IUT Orsay

DUT Informatique
Module Système S4-L
2010 / 2011

Travaux Pratiques n° 3 : Sémaphores

Nom(s) :
Groupe :
Date :

Objectifs : être capable de construire des applications dont le mécanisme d'exclusion mutuelle repose sur les sémaphores.

1 Avant propos

Les rappels nécessaires à la réalisation de ce TP sont fournis en notes de cours au début du TD n° 2 et 3. Par ailleurs il est vivement conseillé de se référer à votre cours de Système S2. Bien sûr les pages de `man` sont comme toujours de précieuses alliées.

2 Utilisation d'un tube par plusieurs processus

Le programme *tp3.c* ci-après [1] crée un tube et deux processsus fils, appelés `fils1` et `fils2`. Les processus `fils1` et `fils2` écrivent 10 fois chacun dans le tube un message formé du numéro de message sur 3 chiffres, d'un espace et de la chaîne «Message de fils1» (ou «Message de fils2» selon le cas). Les processus s'endorment une ou deux secondes (durée choisie aléatoirement) après avoir écrit sur le tube. Le processus père lit les messages et les affiche sur la voie standard de sortie.

Compilez, exécutez plusieurs fois le programme puis comparez les résultats. Qu'observez-vous ? Expliquez.

1. Vous trouverez ce code sur la page du Système S4, `http://www.lri.fr/~bastoul/teaching/systeme`.

```c
#include <stdio.h>                  // Pour entrees/sorties
#include <stdlib.h>                 // Pour exit
#include <sys/types.h>              // Pour pid_t
#include <sys/wait.h>               // Pour wait
#include <unistd.h>                 // Pour pipe, fork, write, read, close
#define LONGUEUR_MESSAGE 21         // 20 lettres + '\0'
#define NB_MESSAGES      10

int tube[2];                        // Descripteurs du tube (variable globale)

void fils1()                        // Code de fils1
{ int i;
  char message[LONGUEUR_MESSAGE];

  close(tube[0]);                   // On ferme la lecture pour le fils
  srand(getpid());                  // Initialisation du generateur aleatoire
  for(i=1; i<=NB_MESSAGES; i++)     // On ecrit les messages
  { sprintf(message, "%03d Message de fils1", i);
    write(tube[1], message, LONGUEUR_MESSAGE);
    sleep(rand()%2+1);              // On s'endort 1 ou 2 secondes
  }
  close(tube[1]);                   // On ferme l'ecriture pour le fils
  exit(0);                          // Fin de fils1
}

void fils2()                        // Code de fils2.
{ int i;
  char message[LONGUEUR_MESSAGE];

  close(tube[0]);                   // On ferme la lecture pour le fils
  srand(getpid());                  // Initialisation du generateur aleatoire
  for(i=1; i<=NB_MESSAGES; i++)     // On ecrit les messages
  { sprintf(message, "%03d Message de fils2", i);
    write(tube[1], message, LONGUEUR_MESSAGE);
    sleep(rand()%2+1);              // On s'endort 1 ou 2 secondes
  }
  close(tube[1]);                   // On ferme l'ecriture pour le fils
  exit(0);                          // Fin de fils2
}

int main()
{ int nb_car, etat;
  char buffer[LONGUEUR_MESSAGE];
  pid_t pid1, pid2, pid_wait;

  pipe(tube);                       // Creation du tube par le pere
  pid1 = fork();                    // Creation d'un premier fils
  if (pid1 == 0)
    fils1();
  else                              // Suite du code du pere
  { pid2 = fork();                  // Creation d'un second fils
    if (pid2 == 0)
      fils2();
    else                            // Suite du code du pere
    { close(tube[1]);               // On ferme l'ecriture pour le pere

      nb_car = read(tube[0], buffer, LONGUEUR_MESSAGE);
      while (nb_car > 0)            // Lecture messages jusqu'a fin de fichier
      { printf("%s\n",buffer);
        nb_car = read(tube[0], buffer, LONGUEUR_MESSAGE);
      }

      pid_wait = wait(&etat);       // Attente de la fin d'un premier fils
      printf("Pere => fin du fils %d.\n", (pid_wait == pid1)?1:2);
      pid_wait = wait(&etat);       // Attente de la fin d'un second fils
      printf("Pere => fin du fils %d.\n", (pid_wait == pid1)?1:2);
    }
  }
  return(0);
}
```

