
Compte-rendu TP2 OS302

Dougy Hugo & Creton Pierre

Introduction

Durant ce TP, nous allons nous intéresser aux signaux. Nous allons appliquer les connaissances vues en cours pour gérer les processus entre eux.

Exercice 1

Implémentation

Pour cette exercice, nous avons utilisé les fonctions `kill()`, `pause()` et `signal()` pour synchroniser les deux processus. la fonction `pause()` nous permet de mettre le procesus en attente de signal. Grace à la fonction `kill`, on peut lui en transmettre un.

Cependant, exectuer `kill(fils,SIGUSR1)` ne suffi pas pour réveiller le processus fils, car comme vu en cours la fonction `pause()` termine le processus lorsqu'il recevra `SIGUSR1`. Nous devons donc utiliser le handler, ici notre handler est la fonction du même nom qui ne fait rien, car on ne veut rien faire de particulier (juste réveiller le processus, ce que fait `pause` quand il recevra `SIGUSR1`). Nous avons donc utiliser la fonction `signal` pour exécuter la fonction handler (même si elle ne fait rien) à la receprtion du signal `SIGUSR1`.

Code

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  //Fonction handler qui s'execute à la reception du signal SUGUSR1.
7  void handler(int sig){}
8
9  int main (){
10     signal(SIGUSR1,handler); //on exécute la fonction handler à la ré
        ception du SIGUSR1
11     pid_t fils = fork();
12     if (fils == 0){ //si on est dans le fils
13
14         pid_t pere = getppid();
15         for(int i=2;i<=100;i+=2){ //le fils affiche les nombres paires
16
17             pause(); //le processus se met en pause, il attend d'être ré
                veillé par un signal
18
```

```
19     printf("%d\n",i);
20
21     kill(pere,SIGUSR1); //le fils envoie à son tour le signal SIGUSR1
    au père pour le réveiller
22 }
23 }
24 else if (fils == -1){ //cas d'erreur
25     perror("Le fork n'a pas réussi");
26     return EXIT_FAILURE;
27 }
28 else{ //si on est dans le père
29
30     for(int i=1;i<=100;i+=2){ //le père affiche les nombres impaires
31
32         sleep(1); //on attend 1s pour être sur que le fils soit en pause.
33
34         printf("%d\n",i);
35
36         kill(fils,SIGUSR1); //on envoie le signal SIGUSR1 au processus
    fils.
37
38         pause(); //on attend de recevoir un signal du fils pour afficher
    à nouveau.
39     }
40 }
41 return EXIT_SUCCESS;
42
43 }
```

Résultat

À l'exécution de ce programme nous obtenons bien le résultat escompté.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP2$ ./ex1
Père: 1
Fils: 2
Père: 3
Fils: 4
Père: 5
Fils: 6
Père: 7
Fils: 8
Père: 9
Fils: 10
Père: 11
Fils: 12
Père: 13
Fils: 14
Père: 15
```

Exercice 2

Implémentation

Pour ce programme, on utilise une fonction supplémentaire: `alarm()`. Cette fonction va envoyer un signal `SIGALRM` toutes les `n` secondes (`n` donné en argument).

On utilise 3 fonctions qui permettent de gérer les secondes, les minutes et les heures. C'est ces fonctions qui seront affectées à un signal avec la fonction du même nom.

On affecte donc la fonction qui permet de gérer les secondes à la réception du signal `SIGALRM` pour le petit fils. Une fois `MAX_S` atteint, on passe aux minutes grâce à la fonction `M` affecté à la réception du signal `SIGUSR1` pour le fils etc...

On a 3 processus différents pour gérer le temps.

