

Rapport 1

Gestion des processus / signaux

Mahmoud Bacha rabah sami

M1 RSD - GROUP : 02

Exercice 01 :

question 2 :

Le role des fonctions :

- **getpid :** `pid_t getpid(void)`
Elle recupere le pid du processus appelant la fonction
- **getppid :** `pid_t getppid (void)`
elle récupère le pid du processus appelant la fonction
- **exit :** `void exit(int return_code)`
elle termine un processus en retournant une valeur int donné comme argument

Modification :

On rajoute seulement li pid dans l'affichage du processus père car pid stock la valeur du pid du fils .

question 3 :

La commande ps aux permet d'afficher les informations sur les processus qui sont entrain de s'exécuter (USER , PID , CPU , MEM ...) mais elle ne donne pas le ppid

Pour avoir le ppid , on utilise la commande : **ps l**

```
sami@laptop:~/SE_RSD/tp1$ ps l
F  UID      PID      PPID  PRI  NI       VSZ      RSS WCHAN  STAT TTY          TIME COMMAND
4  1000     1199     1131   20    0  162424    6272 do_pol  Ssl+ tty2        0:00 /usr/libexec/gdm-x-se
4  1000     1202     1199   20    0  418636   184804 ep_pol  Sl+  tty2        4:31 /usr/lib/xorg/Xorg vt
0  1000     1280     1199   20    0  223032   15488 do_pol  Sl+  tty2        0:00 /usr/libexec/gnome-se
0  1000     2398     2365   20    0   11136    3968 do_wai  Ss    pts/0        0:00 bash
0  1000     2645     2622   20    0   11164    4224 do_sel  Ss+   pts/1        0:00 /usr/bin/bash --init-
0  1000     3043     2365   20    0   11136    4736 do_wai  Ss    pts/2        0:00 bash
0  1000     3142     2622   20    0   11164    4224 do_sel  Ss+   pts/3        0:00 /usr/bin/bash --init-
4  1000     3335     3334   20    0   11004    4992 do_sel  S+    pts/4        0:00 bash
1  1000     5659     1153   20    0    2772     896 hrtime  S     pts/4        0:00 ./exo1
4  1000     5669     3043   20    0   12672    3328 -      R+    pts/2        0:00 ps l
```

Dans notre cas , le processus de l'exo1 a pid = 5659 et son père est : 1153

```
sami@laptop:~/SE_RSD/tp1$ ps l 1153
F  UID      PID      PPID  PRI  NI       VSZ      RSS WCHAN  STAT TTY          TIME COMMAND
4  1000     1153         1   20    0   17952   10624 ep_pol  Ss    ?          0:02 /lib/systemd/systemd --user
```

Le père est donc le process systemd , son père est le processus init ayant pid = 1

Question 4 :

Le script : ici n est fixé a 4

```

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h> // fork
#include <stdlib.h> // exit

#define n 4

int main(int argc , char *argv[])
{
    int i=0, pid;
    while (i<n)
    {
        i++;
        pid= fork();
        if (pid == - 1)
        {
            /* code si échec : */
            perror ("fork") ;
            exit(1) ;
            //sortir sur un code d'erreur
        }
        if (pid==0)
        {
            // Code du fils
            printf("Bonjour c le fils : %d, papa est : %d\n", getpid(), getppid());
            sleep(6);
            printf("Au revoir c le fils : %d, papa est : %d\n", getpid(), getppid());
            exit(0);
            // Fin du processus fils
        }
        // Suite code du père, si pid > 0
        sleep(2);
        printf("je suis le père pid : %d, ppid=%d , mon fils est : %d: \n", getpid(),
        getppid() ,pid);
    }
}

```

Resultat d'exécution :

```

sami@laptop:~/SE_RSD/tp1$ ./exo1_q4
Bonjour c le fils : 6326, papa est : 6325
je suis le père pid : 6325, ppid=3043 , mon fils est : 6326:
Bonjour c le fils : 6328, papa est : 6325
je suis le père pid : 6325, ppid=3043 , mon fils est : 6328:
Bonjour c le fils : 6329, papa est : 6325
Au revoir c le fils : 6326, papa est : 6325
je suis le père pid : 6325, ppid=3043 , mon fils est : 6329:
Bonjour c le fils : 6330, papa est : 6325
je suis le père pid : 6325, ppid=3043 , mon fils est : 6330:
Au revoir c le fils : 6328, papa est : 6325
sami@laptop:~/SE_RSD/tp1$ Au revoir c le fils : 6329, papa est : 1153
Au revoir c le fils : 6330, papa est : 1153

```

Interprétation de résultats :

- Les 4 processus fils affiche leur pid et ppid avant et après faire un sleep de 6 seconds alors que le père ne fait un sleep que de 2 seconds
- Tous les processus affiche et le ppid correct de leur père avant se mettre en pause car le processus père existait toujours
- Après le sleep , les deux premiers processus ont affiché le ppid du père qui est 6325 , **la raison est que lorsque ses deux derniers sont revenus de leur sleep , le processus père ayant pid=6325 a terminé son exécution et donc ils sont devenu orphelins.**
- Le OS fait adopté les process orphelins au processus systemd ayant pid = 1153 qui appartient a init

