

Introduction aux Systèmes et Réseaux

TD n°3 : Flots d'entrées-sorties et communication par tubes

L'objectif de ce TD¹ est d'illustrer l'accès à des fichiers, la manipulation de flots d'entrées-sorties et la communication par tubes dans un système Unix.

1 Redirection des entrées/sorties d'un processus

Chaque processus manipule trois flots d'entrées/sorties : l'entrée, la sortie et la sortie d'erreur. Par défaut, ceux-ci sont positionnés pour correspondre respectivement aux flots d'entrée standard (le clavier), le flot de sortie standard (l'écran) et la sortie d'erreur standard (l'écran également).

Les flots manipulés par un processus peuvent être modifiés soit en ligne de commande, soit de manière programmatique. Dans la suite de cette section, nous considérons la ligne de commande.

1.1 Les commandes `>` et `>>` permettent de rediriger la sortie d'un processus vers un fichier.

Que fait `ls -l > toto` ?

Que fait `cat > toto` ?

Que fait `ls -l » toto` ?

1.2 La commande `<` permet d'envoyer le contenu d'un fichier en entrée d'un processus.

Que fait `sort < numbers.txt` ?

Que fait `cat < toto` ?

Quelle est la différence par rapport à `cat toto` ?

1.3 Enfin, en utilisant un *pipe* (tube en français) `|`, vous pouvez rediriger la sortie d'un processus vers l'entrée d'un autre processus.

Que fait `ls -l | less` ?

Que fait `find . -type f -print | wc -l` ?

2 Descripteurs de fichiers

Un descripteur de fichier est créé lors de l'ouverture d'un fichier par un processus. Un descripteur est identifié par un entier (numéro de descripteur), qui est une entrée dans la

1. Certains exemples sont empruntés (avec des adaptations) aux ouvrages suivants : R. E. Bryant, D. O'Hallaron. *Computer Systems: a Programmer's Perspective*, Prentice Hall, 2003, et W. R. Stevens, *UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications*, Prentice Hall, 1999.

table des descripteurs du processus. Dans la suite, lorsqu'il n'y a pas d'ambiguïté, nous utilisons aussi le terme de *descripteur* pour désigner les numéros de descripteurs. Les trois descripteurs 0, 1, et 2 sont respectivement attribués, dans chaque processus, au flot d'entrée standard `stdin`, au flot de sortie standard `stdout` et au flot de sortie d'erreur `stderr`.

Les descripteurs sont utilisés dans les fonctions de manipulation de fichiers de l'interface système. Les principales fonctions sont :

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);

#include <unistd.h>
int close(int fildes);

#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

La fonction `open` ouvre un fichier et retourne un descripteur de fichier au processus appelant. Les flags définissent le mode de manipulation autorisé des fichiers, quelques exemples sont

<code>O_RDONLY</code>	open for reading only\\
<code>O_WRONLY</code>	open for writing only\\
<code>O_RDWR</code>	open for reading and writing
<code>O_NONBLOCK</code>	do not block on open or for data to become available
<code>O_APPEND</code>	append on each write
<code>O_CREAT</code>	create file if it does not exist
<code>O_TRUNC</code>	truncate size to 0
<code>O_EXCL</code>	error if <code>O_CREAT</code> and the file exists
<code>O_SHLOCK</code>	atomically obtain a shared lock
<code>O_EXLOCK</code>	atomically obtain an exclusive lock

Ce même descripteur est utilisé pour fermer le fichier (indiquer que l'on ne l'utilise plus), ainsi que pour lire depuis ou écrire dans le fichier.

2.1 Écrire un programme `mycopy.c` qui utilise les primitives `read()` et `write()` pour copier un fichier quelconque, de taille arbitraire, vers `stdout`. Le nom du fichier doit être passé en paramètre de la commande `mycopy`

2.2 Qu'affiche le programme ci-après ?

```
1  #include "csapp.h"
2
3  int main(){
4      int fd1, fd2;
5      fd1 = Open("toto.txt", O_RDONLY, 0);
6      Close(fd1);
7      fd2= Open("titi.txt", O_RDONLY, 0);
8      printf("fd2 = %d\n", fd2);
9      exit(0);
10 }
```