Code

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
```

```
5
6 #define MAX_S 60
7 #define MAX_M 60
8 #define MAX_H 24
9
10
11 void H(int sig){ //fonction pour gérer les heures
12     static int heure = 0;
13     heure++;
14     printf("\t\t\t\t%d heure(s)\n",heure);
15     if( heure == MAX_H)heure = 0;
16 }
17
18 void M(int sig){ //fonction pour gérer les minutes
19     static int minute = 0;
20     pid_t pere = getppid();
21     minute++;
22     if (minute==MAX_M){ //quand on atteint 60 minutes, on envoie le
23         signal SIGUSR1 à son père pour compter les heures
24         kill(pere,SIGUSR1);
25         minute=0;
26     }
27     else{
28         printf("\t\t\t\t%d minute(s)\n",minute);
29     }
30 }
31
32 void S(int sig){ //fonction pour gérer les secondes
33     static int seconde = 1;
34     printf("%d Seconde(s)\n",seconde);
35     seconde++;
36
37     pid_t pere = getppid();
38
39     if (seconde==MAX_S){ //quand on a atteint 60s, on envoie le signal
40         SIGUSR1 à son père pour compter les minutes.
41         kill(pere,SIGUSR1);
42         seconde=0;
43     }
44 }
45
46 int main(){
47
48     signal(SIGUSR1,H); //on exécute la fonction H à la réception du
49     SIGUSR1 pour le processus père
50     pid_t fils_minute = fork();
51
52     if (fils_minute == 0){ //si on est dans fils_minute
53
54         signal(SIGUSR1,M); //on exécute la fonction M à la réception du
55         signal SIGUSR1 pour le processus fils_minute
```

```
52     pid_t fils_seconde =fork();
53
54     if(fils_seconde ==0){ //si on est dans fils_seconde
55         signal(SIGALRM,S); //on exécute la fonction S à la réception du
           signal SIGALRM dans le processus fils_seconde
56         while(1){
57             alarm(1); //la fonction alarm va envoyer un signal SIGALRM après
           une seconde
58             pause(); //on attend de recevoir le prochain SIGALRM de la
           fonction alarm.
59         }
60
61     }
62     else if (fils_seconde == -1){ //cas d'erreur
63         perror("Le fork n'a pas réussi");
64         return EXIT_FAILURE;
65     }
66     else{ //minute
67         while(1){
68             pause();
69         }
70     }
71 }
72 else if (fils_minute == -1){ //cas d'erreur
73     perror("Le fork n'a pas réussi");
74     return EXIT_FAILURE;
75 }
76 else{ //heure
77     while(1){
78         pause();
79     }
80 }
81 return 0;
82 }
```

Résultat

On obtient ainsi le résultat suivant:

