IUT Orsay

DUT Informatique
Module Système S4-L
2010 / 2011

Travaux Dirigés n° 4 : Segments de Mémoire Partagée

Objectifs : connaître les principes d'utilisation de segments de mémoire partagée entre processus et repérer les problèmes de synchronisation qui en découlent.

1 Notes de cours

Contrairement aux mécanismes tels que signaux ou tubes dont les informations transitent au travers du système d'exploitation, les segments de mémoire partagée permettent aux processus de partager de l'information, directement et sans contrôle. C'est en fait le *seul* moyen de *partager* de l'information entre processus. Il s'agît d'allouer une zone de la mémoire centrale qui sera accessible directement par les processus connaissant la *clé* de cette zone, au travers d'un pointeur. Il y a simplement quatre primitives fondamentales (voir le polycopié *Primitives Système sous UNIX* pour les informations détaillées) :

```
#include <sys/shm.h>
int     shmget(key_t cle, size_t taille, int flag);
void * shmat (int identificateur, NULL, int option);
int     shmdt (const void * adresse);
int     shmctl(int identificateur, int operation, NULL);
```

1. `shmget(cle,taille,flag)` retourne l'identificateur d'un segment à partir de sa clé (`cle`) ou -1 en cas d'échec. Le segment sera créé s'il n'existait pas encore. On peut utiliser la clé `IPC_PRIVATE` pour la création quand il n'est pas utile ensuite d'acquérir l'identificateur. Le paramètre `taille` donne le nombre d'octets du segment (s'il a déjà été créé, la taille doit être inférieure ou égale à la taille de création). Le paramètre `option` est une combinaison (par OU bit à bit) de constantes (telles que `IPC_CREAT` pour la création) et de droits d'accès (comme `0666`). Par exemple pour créer un segment on utilisera typiquement l'option `IPC_CREAT|0666`, et pour l'acquisition simplement `0666`.
2. `shmat(identificateur,NULL,option)` sert à *attacher* un segment, c'est à dire à obtenir une fois que l'on connaît son identificateur, un pointeur vers la zone de mémoire partagée. L'option sera `SHM_RDONLY` pour un segment en lecture seule ou `0` pour un segment en lecture/écriture. Cette primitive retourne l'adresse de la zone de mémoire partagée ou `(void *)(-1)` en cas d'échec.
3. `shmdt(adresse)` sert à *détacher* le segment attaché à l'adresse passée en paramètre. Retourne 0 en cas de succès, ou -1 en cas d'échec.
4. `shmctl(identificateur,operation,NULL)` sert au contrôle (par exemple la suppression) du segment dont l'identificateur est `identificateur`. Pour supprimer le segment, la valeur du paramètre `operation` est `IPC_RMID` (la suppression ne sera effective que lorsque plus aucun processus n'attachera le segment). Retourne 0 en cas de succès, ou -1 en cas d'échec.

2 Exercices

2.1 Exercice 1 : transmission d'information

Un processus père et un processus fils s'échangent des informations très rapidement en utilisant une variable entière partagée. Le père écrit un à un tous les entiers entre 0 et `SHRT_MAX` (sans se soucier du fils), le fils lit `SHRT_MAX` fois la variable partagée et affiche à chaque fois la valeur lue (sans se soucier du père).

1. Réalisez cette application.
2. Est-ce que le processus fils va à coup sûr afficher tous les entiers entre 0 et `SHRT_MAX` ?
3. Décrivez le besoin en synchronisations pour que cela soit le cas.

2.2 Exercice 2 : calculateur parallèle

Le but de cet exercice est de simuler un calcul sur une machine parallèle disposant de n processeurs. Soit `T` un tableau de `n` entiers (`n` étant une puissance de 2). On souhaite calculer la somme `S` (ou toute autre opération associative comme $\times$, min, max, pgcd, etc.) de tous les éléments du tableau, donnée par la formule suivante :

$$S = \sum_{i=1}^{n} T[i]$$

1. Décrivez l'organisation d'un programme permettant de calculer `S` et utilisant `n` processus fils. Les segments de mémoire partagée seront l'unique moyen de communication.
2. Des synchonisations entre les processus sont-elles nécessaires ? Si oui, pourquoi (donnez des exemples de situations incohérentes) ?
3. Proposez une solution utilisant un sémaphore d'exclusion mutuelle pour la synchronisation (on suppose que les primitives `Init`, `P` et `V` sont disponibles). Implémentez votre solution.
4. Combien d'opérations sont nécessaires (complexité en nombre additions) dans votre première solution ? Peut-on réduire le nombre de processus tout en gardant la même complexité de calcul ? Si oui, expliquez comment.

