TRAVAUX PRATIQUES

TP 2 SIGNAUX

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


#define NB_ENFANTS 10


struct sigaction action;
sigset_t mask_nv;
sigset_t mask_anc;


void handler(int sig)
{
    for (int i = 0; i < NB_ENFANTS; i++)
    {
        printf("Attente d'un fils\n");
        wait(NULL);
    }
}


void creation10fils(pid_t pere)
{
    for (int i = 0; i < NB_ENFANTS && getpid() == pere; i++)
    {
        fork();
    }
}


int main(int argc, char* argv[])
{
    // bloque SIGUSR1
    sigemptyset(&mask_nv);  // Vide le set
    sigaddset(&mask_nv, SIGUSR1); // A decommenter  // ajoute SIGUSR1 au set
    sigprocmask(SIG_BLOCK, &mask_nv, &mask_anc);    // defini le masque du processus comme l'union de
l'ancien
    // (stocké ensuite dans le 3° param) et du 2° param


// Définition du handler
    action.sa_handler = handler;
    sigaction(SIGUSR1, &action, NULL);


    pid_t pere = getpid();
    printf("PID PERE : %d\n", pere);


    creation10fils(pere);


    if (getpid() != pere)
    {   // fils
        printf("Je suis un fils\n");
```

```
        sleep(20); // le fils fait des trucs
        printf("Terminé\n");
    }
    else
    {   // pere
        sigsuspend(&mask_anc);      // redefini le masque le tps d'un signal (ici pr laisser passer SIGUSR1)
    }


    exit(0);
}
// dans un terminal :  kill -s SIGUSR1 pid
// avec pid le pid du pere
```

TP 3 SEMAPHORES EN C

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


int makeSem(int init_value){


 int mysem = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
  if(mysem < 0) perror("Error: semget");


  int j;
  int retval;
  union semun
  {
    int val;
    struct semid_ds *buf;
    ushort *array;
  } arg;
  arg.val = init_value;  // Valeur d'initialisation
  retval = semctl(mysem, 0, SETVAL, arg);
  if(retval < 0) perror("Error: semctl");


return mysem;
}


void rmSem(int sem){
    int retval = semctl(sem, 0, IPC_RMID, 0);
    if(retval < 0) perror("Error: semctl");
}


void P(int sem){


    struct sembuf op;
    op.sem_num = 0;
    op.sem_op = -1; //1=V, -1=P
    op.sem_flg = 0;


    int retval = semop(sem, &op, 1);
    if(retval != 0) perror("error: semop");
```

```
}

void V(int sem){
    struct sembuf op;
    op.sem_num = 0;
    op.sem_op = 1; //1=V, -1=P
    op.sem_flg = 0;


    int retval = semop(sem, &op, 1);
    if(retval != 0) perror("error: semop");
}


int main() {


    int synA = makeSem(0);
    int synBC = makeSem(2);


                    // 1. Le processus père est le p1 du sujet de TP
    pid_t p2 = fork();  // 2. Le p1 crée un fils p2
    pid_t p3 = 0;
    if (p2!=0)          // 3. Le p2 crée un fils p3
        p3 = fork();


    // Début des prints ABC


    if (p2 == 0) {  // fils p2 : B


        while(1){
            P(synA);
            printf("B\n");
            V(synBC);


        }


    }else if(p3 == 0){   // fils : C


        while(1){


            P(synA);
            printf("C\n");
            V(synBC);
        }


    }else{  // père : A


        while(1){
            P(synBC);
            P(synBC);
            printf("A\n");
            V(synA);
            V(synA);


        }
```

```
    }

    return 0;
}
```

SEMAPHORE JAVA

```
import java.util.concurrent.Semaphore;
public class Exo23 extends Thread{

    public static final Semaphore semA;
    public static final Semaphore semB;

    static{
        semA = new Semaphore(0, true);
        semB = new Semaphore(1, true);
    }

    public void run() {
        while(true){

            try{
                Exo23.semA.acquire();
            }catch(InterruptedException e)
            {
                System.out.println(e.getMessage());
            }
            System.out.println("B");
            Exo23.semB.release();
        }
    }

    public static void main(String[] args) {
        Exo23 threadB = new Exo23();
        threadB.start();

        while(true){
            try{
                Exo23.semB.acquire();
            }catch(InterruptedException e)
            {
                System.out.println(e.getMessage());
            }
            System.out.println("A");
            Exo23.semA.release();
        }

    }

}
```