```
55 Seconde(s)
56 Seconde(s)
57 Seconde(s)
58 Seconde(s)
59 Seconde(s)
    1 minute(s)
60 Seconde(s)
61 Seconde(s)
62 Seconde(s)
63 Seconde(s)
64 Seconde(s)
65 Seconde(s)
66 Seconde(s)
^C
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP2$
```

Quand on a compté 86 400 secondes soit une journée, on remet le compteur des heures à 0 et on recommence à compter.

Exercice 3

Implémentation

Nous avons utilisé les mêmes fonctions que dans les exercices précédent pour cet exercice.

Il fonctionne comme ceci : les fils attendent un signal du père, tandis que le père lui agit differement en fonction des signaux qu'il recoit : soit il continue l'exécution, soit il arrete tout à la mort d'un fils

On a créé 3 handler :

fin : tuer les fils lorsqu'un est meurt, grâce à la fonction `kill(0,SIGUSR2)` qui va faire echo avec le deuxieme handler : **stop**. Le 0 dans le kill permet de d'envoyer un signal à tous les signaus du même groupe, donc dans notre le cas le père envoie le signal SIGUSR2 à lui même et à ses fils.

stop : pour arreter les 5 fils encore vivant.

wake_up : pour ne rien faire lorsque l'on recoit un signal, cela permet de pourvoir juste attendre la réception d'un signal

Cette instruction : `signal(SIGUSR2, wake_up)` ; permet ausssi au père de ne rien faire lorsqu'il recoit SIGUSR2 (sans cela, à la fin du programme quand on `kill(0,SIGUSR2)` un message disant que le signal

SIGUSR2 est reçu était écrit dans le terminal)

Code

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  #define N 6
7  #define TAILLE_MAX 20
8
9  int lire_valeur(const char *path);
10
11 void wake_up(int sig){} //fonction qui ne fait rien pour pouvoir juste
    réveiller un processus mis en pause
12 void stop(int sig);
13 void fin(int sig);
14
15
16 int main (int argc, char *argv[]){
17
18     if (argc != 2) {
19         printf("Utilisation : %s nbre-de-processus !\n", argv[0]);
20         return EXIT_FAILURE;
21     }
22
23
24     signal(SIGUSR1,wake_up);
25     signal(SIGUSR2,wake_up); //pour ne rien faire à la réception de
        SIGUSR2, car le père va se ping lui meme avec se signal avec un
        kill(0,SIGUSR2)
26     signal(SIGCHLD,fin); //quand le fils est mort (signal SIGCHILD) on ex
        éctue la fonction fin qui va envoyé le signal de "libération" aux
        autres fils
27
28
29     pid_t fils[N]; //on créer un tableau de N processus fils
30
31     for (int i = 1 ; i <= N ; i++) {
32
33         fils[i]= fork();
34
35         if (fils[i] == 0){ // on est dans le ième fils
36             printf("%d est créé\n",getpid());
37
38             if (fils[i] == -1){ //cas erreur dans la création du fils
39                 perror("Le fork n'a pas réussi");
40                 return EXIT_FAILURE;
```

