Cours 5
Généralités sur les processus

Qu'est-ce qu'un processus ?

Définition : Un processus est une unité d'exécution, c'est à dire de partage du temps processeur et de la mémoire.

Rappel : donc ne pas confondre
- **programme** : code sur disque (en langage source, en langage machine)
- **processus** : code en mémoire centrale (en langage machine).

Deux idées à comprendre :

1. multitâche : plusieurs unités d'exécution sont chargées en machine et l'on passe de l'une à l'autre pendant l'exécution.
   - mode batch : quand un processus fait une E/S, on passe à un autre processus.
   - time-sharing : cas particulier ou les entités s'exécutent en pseudo parallèle

2. une unité d'exécution (processus) est autre chose qu'un programme parce que
   - le même programme peut être lancé plusieurs fois et se trouver dans plusieurs unités d'exécution en même temps
   - En plus du programme, l'unité d'exécution a à chaque instant un état courant (pointeur d'instruction, état de la pile, valeur des variables...) qui n'est pas le même dans différentes UE du même programme.
   - on peut même conserver la même UE en changeant le code qu'elle exécute (instruction `execl()` du C)

Les commandes ps et top permettent de voir les processus.qui s'exécutent sur la machine.

Tâche de premier plan et tâche de fond

Quand plusieurs processus sont rattachés au même terminal, ils ne peuvent tous y avoir accès en même temps.
Ex: quand on appuie sur une touche clavier, …….
…….
…….

Conclusion: un seul processus a les droits d'accès au clavier à un moment donné. Ceux qui ont accès à l'écran sont les processus en tâche de 1er plan de ce terminal. Les autres processus rattachés à ce terminal sont en tâche de fond.
Un processus en tâche de fond continue à calculer (il reste actif), sauf quand il doit accéder au terminal. Il est alors stoppé jusqu'à ce qu'il passe au premier plan.

La commande **jobs** permet de voir les tâches rattachées au terminal sur lequel on l'eploie. **fg** passe une tâche en 1er plan, **bg** la passe en tâche de fond, **^Z** stoppe la tâche de 1er plan, **^C** la tue.

Exemple (On est dans une fenètre Nxterm de l'environnement graphique sous linux) :

```
levy:essais:26> set -b                      # pour voir les messages
levy:essais:27> more longlasting.c &        # lance more en tâche de fond
[1] 384                                     # c'est la tâche 1, n°de processus 384
[1]+  Stopped (tty output)    more longlasting.c  # ……..
levy:essais:28> ./longlasting &             # lance longlasting en tâche de fond
[2] 385                                     # c'est la tâche 2, n°de processus 385
levy:essais:29> jobs                        # ……..
[1]+  Stopped (tty output)    more longlasting.c
[2]-  Running                 ./longlasting &
```

```
levy:essais:30> %1                          # ……..
more longlasting.c
int main(int argc, char * argv[]) {
  int i=1;
  while (i!=0) {
    i = (i * 2) % 1513 ;                    # Pas étonnant que ça dure longtemps !
}  }
levy:essais:31> jobs
[2]+  Running                 ./longlasting &    # La tâche 1 est terminée
levy:essais:32> Xemacs longlasting.c             # On a oublié le & ⇒  ne rend pas la main
                                                 # ^Z au clavier stoppe la tâche de premier plan
[3]+  Stopped                 Xemacs longlasting.c     # Xemacs ne fonctionne plus
levy:essais:33> jobs                                   # parce que ……..
[2]-  Running                 ./longlasting &
[3]+  Stopped                 Xemacs longlasting.c
levy:essais:34> bg                               # ……
[3]+ emacs longlasting.c &
levy:essais:35> fg %2                            # ……
./longlasting
                                                 # Maintenant, longlasting reçoit ^C, qui le termine
levy:essais:36>
```

Un processus qui n'est rattaché à aucun terminal (parce qu'il n'aura jamais besoin de lire ou d'écrire) est un processus démon. Ex.: httpd est un processus qui prend les connexions http

Autre exemple: un environnement graphique.

(la commande ps liste les processus)

```
levy:essais:57> ps f
  PID TTY STAT TIME COMMAND
  176   1 S    0:00 -bash
  223   1 S    0:00  \_ sh /usr/X11R6/bin/startx
  224   1 S    0:00    \_ xinit /usr/X11R6/lib/X11/xinit/xinitrc --
  228   1 S    0:00      \_ fvwm2 -cmd FvwmM4 -debug /etc/X11/AnotherLevel/fvwm2rc.m4
  300   1 S    0:00        \_  /usr/X11R6/lib/X11/fvwm2//FvwmTaskBar 9 4 /tmp/fvwmrca00239 0 8
  301   1 S    0:00        \_ /usr/X11R6/lib/X11/fvwm2//FvwmButtons 11 4 /tmp/fvwmrca00239 0 8
  304   1 S N  0:00        \_ xload -nolabel -geometry 32x20+0+0 -bg grey60 -update 5
  305   1 S    0:00        \_ /usr/X11R6/lib/X11/fvwm2//FvwmPager 11 4 /tmp/fvwmrca00239 0 8 0 1
  306  p0 S    0:00 bash
  386  p0 S    0:01 \_ emacs longlasting.c
  472  p0 R    0:00 \_ ps f
  310   1 S    0:01 xfm
  390  p1 S    0:00 bash
levy:essais:58>
```

PID = numéro de processus ; TTY = terminal (TeleTYpe) ; STAT = état (Sleeping ou Running)

Comment sont créés les processus

C'est l'appel système fork() qui crée un processus fils.Le père et le fils ont le même code et s'exécutent en même temps à partir du retour du fork(). Dans le père, la valeur de retour de fork() est le numéro (pid) du fils (jamais nul). Dans le fils, c'est 0. D'où la forme standard pour utiliser fork() :

```
int pid ;
pid = fork();
if (pid != 0) {/* code du père */}
else { /* code du fils */}
```

Exemple :

```
main() {
      int pid ;
      pid = fork();
      if (pid != 0) {         /* père */
            printf("Je suis le processus pere");
      }
      else        {           /* fils */
            printf("Je suis le processus fils");
      }
```

Qu'est-ce qui s'affiche a l'écran ? : ……
…….. …… …… ……

Comment on change le code d'un processus

Un processus peut changer son code par un appel système commençant par exec. Voici le prototype de execl :

```
#include <unistd.h>
int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/);
```

Exemple :

```
main(){
      execl("/usr/bin/cc", "cc", "-o", "mymain", "mymain.c", "util.c",NULL);
}
```

Attention :
a. il faut faire attention à ce qu'on écrit

```
¬  /* Fichier: prog1.c */
    main(){
       execl ("./prog2.c", "prog2.c",NULL);
    }
¬  /* Fichier: prog2.c */
    main(){
       execl ("./prog1.c", "prog1.c",NULL);
     }
```

→ boucle infinie !

b. Le code après le `execl()` n'est jamais exécuté. Par exemple :

```
main(){
execl("./monprog", "monprog", NULL);
printf("monprog a été lancé"); // ce code n'est jamais atteint
}
```

On peut aussi passer un tableau d'environnement avec une autre forme de exec (voir cours 6) :

```
int execle (const char *path, const char* arg0[], ... , const char *argn, char * /*NULL*/,
const char* envp[]);
```