3 Tubes unidirectionnels pour implanter des sémaphores

3.1 Sémaphores d'exclusion mutuelle

Un tube peut servir à implanter un sémaphore, mais de manière inefficace. L'implantation d'un sémaphore nécessite les possibilités suivantes :
– la mise à jour en exlusion mutuelle d'un compteur,
– un mécanisme d'attente/réveil associé à ce compteur.
Dans le cas de sémaphores d'exclusion mutuelle, la valeur du compteur est binaire et varie de 0 à 1. On peut donc implanter un sémaphore d'exclusion mutuelle (`mutex`) à l'aide d'un tube de la façon suivante :
– la valeur binaire du compteur est représentée par la présence ou non d'un seul caractère dans le tube ;
– l'attente est réalisée par une tentative de lecture dans le tube, ce qui provoquera une attente tant qu'il n'y aura pas de caractère dans le tube ;
– le réveil sera réalisé par une écriture d'un caractère dans le tube.
On peut donc implanter la primitive `P` en lisant un caractère dans un tube et `V` en écrivant un caractère dans ce tube. On peut implanter plusieurs sémaphores `mutex` en les identifiant avec l'adresse du tableau qui contient les descripteurs des deux extrémités du tube. L'exclusion mutuelle est réalisée par la lecture et l'écriture atomique d'un seul caractère sur le tube. On donne les spécifications suivantes placées dans le fichier mutextube.h fourni :

```
/* Cree un tube dont les descripteurs seront places dans le
 * tableau sem, et ecrit 1 caractere dans ce tube. Le tableau
 * sem servira ensuite comme "identifiant" du semaphore.
 */
void Init(int * sem);

/* Effectue une operation V(1) sur le semaphore sem. */
void V(int * sem);

/* Effectue une operation P(1) sur le semaphore sem. */
void P(int * sem);
```

Programmez ces trois primitives dans un fichier mutextube.c et **joignez ce fichier à votre compte rendu de TP**.

3.2 Synchronisation avec les sémaphores précédents

On reprend le programme *tp3.c* présenté en section 2. Modifiez ce programme en l'appelant tp4sync.c pour qu'il synchronise les deux processus écrivains `fils1` et `fils2`, de telle manière que ce soit d'abord `fils1` qui écrive, puis une stricte alternance des écritures entre `fils1` et `fils2`. On utilisera pour cette synchronisation des sémaphores `mutex` implantés dans la section précédente.

Combien de sémaphores sont nécessaires à cette synchronisation ? Décrivez votre solution.

Programmez tp4sync.c, vérifiez bien que les exécutions se déroulent toujours de la même manière et **joignez ce fichier à votre compte rendu de TP**.

3.3 Implantation de Sémaphores quelconques avec des tubes

On souhaite généraliser la réalisation de sémaphores quelconques avec des tubes qui implanteraient la spécification suivante placée dans le fichier mutextuben.h fourni :

```
/* Cree un tube dont les descripteurs seront places dans le
 * tableau sem, et ecrit n caracteres dans ce tube. Le tableau
 * sem servira ensuite comme "identifiant" du semaphore.
 */
void Initn(int * sem, int n);

/* Effectue V(n) sur le semaphore sem (ajoute n au compteur). */
void Vn(int * sem, int n);

/* Effectue P(n) sur le semaphore sem (retire n au compteur). */
void Pn(int * sem, int n);
```

Cette solution est-elle efficace et fiable ? Expliquez. Qu'en concluez-vous sur l'utilisation de tubes pour réaliser des sémaphores ?

Auriez-vous une proposition, utilisant les sémaphores, qui permette de fiabiliser cette solution ? Décrivez-la.

4 Sémaphores d'exclusion mutuelle et sémaphores Unix

On souhaite implanter les primitives `Init`, `P` et `V` des sémaphores d'exclusion mutuelle cette fois-ci à l'aide des primitives efficaces et robustes d'Unix (voir exercice 3 du TD Sémaphores).

Implantez et testez cette version sur le problème en section 3.2. **Joignez à votre compte-rendu les fichiers mutex.h et mutex.c correspondants**, conservez-en une copie : ils seront nécessaires pour un prochain TP.

Commentaires personnels sur le TP (résultats attendus, difficultés, critiques etc.).