
Communication par tubes

Louis Mandel

`louis.mandel@lri.fr`

Laboratoire de Recherche en Informatique

Université Paris-Sud 11

IFIPS

Cycle ingénieur de la filière étudiant

année 2008/2009

Communication inter-processus : les tubes (pipes)

- ▶ Tubes
 - ▷ moyen de communication entre lecteur(s) et écrivain(s)
- ▶ Les tubes sont des objets du système de fichier
 - ▷ accès via un descripteur de fichier
 - ▷ opérations read et write
 - ▷ redirection des entrées/sorties standards depuis/vers un tube
 - ▷ etc.
- ▶ Communication unidirectionnelle
 - ▷ un descripteur pour écrire (mode O_WRONLY)
 - ▷ un descripteur pour lire à l'autre bout (mode O_RDONLY)
- ▶ Deux types de tubes
 - ▷ tubes anonymes, *pipe*
 - ▷ tubes nommés, *fifo*

Redirection vers/depuis un tube

- ▶ Motivation première des tubes, combiner
 - ▷ connexion de la sortie standard d'un processus vers un tube
 - ▷ connexion de l'entrée standard d'un processus depuis un tube
- ▶ Tube liant deux commandes
 - ▷ --> `cmd1 | cmd2`
 - ▷ Création du tube par le shell
 - ▷ Entre `fork` et `exec` de la commande `cmd1` : le shell redirige la sortie standard vers le tube
 - ▷ Entre `fork` et `exec` de la commande `cmd2` : le shell redirige l'entrée standard vers le tube
 - ▷ Destruction du tube automatique à la fermeture des descripteurs

Caractéristiques générales des tubes

- ▶ Communication d'un flot de caractères
 - ▷ un caractère lu depuis un tube est extrait du tube
 - ▷ opérations de lecture indépendantes des opérations d'écritures
 - ▷ Exemple : écrire 5 caractère, en lire 2, en écrire 3, en lire 6...
- ▶ Communication en mode FIFO
 - ▷ First In First Out
 - ▷ premier caractère écrit, premier caractère lu
 - ▷ pas de possibilité de positionnement dans le tube (lseek)
- ▶ Taille bornée
 - ▷ un tube peut être plein
 - ▷ une écriture peut être bloquante

Caractéristiques générales des tubes

- ▶ Nombre de lecteurs
 - ▷ nombre de descripteurs associés à la lecture depuis le tube
 - ▷ s'il n'y a pas de lecteurs, le tube est inutilisable
 - ▷ Les écrivains sont prévenus : signal SIGPIPE
- ▶ Nombre d'écrivains
 - ▷ nombre de descripteurs associés à l'écriture dans le tube
 - ▷ s'il n'y a pas d'écrivains, le tube vide est inutilisable
 - ▷ Les lecteurs sont prévenus : notion de fin de fichier

Tubes anonymes

- ▶ Un fichier sans nom
 - ▷ pas d'entrée dans le système de fichier
 - ▷ on ne peut pas utiliser `open`
 - ▷ création par une opération ad hoc
 - ▷ destruction automatique à la fin de l'utilisation
- ▶ Primitive de création d'un tube
 - ▷ `#include <unistd.h>`
`int pipe(int *fd);`
 - ▷ Initialise le tableau `fd`
 - ▷ `fd[1]` pour l'écriture
 - ▷ `fd[0]` pour la lecture
- ▶ Connaissance du tube
 - ▷ avoir réalisé la création
 - ▷ héritage des descripteurs

Exemple : seul avec un tube

(pp-solo.c)

```
#include <stdio.h>
#include <unistd.h>
#define BSIZE 1024
int main (int argc, char *argv[]) {
    int fds[2]; long n;
    char buf[BSIZE];
    n = atol(argv[1]);
    pipe(fds);
    while (1) {
        putchar('*'); fflush(stdout);
        write(fds[1], buf, n);
        read(fds[0], buf, 256);
    }
    return 0;
}
```

Partage d'un tube

- ▶ Utilisation typique d'un tube
 - ▷ un processus écrivain
 - ▷ un processus lecteur
- ▶ Mécanisme de partage
 - ▷ un tube est créé par un processus père
 - ▷ processus père crée un processus fils
 - ▷ le père et le fils partagent les descripteurs d'accès au tube
 - ▷ chacun va utiliser un « bout » du tube
 - ▷ **chacun détruit le descripteur inutile**

Lecture dans un tube

- ▶ Primitive read
 - ▷ `ssize_t read(int fd, void *buf, size_t nbytes);`
 - ▷ demande de lecture d'au plus `nbytes`
- ▶ Si le tube n'est pas vide (contient `tbytes`) :
 - ▷ `read` extrait `min(tbytes, nbytes)` caractères
 - ▷ écrit à l'adresse `buf`
 - ▷ retourne la valeur `min(tbytes, nbytes)`
- ▶ Si le tube est vide :
 - ▷ Si le nombre d'écrivains est nul
 - ▷ il n'y aura jamais plus rien à lire
 - ▷ c'est le fin du fichier
 - ▷ `read` retourne 0
 - ▷ Si le nombre d'écrivains n'est pas nul
 - ▷ processus bloqué tant qu'il n'y a pas écriture par un écrivain

