

TP 3 Systeme d'exploitation: Outils de Communication Inter-Processus

Il existe plusieurs moyens pour faire communiquer deux processus s'exécutant sur la même machine sous linux.

- ❑ Les pipes ou tubes: communication entre processus père et processus fils a travers des descripteurs de fichiers (communication entre processus liés).
- ❑ Les tubes nommes (pipes named) les FIFO: ce sont des tubes ayant une existence dans le système de fichiers (un chemin d'accès). Communication entre processus non liés.

Ces deux moyens sont gérés comme des entrées sorties et donc sont coûteux en temps d'exécution. Il existe aussi des moyens de communication gérés directement par le noyau Linux et qu'on regroupe sous l'acronyme IPC System V (IPC : Inter-Process Communication, System V est le noyau Unix ancêtre de Linux). Il y a 3 sortes d' IPCs

- ❑ Les files de messages (message queues)
- ❑ Les segments de mémoire partagée (shared memory segments)
- ❑ Les sémaphores

Ce TP va d'abord aborder les IPCs System V plus particulièrement les sémaphores puis les pipes et les FIFOs.

I- Identification unique des ressources IPC quel que soit leur type

Les ressources IPC sont identifiées par une clé unique dans tout le système.

Avant de créer une ressource IPC, on doit fournir une clé unique qu'on peut obtenir avec la fonction **ftok** (file to key) qui permet de créer une clé à partir d'un chemin valide et d'une lettre.

key_t key = ftok (char* pathname, char project)

Exp: key_t key =ftok ("/home/etudiant/", 'c')

Un autre identifiant est généré par le système lors de la création d'une ressource IPC d'un type donné et qui permet d'identifier les ressources de même type.

II- Commandes Linux pour lister et manipuler les IPCs en ligne de commande:

Les IPCs System V ont la particularité de rester en mémoire même après que le processus les ayant créés ait terminé son exécution (si ce dernier n'a pas supprimé les ressources qu'il a créées). Pour créer, lister, supprimer des IPCs en ligne de commande on utilise les commandes suivantes:

Création d'une ressource IPC	ipcmk (options: -s sémaphores, -m mémoire, -q file de messages)
Lister toutes les ressources IPC existantes dans le système	# ipcs -a (ou -s, -m, -q pour lister chaque type à part)

Lister les details d'une ressource IPC	# ipcs -s/-m/-q -i id Exp. # ipcs -s -i 32768
Suppression d'une ressource IPC	#ipcrm -s/-m/-q id Exp. suppression du semaphore d'id. 6766: # ipcrm -s 6766
Obtenir les limites d'une ressource IPC	# ipcs -m/-s/-q -l
Le dernier processus qui a accede a la ressource (option -p de ipcs)	# ipcs -m -p

II- Creation d'un groupe de sémaphores: la fonction semget

- Inclure le fichier entête `#include <sys/sem.h>`

Les sémaphores system V sont créés par groupes de **n** sémaphores. Un sémaphore est une structure **semid_ds** qui contient toutes les informations du groupe. Les sémaphores d'un même groupe sont numérotés de **0 a n**.

La fonction **semget (key, n, flags)** permet de **créer** ou **obtenir** l'identifiant d'un groupe de sémaphores existants. **N** est le nombre de sémaphores dans le groupe, flags est une combinaison (ou logique) d'options de création. L'option **IPC_CREAT** permet de créer un nouveau groupe de sémaphores, l'option **IPC_EXCL** permet de retourner **-1** si le groupe existe déjà. La 3eme option nécessaire est les droits d'accès au nouveau groupe de sems (0666 pour tout les droits).

Exp: `int semid=semget(key, n, IPC_CREAT| IPC_EXCL| 0666)`

- L'appel **semget (key, n, 0)** retourne l'identifiant du groupe de sémaphores existant ayant la cle key.
- **Le retour de semget:** -1 erreur sémaphore existant si **IPC_EXCL** est indiquée dans le flag , 0 indique aussi un sémaphore existent si **IPC_EXCL** n'est pas utilisée, n > 0 l'identifiant du nouveau groupe de semaphores.

