Faculté des Sciences & Techniques — Université de Limoges

Licence 3ème année

Parallélisme I

Corrections – TD n°1

Manipulation des appels système fork, pipe

Manipulation de fork & pipe

1 – Il y a 8 affichage de « Salut ».

2 – Combien de lignes « salut » affiche chacun de ces programmes :

Programme 1 :

```
1 int main() {
2    int i;
3    for (i=0; i<2; i++)
4       fork();
5    printf("salut\n");
6    exit(0);
7    }
```

Il y a 4 affichages.

Programme 2 :

```
1 void go() {
2    fork();
3    fork();
4    printf("salut\n");
5 }
6 int main() {
7    go();
8    printf("salut\n");
9    exit(0);
10 }
```

Il y a 8 affichages.

Programme 3 :

```
1 int main() {
2    if (fork())
3       fork();
4    printf("salut\n");
5    exit(0);
6    }
```

Il y a 3 affichages.

Programme 4 :

```
1 int main() {
2    if (fork() == 0)
3    { if (fork())
4       {
5          printf("salut\n");
6       }
7    }
8 }
```

Il y a 1 affichage.

3 – Écrire un programme qui crée 10 processus fils :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void travail_enfant(int rang)
5 {   int i;
6    for(i=0; i< 10; i++)
7    {   sleep(1);
8       printf("%d\n",rang);
9    }
10   exit(0);
11 }
12 int main()
13 {   int i;
14   for (i=0; i<10; i++)
15   {
16      if (fork() == 0)
17      { /* je suis l'enfant */
18         travail_enfant(i);
19      }
20   }
21 }
```

4 – Crible d'Erathostène :

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

#define LECTURE 0
#define ECRITURE 1

int main()
{
   int i;
   int tube_successeur[2];

   pipe(tube_successeur);
   if (fork() != 0)
   {
      /* je suis le pere */
      close(tube_successeur[LECTURE]);
      for(i = 2; i < 100; i++)
         write(tube_successeur[ECRITURE], &i, sizeof(int));
      close(tube_successeur[ECRITURE]);
      wait(NULL); /* On attend la fin du premier fils */
   }
   else
   {
      /* je suis le fils */
      int premier, tube_predecesseur[2], valeur, existe_successeur = 0;

      close(tube_successeur[ECRITURE]);
      tube_predecesseur[LECTURE] = tube_successeur[LECTURE];
      read(tube_predecesseur[LECTURE], &premier, sizeof(int));
      printf("Valeur premiere : %d\n", premier);

      while (1) /* Boucle infinie, la sortie est prévue lors de la fermeture du tube */
      {
         int resultat;

         resultat = read(tube_predecesseur[LECTURE], &valeur, sizeof(int));
         if (!resultat)
            exit(0); /* Sortie lorsque le tube est fermé, et la lecture impossible */
         if ((valeur % premier) != 0)
         {
            if (existe_successeur)
            {
               write(tube_successeur[ECRITURE], &valeur, sizeof(int));
            }
            else
            {
               pipe(tube_successeur);
               if (fork() != 0)
               {   /* je suis le pere */
                  close(tube_successeur[LECTURE]);
                  existe_successeur = 1;
                  write(tube_successeur[ECRITURE], &valeur, sizeof(int));
               }
               else
               {   /* je suis le fils */
                  close(tube_predecesseur[LECTURE]);
                  close(tube_successeur[ECRITURE]);
                  tube_predecesseur[LECTURE] = tube_successeur[LECTURE];
                  read(tube_predecesseur[LECTURE], &premier, sizeof(int));
                  printf("Valeur premiere : %d\n", premier);
               }
            }
         }
      }
   }
}
```

Manipulation des signaux

5 – Écrire un programme P créant deux fils :

- P envoie le signal SIGUSR1 à son second fils ;
- à la réception de ce signal, le second fils envoie le signal SIGUSR2 au premier fils (qui provoque sa terminaison) avant de s'arrêter.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <signal.h>
 4
 5 int pid_enfant1;
 6
 7 void enfant1(int numero_signal)
 8 {
 9    fprintf(stderr,"Enfant1 reception signal de enfant2\n");
10    exit(0);
11 }
12
13 void enfant2(int numero_signal)
14 {
15    fprintf(stderr,"Enfant2 envoi signal a enfant1\n");
16    kill(pid_enfant1, SIGUSR2);
17    exit(0);
18 }
19
20 int main()
21 {
22    int pid_enfant2;
23    pid_enfant1 = fork(); /* renvoie 0 au fils et pid au pere */
24    if (!pid_enfant1)
25    {
26       /* Je suis l'enfant 1 */
27       fprintf(stderr, "Lancement de l'enfant 1\n");
28       signal(SIGUSR2, enfant1);
29       while(1);
30    }
31    else
32       fprintf(stderr, "Creation du processus enfant1 %d\n", pid_enfant1);
33    pid_enfant2 = fork();
34    if (!pid_enfant2)
35    {
36       /* Je suis l'enfant 2 */
37       fprintf(stderr, "Lancement de l'enfant 2\n");
38       signal(SIGUSR1, enfant2);
39       while(1);
40    }
41       else
42       fprintf(stderr, "Creation du processus enfant2 %d\n", pid_enfant2);
43    sleep(1);
44    fprintf(stderr, "Parent envoi signal a enfant2\n");
45    kill(pid_enfant2,SIGUSR1);
46    fprintf(stderr,"Attente Enfant 1\n");
47    waitpid(pid_enfant1, NULL, 0);
48    fprintf(stderr,"Attente Enfant 2\n");
49    waitpid(pid_enfant2, NULL, 0);
50    fprintf(stderr,"Fin\n");
51 }
```

6 – Écrire un programme qui se termine uniquement au 5ème « `Ctrl-C` ».

Avec l'utilisation de « `sigaction` » :

En comptant 5 :

```
 1 #include <stdio.h>
 2 #include <signal.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5
 6 void countFive (int signal) {
 7    static int i=1;
 8
 9    if (i == 5)
10       exit (0);
11    i++;
12 }
13
14 int main () {
15    struct sigaction nvt, old;
16    memset (&nvt, 0, sizeof(nvt));
17    nvt.sa_handler=countFive;
18    sigaction (SIGINT, &nvt, &old);
19    for (;;);
20 }
```

En comptant 4 et en désactivant le support du signal :

```
 1 #include <stdio.h>
 2 #include <signal.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5
 6 struct sigaction nvt, old;
 7
 8 void countFive (int signal) {
 9    static int i=1;
10
11    if (i == 4)
12       sigaction(SIGINT, &old, NULL);
13    i++;
14 }
15
16 int main () {
17    memset (&nvt, 0, sizeof(nvt));
18    nvt.sa_handler=countFive;
19    sigaction (SIGINT, &nvt, &old);
20    for (;;);
21 }
```

Avec l'utilisation de « `signal` » :

```
 1 #include <signal.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 /* int nombre_occurence = 0; */
 6
 7 void compteur(int numero_signal)
 8 {
 9    static int nombre_occurence = 0;
10    nombre_occurence++;
11    if (nombre_occurence == 5)
12       exit(0);
13    printf("Meme pas mal\n");
14 }
15
16 int main(int argc, char **argv)
17 {
18    int n;
19 signal(SIGINT, compteur);
20 for( ; ; );
21 }
```