire l'explication sur l'interblocage dans le manuel entre les pages 246 et 249.



L'utilisation de **verrous** permet de bloquer l'accès à des ressources en accès exclusif mais présente le risque d'**interblocage**, par exemple si deux processus se bloquent mutuellement la ressource dont ils ont besoin . On peut caractériser une situation d'interblocage par la vérification de certaines conditions. Par exemple avec deux processus P1 et P2 et deux ressources en accès exclusifs :

- *Ressources en accès exclusif :* le processus P1 possède un verrou sur la ressource A et le processus P2 un verrou sur la ressource B ;
- *Dépendance circulaire :* P1 demande B avant de libérer A et P2 demande A avant de libérer B ;
- *Non préemption :* il est impossible de préempter la ressource d'un processus.

Si plusieurs processus utilisent des verrous, une situation d'interblocage peut être prévenue par le programmeur si ***le même ordre d'acquisition*** *des verrous est défini pour tous les codes.*