Communication Inter Processus (IPC) : Tubes

Principe

- À partir de maintenant, on veut faire communiquer plusieurs processus.
- Un tube est un moyen de le faire.

Note :

- Faire communiquer des processus sur une même machine par tubes peut sembler archaïque, et pas très efficace (comparé à la mémoire partagée).
- Mais : les communications réseau (par sockets) se feront de manière similaire
- les principes/fonctions expliqués dans ce chapitre seront toujours valables.

Principe

- Tube : canal de communication FIFO (First In First Out)
- Utilise 2 descripteurs de fichiers : un pour l'écriture (l'entrée), et un pour la lecture (la sortie)
- L'écriture dans l'entrée sera mise en attente dans un tampon
- La lecture dans la sortie lira les données du tampon, dans l'ordre (FIFO).
- La lecture et l'écriture se font comme pour les fichiers réguliers : read et write
- Il n'y a pas de curseur : lseek est impossible !

Principe

Par exemple, lorsque l'on exécute :

```
cat fichier.txt | grep password
```

- le shell lance deux nouveaux processus : un pour `cat` et un pour `grep`
- le shell crée un tube
- la sortie standard de `cat` sera le côté "écriture" du tube
- l'entrée standard de `grep` sera le côté "lecture" du tube

Principe

Plus précisement :

```
cat fichier.txt | grep password
```

- le shell crée un tube
- le shell lance deux nouveaux processus (deux `fork()`) : un pour `cat` et un pour `grep`
- la sortie standard de `cat` est ecrasée par le côté "écriture" du tube (via par exemple `dup2`)
- l'entrée standard de `grep` est écrasée par le côté "lecture" du tube
- les fils se recouvent (`exec...`) en cat et grep.

Principe

```
cat file.txt | grep passwd | sed 's/.*passwd=\(\w*\).*/\1/'

$ lsof
...
cat   5223   mrao   0u   CHR  136,1   0t0   4       /dev/pts/1
cat   5223   mrao   1w FIFO    0,10   0t0   27854   pipe
cat   5223   mrao   2u   CHR  136,1   0t0   4       /dev/pts/1
...
grep  5224   mrao   0r FIFO    0,10   0t0   27854   pipe
grep  5224   mrao   1w FIFO    0,10   0t0   27856   pipe
grep  5224   mrao   2u   CHR  136,1   0t0   4       /dev/pts/1
...
sed   5225   mrao   0r FIFO    0,10   0t0   27856   pipe
sed   5225   mrao   1u   CHR  136,1   0t0   4       /dev/pts/1
sed   5225   mrao   2u   CHR  136,1   0t0   4       /dev/pts/1
```

Créer un tube (par syscall)

```c
#include <unistd.h>

int pipe(int pipefd[2]);
```

- Ouvre les 2 descripteurs de fichier associés a un nouveau tube
- Prend en argument un tableau de deux entiers :
- Renvoie dans `pipefd[0]` la sortie du tube (le descripteur en lecture)
- Renvoie dans `pipefd[1]` l'entrée du tube (le descripteur en écriture)

Exemple : pipe

```
main() {
   int fd[2],r;
   char buffer[10];

   pipe(fd);

   r=write(fd[1],"hello",5);
   assert(r==5);

   r=read(fd[0],buffer,10);
   assert(r==5);

   buffer[r]=0;
   printf("recu: %s\n",buffer);
}
```

Exemple : pipe + fork

Créer un tube nommé ("fichier tube")

- Un autre moyen de créer un tube est de créer et ouvrir un "tube nommé"
- Il s'agit d'un fichier spécial (non "régulier")
- Commande shell pour créer un tube nommé : `mkfifo`.
- Appels systèmes : `mkfifo` ou `mknod`.

Quand un fichier tube est ouvert en lecture, et ouvert par un autre processus en ecriture, le comportement sera le même qu'un tube crée par `pipe`

Mode "flot" (stream)

Mode flot : les envois successifs d'informations s'additionnent.
Il n'y a pas de "séparations" entre elles.

Exemple :

- `write(in,"ABC",3)`
  - le tube contient "ABC"
- `write(in,"123",3)`
  - le tube contient "ABC123"
- `read(out,bf,4)`
  - renvoie 4, et bf contient "ABC1"
  - le tube contient "23"
- `read(out,bf,4)`
  - renvoie 2, et bf contient "23"
  - le tube est vide
- `read(out,bf,4)`
  - bloque jusqu'à ce qu'un processus écrive dans le fifo...

Nombre de lecteur et écrivains

- Un tube peut avoir un nombre de lecteur (ou d'écrivain) différent de un.
- Si un tube a 0 lecteur : l'écriture échouera (signal SIGPIPE)
- Si plus d'un lecteur : premier arrivé, premier servi
- Si un tube a 0 écrivain (et le tube est vide), la lecture renverra 0 (i.e. comme pour un fin de fichier)
- Comme toujours, on ferme les descripteurs qui ne servent plus.

Caractère bloquant

- Par défaut, la lecture dans un tube vide sera bloquant
- Il est possible de rendre la lecture non bloquante, en changeant l'option O_NONBLOCK du descripteur de fichier
- Dans ce cas, la lecture dans un tube vide échouera (retour -1), avec errno = EAGAIN
- Attention, un tube a aussi une capacité limitée (PIPE_BUF=4096).
  Quand un tube est plein, une écriture sera également bloquante.