Exercice 1:

Ecrire un programme `semCreate.c` qui crée un ensemble de quatre sémaphores (appel a la fonction `semget`). On s'assurera que la clé est unique sur le système en utilisant la fonction `ftok()`. On s'assurera qu'aucun groupe de sémaphores n'est associé à la clé. Si le groupe existe déjà, on récupère son identifiant (avec un deuxième appel a `semget`) et on l'affiche . Utiliser la commande **ipcs** pour visualiser le sémaphore crée.

III- Manipulation de semaphores: la fonction semctl

La fonction **int semctl (semid, semnum, command, arg)** permet d'initialiser la valeur d'un sémaphore, récupérer la valeur, supprimer un groupe de sémaphores etc. La liste complète des **commandes** possible est disponible dans le manuel (man semctl).

Elle a besoin de quatre arguments : un identificateur de l'ensemble de sémaphores (semid) retourné par semget(), le numéro du sémaphore (dans le groupe) à examiner ou à changer (semnum), un paramètre de commande (cmd). Les options (qui dépendent de la commande appliquée au sémaphore) sont passées via un paramètre de type union semnum (arg).

- **Initialisation d'un sémaphore:** on utilise la fonction semctl avec la commande **SETVAL**. **arg** a une interprétation différente selon la commande utilisée. Dans le cas de **SETVAL**, **arg** sera de type **int** et prendra la valeur que l'on veut affecter à notre sémaphore.

- **Suppression d'un groupe de semaphore:**

Pour supprimer un groupe de sémaphores existant on utilise la fonction semctl avec la commande **IPC_RMID**: semctl (semid, 0, IPC_RMID,0) le deuxième argument est ignoré ici.

Exercice 2:

En supposant qu'un sémaphore a déjà été créé dans l'exercice 1.

1- Ecrire un programme semInit.c qui met le troisième sémaphore de l'ensemble à 1, affiche la valeur du troisième sémaphore, affiche le pid du processus qui a effectué la dernière modification et finalement détruit l'ensemble de sémaphores.

2- Modifier le programme pour initialiser tout les sémaphores du groupe avec un seul appel à la fonction semctl (utiliser la commande **SETALL** et **GETALL** pour récupérer les valeurs).

IV - Opérations P() et V(): La fonction semop

Les opérations **P** et **V** permettent respectivement de décrémenter (avant **SC**) et incrémenter la valeur d'un sémaphore (après **SC**). Une troisième opération est possible aussi **Z** et qui permet de bloquer un processus jusqu'à ce que la valeur d'un sémaphore atteigne 0. L'incrémentation et décrémentation de la valeur d'un sémaphore est modélisée dans une structure **sembuf** définie comme suit :

```
struct sembuf { short sem_num; short sem_op; short sem_flg; };
```

sem_num est le numéro du sémaphore à utiliser, **sem_op** est l'opération à réaliser et **sem_flg** contient les paramètres de l'opération (dans notre cas, il n'est pas important, on le met donc à 0).

sem_op prend les valeurs possibles suivantes:

- **Supérieure à zéro (V)** : La valeur du sémaphore est augmentée de la valeur correspondante. Tous les processus en attente d'une augmentation du sémaphores sont réveillés.
- **Egale à zéro (Z)** : Teste si le sémaphore a la valeur 0. Si ce n'est pas le cas, le processus est mis en attente de la mise à zéro du sémaphore.

❑ **Inférieur à zéro (P)** : La valeur (absolue) est retranchée du sémaphore. Si le résultat est nul, tous les processus en attente de cet événement sont réveillés. Si le résultat est négatif, le processus est mis en attente d'une augmentation du sémaphore.

Une fois, la structure bien remplie par l'opération, elle sera transmise à la fonction semop :

int semop(int semid, struct sembuf *ops, unsigned nbops).

nbops est le nombre d'opérations à exécuter c'est à dire le nombre de sémaphores à manipuler dans le groupe identifié par **semid**. Qui est aussi égal au nombre de structures sembuf passées grâce au pointeur *ops.

Exercice 3:

On souhaite simuler l'accès concurrent à une seule imprimante par plusieurs processus. Un seul processus utilise l'imprimante à un instant donné. Écrivez un programme Spool.c qui se duplique pour créer N fils (La valeur de N est transmise par clavier). Chaque fils tente d'accéder à la ressource : une fois dedans, il dort un temps aléatoire entre 1 et 3 secondes, affiche un message indiquant qu'il a terminé d'utiliser la ressource ainsi que la durée d'utilisation, et ensuite il libère la ressource. Le père attend la terminaison de tous ses fils avant d'indiquer sa terminaison.

