TP UNIX N°1
BUT - INFO 2°Année

1_ FIFO

Une file premier entré, premier sorti (First-in, First-out, FIFO) est un tube qui dispose d'un nom dans le système de fichiers. Tout processus peut ouvrir ou fermer la FIFO ; les processus raccordés aux extrémités du tube n'ont pas à avoir de lien de parenté (contrairement au pipe). Les FIFO sont également appelés canaux nommés.

Vous pouvez créer une FIFO via la commande *mkfifo*. Indiquez l'emplacement où elle doit être créée sur la ligne de commande. Par exemple, créez une FIFO dans /tmp/fifo en invoquant ces commandes :

```
$ mkfifo /tmp/fifo
$ ls -l /tmp/fifo
prw-rw-rw-  1  ab  users  0  nov 16 14:00  /tmp/fifo
```

Le premier caractère affiché par ls est p ce qui indique que le fichier est en fait une FIFO (canal nommé, named pipe).

Dans une fenêtre, lisez des données depuis la FIFO en invoquant cette commande :

```
$ cat < /tmp/fifo
```

Dans une deuxième fenêtre, écrivez dans la FIFO en invoquant cela :

```
$ cat > /tmp/fifo
```

Puis, saisissez du texte. À chaque fois que vous appuyez sur Entrée, la ligne de texte est envoyé dans la FIFO et apparaît dans la première fenêtre. Fermez la FIFO en appuyant sur Ctrl+D dans la seconde fenêtre. Supprimez la FIFO avec cette commande :

```
$ rm /tmp/fifo
```

Créer une FIFO

Pour créer une FIFO par programmation, utilisez la fonction mkfifo. Le premier argument est l'emplacement où créer la FIFO ; le second paramètre spécifie les permissions du propriétaire du tube, de son groupe et des autres utilisateurs.

Accéder à une FIFO

L'accès à une FIFO se fait de la même façon que pour un fichier ordinaire. Pour communiquer via une FIFO, un programme doit l'ouvrir en écriture. Il est possible d'utiliser des fonction d'E/S de bas niveau (open, write, read , close , etc) ou des fonctions d'E/S de la bibliothèque C (fopen, fprintf, fscanf, fclose , etc .).

Par exemple, pour écrire un tampon de données dans une FIFO en utilisant des routines de bas niveau, vous pourriez procéder comme suit :

```
int fd = open ( fifo_path , O_WRONLY ) ;
write (fd , data , data_length );
close ( fd ) ;
```

Pour lire une chaîne depuis la FIFO en utilisant les fonctions d'E/S de la bibliothèque C, vous pourriez utiliser ce code :

```
FILE * fifo = fopen ( fifo_path , "r");
fscanf ( fifo , "%s " , buffer ) ;
fclose ( fifo ) ;
```

En vous aidant du manuel (commande : man), créez deux programme C permettant d'échanger des messages contenus dans un fichier (donné en paramètre à l'un des deux processus) à travers une FIFO. Le premier processus créé la FIFO et ouvre le fichier en lecture pour lire les données et les écrire dans la FIFO. Le second, lit dans la FIFO et affiche à l'écran les données.

2_ Les queues de message

Avec la volonté d'uniformiser les interfaces de programmation des systèmes Unix par l'intermédiaire de la norme Posix, sont apparues de nouvelles files de messages, dont l'interface est simple, et dont les descripteurs sont proches de ceux des fichiers. Les *message queues* Posix n'ont été intégrées qu'assez tardivement dans Linux (dans la version 2.6.10).

Les appels-système sont :

```
#include <mqueue.h>

mqd_t   mq_open    (const char * nom, int flags, mode_t mode, struct mq_attr * attr);
int     mq_send    (mqd_t mq, const char * msg, size_t lg, unsigned int prio);
ssize_t mq_receive (mqd_t mq, char * msg, size_t lg, unsigned int * prio);
int     mq_close   (mqd_t mq);
int     mq_unlink  (const char * nom);
```

On peut se reporter aux pages de manuels respectives de ces fonctions pour avoir plus de détails.

En utilisant ces appels système écrivez deux programmes C échangeant dix messages grâce à ce mécanisme.

Pour compiler : *gcc source.c -o destination -lrt*

3_ Le système de fichiers /proc

Essayez d'invoquer la commande mount sans argument, elle liste les systèmes de fichiers actuellement montés sur votre système GNU/Linux. Vous apercevrez une ligne de ce type :

proc on /proc type proc (rw)

A l'aide du man 5 proc, explorez ce répertoire, découvrez les informations de votre système et établissez une cartographie des processus en cours d'exécution.

Que contiennent les fichiers **cpuinfo**, **devices**, **dma**, **filesystems**, **interrupts**, **partitions**, **meminfo**, **modules**, **uptime**, **version**.

A quoi correspondent les répertoires dont le nom est un numéro, listez leur contenu, comparez cette liste de répertoires avec le résultat de la commande *ps -x*.
que contiennent les fichiers : **cmdline, cwm, environ, exe, fd, maps**

Recherchez un processus en cours d'exécution tel que votre navigateur ou un éditeur de texte à l'aide de la commande :

ps -efa | grep monprog
puis ls -l /proc/Pid