1 ère année Master : RSD **Année : 2021/2022**

Cours : Programmation réseaux et systèmes

Chapitre : Communication entre processus

Introduction

Dans ce chapitre, nous voyons comment faire communiquer des processus entre eux par des tubes. Pour le moment, les processus qui communiquent doivent être des processus de la même machine. Cependant, le principe de communication avec les fonctions read et write sera réutilisé par la suite dans la programmation réseau, qui permet de faire communiquer des processus se trouvant sur des stations de travail distinctes.

La fonction pipe

Un tube de communication est un tuyau (en anglais pipe) dans lequel un processus peut écrire des données et un autre processus peut lire.

On crée un tube par un appel à la fonction pipe, déclarée dans unistd.h : int pipe(int descripteur[2]);

La fonction renvoie 0 si elle réussit, et elle crée alors un nouveau tube.

La fonction pipe

La fonction pipe remplit le tableau descripteur passé en paramètre, avec :

• descripteur[0] désigne la sortie du tube (dans laquelle on peut lire des données) ;

• descripteur[1] désigne l'entrée du tube (dans laquelle on peut écrire des données) ;

Le principe est qu'un processus va écrire dans descripteur[1] et qu'un autre processus va lire les mêmes données dans descripteur[0].

La fonction pipe

Le problème est qu'on ne crée le tube dans un seul processus, et un autre processus ne peut pas deviner les valeurs du tableau descripteur. Pour faire communiquer plusieurs processus entre eux, il faut appeler la fonction pipe avant d'appeler la fonction fork. Ensuite, le processus père et le processus fils auront les mêmes descripteurs de tubes, et pourront donc communiquer entre eux. De plus, un tube ne permet de communiquer que dans un seul sens. Si l'on souhaite que les processus communiquent dans les deux sens, il faut créer deux pipes.

Les fonctions du tube

pour écrire dans un tube, on utilise la fonction write:

ssize_t write(int descripteur1, const void *bloc, size_t taille);

Le descripteur doit correspondre à l'entrée d'un tube.

La taille est le nombre d'octets qu'on souhaite écrire, et le bloc est un pointeur vers la mémoire contenant ces octets.

Pour lire dans un tube, on utilise la fonction read:

ssize_t read(int descripteur0, void *bloc, size_t taille);

Les fonctions du tube

Le descripteur doit correspondre à la sortie d'un tube, le bloc pointe vers la mémoire destinée à recevoir les octets, et la taille donne le nombre d'octets qu'on souhaite lire. La fonction renvoie le nombre d'octets effectivement lus. Si cette valeur est inférieure à taille, c'est qu'une erreur s'est produite en cours de lecture (par exemple la fermeture de l'entrée du tube suite à la terminaison du processus qui écrit). Dans la pratique, on peut transmettre un buffer qui a une taille fixe (256 octets dans l'exemple ci-dessous). L'essentiel est qu'il y ait exactement le même nombre d'octets en lecture et en écriture de part et d'autre du pipe.

Exemple

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#define BUFFER_SIZE 256
int main(void)
{
pid_t pid_fils;
int tube[2];
unsigned char bufferR[256], bufferW[256];
puts("Création d'un tube");
if (pipe(tube) != 0) {/* pipe */
    { fprintf(stderr, "Erreur dans pipe \n");
exit(1); }
```

Exemple

```
pid_fils = fork(); {/* fork */
if (pid_fils == -1)
      {  fprintf(stderr, "Erreur dans fork\n");
exit(1);
}
if (pid_fils == 0) {/* processus fils */
     {  printf("Fermeture entre dans le fils (pid = %d)\n", getpid());
close(tube[1]);
read(tube[0], bufferR, BUFFER_SIZE);
printf("Le fils (%d) a lu : %s\n", getpid(), bufferR);
}
else {/* processus père */
{
printf("Fermeture sortie dans le père (pid = %d)\n", getpid());}
```

Exemple

```
pid_fils = fork(); {/* fork */
if (pid_fils == -1)
      {  fprintf(stderr, "Erreur dans fork\n");
exit(1);
}
if (pid_fils == 0) {/* processus fils */
     {  printf("Fermeture entr e dans le fils (pid = %d)\n", getpid());
close(tube[1]);
read(tube[0], bufferR, BUFFER_SIZE);
printf("Le fils (%d) a lu : %s\n", getpid(), bufferR);
}
else {/* processus père */
{
printf("Fermeture sortie dans le pere (pid = %d)\n", getpid());}
```

Tubes nommés

On peut faire communiquer deux processus à travers un tube nommé. Le tube nommé passe par un fichier sur le disque. L'intérêt est que les deux processus n'ont pas besoin d'avoir un lien de parenté. Pour créer un tube nommé, on utilise la fonction mkfifo de la bibliothèque sys/stat.h.

Dans le code suivant, le premier programme transmet le mot "bonjour" au deuxième programme. Les deux programmes n'ont pas besoin d'être liés par un lien de parenté.

Exemple

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
int main()
{
int fd;
FILE *fp;
char *nomfich="/tmp/test.txt"; /* nom du fichier */
char *nomfich="/tmp/test.txt"; /* nom du fichier */
if(mkfifo(nomfich, 0644) != 0) /* création du fichier */
{
perror("Problème de création du noeud de tube");
exit(1);
}
fd = open(nomfich, O_WRONLY); /* ouverture en écriture */
fp=fdopen(fd, "w"); /*ouverture du flot */
fprintf(fp, " bonjour \n"); /* écriture dans le flot */
unlink(nomfich); /* fermeture du tube */
return 0;
}
```

Exemple

La fonction mkfifo prend en paramètre, outre le chemin vers le fichier, le masque des permissions (lecture, écriture) sur la structure fifo

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
int main()
{
int fd;
FILE *fp;
char *nomfich="/tmp/test.txt", chaine[50];
fd = open(nomfich, O_RDONLY); /* ouverture du tube *
fp=fdopen(fd, "r"); /* ouverture du flot */
fscanf(fp, "%s", chaine); /* lecture dans le flot */
puts(chaine); /* affichage */
unlink(nomfich); /* fermeture du flot */
return 0;
}
```