# Séance n°1 : Etude de l'ordonnancement sous Linux (PC sous Debian)

*Sauf avis contraire vous travaillerez sous l'utilisateur tpuser.*

## I.1 Visualisation de la liste des processus et de leur état.

### I.1.1 Commande *ps*

La commande ps va vous permettre de visualiser la liste des processus lancés sur votre machine ainsi que leurs caractéristiques. Cette commande possède de nombreuses options et 3 syntaxes différentes issues de *UNIX* (options avec un tiret) de *BSD* (options sans tiret) ainsi que la syntaxe *GNU* (avec 2 tirets).

Placer vous dans une fenêtre console de votre machine virtuelle.

a) en utilisant la commande « *ps aux* » (syntaxe *BSD*) visualiser la liste de tous les processus. En remontant sur la première ligne de la liste vous allez trouver la signification des colonnes :

| *USER* | *PID* | *%CPU* | *%MEM* | *VSZ/RSS* | *STAT* | *START* | *TIME* | COMMAND |
|---|---|---|---|---|---|---|---|---|
| *Owner user* | *ID task* | *CPU usage* | *Memory usage* | *Resident & virtual memory* | *Status* | *Time of start* | *Elapsed time in CPU* | command |

**Taper la commande *man ps* pour connaitre la signification des différents champs.**

b) colonne *PID*
Les numéros de *PID* sont attribués dans l'ordre de lancement : retrouver le nom du premier processus lancé ainsi que le *PID* de la fenêtre qui a permis de lancer la commande ps (vous pouvez vous aider de l'horodatage : colonne *START*).

c) Utilisation du *CPU*
En utilisant les colonnes *%CPU* et *TIME*, retrouver quels processus ont utilisé le plus de temps CPU. Pourquoi le %CPU est-il toujours presque nul ?

d) colonne STAT :

| Première lettre | Lettres suivantes |
|---|---|
| *R running or runable (file d'attente tâches prêtes)* | *< high priority* |
| *S interruptible sleep : en attente d'un événement* | *N low priority* |
| *D non interruptible sleep : IO exemple NFS* | *L page locked in memory* |
| *T stop (arrêter par un signal de contrôle : debug...)* | *s session leader* |
| *Z zombie (pas encore supprimé par le parent)* | *l multithread* |
| *X dead (ne devrait pas exister mais temps maj liste)* | *+ foreground (visible à l'écran : shell, commandes)* |

Comment sont la plupart des processus ? Pour l'instant quel est le seul processus dans la file d'attente des tâches prêtes ? Quels sont les processus visibles à l'écran ?

e) Nous allons maintenant changer de syntaxe de représentation (syntaxe UNIX) pour pouvoir choisir nos colonnes qui nous serviront plus tard. Nous allons utiliser la commande ps avec l'option -o :

ps -eo liste de colonnes à afficher séparées par des virgules (ex : ps -eo pid,user)

Taper : *ps -eo pid,user,time,stat,command,**pri***
qui va nous donner une représentation à peu près équivalente à la précédente mais un peu plus réduite.

***pri=19-nice (nice=0 par défaut)*** ***pri=pd (vu en cours)***

Quelles sont les valeurs de *pri* pour la plupart des processus ? Identifier les processus de plus haute priorité (*pri=39*). Chercher sur internet la signification de 3 processus Linux qui ont une priorité de 39. On verra juste un peu plus loin pourquoi certain processus ont des priorités de 139

Taper : *ps -eo pid,user,time,stat,command,**priority***

Quelles valeurs ont maintenant les priorités ? Quelle est la relation entre pri et priority ?
Différentes définitions de la priorité existent. En effet, suivant l'usage qui en est fait les valeurs sont translatées soit pour permettre un affichage plus compréhensible, soit pour simplifier l'algorithme qui les utilise.

Taper maintenant : *ps -eo pid,user,time,stat,command,**pri,class***

Que veut dire *TS* pour le champ *class* ? Chercher la réponse en tapant *man ps*...
Identifier les processus avec un priorité de 139 ? Quelle est leur classe ? Expliquer

Taper maintenant : *ps -eo pid,user,time,stat,command,**pri,class,nice***

Identifier les processus avec les valeurs de nice par défaut ( nice=0) et les processus qui ont une priorité plus grande ( ici nice négatif). Quel sont les processus qui n'ont pas de nice ?

## I.1.2 Commande top (affichage permanent)

**Pour connaître en permanence** la liste et l'état des tâches, vous pouvez utiliser la commande *top*. C'est une commande qui rafraîchit l'affichage des tâches avec une périodicité de quelques secondes.

Taper maintenant : *top*
Quelle est la priorité utilisée pour la commande *top* ?
Identifier en haut de la liste les 2 à 3 processus qui occupe la *CPU* ? Chercher sur Internet ce qu'ils font
Pourquoi *top* en fait partie ?

Une information intéressante est qu'il faut taper 'q' pour libérer la fenêtre de l'affichage et arrêter « top ».

Les informations affichées sont à peu près équivalentes à *ps* (nous verrons quelques différences dans la suite de la séance).

# I.2 Ordonnancement et priorité d'un processus

Les sources des programmes se trouvent dans le répertoire ~/str/TD1.

