Module ITC313 - Informatique

Partie C / C++

TD4 Communication par tubes

Benoît Darties - benoit.darties@u-bourgogne.fr
Université de Bourgogne

Année universitaire 2016-2017

1 Communication par tubes via le shell

Les tubes sont un outil de communication entre processus qui peuvent stocker un certain nombre d'octets de manière temporaire. Les octets sont envoyés dans un tube par des processus écrivains, et lu par des processus lecteurs. Le tube est vu comme une file d'attente en mode `fifo`. A chaque fois qu'un octet est lu, il est sorti du tube. Nous allons tout d'abord manipuler les tubes via le shell

Exercice 1 : *Comportement d'un tube en presence de differents lecteurs/ ecrivains*

Dans cet exercice, nous allons manipuler les tubes exclusivement via le shell. Nous allons pour cela utiliser un tube nommé, puis remplir le tube avec des commandes usuelles disponibles sous UNIX dans le seul but d'illustrer les mécanismes de fonctionnement des tubes. Un premier objectif consistera à écrire un message depuis un terminal et à l'afficher sur un autre terminal.

Dans cet exercice, nous allons manipuler un tube et gérer une communication entre deux processus créés dans des terminaux différents. Toutes les lectures et écritures dans les différents tubes que nous utiliseront se feront à l'aide de la commande `cat`.

1. **Manipulation de la commande `cat` depuis l'entrée standard :**
   (a) La commande `cat` a été précedemment utilisée pour afficher le contenu d'un fichier. Pour rappel, récupérer un fichier contenant uniquement du texte, nommez le `fichier` et affichez son contenu à l'écran avec la commande `cat fichier`. Quel est le résultat de la commande `cat fichier > fichier2` ?
   (b) Si l'on exécute la commande `cat > fichier3`, la commande va non pas lire le contenu provenant d'un fichier, mais directement sur l'entrée standard et copier ce qu'elle lit dans `fichier3`. Exécutez cette commande, tapez quelques lignes, y compris le caractère "retour chariot" (entrée). Le programme ne s'arrête pas tant qu'il n'a pas reçu le caractère de fin de fichier et continue de tourner. Pour arrêter la saisie, tapez le caractère `ctrl+d`. Vérifiez que la commande s'arrête. Quel est alors le contenu du fichier `fichier3` ?
2. **Communication unidirectionnelle entre un lecteur et un écrivain via un tube nommé en shell**
   (a) La commande **mkfifo** permet de créer un tube nommé. En utilisant le manuel de cette commande, créez un tube nommé appelé `vortex`. Affichez ses caractéristiques avec la commande `ls -l`. Que constatez-vous ? quel élément permet de caractériser ce fichier comme étant un tube ?

(b) Rappelez sous quelles conditions un tube nommé peut être accédé correctement en écriture ?

(c) Dans un premier terminal, lancez un `lecteur`, c'est à dire un processus qui sera chargé de lire dans le tube. Ce lecteur se lance simplement avec la commande `cat vortex`. Le tube est vide, il n'y a pas d'écrivains, le processus lecteur se met en attente d'écrivains.

(d) dans un second terminal, lancez un `ecrivain`, , c'est à dire un processus qui sera chargé d'écrire dans le tube. Cet écrivain se lance simplement avec la commande `cat > vortex`. Tapez quelques caractères dans ce terminal et validez avec entrée. Que se passe-t'il dans le terminal dans lequel est lancé le `lecteur` ?

(e) ajoutez quelques lignes, puis terminez l'écriture en saisissant le caractère de fin de fichier `ctrl+d`. Que se passe-t'il du coté du processus `lecteur` ? Expliquez pourquoi.

3. **Extension à plusieurs écrivains et lecteurs**

(a) En utilisant 3 terminaux, créez maintenant un `lecteur` et deux processus `ecrivain`. Saisissez du texte depuis l'un ou l'autre des écrivains (sans utiliser `ctrl+d`), que se passe-t'il ?

(b) Dans l'un des `ecrivains`, terminez votre saisie en saisissant `ctrl+d`. Le comportement de l'écrivain est-il similaire à précédemment ? pourquoi ?

(c) Lancez un second `lecteur` dans le terminal libéré. Dans le terminal de l'écrivain restant, saisissez plusieurs lignes (sans utiliser `ctrl+d`) en validant à chaque fois sur entrée . Que se passe-t'il ? Ce comportement est-il normal ?

(d) Utilisez un quatrième terminal pour recréer un second processus `ecrivain` et écrivez plusieurs lignes depuis les deux terminaux `ecrivain`. Observez que le tube fait office de buffer global et envoie les octets qui viennent indépendemment des deux écrivains vers l'un ou l'autre des lecteurs.

