# TP N°5
# Création de processus

Sylvain Chevallier

## Préambule

Vous testerez vos programmes avec des jeux d'essai, c'est-à-dire différentes valeurs pour vérifier son bon fonctionnement. Vous indiquerez en commentaires dans le code source ou dans un fichier séparé, le jeu d'essai que vous avez testé et les résultats obtenus.

Les exercices possédant une **(*)** sont facultatifs. À la fin de la séance, vous ferez une archive compressée, avec `tar` du dossier du TP que vous enverrez par mail à votre enseignant.

Vous pouvez utiliser la commande `ps` dans la console pour afficher l'ensemble des processus en cours d'execution sur votre terminal. En utilisant `ps -elf`, vous afficherez l'ensemble des processus en cours sur le système et pour chaque processus: son propriétaire, son PID, son PPID, sa priorité, son nom, etc. Vous remarquerez que le processus `init` a pour PID `1` et pour PPID `0`. Les processus dont les noms sont entre `[]`, comme `[kthreadd]`, indiquent des processus générés par le noyau.

Vous pouvez également afficher la liste des processus en cours sous forme d'arbre en utilisant la commande `pstree`. Le lien père-fils entre les processus apparaît ainsi directement.

## 1 Création de plusieurs processus

Écrivez un programme qui crée N processus (tous les processus sont ses fils). Le processus père donne un numéro a chacun des fils: le premier fils est 1, le deuxième 2, etc. Chaque processus affiche le message

"Je suis le processus : <numero-donné-par-le-père>, pid et je calcule les multiples du <numero-donné-par-le-père>"

Et chaque processus affiche les premiers 100 multiples de numéro reçu.

## 2 La fonction execl

La fonction `execl()` est complémentaire de la création de processus avec `fork()`. Le principe que nous allons voir ici revient à modifier le code du programme à exécuter pendant l'exécution du code en cours.

Écrire un programme appelé `bonjour` dont le code affiche à l'écran la chaîne `"bonjour"`. Écrire et lancer le programme suivant qui fait appel au programme `bonjour` en utilisant la fonction `execl()`. Qu'en concluez vous ? Que devient l'affichage suivant l'appel à `execl` ?

```c
#include <stdio.h>
#include <unistd.h>

int main ( ) {
  printf ("avant l'execution de bonjour\n");
  execl ("./bonjour", "./bonjour",(char*) 0);
  printf (" apres l'execution de bonjour\n ");
}
```

## 3 La fonction execv

La fonction `execv()` utilise la notation `intensive` des paramètres passés à l'appel du programme principal. Écrire un programme appelé `necho` qui affiche les paramètres passés en argument lors de son appel. Compilez ce programme et générez un exécutable appelé `necho`. Écrivez et exécutez le programme suivant qui appelle

le programme `necho` avec la fonction `execl`. Vous remplacerez l'appel à `execl` par un appel à `execv`, afin d'obtenir le même résultat.

Vous analyserez les résultats obtenus, en prêtant bien attention aux arguments affiché par `necho` lorsque vous modifiez la ligne `arguments[4] = (char *) 0;` par `argument[1] = ...,argument[2] = ...,` etc.

```
#include <stdio.h>
#include <unistd.h>

char *arguments[5] = { "./necho", "au revoir", "123", "3.14159" };
int main ( ) {
  arguments[4] = (char *) 0;
  execl ("./necho", arguments[0], arguments[1], arguments[2], arguments[3], (char *) 0);
}
```

## 4 Création de Xterm à la volée (*)

On désire écrire un programme qui permet de lancer un "xterm" à chaque appui de la touche retour-chariot.

1) Le programme suivant est une première tentative :

```
int main() {
  pid_t idpr;
  int nbFils=0;
  while (1){
    while (getc(stdin) != '\n');
    idproc= fork();
    if (idpr==-1) {
      perror ("echec du fork'');
      exit(EXIT_FAILURE);
    }
    if (idpr==0) {
      execlp ("xterm", "xterm", NULL);
      perror ("erreur exec ");
      exit(EXIT_FAILURE);
    }
    nbFils++;
  }
  exit (EXIT_SUCCESS);
}
```

a) Commentez ce programme. Ce programme permet-il de lancer un nombre quelconque de xterm ? Justifiez votre réponse.

b) Que se passe-t-il si on ferme des xterm ?

2) On modifie le programme précédent pour pouvoir limiter à 4 le nombre de fenêtres xterm actives simultanément (comprendre: des xterm créées par ce programme).

Nouveau programme :

```
#define NX 4 // nombre max de xterm simultanees autorisees
int main() {
  pid_t idproc;
  int nbXterm=0; //nombre xterm
  while (1){
    while (getc(stdin) != '\n');
    if ( nbXterm >= NX){ //
      if (wait(NULL)>0)  //
      nbXterm--;         //
    }
    //
    idpr= fork();
    if (idpr==-1 ) {
```

```
        perror ("echec fork");
15      exit (EXIT_FAILURE);
      }
17    if (idpr==0){ // processus fils
        execlp ( "xterm" , "xterm" , NULL);
19      perror ("erreur exec ");
        exit (EXIT_FAILURE);
21    }
      nbXterm++;  // on a lance 1 xterm de plus
23  }
}
```

Expliquer le fonctionnement. Cela résoud-il les problèmes soulevés précédemment ?