Final LO41 - 28/06/2000

Durée 2 heures ; notes de cours, de TD et de TP autorisées ; 1 copie par partie

PARTIE A

1. Utilisation des signaux pour communiquer (3pts)

Écrire un programme (en C ou en pseudo-code) selon les exigences suivantes :

. Le processus principal (père) crée un autre processus (fils).
. Le père et le fils vont se communiquer à l'aide du signal SIGUSR1. La synchronisation correcte de cet échange étant :
  - le fils envoie un signal au père, le père affichera alors "*simulation de requête*",
  - le père retourne un signal au fils, le fils affichera alors "*simulation de réponse*".
. Après la terminaison du fils, le père affichera "*fin du fils*".

2. Utilisation des IPC files de messages (6pts)

Écrire un programme qui permet de tester les fonctionnalités des files de messages. Ce programme propose un menu d'actions et accède à une file donnée via une clé. Voici les fonctions proposées à l'utilisateur (vous y ajouterez les arguments nécessaires) :

. clé() :	permet de changer la clé utilisée.
. création() :	crée une file de messages en utilisant la clé. On doit pouvoir savoir si la clé correspond à une file déjà ouverte ou non.
. envoi() :	demande le type et le contenu du message à envoyer sur la file. Signale si la file est pleine.
. réception() :	demande un type entier et lit un message. Signale si la file est vide.
. info() :	affiche les caractéristiques de la file.
. fin().	

PARTIE B

1. Simulation de fonctionnement d'un ascenceur (6pts)

Le but de cet exercice est d'écrire, sous forme d'algorithme ou de code C, un programme qui simule le fonctionnement d'un ascenceur. Il s'agit de créer 4 processus fils qui vont simuler les différents étages (0, 1, 2 et 3) tandis que le processus père simule l'ascenceur.

Initialement l'ascenceur est à l'étage 0. Son comportement en boucle consiste alors à monter jusqu'au 3ème puis redescendre à l'étage 0, toujours dans cet ordre. L'ascenceur s'arrête à un étage si et seulement s'il y a des passagers qui veulent descendre ou monter. La capacité maximale de l'ascenceur est de 5 personnes. Chaque étage gère une file de personnes, chacune ayant un étage destination. La constitution de ces files se fait aléatoirement.

Pour vous aider, vous pouvez utiliser les primitives P et V vues durant l'UV. Vous pouvez également utiliser les primitives de manipulation de files : Créer, Insérer et Premier. On peut considérer que chaque élément de la file est un entier qui est l'étage où la personne désire se rendre.

La solution devra comporter :

. une partie création des processus,

. la boucle sans fin qui gère le comportement de l'ascenceur,

. la gestion des différentes files de personnes en attente,

. les instructions qui permettent de synchroniser l'ascenceur pour qu'il :

- s'arrête aux étages où des personnes attendent/veulent descendre,
- ne transporte jamais plus de 5 personnes.

2. Amélioration de synchronisations artificielles (5pts)

Soit le programme C suivant :

```
$ cat exemple.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

char ch[10];
pid_t pid;
int tube[2], d1, d2, n;

void main ( ) {
      pipe (tube);
      d1=open ( "fic", O_RDWR | O_CREAT | O_TRUNC, 0666 );
      switch ( fork( ) ) {
            case -1 :
                        perror ( "fork" );
                        exit (1);
            case 0 :
                        sleep (1);
                        d2 = open ( "fic", O_RDWR ); sleep (3);
                        n = read ( tube[0], ch, 10 );
                        printf ( "Fils : n = %d --> %s\n", n, ch );
                        write ( d2, ch, n ); sleep (5);
                        n = read ( d1, ch, 1 );
                        printf ( "Fils : n = %d --> %s\n", n, ch ); sleep (5);
                        write ( d1, "12", 2 );
            default :
                        d2 = open ( "fic", O_RDWR ); sleep (1);
                        write( tube[1], "abcde", 5 ); sleep (5);
                        write( tube[1], "fghij", 5 );
                        n = read ( tube[0], ch, 10 );
                        printf ( "Père : n = %d --> %s\n", n, ch );
                        write ( d2, ch, n ); sleep (5);
                        n = read ( d1, ch, 2 );
                        printf ( "Père : n = %d --> %s\n", n, ch );
                        write (d1, "AB", 2); sleep (5);
                        n = read ( d2, ch, 1 );
                        printf ( "Père : n = %d --> %s\n", n, ch );
                        n = read ( d1, ch, 1 );
                        printf ( "Père : n = %d --> %s\n", n, ch );
      }
}
```

a) Que contient le fichier de nom fic lorsque les 2 processus sont terminés?

b) Remplacer les «synchronisations artificielles», réalisées au moyen de temporisations des processus par l'intermédiaire d'appels à la fonction *sleep*, par des synchronisations (une de votre choix) plus naturelles réalisées par des mécanismes synchronisants disponibles dans UNIX; l'exécution conduisant aux mêmes résultats.