Corrigé TP Programmation systèmes

Exercie 1 : Ecrire un programme qui affiche les information suivantes associées à un processus:
- Le numero du processus (pid)
- le nuùero du père du processus (ppid)
- l'UID réel du processus (uid)
- l'UID effectif du processus (euid)
- le GID réel du processus (gid)
- le GID effectif du processus (egid)

Un exemple d'exécution est :

```
./a.out
Je suis le processus de pid      : 20011
Mon père est le processus de pid : 5411
Mon uid                          : 322
Mon euid                         : 322
Mon gid                          : 100
Mon egid                         : 100
```

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


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

  pid_t pid;
  pid_t ppid;

  uid_t uid;
  uid_t euid;

  gid_t gid;
  gid_t egid;

  pid=getpid();
```

```
  ppid=getppid();
  uid=getuid();
  euid=geteuid();
  gid=getgid();
  egid=getegid();


  printf("Je suis le processus de pid      : %d",pid);
  printf("\nMon père est le processus de pid : %d ",ppid);
  printf("\nMon uid                          : %d ",uid);
  printf("\nMon euid                         : %d ",euid);

  printf("\nMon gid                          : %d ",gid);
  printf("\nMon egid                         : %d ",egid);

  printf("\n");
  return(0);
}
```

Exercie 2 : Ecrire un programme qui crée un processus fils et qui affiche les informations pid et ppid de chaque processus créer

```
./a.out

Valeur de fork = 22723
Je suis le processus père : pid=22722, ppid=5411,pid fils = 22723

Valeur  de fork = 0
Je suis le processus fils : pid=22723, ppid 22722
```

Exercie 3 : Rependre l'exercice 1 et afficher les informations relatives aux processus père et fils

./a.out

```
Valeur fork = 0
Je suis le processus de pid      : 22851
Mon père est le processus de pid : 22850
Mon uid                          : 322
Mon euid                         : 322
```

```
Mon gid                          : 100
Mon egid                         : 100
Mon repertoire de travail        : "/cergy/homep/profs/rc/exemple"

Valeur fork = 22851
Je suis le processus de pid      : 22850
Mon père est le processus de pid : 5411
Mon uid                          : 322
Mon euid                         : 322
Mon gid                          : 100
Mon egid                         : 100
Mon repertoire de travail        : "/cergy/homep/profs/rc/exemple"
```

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

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

  pid_t pidf;
  pid_t pid;
  pid_t ppid;

  uid_t uid;
  uid_t euid;

  gid_t gid;
  gid_t egid;

  char path[128];
  char* retgetcwd;

  pidf = fork ();

  if (pidf > 0) {
    /* Processus père      */
    sleep(3);
    printf("\n\nValeur de fork = %d",pidf);
```

```
    pid=getpid();
    ppid=getppid();
    uid=getuid();
    euid=geteuid();
    gid=getgid();
    egid=getegid();

    printf("\nJe suis le processus père de pid : %d",pid);
    printf("\nMon père est le processus de pid : %d ",ppid);
    printf("\nMon uid                          : %d ",uid);
    printf("\nMon euid                         : %d ",euid);

    printf("\nMon gid                          : %d ",gid);
    printf("\nMon egid                         : %d ",egid);
    retgetcwd = getcwd(path, 128);
    printf("\nMon répertoire de travail        : \"%s\"\n",path);


  }
else if (pidf == 0)
 {
    /* Processus fils      */
    printf("\nValeur de fork = %d",pidf);
    pid=getpid();
    ppid=getppid();
    uid=getuid();
    euid=geteuid();
    gid=getgid();
    egid=getegid();

    printf("\nJe suis le processus fils de pid : %d",pid);
    printf("\nMon père est le processus de pid : %d ",ppid);
    printf("\nMon uid                          : %d ",uid);
    printf("\nMon euid                         : %d ",euid);

    printf("\nMon gid                          : %d ",gid);
    printf("\nMon egid                         : %d ",egid);
    retgetcwd = getcwd(path, 128);
```

```
    printf("\nMon repertoire de travail        : \"%s\"\n",path);

  }
  else
  {
    /* Traitement d'erreur */
    printf("\nErreur de création\n");
  }

  /* On termine pour chaque process.  */
  /* il n'y a pas de synchronisation. */

  exit(0);
}

/* Exemple utilisation primitive fork() sous UNIX */

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

 void main(void)
{
    int pid; /* PID du processus fils */
    int i;
    pid = fork();
    switch (pid)
    {
     case -1:
       printf("Erreur: echec du fork()\n");
       exit(1);
       break;
     case 0:
       /* PROCESSUS FILS */
       printf("Processus fils : pid = %d et mon ppid = = %d\n", getpid(),getppid());
       exit(0); /* fin du processus fils */
       break;
     default:
        /* PROCESSUS PERE */
```

```
        printf("Ici le pere: le fils a un pid=%d\n", pid );
        wait(0); /* attente de la fin du fils */
        printf("Fin du pere.\n");
  }
}
```

