Solution : Exercises

Exercise n°1

Write a program that creates N child processes in parallel, then waits for them to terminate. Each process created must display its pid, the pid of its parent and its creation order number.

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

int main () {
int N = 4;
for (int i =0 ; i < N ; i++ ) {
if (fork() == 0) { // code child
 printf("Child %d, PID %d, PPID %d \n"i, getpid(), getppid());
 exit(i);
 }
}
for (int i =0 ; i < N ; i++ ) {
wait(nullptr);
}
return 0;
}
```

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

int main () {
int N = 4;

for (int i =0 ; i < N ; i++ ) {
if (fork() == 0) { // code child
  printf("Child %d, PID %d, PPID %d \n"i, getpid(), getppid());
  exit(i);
 }
}

for (int i =0 ; i < N ; i++ ) {
int status;
int pid = wait(&status);

 printf("Termination of child whose pid:  %d, return code: %d \n" pid, WEXITSTATUS(status));
 }
return 0;
}
```

Exercise n°2

Write a program using the functions *fork()*, *exit()* and *wait()* to transmit an ASCII character from the child process to the parent process. The child process reads the character using the function *getchar()*. The parent process displays the ASCII code of this character and transforms it into uppercase using the function *toupper(char c)*.

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

int main(void) {
int car;
int pid;
pid =fork();
if (pid) {
 wait(&car);
 printf("Process parent displays the ascii code of the character :%d\n",car>>8);
 printf("The uppercase of this character :%c\n",toupper(car>>8));
 exit(0);
 }
else {
 printf("Child process: enter a character \n");
 car = getchar();
 exit(car);
 }
return 0;
}
```

Exercise n°3

Use the compiler “c” under Linux “gcc” to compile the following program.

#gcc -c filename.c

#gcc -o exe-name filename.o

#./exe-name

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

int main(void){
int x,pid,s;

pid =fork();
if (pid){
         wait(&x);
         s = x>>8;
         s = s*s;
         printf("The parent process calculates the square of x :%d\n",s);
         exit(0);
         }
else {
      printf("entrer la valeur de x : \n");
      scanf("%d",&x);
      exit(x);
     }
return 0;
}
```

1) Indicate the result displayed by this program.
2) Indicate the relation between *exit()* and *wait()*.

Exercise n° 4

Write a program whose parent, after creating three children (f1, f2, f3), waits for these three children to return before performing the calculation 3 × 10 + 5.

The data:

- The child f1 returns the value 5;
- The child f2 returns the value 10;
- The child f3 returns the value 3.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <wait.h>
int status, i, som=0, val[2];
int main() {
pid_t pid[2],retpid;
if ((pid[0] = fork()) == 0)  exit(5);              /* fils 1 */
if ((pid[1] = fork()) == 0)   exit(10);            /* fils 2 */
if ((pid[2] = fork()) == 0)   exit(3);             /* fils 3 */
/* le pere attend la fin de ses fils */
i = 0;
while (i<=2) {
 retpid = waitpid(pid[i], &status, 0);
if (WIFEXITED(status)){
   printf("Le fils %d s’est terminé normalement avec le code %d\n",
retpid, WEXITSTATUS(status));
   val[i]=WEXITSTATUS(status);
  }
else  printf("Le fils %d s’est terminé anormalement\n", retpid);
i++;
}
som = val[2]*val[1]+val[0];
printf("la valeur retournee est : %d \n", som);

//if (errno != ECHILD)
//perror("erreur dans waitpid");
exit(0);
}
```

Exercise n° 5

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

int main(void) {
  if (fork()==0) {  //child 1
    printf("T");
    fflush(stdout);
    if(fork()==0){  //child 1.1
      printf("E");
      //fflush(stdout);
      exit(0);
    }
    wait(NULL);
    exit(0);
  }else{
      wait(NULL);
  if (fork()==0) {  //child 2
    printf("S");
    fflush(stdout);
    if(fork()==0){  //child 2.1
      printf("T ");
      fflush(stdout);
      exit(0);
    }
    wait(NULL);
    exit(0);
  }else{
    wait(NULL);
  if (fork()==0) {  //child 3
    printf("T");
    fflush(stdout);
    if(fork()==0){  //child 3.1
      printf("P\n");
      fflush(stdout);
      exit(0);
    }
    wait(NULL);
    exit(0);
      }
    }
   }
}
```