Manipuler un descripteur de fichier : fcntl

`fcntl` permet de manipuler les descripteurs de fichiers.

Elle permet (entre autres) de changer les options (modes) des descripteurs de fichiers. En particulier :

- O_NONBLOCK : caractère non bloquant d'un descripteur

Pour passer un descripteur en mode non bloquant :

```
int flags = fcntl(fd, F_GETFL, 0);
fcntl(fd, F_SETFL, flags | O_NONBLOCK);
```

Attente sur plusieurs descripteurs de fichiers : select

`select` permet d'attendre (avec un temps limite) sur un ensemble de descripteurs de fichiers en un seul appel.

Pour utiliser `select`, il faut au préalable manipuler une structure qui représente un ensemble de descripteurs de fichiers. Cela se fait via les primitives suivantes :

```
#include <sys/select.h>

void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

Attente sur plusieurs descripteurs de fichiers : select

```
#include <sys/select.h>

int select(int nfds, fd_set *readfds, fd_set *
    writefds, fd_set *exceptfds, struct timeval
    *timeout);
```

- nfds : le plus grand descripteur de fichiers à vérifier +1
- readfds : l'ensemble des descripteurs à vérifier en lecture
- writefds : l'ensemble des descripteurs à vérifier en ecriture
- execptfds : l'ensemble des descripteurs à vérifier en exception
- timeout : temps maximal à attendre.

À sa sortie, `select` modifie les ensembles de telle façon qu'il ne reste que les descripteurs de fichiers sur lesquels il y a quelque chose à lire ou écrire.

Attente sur plusieurs descripteurs de fichiers : poll

`poll` permet également d'attendre sur un ensemble de descripteurs de fichiers, mais plus finement.

```
#include <poll.h>

int poll(struct pollfd *fds, nfds_t nfds, int timeout);

struct pollfd {
 int    fd;             /* file descriptor */
 short  events;         /* requested events */
 short  revents;   /* returned events */
};
```

- fds : un table de pollfd à surveiller,
- nfds : taille de fds
- timeout : temps maximum (en millisecondes)
- cmdevents et `revents` sont des conjonctions de :
  - POLLIN : il y a quelque chose à lire
  - POLLOUT : il est possible d'y écrire
  - POLLERR : il y a une erreur
  - POLLHUP : pipe ou socket fermé de l'autre côté

Un premier pas vers les communications réseau

Une socket est un point de communication où il est possible d'envoyer et de recevoir des informations.
On en reparlera longuement au moment de la programmation réseau

Les sockets communiquent par pair. Il y a plusieurs moyen de les faire communiquer (différents protocoles réseau, ou en local).

On peut créer une paire de socket en communication locale, qui fonctionnera similairement deux tubes :

- l'entrée de la 1ere socket sera l'entrée du 1er tube et la sortie de la 2eme socket sera la sortie du 1er tube
- l'entrée de la 2eme socket sera l'entrée du 2eme tube et la sortie de la 1ere socket sera la sortie du 2eme tube

Un premier pas vers les communications réseau

```
#include <sys/types.h>
#include <sys/socket.h>

int socketpair(int domain, int type, int
    protocol, int sv[2]);
```

Crée 2 sockets associées.
Pour le faire via une communication locale :

- domain = AF_UNIX
- protocol = 0
- type :
  - SOCK_STREAM : communication par flot (comme pour les tubes)
  - SOCK_DGRAM : communication en mode paquet
- sv : un tableau de 2 entiers, pour le renvoi des 2 descripteurs de fichiers (les 2 sockets)

mode paquet (DGRAM)

Au contraire du mode flot (STREAM), chaque information envoyée constitue une entité indivisible.
Exemple :

- `write(in,"ABC",3)`
  - la file de messages contient "ABC"
- `write(in,"123",3)`
  - la file contient "ABC","123"
- `read(out,bf,10)`
  - renvoie 3, et bf contient "ABC"
  - la file contient "123"
- `read(out,bf,10)`
  - renvoie 3, et bf contient "123"
  - la file vide
- `read(out,bf,4)`
  - bloque jusqu'à ce qu'un processus écrive dans le socket...

Autres IPC

D'autres moyens de communication inter-processus existent (POSIX et SysV).

Nous n'ont parlerons pas, car les mécanismes sont similaires à des mécanismes déjà vus (pipe/socket), ou que l'on verra plus tard (threads)

Ce sont :

- Les files de messages (POSIX : `man mq_overview`)
  - Similaire aux sockets en mode paquet (DGRAM)

Autres IPC

- La mémoire partagée (POSIX : `man shm_overview`)
  - Un segment mémoire est partagé entre plusieurs processus. C'est un moyen de communication très rapide (au sein d'une même machine), mais il faut faire attention aux synchronisations.
- Les sémaphores (POSIX : `man sem_overview`)
  - Il s'agit de mécanisme de synchronisation (exclusion mutuelle). On parlera de sémaphores et mutex en même temps que les threads.

Pour voir les mécanismes System V : `man svipc`

Suite :

- gdb
- Threads
- Réseau...