Exercice 4:

Nous considérons une mémoire partagée entre plusieurs processus accédant à la zone soit pour écrire des données ou pour lire le contenu. Proposez un code pour les processus lecteurs et le code d'un processus rédacteur. Le processus principal créera 1 processus fils rédacteur et n autres processus fils lecteurs (n est entré au clavier). Il y a exclusion mutuelle entre le rédacteur et les lecteurs tandis que plusieurs lecteurs peuvent accéder en même temps à la zone commune.

Pour créer une zone mémoire partagée entre un groupe de processus on utilise la commande shmget (man shmget) qui s'utilise de la même manière que semget pour les sémaphores.

TP 3 Systeme d'exploitation: Semaphores - suite

TP semaphores suite..

Exercice 5: Producteur-Consomateur

Un processus producteur et un processus consommateur se partagent l'accès a une zone mémoire permettant de stocker 10 valeurs entières (un tableau) que le producteur dépose et que le consommateur récupère pour les afficher (une seule valeur a la fois).

- ☐ Producteur et consommateur peuvent accéder a la zone en même temps pour écrire et lire dans des cases différentes du tableau. Deux indexes idxr et idxl permettraient d'indiquer la case actuelle d'écriture et de lecture (qui est aussi la dernière case écrite). Le tableau est gère comme une pile.
- ☐ Le producteur se bloque si la zone est pleine en attendant d'être réveillé par un consommateur qui libère une case du tableau (la lit).
- ☐ Le consommateur se bloque s'il n'y a aucune case écrite et est réveillé par le producteur des qu'il fini d'écrire dans une case du tableau.
- ☐ Si il y a plusieurs consommateurs et plusieurs producteurs, il y a exclusion mutuelle entre les producteurs de même entre consommateurs pour l'accès a toute la zone.

1- Ecrire deux programmes producteur.c et consommateur.c qui vont écrire et lire dans une zone préalablement crée par un autre programme creat.c (qui crée aussi les sémaphores nécessaires).

2- Généraliser la solution a n consommateurs et n producteurs (ajouter deux sémaphores binaires pour réaliser l'exclusion mutuelle entre les producteurs et entre les consommateurs).

Implémenter la solution théorique suivante a ce problème:

Sémaphores **nvide**=10, **nplein**=0.

Sémaphores binaires **mutex** (pour protéger idxl), **mutexc** (EM entre consommateurs), **mutexp** (EM entre producteurs).

int idxr=0, idxl=0

Processus producteur:	Processus consommateur
Debut: Repeter produire (article); P(nvide) P(mutexp); deposer (idxr, article); P(mutex) idxl=idxr idxr++ V(mutex) V(mutexp); V(nplein); Tantque (vrai) Fin.	Debut: Repeter P(nplein); P(mutexc); Prelever (article, idxl); P(mutex) idxr=idxl idxl-- V(mutex) V(nvide); V(mutexc); Consommer (article); Tantque (vrai) Fin.

- | | |
|--|--|
| | |
|--|--|
- Pour implémenter la zone partagée on utilisera les fonctions **shmget** et **shmat**. Les données a partager sont décrites dans une structure data décrite dans le code suivant.

```
#include <stdio.h>
# include <unistd.h>
#include <sys/shm.h>

typedef struct data{
int idxl;
int idxr;
int tab[10];
}sdata;

int main(){
//creation du segment a faire dans un seul processus puis passer l'identifiant
aux autres

key_t key=ftok("/Users/user",5); //modifier le chemin

int shmid=shmget(key,sizeof(sdata),IPC_CREAT|IPC_EXCL|0666);

if(shmid==-1){
shmid=shmget(key,sizeof(sdata),0);

printf("Segment existe deja d'id:%d\n",shmid);
}else printf("Segment mémoire d'id:%d\n",shmid);

//tout les processus doivent appeler shmat pour attacher une adresse a la zone
memoire le //pointeur permet par la suite d'ecrire directement des donnees dans
la zone partagee.

sdata *sd=shmat(shmid,sd,0);

//maintenant on peut ecrire dans la zone via la structure.
sd->idxl=10;

printf("val ecrites:%d, %d\n",sd->idxl,sd->idxr);
return 0;
}
```