2 Création et manipulation de tubes en C

Exercice 2 : *Etude d'un exemple*

On vous donne le code de 4 programmes. On suppose qu'un utilisateur lance exactement une fois chaque programme à partir de son répertoire personnel.

1. Représentez sur un schéma l'organisation des différents processus issus de l'exécution de ces 4 programmes en supposant des conditions normales de fonctionnement, ainsi que les outils de communication entre ces derniers en respectant les points suivants :
   — Les processus seront représentés par des rectangles à l'intérieur desquels figure le nom du programme correspondant ;
   — les tubes seront représentés par des cylindres à l'intérieur desquels figure le type de tube, à savoir "tube anonyme" ou, s'il s'agit d'un tube nommé, le nom du fichier identifiant ce tube ;
   — vous ferez figurer pour chaque processus les différents descripteurs identifiant un accès en écriture ou en lecture sur ces tubes au moyen d'une flèche reliant le processus au tube, en y collant le nom de la variable du processus identifiant ce descripteur. S'il s'agit d'un descripteur en lecture, la flèche ira du tube vers le processus. S'il s'agit d'un descripteur en écriture, la flèche ira du processus vers le tube. S'il s'agit d'un descripteur en lecture-écriture, les deux extrémités seront fléchées ;
   — S'il existe des relations de parenté entre ces processus (processus parent-enfant), représenterez ces dernières au moyen de flèches en pointillé allant du processus enfant vers le processus parent.
2. En quelques mots, que fait l'instruction pipe() de la ligne 4 de *Programme1.c* ?
3. En quelques mots, que fait l'instruction fork() de la ligne 5 de *Programme1.c* ? Citez une circonstance au cours de laquelle la condition de la ligne 7 est validée. Que dire des deux blocs d'instructions situés entre les lignes 09 et 25 d'une part, et 27 et 40 d'autre part ?

4. Sur *Programme1.c*, l'ordre de fermeture des descripteurs (lignes 22 à 24) a t'il une importance et si oui pourquoi ?
5. Sur *Programme1.c*, que manque-t'il entre les lignes 24 et 25 pour terminer proprement l'exécution du programme sans apparition de zombies ?
6. Combien de processus tentent de créer le tube nommé "tube" ? Que se passe-t'il au niveau d'un processus si le tube existe déjà ?
7. L'utilisateur aurait-il pu lancer ces 4 programmes à partir d'un répertoire de travail dont les droits d'accès auraient été `r-xr-xr-x` et pourquoi ?
8. Que se passe-t'il au niveau des tubes et des processus , si pendant l'exécution des programmes le fichier *./tube* est supprimé après que les différents accesseurs aient été ouverts ?
9. Qu'est censé faire le processus correspondant à *Programme4.c* ? Affichera-t'il toujours la même phrase à l'écran et pourquoi.

"Programme programme1.c"

```
1  int main() {
2      int p[2], fd1, fd2, result;
3
4      pipe(p);
5      result=fork();
6
7      if (result ==-1) exit(EXIT_FAILURE);
8
9      if result > 0) {
10         close(p[0]);
11         mkfifo("./vortex", 666);
12         mkfifo("./tunnel", 666);
13
14         fd1 = open("./vortex", O_WRONLY);
15         fd2 = open("./tunnel", O_WRONLY);
16
17         write(fd1, "Une poule", 9);
18         write(p[1], " sur un mur", 11);
19         write(fd2, " qui picore ", 11);
20         write(p[1], " du pain dur.", 13);
21
22         close(p[1]);
23         close(fd2);
24         close(fd1);
25     }
26
27     else {
28         char buffer[200];
29         int nb_lus;
30         close(p[1]);
31         mkfifo("./tube", 666);
32         fd  = open("./tube", WR_ONLY);
33         nb_lus = read (p[0], buffer, 200);
34         while (nb_lus > 0) {
35             write (fd, buffer, nb_lus);
36             nb_lus = read (p[0], buffer, 200);
37         }
38         close(p[0]);
39         close(fd);
40     }
41     exit(EXIT_SUCCESS);
42 }
```


Remarques / questions : benoit.darties@u-bourgogne.fr

"Programme programme2.c"

```
1  int main() {
2      int fd1, fd2, nb_lus;
3      char buffer[200];
4
5      mkfifo("./vortex", 666);
6      mkfifo("./tube", 666);
7      fd1 = open("./vortex", O_RDONLY);
8      fd2 = open("./tube", O_WRONLY);
9
10     nb_lus = read (fd1, buffer, 200);
11     while (nb_lus > 0) {
12         write (fd2, buffer, nb_lus);
13         nb_lus = read (fd1, buffer, 200);
14     }
15     close (fd1);
16     close (fd2);
17     exit(EXIT_SUCCESS);
18 }
```