2.3 Exercice 3 : échange d'informations en ressources limitées

Un processus père `p1` dispose d'un tableau de 10 caractères (contenant une chaîne quelconque que nous nommerons `A`). De même, un processus fils `p2` dispose d'un tableau de 10 caractères (contenant une chaîne quelconque que nous nommerons `B`). Le but est d'échanger les chaînes respectives de `p1` et `p2` au moyen d'un unique segment de mémoire partagée de taille 2 caractères. Vous utiliserez des sémaphores d'exclusion mutuelle pour la synchronisation (on suppose que les primitives `Init`, `P` et `V` sont disponibles).

1. Analysez le problème, en particulier le besoin en synchronisations.
2. Implémentez votre solution.

2.4 Exercice 4 : réorganisation d'un tableau **

Un tableau `T` de `2n` éléments est partagé en deux moitiés : on souhaite regrouper dans la première moitié les `n` plus grands éléments et dans la seconde les `n` plus petits. On se propose de donner une solution au problème en utilisant deux processus partageant le tableau en mémoire :

l'un accède à la première moitié et recherche le plus petit élément, et l'autre recherche dans la seconde moitié le plus grand élément. Lorsque les deux processus ont terminé leur recherche, il y a éventuellement échange des deux éléments dans le tableau et itération du mécanisme. Le seul moyen de communication est la mémoire partagée.

1. Analysez le problème, en particulier le besoin en synchronisations.
2. Implémentez votre solution.

3 Entraînement : exercice corrigé

3.1 Énoncé : exclusion mutuelle

Une solution au problème de l'exclusion mutuelle entre deux processus `p0` et `p1` par variables partagées est due à G.I. Peterson (1981, voir exercice corrigé du TD n° 1). Les algorithmes des processus autour des sections critiques sont les suivants, avec `tour`, `D0` et `D1` trois variables booléenes partagées (des entiers en C) :

Processus `p0`	Processus `p1`
`/* Avant section critique */`	`/* Avant section critique */`
`D0 = 1;`	`D1 = 1;`
`tour = 0;`	`tour = 1;`
`while (D1 && (tour == 0));`	`while (D0 && (tour == 1));`
`/* section critique */`	`/* section critique */`
`D0 = 0;`	`D1 = 0;`
`/* Après section critique */`	`/* Après section critique */`

Implémentez un tel mécanisme à l'aide des primitives Unix sur les segments de mémoire partagée. L'initialisation de `D0` et `D1` est `faux`, celle de `tour` est 0.

3.2 Correction (essayez d'abord ! ! !)

Il y a en tout trois variables partagées, on va donc devoir créer un segment de mémoire partagée de longueur trois entiers. Un processus père se chargera de créer le segment, de l'initialisation des variables partagées et de la création des deux processus `p0` et `p1`. L'adresse du début du segment de mémoire partagée sera contenue dans une variable globale, ainsi tous les processus en auront connaissance.
Un code possible est le suivant. Attention, aucun test d'échec des diverses primitives n'est effectué. C'est à corriger dans tout travail sérieux.

```c
#include <stdlib.h>
#include <sys/shm.h>

int * tab;                          // Adresse du segment partage

void p0()                           // --- Code de p0 ---
{
  // Avant section critique
  tab[0] = 1;
  tab[2] = 0;
  while(tab[1] && (tab[2] == 0));

  //********************
  // Section critique
  //********************

  tab[0] = 0;
  // Apres section critique
  exit(0);
}

void p1()                           // --- Code de p1 ---
{
  // Avant section critique
  tab[1] = 1;
  tab[2] = 1;
  while(tab[0] && (tab[2] == 1));

  //********************
  // Section critique
  //********************

  tab[1] = 0;
  // Apres section critique
  exit(0);
}

int main()                          // --- Code du pere ---
{
  int id_shm;                       // Identificateur du segment partage

  // Creation d'un segment de memoire partagee, de taille 3 entiers
  id_shm = shmget(IPC_PRIVATE, 3*sizeof(int), IPC_CREAT|0666);

  // Attachement du segment partage a l'adresse tab
  tab = (int *)shmat(id_shm, NULL, 0);

  // Initialisations :
  tab[0] = 0;                       // D0   : tab[0], initialise a faux
  tab[1] = 0;                       // D1   : tab[1], initialise a faux
  tab[2] = 0;                       // tour : tab[2], initialise a 0

  if (fork() == 0)                  // Creation du processus p0
    p0();

  if (fork() == 0)                  // Creation du processus p1
    p1();

  return 0;
}
```