TP 5 TUBE ANONYME

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

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char ** argv) {
   int fd[2];
   pipe(fd);

   if(fork()==0)
   {
      // Processus fils : ecrit (lance le `who`)
      close(fd[0]);
      dup2(fd[1], STDOUT_FILENO);
      close(fd[1]);
      execl("/usr/bin/who","who",0);
   }
   else
   {
      //Processus pÃ¨re : lit (fait le `grep`)
      close(fd[1]);

      char buffer[10];
      read(STDIN_FILENO, buffer, 10);

      dup2(fd[0], STDIN_FILENO);
      close(fd[0]);

//    execl("/bin/grep", "grep", argv[1],0);
      execl("/bin/grep", "grep", buffer,0);

   }
   return 0;
}
```

TUBE NOMME

CLIENT

```
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>

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

   // 0. Afficher son PID
   printf("Client PID : %d\n", getpid());

   // a. Ouvrir le tube en lecture
```

```
    int file;

    if((file = open("/tmp/PipeNomme", O_RDONLY)) == -1) // S'il a bien été ouvert
    {
        perror("open");
        exit(1);
    }

    // 1. envoyer un signal au Server

    // 1.1 On demande à l'user le PID du serveur
    char pidServer[5];
    read(STDIN_FILENO, pidServer,5);
    int pid = atoi(pidServer);
    // 1.2 On envoie le signal correspondant la l'argv[1] du script
    if(argc == 2 && atoi(argv[1]) == 1)
        kill(pid, SIGUSR1);
    else if(argc == 2 && atoi(argv[1]) == 2)
        kill(pid, SIGUSR2);
    else {
        perror("No signal sent");
        exit(1);
    }



    // b. Lire de l'information dedans
    char buffer[6];
    read(file, buffer, 6);

    printf("%d\n", atoi(buffer));

    // c. Fermer le tube
    close(file);

    return 0;
}
```

SERVEUR

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
//#include <bits/sigaction.h>

int service1();
int service2();
void handler(int sig);

// b. Créer le tube
```

```
int file;
struct sigaction action;

int main() {
    unlink("/tmp/PipeNomme");
    // 0. préparation du sigaction

    action.sa_handler = handler;
    sigaction(SIGUSR1, &action, NULL);
    sigaction(SIGUSR2, &action, NULL);

    // a. Afficher son PID
    printf("Server PID : %d\n", getpid());

    // b. Créer le tube
    if(mkfifo("/tmp/PipeNomme", 0777) != 0) // Si le pipe est bien créé
    {
        perror("mkfifo");
        exit(1);
    }

    // c. L'ouvrir en écriture
    if((file = open("/tmp/PipeNomme", O_WRONLY)) == -1) // S'il a bien été ouvert
    {
        perror("open");
        exit(1);
    }

    char userChoiceBuffer[2];
    while (strcmp(userChoiceBuffer,"q") != 0){

        // on attend le message du client ou le 'q' de l'utilisateur pour quitter
        // On utilise le read bloquant pour mettre en attente (pas active) le processus
        // on attend soit un SIGUSR1/2, soit un 'q'
        read(STDIN_FILENO,userChoiceBuffer, 2);
    }

    // e. Fermer le tube
    close(file);

    // f. Détruire le tube
    unlink("/tmp/PipeNomme");

    return 0;
}

int service1(){
    return 1;
}

int service2(){
    return 2;
}

void handler(int sig){
```

```
    // d. Ecrire de l'information dedans
    printf("Handler called\n");
    char buffer[2];


    if (sig == SIGUSR1){
        printf("SIGUSR1\n");
        sprintf(buffer,"%d",service1());
    }else if(sig == SIGUSR2){
        printf("SIGUSR2\n");
        sprintf(buffer,"%d",service2());
    }else{
        perror("Unkown sig\n");
    }
    write(file,buffer, 2);


}
```