"Programme programme3.c"

```
1
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <sys/uio.h>
5  #include <unistd.h>
6  #include <sys/types.h>
7  #include <sys/stat.h>
8
9  int main() {
10     int fd1, fd2, nb_lus;
11     char buffer[200];
12
13     mkfifo("./tunnel", 0666);
14     mkfifo("./tube", 0666);
15     fd2 = open("./tube", O_WRONLY);
16     fd1 = open("./tunnel", O_RDONLY);
17
18     nb_lus = read (fd1, buffer, 200);
19     while (nb_lus > 0) {
20         write (fd2, buffer, nb_lus);
21         nb_lus = read (fd1, buffer, 200);
22     }
23     close (fd1);
24     close (fd2);
25     exit(EXIT_SUCCESS);
26 }
```


Remarques / questions : benoit.darties@u-bourgogne.fr

"Programme programme4.c"

```
1  int main() {
2      int fd, nb_lus;
3      char buffer[200];
4
5      mkfifo("./tube", 666);
6      fd = open("./tube", O_RDONLY);
7
8      do {
9          nb_lus = read (fd, buffer, 199);
10         buffer[nb_lus]='\0';
11         printf("%s", buffer);
12     } while (nb_lus > 0);
13     close (fd);
14     exit(EXIT_SUCCESS);
15 }
```

Exercice 3 : *Taille d'un tube*

Cet exercice a pour objectif de déterminer la taille d'un tube nommé, puis d'un tube anonyme au moyen de l'expérimentation.

Pour déterminer la taille d'un tube, une approche simple consiste à écrire octet par octet dans le tube, en incrémentant un compteur à chaque itération. Lorsque l'on ne peut plus écrire dans le tube, on obtient ainsi sa taille. Cette approche n'est pas si évidente, et doit être traité en C pour éviter des situations de blocage.

1. Que se passe-t-il si l'on essaye d'écrire dans un tube plein ?
2. Ecrivez un programme C nommé `compteurNomme` permettant de déterminer la taille d'un tube nommé qui aura été précédemment créé. Pour cela, initialisez un compteur à 0, et incrémentez ce compteur à chaque fois que vous ajoutez un octet dans un tube. Pour rappel, vous considèrerez les points suivants :
   — on écrit dans un tube nommé de la même façon que l'on écrit dans un fichier.
   — Ceci nécessite de posséder un descripteur en écriture sur ce tube.
   — L'ouverture d'un tube nommé en écriture est bloquante tant qu'aucun lecteur n'a été ouvert sur le tube. Comment contourner ce problème ?
3. Que se passe-t'il une fois que le tube est rempli ? Comment pourrait-on contourner cela ? regardez notamment la fonction `fcntl()` ainsi que son manuel, et particulièrement l'option `O_NONBLOCK` pour ne plus rendre l'écriture bloquante. Quel est alors le retour de la commande `write()` sur ce tube ?
4. Proposez un programme `C` permettant de déterminer la taille d'un tube nommé et celle d'un tube anonyme (le nombre d'octets que l'on peut écrire avant que ce dernier soit plein).
5. Les deux tailles sont-elles identiques ?

Exercice 4 : *Tube nomme en tube anonyme*

L'objectif de cet exercice est d'illustrer ce qui se passe lorsque le fichier support d'un tube nommé est supprimé alors que le tube était actuellement en cours d'utilisation.

L'ensemble de l'exercice s'effectue dans le même répertoire de travail. Il ne faut pas modifier le mode d'ouverture, qui doit être bloquant par défaut.

Remarques / questions : benoit.darties@u-bourgogne.fr

1. Créez un tube nommé `monTube` avec la commande `mkfifo`.
2. Écrivez un programme `lecteur.c` qui lit toutes les secondes le contenu de `monTube` et l'affiche à l'écran dès que des octets sont écrits dans `monTube`.
3. Écrivez un programme `ecrivain.c` qui écrit périodiquement toutes les secondes (en utilisant la fonction `sleep`) sur `monTube` une suite de caractères de votre choix.
4. Compilez vos programmes, exécutez-les en simultané dans deux terminaux différents, et vérifiez que l'écrivain communique bien vers le lecteur.
5. Durant l'exécution de ces 2 processus, supprimez fichier `monTube`. Que se passe-t-il ? Concluez.
6. Question bonus : proposez une démarche pour rendre le tube anonyme, puis déterminer sa taille . Est-elle identique à la taille d'un tube anonyme classique ?

Remarques / questions : benoit.darties@u-bourgogne.fr