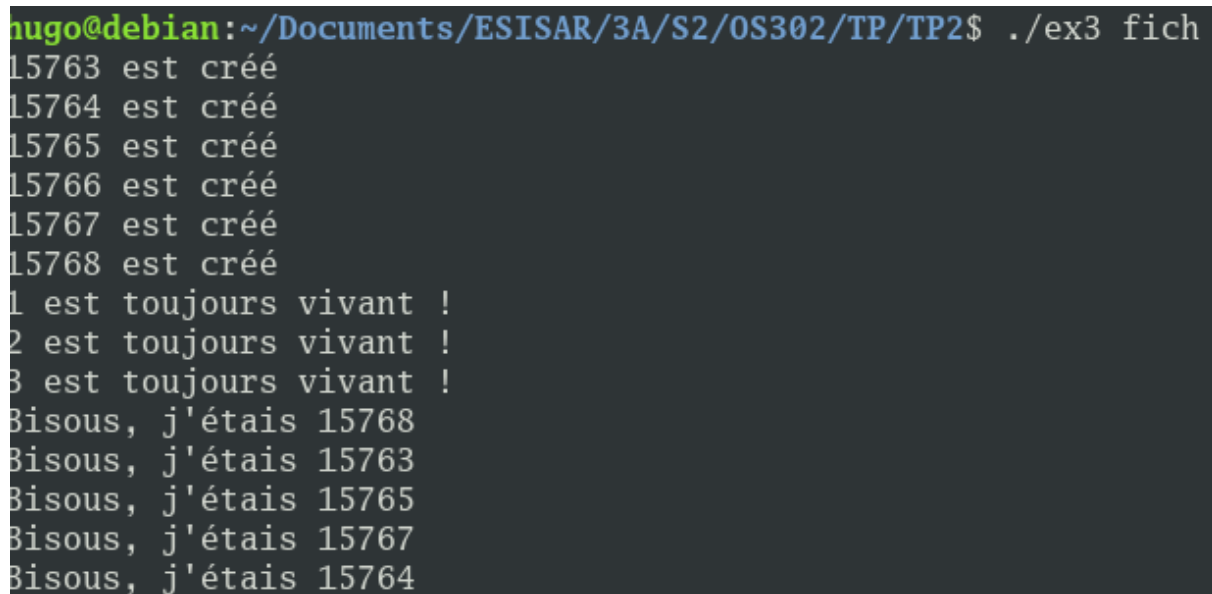


```
41     }
42     signal(SIGUSR1, wake_up);
43     signal(SIGUSR2, stop); //on exécute stop quand on recoit SIGUSR2
                             , donc quand le pere nous "libère"
44
45     while(1){
46         pause(); //on attend d'être réveillé par wake_up, donc par le p
                     ère qui nous envoie un SIGUSR1
47
48         int barillet = lire_valeur(argv[1]); //on lit la valeur du
                     barillet dans le fichier passé en argument
49
50         if (barillet == i){ //si le numéro est celui du fils il se tue.
51             kill(getpid(), SIGKILL); //on lui envoie le signal SIGKILL
                     pour qu'il se tue.
52         }
53         else{ //on affiche les fils qui sont toujours vivants
54             printf("%d est toujours vivant ! \n", i);
55             pid_t pere = getppid();
56             kill(pere, SIGUSR1);
57         }
58     }
59
60 }
61 }
62 sleep(1); //pour etre sur que tous les fils sont créés
63 for (int i = 1 ; i <= N ; i++){
64     kill(fils[i], SIGUSR1); // pour chaque fils on va le réveiller avec
                     le signal SIGUSR1
65     pause(); //on attend d'être réveillé par wake_up, donc soit que le
                     fils nous dise qu'il est toujours vivant, soit qu'un fils est
                     mort
66 }
67 return 0;
68 }
69
70
71 void stop(int sig){
72     pid_t pid = getpid();
73     printf("Bisous, j'étais %d\n", pid);
74     exit(0);
75 }
76
77 void fin(int sig){
78     kill(0, SIGUSR2);
79     sleep(1); //pour être sur que tous les fils se sont bien arrêtés
                     avant d'arrêter le père
80     exit(0);
81 }
82
83 int lire_valeur(const char *path){
```

```
84 FILE *fichier;
85 char chaine[TAILLE_MAX];
86 int valeur;
87 fichier = fopen(path, "r");
88 if (fichier != NULL) {
89     fgets(chaine, TAILLE_MAX, fichier);
90     fclose(fichier);
91     valeur = atoi(chaine);
92 }
93 return valeur;
94 }
```

Résultat

On a créé un fichier `fich` contenant un nombre entre 1 et 6. Pour l'exemple, on prend le nombre 4.



```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP2$ ./ex3 fich
15763 est créé
15764 est créé
15765 est créé
15766 est créé
15767 est créé
15768 est créé
1 est toujours vivant !
2 est toujours vivant !
3 est toujours vivant !
Bisous, j'étais 15768
Bisous, j'étais 15763
Bisous, j'étais 15765
Bisous, j'étais 15767
Bisous, j'étais 15764
```

On remarque bien qu'il n'y a bien que 5 processus qui donne leur identité à la fin. Donc un des fils est bien mort de la roulette russe.

Travail collaboratif

Pour ce TP, nous nous sommes réparti le travail de la manière suivante:

Exercice 1:

- *Hugo & Pierre*: réflexion et documentation
- *Hugo & Pierre*: implémentation & tests

- *Hugo*: commentaires

Exercice 2:

- *Hugo & Pierre*: réflexion et documentation
- *Hugo & Pierre*: implémentation & tests
- *Hugo & Pierre*: commentaires

Exercice 3:

- *Pierre*: réflexion
- *Hugo*: documentation
- *Hugo & Pierre*: implémentations & tests
- *Pierre*: commentaires

Conclusion

Pour conclure sur ce TP, nous avons pu utiliser les fonctions vues en cours et comprendre leur fonctionnement avec des programmes plus ou moins complexes.

Nous avons dû lire les pages man des fonctions pour pouvoir se les approprier correctement.