Lecture dans un tube

- ▶ Réveil d'un processus bloqué en lecture sur un tube vide
 - ▷ si un écrivain réalise une écriture
 - ▷ read retourne une valeur non nulle
 - ▷ si le dernier écrivain ferme son descripteur d'accès au tube
 - ▷ explicitement par un close
 - ▷ automatiquement à sa terminaison
 - ▷ read retourne 0
 - ▷ Bonne pratique
 - ▷ **systématiquement fermer, au plus tôt, tous les descripteurs non utilisés** : garder un descripteur en écriture sur un tube peut potentiellement bloquer un processus

```
#define BSIZE 1024

int main () {
    int fdts[2], /* to son */
        fdfs[2]; /* from son */
    char bufr[BSIZE], bufw[BSIZE];
    pipe(fdts);
    pipe(fdfs);
    switch (fork()) {
    case -1: exit(EXIT_FAILURE);
    case 0:
        read(fdts[0], bufr, 1);
        write(fdfs[1], bufw, 1);
        break;
    default:
        read(fdfs[0], bufr, 1);
        write(fdts[1], bufw, 1);
    }
    printf("bye\n");
    exit(EXIT_SUCCESS);
}
```

Écriture dans un tube

- ▶ Primitive write
 - ▷ `ssize_t write(int fd, const void *buf, size_t nbytes);`
 - ▷ demande d'écriture de `nbytes` caractères
- ▶ Si le nombre de lecteurs dans le tube est non nul
 - ▷ Si `nbytes` est inférieur à `PIPE_BUF` : écriture atomique
 - ▷ sinon découpage par le système : le processus peut passer dans un état endormi
- ▶ Si le nombre de lecteurs dans le tube est nul
 - ▷ Émission du signal `SIGPIPE`
 - ▷ Comportement par défaut : terminaison

```
static int count = 0;
static void alarm_hndlr(int signum) {
    printf("write() blocks after %d bytes\n", count); exit(EXIT_SUCCESS);
}
int main () {
    int fd[2];
    int pipe_size;
    struct sigaction sa;
    sa.sa_handler = alarm_hndlr; sigemptyset(&sa.sa_mask); sa.sa_flags = 0;
    sigaction(SIGALRM, &sa, NULL);
    pipe(fd);
    pipe_size = fpathconf(fd[0], _PC_PIPE_BUF);
    printf("PIPE_BUF: %d\n", pipe_size);
    while (1) {
        alarm(3);
        write(fd[1], "a", 1);
        alarm(0); /* reset alarm */
        count = count + 1;
        if ((count % 1024) == 0) printf("%d bytes in pipe\n", count);
    }
}
```

```
void sigpipe_hndlr(int signum) {
    printf("signal SIGPIPE reçu\n");
}

int main () {
    int fd[2];
    int nbytes;

    struct sigaction sa; sa.sa_handler = sigpipe_hndlr; sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGPIPE, &sa, NULL);

    pipe(fd);
    close(fd[0]);

    if (-1 == (nbytes = write(fd[1], "a", 1))) { perror("write fd[1]"); }
    else { printf("%d bytes written\n", nbytes); }
    exit(EXIT_SUCCESS);
}
```

Fermeture d'un tube

- ▶ Primitive `close`
 - ▷ `int close(int fd)`
- ▶ Fermeture d'un tube
 - ▷ Libère le descripteur
 - ▷ Pour le descripteur d'écriture, agit comme une fin de fichier
- ▶ Fermer tous les descripteurs non utilisés
 - ▷ évite les situations de blocage

Accès non bloquants

- ▶ Par défaut, les lectures et écritures dans un tube sont bloquantes
- ▶ Possibilité d'accès non bloquants
 - ▷ Positionnement du drapeau `O_NONBLOCK`
 - ▷ Exemple pour les écritures

```
fcntl(fd[1], F_SETFL, fcntl(fd[1], F_GETFL) | O_NONBLOCK);
```
- ▶ Écriture non bloquante
 - ▷ `write` retourne `-1` si on ne peut pas écrire de manière atomique les `nbytes` caractères demandés
 - ▷ positionne `errno` à `EAGAIN`
- ▶ Lecture non bloquante
 - ▷ `read` retourne `-1` si le tube est vide et le nombre d'écrivains est non nul
 - ▷ positionne `errno` à `EAGAIN`

