Université de Rouen
Licence informatique
Module : Système I
2003/2004

TP 4 : IPC (InterProcess Communication)

1 Préambule : gestion des signaux

Ecrire un programme qui comptabilise le nombre de signaux SIGINT et SIGQUIT qui lui sont délivrés, et se termine lorsque l'un de ces nombres devient égal à NMAX.

2 Mémoire partagée

Ce mécanisme permet à plusieurs programmes de partager des segments mémoire. Chaque segment mémoire est identifié par une clé à laquelle correspond un identifiant. Losqu'un segment est attaché à un programme, les données qu'il contient sont accessibles par un pointeur.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int shmflg);
char *shmat(int shmid, car *shmadrr, int shmflg);
int shmdt(chmadrr);
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- shmget() donne l'indentifiant du segment ayant la clé key. Un nouveau segment (de taille size) est créé si key est IPC_PRIVATE, ou bien si les indicateurs de shmflg contiennent IPC_CREAT.
  Combinées, les options IPC_EXCL | IPC_CREAT indiquent que le segment ne doit pas exister auparavant. Les bits de poids faible de shmflg indiquent les droits d'acces.
- shmat() attache le segment shmid en mémoire, avec les droits spécifiés dans shmflag (SHM_R, SHM_W, SHM_RDONLY). shmaddr précise où ce segment doit être situé dans l'espace mémoire (la valeur NULL demande un placement automatique). shmat() renvoie l'adresse où le segment a été placé.
- shmdt() sert à détacher le segment si on ne l'utilise plus.
- shmctl() permet de paramétrer ou de supprimer un segment partagé.

Pour constituer une clé, utiliser key_t ftok(char *pathname, char project) ;
Vous baserez la clé sur une donnée personnelle, par exemple votre répertoire d'accueil.

Ecrire un programme impliquant 2 processus, où le fils lit une suite de nombres et effectue le cumul dans une variable en mémoire partagée, et le père lit périodiquement le contenu de la mémoire partagée. Le programme s'arrête par Contrôle-C (signal SIGINT).

La commande ipcs affiche des informations sur les segments qui existent. Vérifiez les ressources utilisées et si besoin utilisez ipcrm pour supprimer un segment.

3 Sémaphores

Lors d'un accès à des ressources communes, il est indispensable de synchroniser les différents acteurs pour éviter les interférences regrettables. Pour cela, on dispose d'un mécanisme IPC servant à synchroniser l'utilisation des mémoires partagées : les sémaphores.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget (key_t key, int nsems, int semflg);
int semop (int semid, struct sembuf *sops, unsigned nsops);
int semctl (int semid, int semnum, int cmd, union semun arg);
```

La fonction semget() demande à travailler sur le sémaphore généralisé identifié par la clé key (même notion que pour les clés des segments partagés), et qui contient nsems sémaphores individuels. Un nouveau sémaphore est créé, avec les droits donnés par les 9 bits de poids faible de semflg, si key est IPC_PRIVATE, ou si semflg contient IPC_CREAT.

La fonction semop() agit sur le sémaphore semid en appliquant simultanément à plusieurs sémaphores individuels les actions décrites dans les nsops premiers éléments du tableau sops. Chaque sembuf est une structure de la forme :

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

- sem_flg est une combinaison des indicateurs IPC_NOWAIT et SEM_UNDO.
- sem_num est le numéro du sémaphore individuel sur lequel porte l'opération.
- sem_op est un entier destiné (sauf s'il est nul) à être ajouté à la valeur courante semval du sémaphore. L'opération se bloque si sem_op + semval < 0. Cas particulier : si sem_op est 0, l'opération est bloquée tant que semval est non nul. Rq : les valeurs des sémaphores ne sont mises à jour que lorsqu'aucun d'eux n'est bloqué.

La fonction semctl permet de réaliser diverses opérations sur les sémaphores, selon la commande demandée. En particulier, on peut fixer le n-ième sémaphore à la valeur val en faisant : semctl(sem, n, SETVAL, val) ;

1) Ecrire les primitives sur les sémaphores : creer_sem, detruire_sem, changer_sem, P, V.

2) Utilisez les sémaphores pour sécuriser l'exercice précédent sur les mémoires partagées.

4 Files de messages

Ce mécanisme permet l'échange de messages par des processus. Chaque message possède un *corps* de longueur variable, et un *type* (entier strictement positif) qui peut servir à préciser la nature des informations contenues dans le corps.

Au moment de la réception, on peut choisir de sélectionner les messages d'un type donné.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
int msgsnd(int msqid, struct msgbuf *msgp, int msgsz, int msgflg);
int msgrcv(int msqid, struct msgbuf *msgp, int msgsz, int msgtyp, int msgflg);
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

msgget() demande l'accès à (ou la création de) la file de message avec la clé key.
msgget() retourne la valeur de l'indentificateur de file.

msgsnd() envoie un message dans la file msqid. Le corps de ce message contient msgsz octets. Il est placé, précédé par le type, dans le tampon pointé par msgp.
Ce tampon est de la forme :

```
struct msgbuf {
  long mtype;
  char mtext[..];
};
```

msgrcv() lit dans la file un message d'un type donné (si type>0) ou indifférent (si type==0), et le place dans le tampon pointé par msgp. La taille du corps ne pourra excéder msgsz octets, sinon il sera tronqué.
msgrcv() renvoie la taille du corps du message.

Ecrire un programme impliquant deux processus, le fils envoyant des messages (lignes de texte) sur une file avec un type donné, et le père affichant les messages reçus.