Question 6 :

Pour éviter qu'un processus devient orphelin il faut s'assurer que le père ne peut se terminer que si tous ses fils sont terminés , on pourra assurer ça par la commande **waitpid**

Exercice 03 :

Déroulement :

P0 créer P1 et P2 et attend leur fin, il créer par la suite P5
Et il se termine

P1 de son tour crée P3 et P4, il attend leur fin
P1 se termine pour donner le feu vert a P0 de lancer le P5

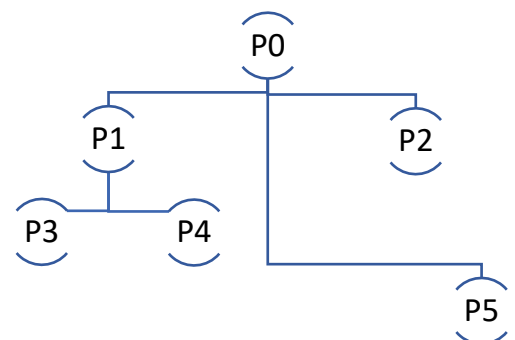
P5 est donc le dernier processus a être créer et a se terminer

Commande utilisé :

Fork : pour la création

wait : attendre la fin des fils

Sleep : endormir les processus



Script :

```

#include<stdio.h>
#include<stdlib.h>
#include<signal.h>
#include<unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char const *argv[])
{

    int tab_pid[5], i=0 , n;
    int pid;

    tab_pid[0]=fork();
    if (tab_pid[0]==0)
    {

        printf("i am p1 , pid is : %d , ppid is : %d \n",getpid(),getppid());
        tab_pid[2] = fork();
        if (tab_pid[2]==0)
        {
            printf ("i am p3 , pid : %d , ppid : %d \n",getpid(),getppid());
            sleep(10);
            printf("p3 will end\n");
            exit(0);
        }
        else{
            tab_pid[3] = fork();
            if(tab_pid[3]==0){
                printf("i am p4 , pid : %d , ppid : %d\n", getpid(), getppid());
                sleep(10);
                printf("p4 will end\n");
                exit(0);

            }else{
                // on est dans p1
                while(wait(NULL)!=-1);
                printf("p1 will end\n");
                exit(0);
            }
        }
    }
    else{
        // on est dans p0

        tab_pid[1]=fork();
        if (tab_pid[1]==0)
        {
            printf("i am p2 , pid : %d , ppid : %d\n",getpid(),getppid());
            sleep(10);
            printf("p2 will end\n");
            exit(0);
        }
        else{

```

```

// on est dans p0 , on attend que p1 et p2 se termine pour creer p5

while(wait(NULL) != -1);

tab_pid[4]=fork();
if (tab_pid[4]==0)
{
    printf("i am p5 , pid : %d , ppid : %d\n",getpid(),getppid());
    sleep(10);
    printf("p5 will end\n");
    exit(0);
}
else{
    wait(NULL);
    printf("p0 will end\n");
}
}
return 0;
}

```

Affichage :

```

sami@laptop:~/SE_RSD/tp1$ gcc Exo3.c -o Exo3
sami@laptop:~/SE_RSD/tp1$ ./Exo3
i am p0 , pid is : 8996
i am p2 , pid : 8998 , ppid : 8996
i am p1 , pid is : 8997 , ppid is : 8996
i am p4 , pid : 9000 , ppid : 8997
i am p3 , pid : 8999 , ppid : 8997
p2 will end
p4 will end
p3 will end
p1 will end
i am p5 , pid : 9003 , ppid : 8996
p5 will end
p0 will end

```

On execute par la suite la commande pstree -p 8996 pour avoir l'arboressance :

On constate 2 qu'on a deux arbres , P5 n'existe pas dans la première car on doit attendre la fin de tous les processus P1 , P2 , P3 , P4 ayant respectivement les pid , 8997 , 8998 , 8999 , 9000 pour qu'il soit crée

Avant création de P5 :

```
sami@laptop:~/SE_RSD/tp1$ pstree -p 8996
Exo3(8996)─Exo3(8997)─Exo3(8999)
              │       └─Exo3(9000)
              └─Exo3(8998)
```

Apres création de P5 :

```
sami@laptop:~/SE_RSD/tp1$ pstree -p 8996
Exo3(8996)─Exo3(9003)
```

Exercice 05 :

