NSI - Terminale

Arhitecture - processus, système de fichiers, shell

qkzk

2020/06/27

Programme vs processus

Un **programme** :

> est une description statique d'une suite d'instruction : du code, généralement en texte.

Un **processus** :

> est un programme en cours d'exécution. C'est une activité dynamique. Un processus a une *vie* : création, exécution, fin.

Un même programme peut-être exécuté plusieurs fois en même temps (ex chrome : un processus par onglet).

Processeur

Le **processeur** :

> est l'entité matérielle qui réalise les instructions. Il permet de faire progresser le processus.

C'est une ressource utilisée par le processus. À un instant donné, un seul processus est exécuté.

Le système d'exploitation décide de **l'ordonnancement** des processus.

```
séquentielle :          1 1 1 1 2 2 2 2
entrelacée   :          1 2 1 2 1 2 1 2
parallèle    :
processeur 1 :          1 1 1 1
processeur 2 :          2 2 2 2
```

L'exécution peut-être séquentielle ou entrelacée. Si plusieurs processeurs sont disponibles, un exécution parallèle est possible.

L'avancement du processus dépend de la disponibilité du processeur.

Ressources du processus

Ressources du processus :

> entité nécessaire à son exécution.
>
> *matérielles* : processeur, mémoire, périphériques...
>
> *logicielles* : variables etc.

Les ressources indispensables sont :

- la mémoire propre (virtuelle)
- contexte d'exécution (= état instantané)

Une ressource peut-être libre ou occupée. Une ressource peut (ou non) avoir plusieurs accès concurrents.

Consulter les processus sous UNIX

Les processus en cours d'exécution : `ps`

```
$ ps aux
```

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0 178824  7692 ?        Ss   mai20   4:50 /usr/lib/systemd/systemd
root           2  0.0  0.0      0     0 ?        S    mai20   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   mai20   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   mai20   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   mai20   0:00 [kworker/0:0H-kblockd]
...
```

`PID` : Processus Identification : numéro unique.

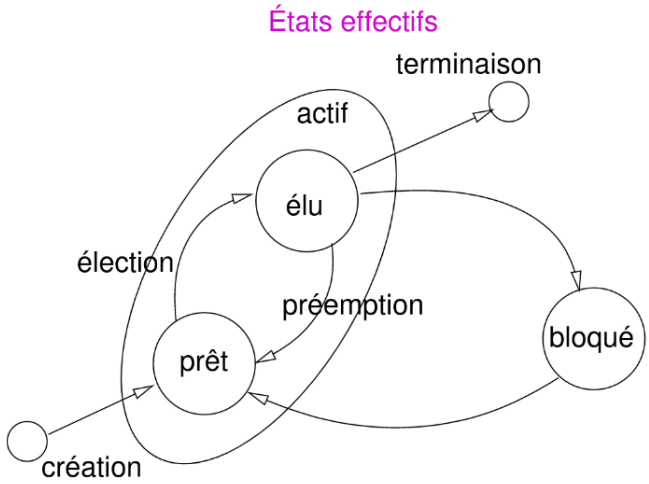
Filtrer avec `grep` : **tous les processus python**

```
$ ps aux | grep python
quentin      892  0.0  0.0 454308 14848 tty1     Ss+  mai20  19:35 /usr/bin/python /home/quentin/prog1.py
quentin     1201  0.0  0.0  16604  5452 ?        S    mai20   8:32 /usr/bin/python /home/quentin/prog2.py
```

Tuer un processus avec `kill`

```
$ kill 892
```

États d'un processus



Attributs d'un processus

Identification univoque

- PID *process ID*
- numéro entier `pid_t`

Propriétaire

- utilisateur qui a lancé le processus, son groupe
- détermine les droits du processus

Répertoire de travail

- origine de l'interprétation des chemins relatifs

Hiérarchie des processus

- création de processus... par un processus
- chaque processus a donc un processus père
- processus `init` ancêtre de tous les processus
- héritage — répertoire de travail, etc.

Interblocage (deadlock)

Soit 2 processus P1 et P2, soit 2 ressources R1 et R2. Initialement, les 2 ressources sont “libres”.

Le processus P1 commence son exécution (état élu), il demande la ressource R1. Il obtient satisfaction puisque R1 est libre, P1 est donc dans l'état “prêt”.

Pendant ce temps, le système a passé P2 à l'état élu : P2 commence son exécution et demande la ressource R2. Il obtient immédiatement R2 puisque cette ressource était libre.

P2 repasse immédiatement à l'état élu et poursuit son exécution (P1 lui est toujours dans l'état prêt).

P2 demande la ressource R1, il se retrouve dans un état bloqué puisque la ressource R1 a été attribuée à P1 : P1 est dans l'état prêt, il n'a pas eu l'occasion de libérer la ressource R1 puisqu'il n'a pas eu l'occasion d'utiliser R1 (pour utiliser R1, P1 doit être dans l'état élu).

P2 étant bloqué (en attente de R1), le système passe P1 dans l'état élu et avant de libérer R1, il demande à utiliser R2. Problème : R2 n'a pas encore été libéré par P2, R2 n'est donc pas disponible, P1 se retrouve bloqué.

Résumons la situation à cet instant : P1 possède la ressource R1 et se trouve dans l'état bloqué (attente de R2), P2 possède la ressource R2 et se trouve dans l'état bloqué (attente de R1)

Pour que P1 puisse poursuivre son exécution, il faut que P2 libère la ressource R2, mais P2 ne peut pas poursuivre son exécution (et donc libérer R2) puisqu'il est bloqué dans l'attente de R1. Pour que P2 puisse poursuivre son exécution, il faut que P1 libère la ressource R1, mais P1 ne peut pas poursuivre son exécution (et donc libérer R1) puisqu'il est bloqué dans l'attente de R2. Bref, la situation est totalement bloquée !

Cette situation est qualifiée d'interblocage (deadlock en anglais).