2.3 Qu'affiche le programme ci-après ?

```
1  #include "csapp.h"
2
3  int main(){
4      int fd1, fd2;
5      fd1 = Open("toto.txt", O_RDONLY, 0);
6      fd2 = Open("tata.txt", O_RDONLY, 0);
7      Close(fd2);
8      fd2= Open("titi.txt", O_RDONLY, 0);
9      printf("fd2 = %d\n", fd2);
10     exit(0);
11 }
```

2.4 On suppose que le fichier `toto.txt` contient dans cet ordre les 6 caractères ASCII "lambda". Qu'affiche le programme ci-après ?

```
1  #include "csapp.h"
2
3  int main(){
4      int fd1, fd2; char c;
5      fd1 = Open("toto.txt", O_RDONLY, 0);
6      fd2 = Open("toto.txt", O_RDONLY, 0);
7      Read(fd1, &c, 1);
8      Read(fd2, &c, 1);
9      printf("c = %c\n", c);
10     exit(0);
11 }
```

2.5 Un processus fils hérite des descripteurs des fichiers ouverts de son père. Comme dans la question précédente, le fichier `toto.txt` contient les 6 caractères ASCII "lambda". Qu'affiche le programme ci-après ?

```
1  #include "csapp.h"
2
3  int main(){
4      int fd; char c;
5      fd = Open("toto.txt", O_RDONLY, 0);
```

```

6         if (Fork() == 0) {
7             Read(fd, &c, 1);
8             exit(0);
9         }
10        Wait(NULL);
11        Read(fd, &c, 1);
12        printf("c = %c\n", c);
13        exit(0);
14    }

```

2.6 La primitive `dup()` sert à dupliquer un descripteur de fichier en utilisant le plus petit numéro de descripteur non utilisé. Elle prend en argument un descripteur de fichier ouvert, et retourne le descripteur de la copie. Cette primitive sert à rediriger un flot d’entrée ou de sortie.

La primitive `dup2()` est une variante de `dup()`, qui prend deux arguments : le descripteur du fichier à dupliquer, et le nouveau descripteur (qui recevra la copie du premier). Si ce nouveau descripteur est occupé, il sera fermé au préalable.

Le fichier `toto.txt` contient toujours les 6 caractères ASCII “`lambda`”. Qu’affiche le programme ci-après ?

```

1  #include "csapp.h"
2
3  int main(){
4      int fd1, fd2; char c;
5      fd1 = Open("toto.txt", O_RDONLY, 0);
6      fd2= Open("toto.txt", O_RDONLY, 0);
7      Read(fd2, &c, 1);
8      Dup2(fd2, fd1);
9      Read(fd1, &c, 1);
10     printf("c = %c\n", c);
11     exit(0);
12 }

```

3 Retour à la redirection d’entrées-sorties

3.1 Certains shells (dont `bash` et `ksh`) offrent la syntaxe suivante pour rediriger le descripteur désigné par `fd1` vers le fichier associé au descripteur `fd2` :

```
fd1 >& fd2
```

Quelle est la différence d’effet entre les deux commandes suivantes ?

```
./a.out > outfile 2>&1
```

```
./a.out 2>&1 > outfile
```

4 Manipulation de tubes

4.1 Voici le programme qui établit un canal de communication entre processus père et processus fils. Commentez son fonctionnement.

```
#include "csapp.h"
```

```

#define BUFSIZE 10
int main(void) {
    char bufin[BUFSIZE] = "empty";
    char bufout[BUFSIZE] = "hello";
    int bytesin, bytesout;    pid_t childpid;
    int fd[2];

    pipe(fd);
    bytesin = strlen(bufin);
    childpid = Fork();
    if (childpid != 0) {      /* père */
        Close(fd[0]); bytesout = write(fd[1], bufout, strlen(bufout)+1);
        printf("[%d]: wrote %d bytes\n", getpid(), bytesout);
    } else {                  /* fils */
        Close(fd[1]); bytesin = read(fd[0], bufin, BUFSIZE);
        printf("[%d]: read %d bytes, my bufin is {%s} \n »,
                getpid(), bytesin, bufin);
    }
    exit(0);
}

```

4.1 Modifiez le programme pour qu'il établisse une communication dans les deux sens.

4.2 Ecrire un programme qui réalise la commande `ls -l > toto`.