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

static char maj = 'A';
static char min = 'a';
static int cptmin=0,cptmaj=0;
static pid_t pid ;

static void handler() {

    if (pid==0) {

        while (cptmin<26) {
            printf("%c",min);
            fflush(stdout);           // on est dans le fils
            min++;
            cptmin++;
            usleep(5000);
            kill(getppid(),SIGUSR1);
        }

        else{

            while(cptmaj<26) {
```

```

printf("%c",maj);
    fflush(stdout);           // on est dans le pere
maj++;
cptmaj++;
    usleep(5000);

    kill(pid, SIGUSR1);
}
}
}
int main (int argc , char * argv[]){

    if(signal(SIGUSR1,handler) == SIG_ERR){
        perror("sigusr1");
    }

    printf("Output : \n\n");

    pid = fork();
    if(pid == -1)
        perror("fork not working ");

    if(pid == 0) {

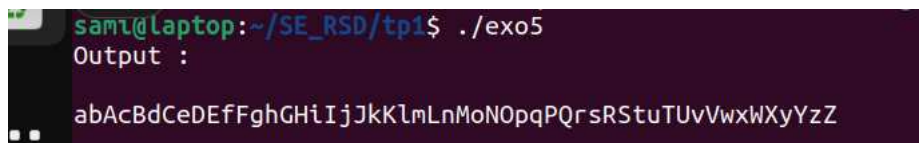
        kill(getppid(),SIGUSR1);}

    else{

        kill(pid,SIGUSR1);
        wait(NULL);
        printf("\n\n");
    }
    return 0;
}

```

Affichage :



```

sami@laptop:~/SE_RSD/tp1$ ./exo5
Output :

abAcBdCeDEfFghGHiIjJkKlLnMoN0pqPQrsRStuTUvVwxWXYyZZ

```


Explication :

Le père et le fils interchange le meme signal SIGUSR1

Dans le handler on doit tester sur le pid du processus appelant pour distinguer le fils et le père car le père et son fils interchange le même signal SIGUSR1.

La fonction usleep permet d'endormir le process durant 5000 ms

Exercice 06 :

Le script :

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

int seconds = 0;
int minutes = 0;
int hours = 0;
pid_t pid_h , pid_m , pid_s , ppid; // sauvgarder les pids dans des var globals pour les
utiliser dans les signaux

void handle_hours(int signum) {
    hours++;
    if (hours == 24) { // incrementer les heures
        hours = 0;
    }
}

void handle_minutes(int signum) {
    minutes++;

    if (minutes == 60) {
        minutes = 0;
        kill(pid_h, SIGUSR2); // envoyer signal SIGUSRÉ a process heure
    }
}

void handle_seconds(int signum) {
    seconds++;
```

```

if (seconds == 60) {
    seconds = 0;
    kill(pid_m, SIGUSR1);           //envoyer signal SIGUSR1 a process minute
}
}

void handler_affichage() {

    if(pid_h==getpid()){
        printf("\n heure : %d\n",hours);
        exit(1);
    }
    else
    if(pid_m==getpid()){
        printf("\n minute : %d\n",minutes);           // ce handler permet d'afficher le
temps                                                    // ce handler sera affecter a SIGINT
        exit(1);
    }
    else                                                    // et aussi terminer l'execution de
tous les process
    if(pid_s==getpid()){
        printf("\n seconds : %d\n",seconds);
        exit(1);
    }
    if(getpid()==ppid) {
        exit(1);
    }
}

int main() {

    ppid= getpid();

    signal(SIGINT , handler_affichage) ;           // appuyer sur CTRL C pour avoir la date

    signal(SIGUSR1, handle_minutes);
    signal(SIGUSR2,handle_hours);

    if (pid_h=fork() == 0) {           // process heur doit attendre avec pause
        pid_h=getpid();
        while(1)pause();
    }else{
        if (pid_m=fork() == 0)           // process minute doit attendre avec pause
        {
            pid_m=getpid();
            while(1)pause();
        }else{

```

```

    if(pid_s=fork()==0) {
        pid_s=getpid();
        signal(SIGALRM, handle_seconds);           // process second recoit un
SIGALRM
        while(1) {
            alarm(1);                               // il se declenche chaque 1 second
            pause();
        }
    }
}
// Le pere attend la fin de tous ses fils

while(wait(NULL) != -1);
return 0;
}

```

Explication :

- Le père crée 3 fils heure minute et seconds , leur pid est sauvegardé dans les variables pid_h , pid_m , pid_s respectivement
- Les processus possède un compteur second , minute et d'heur qui servent a stoqué le temps
- Le processus second s'exécute par un signal SIGALARM de 1 second , on lui affecte le handler des seconds qui incrémente le compteur et teste si on est arrivé a 60 second pour envoyer un signal SIGUSR1 au processus de minute , ce dernier fait incrémente le compteur de minute et si on arrive a la 60 -ème minute , on envoie le signal SIGUSR2 au processus heur pour qu'il incrémente le compteur des heures.
- Le processus père attend la fin de tous ses fils par une boucle sur le wait
- Ce programme ne se termine qu'en envoyant un signal SIGINT par une commande CTRL C le signal sigint est modifié par le handler d'affichage.
Chaque processus affiche son compteur et se termine par un exit , le père est le dernier a se terminer.

Resultat d'execution :

```
sami@laptop:~/SE_RSD/tp1$ gcc exo6.c -o exo6
sami@laptop:~/SE_RSD/tp1$ ./exo6
^C
minute : 10
seconds : 26
heure : 0

sami@laptop:~/SE_RSD/tp1$ gcc exo6.c -o exo6
sami@laptop:~/SE_RSD/tp1$ ./exo6
^C
heure : 0
seconds : 3
minute : 3
```

END 🤖