***Rapport de Devoir TP système ACADC***

Réaliser par :

**Nom : YOUSFI**

**Prénom : Zakaria**

**Matricule : 171732026950**

&

**Nom : MESSILI**

**Prénom : Islem**

**Matricule :181832024108**

**Le Code :**

///////////////////////////////////////////////////////////////////////////////////

//Réaliser par **Yousfi Zakaria** et **Messili Islem**

//Importation de bibliothèque

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

// variables global

int pf=0 ;//Pour distingue entre les processus père et les deux fils

int mes=1 ;//Indique la mesure en cours

int boucle=0 ;//Indique la boucle entraine d’exécuter boucle par l’appel de alarm() (attente de 2mn et 1mn)

int debut=0 ;// Pour manipuler la suspension ou la continuation de prises des mesures (combinaison ctrl+z)

arret=0;//Compteur pour calculer le nombre de click sur la combinaison ctrl+c

pid\_t pid1,pid2; // Variables pour contenir les pid des 2 fils

//Un handler qui traite la suspension ou la continuation de prises des mesures par ctrl+z (signal SIGTSTP)

void handtstp(int sig) {

if(debut==0) {

debut = 1;

if (pf == 0) kill(pid1, SIGUSR1);

}

else if (debut == 1) {

debut = 2;

if(pf == 0) {

kill(pid1, SIGSTOP);//suspendre fis1

kill(pid2, SIGSTOP); //suspendre fis2

}}

else{

debut = 1;

kill(pid1, SIGCONT);//remise fis1

kill(pid2, SIGCONT);//remise fis2

}

}

//Un handler qui traite l’arrêt de tout les processus et la sortie de system (ctrl+c X 3)

void handint(int sig) {

if(pf == 0) {

arret++;

if(arret == 3) {

printf("\nTerminison avec succes\n");

kill(pid1, SIGKILL);//terminaison fis1

kill(pid2, SIGKILL);//terminaison fis 2

exit(0);

} } }

// handler qui traite le signal SIGALRM

void handalrm(int sig) {

if(pf==0) kill(pid1, SIGUSR1);

if(pf==1){

if(mes==2){//la mesure 2

printf("processus1\_pid = %d: mesure2\n",getpid());

kill(pid2, SIGUSR2);//envoi de signal SIGUSR2 au fis 1

mes=1;

}

else if(mes==1){// la mesure 1

printf("\nprocessus1\_pid = %d: mesure1\n",getpid());

alarm(60);

mes=2;

}

}

if(pf==2){

if(boucle == 1) {

kill(getppid(), SIGALRM);// envoi de signal SIGALRM au père boucle = 0;

}

else {

printf("processus2\_pid = %d: mesure\n",getpid());

boucle = 1;

kill(getpid(), SIGUSR2);// envoi de signal SIGUSR2 au fis 2

} } }

// handler qui traite le signal SIGUSER1 de fis 1

void handusr1(int sig) { alarm(1);}

// handler handler qui traite le signal SIGUSER2 de fis 2

void handusr2(int sig){

if(boucle==0) alarm(120);

if(boucle==1) alarm(60);

}

int main(){ // prog du principal

printf("Bienvenue!\n");

//Assignation des handler avec leurs signaux

signal(SIGTSTP, handtstp);

signal(SIGINT, handint);

signal(SIGALRM, handalrm);

signal(SIGUSR1, handusr1);

signal(SIGUSR2, handusr2);

pid2=fork(); // création fils 2 (processus 2)

if (pid2 == -1) printf("erreur de fork()\n");

else if (pid2 == 0) {

pf=2;

printf("Le deuxième fis a été créé !\n");

while(1) {}

}

pid1 = fork(); // création fils 1 (processus 1)

if(pid1 == -1 ) printf("erreur de fork()\n");

else if (pid1 == 0) {

pf=1;

printf("Le premier fis a été créé  \n");

while(1){}

}

else {

printf("Ici c’est le père x2\n");

while(1){// programme de fonds

}

}

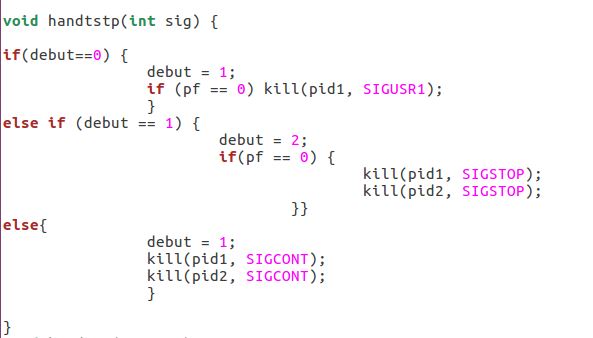
printf("Ici c’est le père x3\n");

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**Explication de code :**

**REMARQUE :** dans le code suivant pour la val d’entrée de la fonction alarm() on a prise 5 second au lieu de 60 seconds et 10 seconds au lieu de 120 seconds juste pour le test



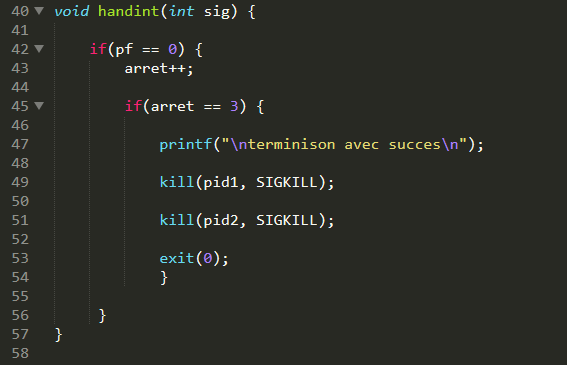
Le handler **handstp** sert à la suspension et la remise en fonction des processus fis 1 et fis 2.