## I.2.1 Ordonnancement de 2 processus

Aller dans le sous répertoire Q2_process_x2.
Analyser les fichiers *process1.c, process2.c* et *Makefile*. Quel est le compilateur utilisé ? Pourquoi ?
Compiler les sources (process1.c et process2.c) avec le Makefile fourni. Les noms des exécutables seront process1 et process2.

Lancer le premier process en tâche de fond (vous permettant d'avoir la main dans le terminal) :
*./process1 &*
Taper top. Qu'en concluez-vous ? Valeur du nice ? Occupation CPU ?

Taper maintenant : *ps -eo pid,user,time,stat,command,**pri,class,nice***
Quelle est la classe du processus *process1* ?

Lancer le deuxième process: *./process2 &*
Taper top. Qu'en concluez-vous ? Occupation CPU ?

Taper maintenant : *ps -eo pid,user,time,stat,command,**pri,class,nice,pcpu***
Que vous donne l'information *pcpu* ? (faire man *ps*). Que pouvez-vous en conclure ?

Pour arrêter le processus en tâche de fond, il faut retrouver son PID et taper la commande « kill -9 PID ».
Arrêtez les 2 processus lancés
Relancez les 2 processus l'un à la suite de l'autre (sans perdre trop de temps) en les 2 lancement.
Taper de nouveau : *ps -eo pid,user,time,stat,command,**pri,class,nice,pcpu***

## I.2.2 Modification de la priorité d'un processus (commande *nice*)

Lors du lancement du programme vous pouvez choisir un niveau de priorité différent de la priorité de base en indiquant au système la priorité choisie : cela se fait via la commande *nice*.

Lancer process1 en tâche de fond sans modifier sa priorité. Puis lancer le process2 en lui donnant une priorité plus basse : *nice -10 ./process2 &* (augmente la gentillesse de 10). **Attention le « - » n'est pas un moins mais un tiret d'une option**
C'est à dire qu'on lui donne une priorité plus faible. Quelle est la valeur de pd=pri ?

Taper top. Qu'en concluez-vous ? Occupation CPU ?
Taper *ps -eo pid,user,time,stat,command,**pri,class,nice,pcpu.*** Quelles sont les valeurs de *nice* et *pri* ?

Arrêter le process2 puis relancer le avec une priorité plus forte.
Pour augmenter la priorité de 10 par exemple, il faut taper la commande *nice - -10 process2* (le premier – c'est pour dire que c'est une option le deuxième est vraiment un signe).
Que se passe-t-il ?
Trouver la solution pour augmenter la priorité.

Vous pouvez aussi modifier la priorité d'un processus en cours d'utilisation par la commande *renice* en précisant le pid. Remettre le processus dans un priorité normale.
Taper *ps -eo pid,user,time,stat,command,**pri,class,nice,pcpu*** afin de visualiser la priorité et le *nice*

Relancer comme précédemment les 2 processus process1 et process2 avec la priorité de base. Mesurer l'évolution de leur temps CPU. Modifier au fur et à mesure (par pas de 1) la priorité du process2, en le rendant d'abord moins prioritaire. Regarder l'évolution du temps CPU.

Faire de même en lui donnant une priorité plus élevée. Tracer la courbe de pourcentage de l'utilisation du CPU en fonction du *nice* (vous constaterez qu'elle n'est pas linéaire).

## I.2.3 Modification de la classe d'un processus

Les processus que nous avons vu jusqu'à présent étaient de class SCHED_OTHER (ou plus exactement SCHED_NORMAL). Ce sont les processus les plus courants sous Linux.

On peut définir des processus qui appartiennent aux classes « temps réel » (SCHED_RR ou SCHED_FIFO). Cette distinction doit se faire au niveau du code source et non par une commande équivalente à *nice*. 2 programmes (process1_RR et process1_FIFO) ont été compilés en même temps que les précédents. Ils sont associés respectivement à chaque classe de processus temps réel.

a) Analyser le code source *process1_RR.c.* Où trouve-t-on l'information que le processus va se lancer en SCHED_RR.A quoi correspond *myparam.sched_priority = 5* ?

b) Lancer process1 en tâche de fond et *process1_RR* en console. Visualiser le partage du temps *CPU*, la classe, la priorité et la priorité temps réel.
Taper *ps -eo pid,user,time,stat,command,**pri,class,nice,pcpu,rtprio***

Pourquoi le *process1_RR* n'a-t-il pas une priorité supérieure ?

c) Relancer le process1_RR en étant root.
Taper de nouveau *ps -eo pid,user,time,stat,command,**pri,class,nice,pcpu,rtprio***

Le fonctionnement est-il alors correct ?

d) Copier l'exécutable process1_RR en process2_RR, relancer les 2 processus (process1_RR et process2_RR), comment est partagé le temps CPU ?

e) En éditant process1_RR.c passer la priorité de 5 à 6. Relancer les 2 processus. Que se passe-t-il ?

f) Reprendre à partir du b) les mêmes questions avec des processus de type FIFO (process_FIFO).

g) Exécuter un processus FIFO et un RR de même priorité. Quels processus sont exécutés ?

h) Lancer un processus FIFO puis un processus RR de priorité plus forte. Quels processus sont exécutés ?