Tubes nommés, contrôle des flots d'E/S

Thèmes

- Tubes nommés (fifo)
- Contrôle des flots d'E/S (`fcntl`)
- Attente sur un ensemble de descripteurs (`select`)
- BE minichat

Ressources

Pour ce TP, comme pour les suivants, vous pourrez vous appuyer sur

- Le polycopié intitulé « Systèmes d'exploitation : Unix », qui fournit une référence généralement suffisante sur la sémantique et la syntaxe d'appel des différentes primitives de l'API Unix. Chaque section du sujet de TP indique la (ou les) section(s) du polycopié correspondant au contenu présenté.
- Les pages du manuel en ligne (commande `man`), et plus particulièrement les sections 2 et 3.

Déroulement de la séance :

- Tubes nommés : 15'
- Exercice de base select/fcntl : 60'
- Prise en mai du BE : 30'

1 Tubes nommés

Les tubes Unix permettent de construire des architectures élaborées de processus communiquant par flots de données, mais présentent une limitation : les processus communiquant par tubes doivent avoir un ancêtre commun, car les références aux tubes sont transmises par héritage de descripteurs entre processus père et processus fils.

Les tubes nommés (« named pipes » ou « `fifos` » dans la terminologie Unix) proposés par Unix lèvent cette limitation en permettant de créer des tubes qui pourront être désignés par un chemin d'accès, à l'instar des autres fichiers. Ainsi, un tube nommé pourra être désigné et utilisé en dehors du contexte d'exécution du processus qui l'a créé. Du point de vue de l'utilisation, le fonctionnement des tubes nommés est similaire à celui des tubes « ordinaires »[1].

Les tubes nommés peuvent être créés depuis le shell, via la commande `mkfifo` (paragraphe 1.1), ou par programme via la primitive `mkfifo(_)` (paragraphe 1.2).

1.1 Utilisation des tubes nommés en ligne de commande

La commande `mkfifo monTube` permet de créer un tube nommé de nom `monTube`. Ce nom est un chemin d'accès qui permettra à d'autres processus de désigner le tube : le tube créé apparaît parmi les fichiers du répertoire dans lequel il a été placé.

1. avec quelques contraintes d'usage (raisonnables), qui seront précisées par la suite

Exercice Le scénario suivant va permettre d'illustrer la création, la désignation et l'usage des tubes nommés pour l'échange de données entre processus indépendants (pensez à répondre aux questions **avant** d'exécuter les commandes) :

- ouvrir deux terminaux différents, positionnés dans le même répertoire courant.
- depuis le premier terminal, créer dans le répertoire courant un tube nommé de nom `monFifo`
- depuis le second terminal, lancer la commande `cat < monFifo`.

  La commande `cat` sans paramètres affiche (sur la sortie standard) le flot provenant de l'entrée standard (le clavier, par défaut). Ici, l'entrée standard a été redirigée vers le tube nommé `monFifo`. Cette commande affichera donc le flot de données provenant de `monFifo`.

  Qu'observez-vous ? Est-ce normal ?
- depuis le premier terminal, exécuter la commande `ls -l`, en redirigeant la sortie de cette commande vers `monFifo`. Que devriez-vous observer ?

Remarque Le résultat de la commande `ls -l` ci-dessus comportera une ligne similaire à :

```
prw-r--r-- 1 philippe wheel 0 23 avr 18:22 monFifo
```

Le caractère `p` en début de ligne indique que le fichier `monFifo` est un tube nommé.

1.2 Utilisation de tubes nommés via l'API Unix

Une opération spécifique permet de créer un tube nommé :

```
int mkfifo(const char * nom_tube, mode_t mode);
```

Le premier paramètre est le chemin qui permettra de désigner le tube. Le second paramètre permet de fixer les droits d'accès à la création du tube, de manière analogue à `creat()` ou `open(-,O_CREAT,-)`.

Les opérations pour utiliser le tube nommé une fois celui-ci créé sont les opérations classiques : `read()`, `write()`, `open()`, `close()`, `unlink()`.

Particularités

- Il n'est pas possible d'ouvrir avec un même appel à `open(-,O_RDWR)` un tube nommé simultanément en lecture et en écriture. Pour accéder au tube à la fois en lecture et en écriture, il faut utiliser deux appels à `open` (l'un en lecture seule, l'autre en écriture seule), et donc deux descripteurs différents.
- L'ouverture d'un tube nommé peut être bloquante. En effet un écrivain (resp. un lecteur) qui demande à ouvrir un tube nommé devra attendre qu'au moins un lecteur (resp. un écrivain) ait ouvert le tube. Dans ce cas, l'ouverture du tube est effectuée simultanément par le premier lecteur et le premier écrivain. C'est le comportement qui peut être observé avec le scénario du paragraphe précédent (1.1). Il s'agit d'une facilité, à laquelle on peut renoncer en rendant les opérations non bloquantes via `fcntl`.
- Les tubes nommés sont implantés en mémoire vive (RAM) et non par des fichiers. Une conséquence positive est une meilleure efficacité des accès au tube nommé. Une conséquence logique, mais moins positive, est qu'il n'est pas possible d'accéder à un tube nommé créé sur une machine A depuis une machine distante B. Ce tube nommé apparaîtra sur la machine distante B si l'on utilise un système de fichiers en réseau, mais il s'agira d'un tube nommé distinct de celui de la machine B. Autrement dit, lorsqu'un tube nommé est créé, chaque machine dispose d'un tube distinct, utilisable par les seuls processus locaux à la machine.

Exemple Le squelette fourni pour la version serveur du `minichat` comporte un exemple de création de tube nommé (`Ecoute`)

2 Contrôle des flots d'E/S : `select`, `fcntl` (polycopié API Unix, section 3.3)

Opérations essentielles

- `fcntl(-)` permet de contrôler les options et modes d'utilisation d'un fichier désigné par un descripteur. L'appel qui vous sera principalement utile par la suite est :

  ```
  fcntl(desc, F_SETFL, fcntl(desc, F_GETFL) | O_NONBLOCK));
  ```

  qui permet de rendre non bloquantes les opérations sur le fichier correspondant. Dans ce cas, une opération comme `read(-)` qui aurait été bloquante (en l'absence de données disponibles sur le descripteur) termine immédiatement et renvoie -1, et la variable `errno` vaut `EAGAIN` (voir polycopié, 3.3.3 et 3.3.4)
- `select(-)` qui permet d'attendre un retour d'opérations d'E/S sur un **ensemble** de descripteurs (3.3.5)

Exercice Implémenter les deux versions de l'exercice vu en TD.
Note : un squelette de base, comportant le code complet du fils et celui de la procédure `traiter(-)` est fourni avec ce sujet de TP.

3 Minichat

Le BE proposé permet d'illustrer l'utilisation des primitives vues dans ce TP (version serveur) et dans le suivant, dans un contexte plus significatif.

Vous pouvez consacrer la fin de ce TP à prendre connaissance du sujet de ce BE, et à parcourir les ressources fournies.

Avec ce que vous venez de voir, et en particulier l'exercice précédent (section 2), vous disposez de tous les éléments pour implémenter la version serveur du `minichat`.