Travaux Dirigés Programmation Système: Feuille 6

Informatique 2ème année. ENSEIRB 2018/2019

—*Mathieu Faverge* - `mfaverge@enseirb.fr` —

Mémoire partagée et threads.

▸**Exercice 1.** *Communication par* ***mémoire partagée*** *:*

1. *Écrivez un programme qui :*
   - *Crée ou récupère un identifiant de mémoire partagée à l'aide à de* `shmget(2)` *et* `ftok(3)`*. Pour la création de la clé à l'aide de* `ftok(3)`*, on utilisera le nom d'un fichier système (par ex.* `/etc/bashrc`*). Pour* `shmget(2)`*, on utilisera une taille de 4 octets et pour le flag, la valeur* `IPC_CREAT | 0644`*.*
   - *Attache le segment ainsi identifié à l'espace virtuel du processus, avec* `shmat(2)`*, comme un tableau d'entiers puis incrémente le premier entier du tableau et l'affiche.*
   - *Détache le segment de l'espace du processus avec* `shmdt(2)` *et quitte.*

   Relancez plusieurs fois votre programme, de plusieurs terminaux. Qu'observez vous ? Placez-vous sur un serveur partagé entre plusieurs étudiants et recommencez l'expérience (il est possible que les droits avec `shmget(2)` *doivent être ajustés).*
2. *Ajoutez avant de quitter un appel à* `shmctl` *pour détruire le segment de mémoire partagée. Refaites l'expérience.*

▸**Exercice 2.** *Threads*

Le but de cet exercice est la manipulation des fonctions de création et d'attente des threads. On pourra faire `man pthread(3)` *pour avoir une liste exhaustive des fonctions utilisables pour les threads. La création de threads se fait avec la fonction* `pthread_create(3)` *qui prend notamment en argument un pointeur sur la fonction qu'exécutera le thread.*

- *Ecrire un programme qui lance 8 nouveaux threads (en plus de celui du programme principal) sur une fonction* `void *start(void *)` *qu'on écrira. Cette fonction affichera le PID du processus. Le PID sera transmis à la fonction* `start` *par* `pthread_create` *par son dernier argument. On compilera avec l'argument* `-lpthread`*.*
- *La fonction* `pthread_join(3)` *permet d'attendre un thread identifié par son THREAD ID (donné lors de l'appel de* `pthread_create`*. Modifiez le programme précédent pour que le thread principal attende tous les threads qu'il a créés puis affiche un message sur la sortie standard.*
- *Ajoutez deux variables entières au programme précédent, l'une globale, l'autre locale à la fonction* `start` *(l'initialiser à 0). Incrémentez dans* `start` *les deux variables, affichez leurs adresses et leurs valeurs. Que se passe-t-il ?*

▸**Exercice 3.** *Chat multithreadé*

Le but de cet exercice est de compléter le "chat" du TD2 pour une gestion multithreadée. On rappelle qu'il y a 2 programmes, le serveur `serveur.c` *et le client* `client.c`*. Le serveur crée deux tubes (FIFOs), avec* `mknod(2)` *(ou* `mkfifo(2)`*) puis ouvre les deux fichiers, l'un en lecture l'autre en écriture. Le client ne fait que l'ouverture des deux fichiers. Ensuite, à tour de rôle, ils écrivent dans le tube ce qu'ils lisent sur l'entrée standard puis écrivent sur la sortie standard ce qu'ils lisent en provenance du tube.*

Code du serveur :

```
#define CHK(X) if ((X)<0) { perror(#X); exit(1); }
...
  // IN et OUT sont les fichiers de FIFO
  CHK(mkfifo(IN,0666));
  CHK(mkfifo(OUT,0666));
  CHK(out = open(OUT, O_WRONLY));
  CHK(in = open(IN, O_RDONLY));
  do {
    CHK(n = read(in, buf, BUFSIZE));
    if (n>0) write(STDOUT_FILENO,buf,n);
    CHK(n = read(STDIN_FILENO, buf, BUFSIZE));
    if (n>0) write(out,buf,n);
  } while (n>0);
...
```

Code du client :

```
#define CHK(X) if ((X)<0) { perror(#X); exit(1); }
...
  // IN et OUT sont les fichiers de FIFO
  // buffer est un tableau de char de taille BUFSIZE
  CHK(in = open(IN, O_RDONLY));
  CHK(out = open(OUT, O_WRONLY));
  do {
    CHK(n = read(STDIN_FILENO, buf, BUFSIZE));
    if (n>0) write(out,buf,n);
    CHK(n = read(in, buf, BUFSIZE));
    if (n>0) write(STDOUT_FILENO,buf,n);
  } while (n>0);
```

- *Compléter les deux programmes ci-dessus avec les déclarations correctes, les includes, et en définissant dans chaque cas le nom des fichiers IN et OUT de FIFO. Tester leur fonctionnement en lançant d'abord le serveur, puis le client. Quelles sont les contraintes au niveau des échanges ?*
- *Modifier les programmes précédents pour qu'après l'ouverture des deux FIFOs, deux threads soient lancés : l'un pour écouter ce qui vient du tube et écrire sur la sortie standard, l'autre pour lire ce qui vient de l'entrée standard et l'écrire dans le tube.*
- *Tester le programme modifié si possible entre deux terminaux (les écritures et les lectures peuvent se dérouler simultanément). Que se passe-t-il ?*