Exécution concurrente des processus père et fils

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

1.  void main()
2.  {
3.          pid_t p;
4.          p=fork();
5.          switch(p)
6.          {
7.          case (0):
8.                  //sleep(15);
9.                  printf("Le fils pid est =%d et mon ppid est=%d\n", getpid(), getppid());
10.                 break;
11.         case (-1):
12.                 //sleep(15);
13.                 printf("Erreur fork\n");
14.                 break;
15.         default:
16.                 printf("Le pere pid est =%d et mon ppid est=%d\n", getpid(), getppid());
17.         }
18.         printf("Fin du processus %d\n",getpid());
19. }
```

2. **Quelques attributs hérités et copie**

A) Enoncé

Reprendre le programme ci-dessus et compléter en affichant l'uid, le gid, et le contenu d'une variable x initialisée à 2 (avant le fork) et modifié selon x+3 par le fils et selon x*5 par le père.

B) Corrigé

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

   1.   void main()
   2.   {
   3.      pid_t p;
   4.      int x = 2;
   5.      p=fork();
   6.      switch(p)
   7.      {
   8.         case (0):
   9.            x = x + 3;
10.     printf("Le fils pid=%d ppid=%d uid=%d gid=%d x=%d\n", getpid(), getppid(), getuid(), getgid(), x);
11.     break;
12.     case (-1):
13.            printf("Erreur fork\n");
14.            break;
15.     default:
16.             x = x * 5;
17.            printf("Le fils pid=%d ppid=%d uid=%d gid=%d x=%d\n", getpid(), getppid(), getuid(), getgid(),
x);
18.     }
19.            printf("Fin programme\n");
     }
```

3. Cas particulier des descripteurs de fichiers

A) Enoncé

Ecrire un programme qui ouvre un fichier nommé « toto », en lecture et écriture, dont le contenu est la suite 123456789. Le programme fork ensuite; le fils écrit ab dans le fichier ensuite il s'endort et il lit 2 caractères; le père s'endort, lit 2 caractères et écrit AB.

B) Corrigé

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

1.  void main()
2.  {
3.          pid_t p;
4.          char chaine[3];
5.          int desc;
6.          desc=open("toto",O_RDWR,0);
7.          p=fork();
8.          switch(p)
9.          {
10.         case (0):
11.                 printf("Le fils PID=%d PPID=%d\n", getpid(), getppid());
12.                 write(desc,"ab",2);
13.                 sleep(10);
14.                 read(desc,chaine,2);
15.                 chaine[2]='\0';
16.                 printf("chaine lue par le fils %s\n", chaine);
17.                 close(desc);
18.                 break;
19.         case (-1):
20.                 printf("Erreur fork\n");
21.                 break;
22.         default:
23.                 printf("Le père Valeur de retour du fork=%d\n", p);
24.                 printf("Le père PID=%d PPID=%d\n", getpid(), getppid());
25.                 sleep(3);
26.                 read(desc,chaine,2);
27.                 chaine[2]='\0';
28.                 printf("chaine lue par le père %s\n", chaine);
29.                 write(desc,"AB",2);
30.                 close(desc);
31.         }
32.                 printf("Fin du Programme\n");
33.         }
```

4. Synchronisation des processus père et fils par la commande wait

A) Enoncé

Ecrire un programme qui prenant une matrice de taille 2*2 en paramètre et crée quatre fils. Chaque fils calcule un élément du carré de la matrice initiale et le renvoi au processus père comme code de retour.

B) Corrigé

#include<sys/types.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>