Quand l’utilisateur appuis la premier fois sur la combinaison **crtl+z** la variable « debut » se met à « 1 » , et le père va envoyer un signal au fils pour débuter la prise de mesure par l’instruction« **kill(pid1,SIGUSR1**); »

pour la deuxième fois il vas exécute le « else » car la variable « **debut** » et déférente de 1, ou il met « debut » a 2 et exécute le deux instruction « **kill(pid1, SIGSTOP); kill(pid2, SIGSTOP);** » ou il suspend les deux processus fis.

Pour la troisième fois il met « debut » a 1 et il exécute les deux instructions « **kill(pid1, SIGCONT) ;kill(pid2, SIGCONT);»** ou il remet les deux processus fis en fonctionnement.

**Il répète la deuxième et la troisième étape tanque le programme est en cours d’exécution**.



Le handler **handint** sert à l’arrêt de tous les processus en cours d’exécution (le père et les 2 fis), après qu’on appuis 3 fois sur le la combinaison **ctrl+c**.

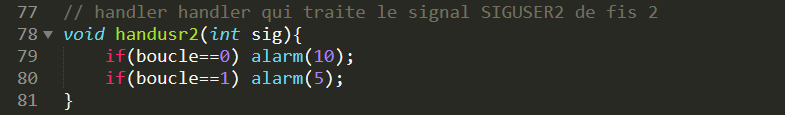
A chaque appuis sur la combinaison **ctrl+c**. la variable « **arret** » s’incrément, et de que la condition de test « **if(arret == 3)** » est vérifier les instruction « **kill(pid1, SIGKILL); kill(pid2, SIGKILL**); **exit(0);** » permet de quitter le programme définitivement.



Le handler **handalrm** permet avec la fonction alarm() de gérer les affichages (mesures) des différents processus, Donc chaque processus va faire un traitement selon les variables mise en jeux. Processus 1 selon la valeur de mes va faire les affichages puis il va envoyer au processus 2 le signal SIGUSR2 pour qu’il fait son mesure en respectant les attente. Puis après le processus 2 va envoyer un signal au processus père pour lui indiquer la fin d’attente et que le processus 1 doit être appelée ce qu’il fait le père.



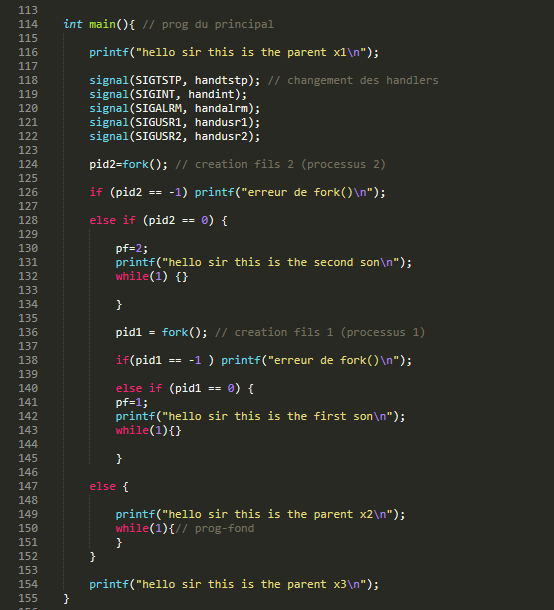
Le handler **handuser1** est utiliser par le processus 1 pour qu’il commence la prise de mesure.



Le handler **handuser2** est utiliser par le processus 2.

Si boucle == 0 donc attente 2mn puis faire la mesure

Si boucle == 1 donc attente 1mn puis envoyer signal au père pour indiquer la fin de mesure et pour débuter la prise de mesure de processus 1



**La fonction main()** permet l’exécution de notre programme.

En commence par assigné les Handler a leur signaux, après en procède a créé les fis par les instruction « pid2=fork(); pid1 = fork(); » ou les variables pid1 et pid2 sont utiliser pour distinguer le instruction que chaque processus de ces 3 processus dois exécuter.

Si un pid est négative alors le sa veut dire que le fis na pas peut être crée, s’il est nul sa veut dire que le processus fis est entrain d’exécution et en utiliser un test « if (pid == 0) » pour préciser les instructions qu’un fis ou un père dois exécuter.

L’instruction « while(1) » est utiliser pour bloquer le processus dans l’état d’exécution et pour qu’il ne quitte pas le system avant qu’on veut.

Le programme père alors et bloque dans une boucle infinie (programme fond) ou il attend la combinaison ctrl+z de l’utilisateur pour commencer les prise des mesure.

**Exécution du programme :**

