USTHB
Departement Informatique
Master 1 SII

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

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

- Les pipes ou tubes: communication entre processus pere et procesus fils a travers des descripteurs de fichiers (communication entre processus lies).

- 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 lies.

Ces deux moyens sont geres comme des entrees sorties et donc sont couteux en temps d'execution. Il existe aussi des moyens de communication geres 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 ancetre de Linux). Il y a 3 sortes d' IPCs

- Les files de messages (message queues)
- Les segments de memoire partagee (shared memory segments)
- Les semaphores

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

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

Les ressources IPC sont identifiees par une cle unique dans tout le systeme.
Avant de creer une ressource IPC, ont doit fournir une cle unique qu'on peut obtenir avec la fonction **ftok** (file to key) qui permet de creer une cle a 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 genere par le systeme lors de la creation d'une ressource IPC d'un type donne et qui permet d'identifier les ressources de meme type.

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

Les IPCs System V ont la particularite de rester en memoire meme apres que le processus les ayant crees ait termine son execution (si ce dernier n'a pas supprime les ressources qu'il a cree). Pour creer lister, supprimer des IPCs en ligne de commande on utilise les commandes suivantes:

Creation d'une ressource IPC	ipcmk (options: -s semaphores, -m memoire, -q fille de messages)
Lister toutes les ressources IPC existantes dans le systeme	# ipcs -a (ou -s, -m, -q pour lister chaque type a 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 resource (option -p de ipcs)	# ipcs -m -p

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

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

Les semaphores system V sont crees par groupes de **n** semaphores. Un semaphore est une structure **semid_ds** qui contient toutes les informations du groupe. Les semaphores d'un meme groupe sont numerotes de **0 a n**.

- La fonction **semget (key, n, flags)** permet de **creer** ou **obtenir** l'identifiant d'un groupe de semaphores existants. **N** est le nombre de semaphores dans le groupe, flags est une combinaison (ou logique) d'options de creation. L'option **IPC_CREATE** permet de creer un nouveau groupe de semaphores, l'option **IPC_EXCL** permet de retourner **-1** si le groupe existe deja. La 3eme option necessaire est les droits d'acces au nouveau groupe de semaphores (0666 pour tout les droits).

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

- L'appel **semget (key, n, 0)** retourne l'identifiant du groupe de semaphore existant ayant la cle key.

- **Le retour de semget:** -1 erreur semaphore existant si IPC_EXCL est indiquee dans le flag ,
  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 deja, on recupere son identifiant (avec un deuxieme appel a semget) et on l'affiche . Utiliser la commande **ipcs** pour visualiser le semaphore cree.

III- Manipulation de semaphores: la fonction semctl

La fonction **int semctl (semid, semnum, command, arg)** permet d'initiaiser la valeure d'un

semaphore, recuperer la valeure, supprimer un groupe de semaphores etc. La liste complete 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 semaphore: 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 semaphores existant on utilise la fonction semctl avec la commande IPC_RMID: semctl (semid, 0, IPC_RMID) le deuxieme argument est ignore 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 semaphores du groupe avec un seul appel a la fonction semctl (utiliser la commande SETALL et GETALL pour recuperer les valeures).

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 (apres SC). Une troisieme operation est possible aussi Z et qui permet de bloquer un processus jusqu'a ce que la valeure d'un semaphore atteint 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 valeures 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 remplit 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 a dire le nombre de semaphores a manipuler dans le groupe identifie par **semid**. Qui est aussi egal au nombre de structures sembuf passees grace 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 base de données où plusieurs processus peuvent lire en même temps mais un seul peut écrire à la fois. Proposez un code pour les processus lecteurs et le code d’un processus écrivains.