```
1.   int main (int argc, char ** argv)
2.   {
3.     pid_t pid_list[4];
4.
5.     int a,b,c,d,i, res[4];
6.
7.     a = atoi(argv[1]);
8.     b = atoi(argv[2]);
9.     c = atoi(argv[3]);
10.    d = atoi(argv[4]);
11.    for (i = 0; i < 4; i++)
12.    {
13.      pid_list[i] = fork();
14.      if (pid_list[i] == 0)
15.      {
16.        break;
17.      }
18.      printf("Fils créé\n");
19.    }
20.    //luxe car le pere a 4 pour valeur de i et ne passe donc pas dans le
```

```
21.    //switch....
22.    if (pid_list[i] == 0)
23.    {
24.      //On calcule la valeur correspondante
25.      switch (i)
26.      {
27.        case 0 : return a*a+b*c;
28.        case 1 : return a*b+b*d;
29.        case 30.        case 2 : return a*c+d*c;
31.        case 3 : return b*c+d*d;
32.      }
33.    }
34.    else
35.    {
36.      for (i = 0; i < 4; i++)
37.      {
38.        waitpid(pid_list[i], res+i, 0);
39.      }
40.      for (i = 0; i < 4; i++)
41.      {
42.        printf("%d\t", WEXITSTATUS(res[i]));
43.      }
44.    }
```

5. Recouvrement d'un processus

A) Enoncé

Ecrire un interpréteur de commandes externes (exemples ps, ls, gcc, ...).

B) Corrigé

```
#include<stdio.h>
#include<errno.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>

void main(int argc, char * argv[])
```

```
{
pid_t pid;
int statut;
char commande[10];
strcpy(commande, "");
while(strcmp(commande, "exit")!=0)
      {
      printf("$");
      fflush(stdin);
      scanf("%s", commande);
      pid = fork();
      switch (pid)
            {
            case -1:
              perror("erreur de fork\n");
              exit(-1);
            case 0:
              printf("%s PPID = %d, PID = %d \n", commande, getppid(),getpid());
              if (strcmp(commande,"exit")!=0) execlp(commande, commande, NULL);
              perror("execlp");
              exit(-1); //sinon le fils continue
            default:
              pid = wait (&statut);
              if (pid==-1) perror("le wait");
              else printf("processus zombi %d\n", pid);
            }
      }

printf ("\n \t tout est termine\n");

}
```

1. **Synchronisation des processus père et fils par la commande wait**

   A) Enoncé

   Ecrire un programme qui prenant une matrice de taille 2 en paramètre et crée quatre fils. Chaque fils calcule un element du carré de la martice initiale et le renvoi au processus père comme code de retour.

   B) Corrigé

```
2.      int main (int argc, char ** argv) {
3.
4.          pid_t pid_list[4];
5.          int a,b,c,d,i, res[4];
6.
7.          a = atoi(argv[1]);
8.          b = atoi(argv[2]);
9.          c = atoi(argv[3]);
10.         d = atoi(argv[4]);
11.         for (i = 0; i < 4; i++)
12.         {
13.             pid_list[i] = fork();
14.             if (pid_list[i] == 0)    {   break;  }
15.             printf("Fils créé\n");
16.         }
17.
18.         if (pid_list[i] == 0)
19.         {
20.             //On calcule la valeur correspondante
21.             switch (i)
22.             {
23.                 case 0 : return a*a+b*c;
24.                 case 1 : return a*b+b*d;
25.                 case 2 : return a*c+d*c;
26.                 case 3 : return b*c+d*d;
27.              }
28.         }
29.         else
30.         {
31.              for (i = 0; i < 4; i++)
32.             {
33.                 waitpid(pid_list[i], res+i, 0);
34.             }
35.             for (i = 0; i < 4; i++)
36.             {
```

```
37.                 printf("%d\t", WEXITSTATUS(res[i]));
38.             }
39.         }
    }
```