Les fonctions `popen` et `pclose`

- ▶ Primitives de la bibliothèque standard

- ▷ `#include <stdio.h>`

- `FILE *popen(const char *command, const char *type);`

- `int pclose(FILE *stream);`

- ▶ Création d'un tube, puis d'un processus fils

- ▷ le processus fils exécute la commande `command` (via un appel à `system`)

- ▷ la sortie ou l'entrée standard de ce processus est redirigée vers/depuis le tube en fonction de `type`

- ▷ retourne le descripteur du tube permettant d'accéder à cette sortie ou entrée standard

- ▶ Argument `type`

- ▷ `r` : le descripteur est un accès en lecture depuis lequel le fils écrit

- ▷ `w` : le descripteur est un accès en écriture depuis lequel le fils lit

Tubes nommés

- ▶ Inconvénient des tubes anonymes
 - ▷ les processus communicants doivent avoir un processus parent commun... qui a créé le tube
 - ▷ pas de rémanence du tube : détruit, au plus tard, à la terminaison des processus
- ▶ Tube nommé ou fifo
 - ▷ même comportement en lecture/écriture que les tubes anonymes
 - ▷ rémanence du tube
 - ▷ liaison dans le système de fichier : nom du tube

Création d'un tube nommé

► Commande mkfifo

- ▷ `mkfifo file...`

- ▷ crée une entrée dans le système de fichier qui est un tube

- ▷ Exemple :

 - > `mkfifo /tmp/tube`

 - > `ls -l /tmp/tube`

 - `prw-r--r-- 1 louis wheel 0 Mar 20 20:02 /tmp/tube`

► Primitive mkfifo

- ▷ `#include <sys/types.h>`

 - `#include <sys/stat.h>`

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

- ▷ Création d'un i-nœud de type `S_IFIFO`

► Exemple :

 - 1--> `cat > /tmp/tube`

 - 2--> `cat < /tmp/tube`

Ouverture d'un tube nommé

- ▶ Primitive habituelle d'ouverture open
 - ▷ `int open(const char *path, int oflag, ...);`
- ▶ Impératif de choisir un mode lecture ou (exclusif) écriture : `O_RDONLY` ou `O_WRONLY`
 - ▷ pas de mode `O_RDWR`
- ▶ Ouverture bloquante par défaut (particularité des tubes)
 - ▷ l'ouverture en lecture est bloquante
 - ▷ s'il n'y a aucun écrivain, et
 - ▷ aucun processus bloqué en ouverture en écriture
 - ▷ ouverture en écriture bloquante
 - ▷ s'il n'y a aucun lecteur, et
 - ▷ aucun processus bloqué en ouverture en lecture
 - ▷ donc synchronisation des ouvertures en lecture et écriture
 - ▷ attention aux interblocages en cas d'ouverture de plusieurs tubes

Ouverture d'un tube nommé

- ▶ Possibilité d'ouverture non bloquante
 - ▷ option `O_NONBLOCK` du mode d'ouverture
 - ▷ Exemple : `fd = open("tube", O_WRONLY | O_NONBLOCK);`
- ▶ Ouverture non bloquante en lecture
 - ▷ succès même s'il y a aucun écrivain
 - ▷ les lectures seront bloquantes
- ▶ Ouverture non bloquante en écriture
 - ▷ échec s'il n'y a aucun lecteur
 - ▷ `open` retourne `-1` et `errno` est positionnée à `ENXIO`
 - ▷ si l'ouverture réussit, les écritures sont non bloquantes

Exemple :

(ff-svr.c/ff-clt.c)

- ▶ Application client/serveur (minimale !)
 - ▷ un serveur lit sur un tube nommé des messages de 64 caractères et les affiche sur la sortie standard
 - ▷ des clients écrivent des messages de 64 caractères dans ce tube nommé

```
#define MSIZE 64
#define FIFO "/tmp/ff-fifo"
int main () {
    int fd; int status; int nbytes;
    char mbuf[MSIZE];
    status = mkfifo(FIFO, S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP);
    assert(status != -1 || errno == EEXIST);
    fd = open(FIFO, O_RDONLY);
    assert (fd >=0);
    while (1) {
        nbytes = read(fd, mbuf, MSIZE);
        assert(nbytes >= 0);
        if (nbytes > 0) printf("message: %s\n", mbuf);
    }
    exit(EXIT_SUCCESS); }
```

Exemple :

(ff-svr.c/ff-clt.c)

```
#define MSIZE 64
#define FIFO "/tmp/ff-fifo"

int main (int argc, char *argv[]) {
    int fd;
    char mbuf[MSIZE];
    int i;
    fd = open(FIFO, O_WRONLY);
    assert (fd >=0);
    for(i=1; i< argc; i++) {
        strncpy(mbuf, argv[i], MSIZE-1);
        mbuf[MSIZE-1] = '\0';
        write(fd, mbuf, MSIZE);
    }
    exit(EXIT_SUCCESS);
}
```