STI 1ère année – Programmation Système

TD 7: processus - suite

J. Briffaut

1 Processus

L'objectif de ce TD est d'éudier :
- Processus orphelins et zombis
- Mutation

1.1 Processus orphelins

Exercice 1 Créer un programme C qui créé un processus fils qui deviendra orphelin (il se rattachera au processus init de PID 1)

Exercice 2 Pour quel type de programme la création de processus orphelins et généralement utilisée ?

Listing 1 – Solution de l'exercice 2

```
#include<stdlib.h>

int main() {
   pid_t status;
   printf("[%d] Je vais engendrer\n", getpid());
   status = fork();
   switch (status) {
      case -1 :
         perror("Creation processus");
         exit(EXIT_FAILURE);
      case 0 :
         sleep(2);
         printf("[%d] Je viens de naitre\n", getpid());
         printf("[%d] Mon pere est %d\n", getpid(), getppid
              ());
         break;
      default:
         printf("[%d] J'ai engendre\n", getpid());
         printf("[%d] Mon fils est %d\n", getpid(), status);
   }
   printf("[%d] Je termine\n", getpid());
   exit(EXIT_SUCCESS);
}
```

1.2 Processus zombi

Exercice 3 Créer un programme C qui créé un processus fils qui deviendra zombi.

Exercice 4 Vérifier avec `ps aux` que le processus fils devient bien un processus zombi (il doit apparaître entre ())

Exercice 5 Quel est le danger potentiel de ce type de processus ?

Listing 2 – Solution de l'exercice 5

```
#include<stdlib.h>

int main() {
   pid_t status;
   int statusf;
   printf("[%d] Je vais engendrer\n", getpid());
   status = fork();
   switch (status) {
      case -1 :
         perror("Creation processus");
         exit(EXIT_FAILURE);
      case 0 :
         printf("[%d] Je viens de naitre\n", getpid());
         printf("[%d] Mon pere est %d\n", getpid(), getppid
              ());
         break;
      default:
         sleep(10);
         printf("[%d] J'ai engendre\n", getpid());
         printf("[%d] Mon fils est %d\n", getpid(), status);
         wait(&statusf);
   }
   printf("[%d] Je termine\n", getpid());
   exit(EXIT_SUCCESS);
}
```

1.3 Création de processus : Mutation

Exercice 6 Créer un programme C qui prend en paramètre une commande à exécuter. Vous utiliserez pour cela la fonction *execve()*

Listing 3 – Solution de l'exercice 6

```
#include<sys/types.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[], char **arge)
{
        execve(argv[1], argv+1, arge);
        perror ("recouvrement");
        exit(EXIT_SUCCESS);
}
```

Exercice 7 Créer un programme C qui va simuler un mini-shell. Une fois lancé, ce programme attendra sur l'entrée standard que l'on entre une commande. Cette commande sera ensuite exécutée, le processus père de votre mini-shell attendra alors que le processus finisse avant de demander à nouveau d'entrée un commande. Si la commande correspond a *exit*, votre shell s'arrête. Si il y a un problème lors de l'exécution de la commande, vous afficherez l'erreur corresponte.

Listing 4 – Solution de l'exercice 7

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

#define BUFFER 512
#define DEBUG 1

int main() {
        char command[BUFFER];
        char* arg[BUFFER];
        int nb_args,i;
        while(1){
                //on rÃľcupÃíre une commande a exÃľcuter
                nb_args=read_command(command,&arg);
                //si la commande correspond a exit, on
                     quitte
                if(nb_args==-1){
                        printf("Exit");
                        exit(EXIT_SUCCESS);
                }

                if(DEBUG){
                        printf("commande a exÃľcuter: %s, %d
                              arguments [",command,nb_args)
                             ;
                        for(i=0;i<nb_args;i++)
                                printf("%s , ",arg[i]);
                        printf("]\n");
                }

                //on execute la commande
                run_command(command,arg);
        }
}

int read_command(char* command, char* arg[]){
        int cpta=0;
        int c,i;
        char * chaine;
        // lire la command
        scanf("%s",command);
        if (strcmp(command,"exit")==0) {
                return -1;
        }

        //lire les arguments
        while((c=getchar())!='\n'){
                chaine=malloc(BUFFER * sizeof(char));
                scanf("%s",chaine);
                arg[cpta++]=chaine;
        }

        //on retourne le nombre d'arguments
        return cpta;
}

int run_command(char * command, char * arg[]) {
        pid_t status;
        status = fork();
        switch (status) {
                case -1 :
                        perror("Creation processus");
                        exit(EXIT_FAILURE);
                case 0 :
                        //utilisation de execv pour
                             exÃľcuter la commande
                        if(execv(command,arg)==-1){
                                perror("Erreur : ");
                                exit(EXIT_FAILURE);
                        }
                        exit(EXIT_SUCCESS);
                        break;
                default:
                        //attente de la fin de la commande
                        wait(&status);
                        if (WIFEXITED(status))
                                printf("commande executÃľe
                                     sans erreur\n");
                        else {
                                printf("commande executÃľe
                                     avec erreur\n");
                        }

        }
}
```