Programmation Système avancée

Wieslaw Zielonka
zielonka@irif.fr

Résumé de cours précédent

Avec les méthodes de cours précédents pour obtenir une mémoire partagée entre des processus il faut :

- soit disposer d'un fichier qui nous sert à initialiser la mémoire (et qui par le mécanisme de synchronisation permet de sauvegarder le contenu de la mémoire partagée)

- soit utiliser la mémoire partagée anonyme. Mais la mémoire partagée anonyme permet uniquement le partage de mémoire entre un processus et ces descendants (à condition que les descendants n'exécutent jamais exec()). Mémoire partagée anonyme ne fait pas partie de Single Unix Specification (POSIX).

- comment faire partager la mémoire entre les processus sans lien de parenté et sans fichier à partager?

Ces problèmes résolus grâce aux objets mémoire POSIX

(Mais il nous manquera toujours un mécanisme de synchronisation entre les processus qui partagent la mémoire.)

objets mémoire POSIX

Objets mémoire

Objets mémoire peuvent peuvent être vu comme de fichiers et dans Linux ils sont implémentés comme fichiers dans un système de ficher virtuel /dev/shm .

```
int shm_open(char *name, int oflag, mode_t mode)
```

crée et ouvre un objet mémoire et retourne le descripteur vers cet objet ou -1 en cas d'erreur.

`name` - le nom de l'objet. Ce nom doit commencer par '/' et ne doit pas contenir d'autres caractères '/'

`oflag` - bitmask de O_CREATE, O_EXCL, O_RDONLY, O_RDWR, O_TRUNC avec la même signification que pour open sur les fichiers ordinaires. Si O_CREATE n'est pas spécifié alors le fichier doit déjà exister.

`mode` - les permissions, les mêmes valeurs que pour les fichiers ordinaires.

Objets mémoire

```
int fd = shm_open("/monobjet",
                  O_CREAT | O_RDWR ,
                  S_IRUSR | S_IWUSR);
```

ouverture de l'objet mémoire "/monobjet" avec la création si l'objet n'existe pas.

L'objet sera ouvert en lecture et écriture.

Si l'objet n'existe pas le nouveau objet aura les permission read+write pour le propriétaire.

Objets mémoire

`fstat()` peut être appliqué à un objet mémoire ouvert et le champ `st_size` de `struct stat` donnera la taille de l'objet mémoire en octet :

```
int fd= shm_open("/monobjet", O_RDWR, 0);

struct stat bufStat;

fstat( fd, &bufStat);

printf( "longueur objet %d\n",

        (int) bufStat.st_size);
```

Objets mémoire

La taille d'objet mémoire juste après la création est `0`.

Donc la première chose à faire après la création c'est appliquer `ftruncate()` pour fixer la taille de l'objet mémoire

```
int fd = shm_open("/monobjet",
                  O_CREAT | O_RDWR ,
                  S_IRUSR | S_IWUSR);
if( fg == -1 ){ perror("shm_open"); exit(1);}

off_t len = 1024;

/* ftruncate permet de changer la taille de l'objet
partagé comme pour les fichiers ordinaires */

if( ftruncate( fd, len) == -1){perror("ftruncate");
exit(1);}
```

modifier/lire objet mémoire

Sous Linux on peut appliquer read()/write() pour lire et écrire dans un objet mémoire, comme pour les fichiers ordinaires. Mais cette possibilité vient de l'implémentation d'objets mémoire sous linux, sous linux les objets mémoire sont implémentés par les fichiers virtuels.

Single Unix Specification permet l'implémentation des objets mémoires pas des fichiers mais permet aussi d'autres implémentations.

Par exemple, **il n'est pas possible** d'utiliser read/wrire sur les objets mémoire sous MacOS où les objets mémoires sont implémentés mais pas comme les fichiers.

Comme Linux implémente les objets mémoire comme les fichiers virtuels, sous Linux il est possible d'ouvrir un objet existant et changer sa taille à l'aide de `ftruncate()`.

Sous MacOS, on peut (**et il faut**) appliquer `ftruncate()` seulement après la création de l'objet mémoire quand sa taille est 0. La tentative de modifier la taille de l'objet qui a déjà une taille différente de 0 provoque erreur.

modifier/lire objet mémoire

La méthode applicable sur tous les systèmes : faire la projection de l'objet mémoire dans l'espace d'adressage du processus avec `mmap()` :

```
int fd = shm_open(”/monobjet” , ORDWR, 0 );
struct stat bufStat ;
fstat(fd, &bufStat);
void *adr = mmap(NULL, bufStat->st_size ,
                 PROT_READ | PROT_WRITE,
                 MAP_SHARED, fd, 0) ;
```

Mais n'oubliez pas `ftruncate()` si `O_CREAT` est spécifié dans `shm_open()`, `ftruncate()` qui doit être appliqué **avant** `mmap()`

durée de vie d'objet mémoire

L'objet mémoire est supprimé avec

```
int  shm_unlink(cont char *name);
```

L'objet mémoire sera effectivement supprimé quand il n'y a plus de processus qui possèdent une référence vers cet objet.

Sous Linux on peu retrouver les objets mémoire dans le répertoire `/dev/shm` et le supprimer avec la commande `rm`.

Si l'objet mémoire n'est pas supprimé alors **il garde son contenu jusqu'au redémarrage (reboot)** du système (comparer avec les tubes nommés `fifo`).

compilation sous Linux

Les objets mémoire sont implémentés dans la bibliothèque real-time donc dans l'étape de l'édition de liens ajouter

```
-lrt
```

ou encore mieux dans Makefile

```
LDLIBS = -lrt
```

(Cette remarque ne concerne pas MacOS).