EFREI *Novembre 2014*

M1 IL

TP (séances 1-2)

1-
Ecrire un programme qui montre que si l'on essaie d'écrire dans un tube fermé en lecture, on reçoit le signal SIGPIPE.

2-
Exécutez le programme suivant, identifiez le bug qu'il contient et corrigez-le :

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

void * additionneur(void *);

#define  nbt  2
pthread_t  tid[nbt];
int  nombre = 0;

main( int argc, char *argv[] )
{
  int i;
  for (i=0; i<nbt; i++) {
    pthread_create(&tid[i], NULL, additionneur, NULL);
  }
  for ( i = 0; i < nbt; i++)
    pthread_join(tid[i], NULL);

  printf("main signale que les %d threads ont terminé\n", i);
  printf("Je suis main ! nombre=%d\n", nombre);
}

void * additionneur(void * parm)
{
   int i;
   printf("Je suis une nouvelle thread !\n");
   for(i=0;i<200000;i++) {
      nombre++;
   }
   pthread_exit(0);
}
```

3- Threads, mutexes et variables de conditions

Les *mutexes* servent à implémenter la synchronisation des threads en contrôlant l'accès aux données partagées.
Les *variables de conditions* servent à contrôler la synchronisation des threads sur la valeur d'une variable partagée.
En utilisant ces notions, écrire un programme qui crée 3 threads. Deux des threads incrémentent une variable *compteur* et la troisième surveille la valeur de ce *compteur*.
Quand le *compteur* atteint une valeur prédéterminée, l'une des 2 threads le signale à la troisième thread qui attend cet événement.

4-
Ecrire un programme qui envoie un courrier électronique à un utilisateur via un tube (le père écrit le message dans le tube et le fils envoie le contenu du tube à l'utilisateur).

5-
La communication entre processus peut se réaliser par l'envoi de messages.
Il s'agit alors d'écrire un programme client et un programme serveur qui communiquent via une file de messages :
a- messages échangés via une seule file mais en utilisant 2 types différents.
b- messages échangés via 2 files différentes

Exemple d'exécution avec la méthode 1 (on lance le serveur *mqfile1serv* en arrière plan et ensuite le client *mqfile1cli*) :

```
% ./mqfile1serv &
[1] 24161
% ./mqfile1cli

mqfile1cli: contenu du message envoyé: Ici l'EFREI: bonjour M1 IL
Le serveur 24161 a reçu un message du client 24165
Contenu du message reçu: Ici l'EFREI: bonjour M1 IL
mqfile1cli: attente d'un message
le client  24165 a reçu un message du serveur 24161
```

6-
Il s'agit de simuler un point de rencontre de trois processus à l'aide de sémaphores et de la mémoire partagée.
Le programme *init-mp+sem* crée un sémaphore pour l'exclusion mutuelle et un autre pour le rendez-vous et un segment de mémoire partagée pour permettre aux processus de connaître à leur arrivée le nombre de processus présents au point de rendez-vous.
Le programme *point-rdv* simule les processus arrivant en ce point.

7- Problème du dîner des philosophes

C'est un problème théorique proposé et résolu par Dijkstra.
Cinq philosophes sont réunis autour d'une table sur laquelle il y a cinq plats de spaghettis et cinq fourchettes. Un philosophe passe son temps à manger et à penser. Pour manger son plat de spaghettis, un philosophe a besoin de deux fourchettes qui sont de part et d'autre de son plat.
La solution de ce problème ne doit pas produire d'interblocage. Elle utilise un tableau appelé *etat* pour suivre l'état du philosophe (il mange, il pense, ou il a faim et tente de s'emparer des fourchettes). Un philosophe donné ne peut passer à l'état « manger » que si aucun de ses voisins ne mange à ce moment-là.
Écrire un programme en C qui permet à chaque philosophe de se livrer à ses activités (penser et manger) sans jamais se bloquer.

(Source : Wikipedia)

Canevas du programme :

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

#define N 5                     /* Nombre de philosophes */
#define PENSE      0            /* Le philosophe pense */
#define AVOIR_FAIM    1         /* Le philosophe essaie de prendre les fourchettes */
#define MANGE    2              /* Le philosophe mange */
#define GAUCHE  (numero_phil+N-1)%N         /* numéro du voisin de gauche */
#define DROITE    (numero_phil+1)%N         /*  numéro du voisin de droite */

sem_t        mutex;             /* Exclusion mutuelle pour les sections critiques */
sem_t        S[N];              /* Un sémaphore par philosophe */


void * philosphe(void *num);
void prendre_fourchettes(int);
void poser_fourchettes(int);
void test(int);
int etat[N];
int numero_phil[N]={0,1,2,3,4};

int main()
{
   int i;
   pthread_t thread_id[N];
   sem_init(&mutex,0,1);
   for(i=0;i<N;i++)
      sem_init(&S[i],0,0);
   for(i=0;i<N;i++)
   {
      pthread_create(&thread_id[i],NULL,philosophe,&numero_phil[i]);
      printf("Philosophe %d est en train de penser\n",i+1);
   }
   for(i=0;i<N;i++)
      pthread_join(thread_id[i],NULL);
}
```

```
void  *philosophe(void *num)
{
   while(1)
   {
      int *i = num;
      sleep(1);
      prendre fourchettes(*i);
      sleep(1);
      poser_fourchettes(*i);
   }
}
```

