UNIVERSITE A.MIRA-BEJAIA
FACULTE DES SCIENCES EXACTES
DEPARTEMENT INFORMATIQUE

CORRIGE DES TRAVAUX PRATIQUES
PROGRAMMATION SYSTEME
Série 1 : Gestion de Processus

Exercice1 :

Q1) Dérouler le programme et expliquer chacune des instructions.

```
#include <stdio.h>
int main( void ) {
      int pid = fork();
      if ( pid == 0 ) {
            printf( "C'est le processus fils qui affiches\n" );
      } else {
            printf( "C'est le processus père qui affiche:\n"
                    " Le pid du fils est: %d\n", pid );
      }
      return 0; }
```

Q2) Dérouler le programme plusieurs fois et vérifier si le processus père s'exécute avant le processus fils.

Q3) Ecrire un programme qui crée un nouveau processus et qui affiche pour les deux processus (père et fils) les caractéristiques générales d'un processus :

- Identifiant du processus.
- Identifiant du père du processus.
- Répertoire de travail du processus.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main( void ) {
      int pid = fork();
      if ( pid == 0 ) {
            printf( "C'est le processus fils qui affiches\n" );
            int pidf=getpid() ;
            int pidp=getppid() ;
            char buf[20];
            char* S=getcwd(buf,20);
           printf("mon pid:%d, pid du pere:%d, repertoire de
travail :%s\n ", pidf, pidp,,buf) ;
      } else {
            printf( "C'est le processus père qui affiche:\n"
                    " Le pid du fils est: %d\n", pid );
                    int monpid=getpid() ;
                    int pidp=getppid() ;
                    char buf[20];
                    char* S=getcwd(buf,20);
                  printf("mon pid: %d, pid du pere:%d, repertoire de
travail :%s\n ", pidf, pidp, buf) ;
      }
      return 0;}
```

Exercice2 : Les primitives Exit() et Wait() :
Q1) Ecrire un programme qui positionnes une attente de quelques secondes (`sleep(10)`) dans le processus fils pour que le processus père se termine avant le fils.

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

int main( void ) {
     int pid = fork();
     if ( pid == 0 ) {
          printf( "C'est le processus fils qui affiches\n" );
          sleep(10) ;
     } else {
          printf( "C'est le processus père qui affiche:\n");
     }
     return 0; }
```

Q2) Puis ajouter les instructions qui permettront au processus père d'attendre la terminaison de son fils et qui afficheront le code retour de celui-ci.

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

int main( void ) {
     int etat;
     int pid = fork();
     if ( pid == 0 ) {
          printf( "C'est le processus fils qui affiches\n" );
          sleep(10) ;
          exit(0) ;
     } else {
          printf( "C'est le processus père qui affiche:\n");
          int pidf=wait(&etat) ;
          printf( "Le pid du processus fils: %d\n", pidf);
     }
     return 0; }
```

Exercice3 : Les primitives de la famille exec() :

Q) A l'aide des primitives fork et execl, écrire un programme qui crée 2 processus l'un faisant la commande ls -l, l'autre ps -l. Le père devra attendre la fin de ses deux fils et afficher quel a été le premier processus à terminer.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <unistd.h>
int main( ) {
pid_t pid1,pid2, pid_premier;
int status;
```

```
pid1=fork( );
if (pid1== 0) {
execl("/bin/ls","ls",”-l”, NULL);
exit(0);
}

pid2=fork( );
if (pid2== 0) {
execl("/bin/ps","ps",”-l”, NULL);
exit(0);
}

pid_premier = wait(&status);
wait(&status);
printf("Premier processus a finir : %d\n", (pid_premier==pid1)?1:2);
}
```

Exercice4 : Gestion de threads :

Q1) Ecrire un programme qui permet de créer 3 threads et affichent leur ID.
Q2) Ecrire un programme à trois threads qui font appel à une fonction qui modifie une variable i partagée par tous.

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

int i;
void addition (){
int tid=pthread_self();
i=i+10;
printf("thread fils %d  i= %d/n", tid, i);
}

main()
{
pthread_t th1, th2, th3;
i=0;

pthread_create(&th1,NULL, &addition, NULL);
pthread_create(&th2,NULL, &addition, NULL) ;
pthread_create(&th3,NULL, &addition, NULL);

i=i+1000;

printf("thread principal %d\n",i);

pthread_join(th1,NULL);
pthread_join(th2,NULL);
pthread_join(th3,NULL);
}
```

-FIN-