Ecrire les procédures *prendre_fourchettes(int numero_phil)* et *poser_fourchettes(int numero_phil)* qui utilisent la procédure *test(int numero_phil)*

```
void test(int numero_phil)
{
   if (etat[numero_phil] == AVOIR_FAIM && etat[GAUCHE] != MANGE &&
etat[DROITE] != MANGE)
   {
      etat[numero_phil] = MANGE;
      sleep(2);
        printf("Philosophe %d prend fourchette %d et
%d\n",numero_phil+1,GAUCHE+1,numero_phil+1);
      printf("Philosophe %d est en train de manger\n",numero_phil+1);
      sem_post(&S[numero_phil]);
   }
}
```

Voici un exemple d'exécution :

```
Philosophe 1 est en train de penser
Philosophe 2 est en train de penser
Philosophe 3 est en train de penser
Philosophe 4 est en train de penser
Philosophe 5 est en train de penser
Philosophe 1 a faim
Philosophe 1 prend fourchettes 5 et 1
Philosophe 1 est en train de manger
Philosophe 3 a faim
Philosophe 3 prend fourchettes 2 et 3
Philosophe 3 est en train de manger
Philosophe 4 a faim
Philosophe 5 a faim
Philosophe 2 a faim
```

Philosophe 1 pose fourchettes 5 et 1
Philosophe 1 est en train de penser
Philosophe 5 prend fourchettes 4 et 5
Philosophe 5 est en train de manger
Philosophe 3 pose fourchettes 2 et 3
Philosophe 3 est en train de penser
Philosophe 2 prend fourchettes 1 et 2
Philosophe 2 est en train de manger
Philosophe 5 pose fourchettes 4 et 5
Philosophe 5 est en train de penser
....
....
....

8- Héritage de descripteurs

Ecrire le programme *heritdesc.c* qui crée un tube, un fils et écrit un message dans le tube. Le fils peut lire dans le tube car il hérite des descripteurs créés par le père. Comme les codes du père et du fils sont dans 2 programmes différents, le fils ne connaît pas les numéros de ces descripteurs. Il faut donc passer au fils le numéro du descripteur concerné en paramètre.

```
 /* heritdesc.c */
                /*** Remplacer les points par les instructions
                *** ou les paramètres manquants ***/

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

void fils(numero)
int numero;
{
 int desc ;
 int nlect ;
 char message[100] ;

    desc = numero ;
    printf("Le descripteur est:%d\n",desc) ;
```

```
/*
    * lecture dans le tube
    */
    switch (nlect=read(...,...,...)) {
    case -1:
                ...
    case 0:
                ...
    default:
        printf("Lecture %d octets:%s\n",nlect,message) ;
    }
}
```

```
main()
{
 int desc[2] ;
 char chaine[100] ;
   /*
    *Création du tube
    */
        ...
   /*
    * Création du fils
    */
    switch (fork()){
    case -1:
                ...
    case 0:
          ...
        fils(desc[0]) ;
        exit(0) ;
    }
   /*
    * Ecrire dans le tube
    */
        ...
}
```