TP Tube (Partie 1)

# Introduction

Les tubes permettent de transmettre des informations entre processus. Vous avez déjà utilisé les tubes dans un shell grâce au pipe: |

Le symbole | indique que vous souhaitez créer une laison entre la sortie standard du processus à gauche et l’entrée standard du processus à droite.

## Mise en application

Lancer la commande suivante afin d’utiliser le programme bc et évaluer l’expression affichée par echo:

echo “3+4” | bc

Curiosité: nous pouvons réaliser des fonctions élémentaires en utilisant la commande de substitution sed

plus=“x+y”  
echo $plus | sed 's/x/5/' | sed 's/y/10/' | bc

# Les tubes en C:

Créer des “tubes” avec le langage de programmation C est un peu plus compliqué que nos exemples de shell.

Pour créer un tube simple avec C, nous utilisons l’appel système pipe().

Un seul argument est requis, un tableau de deux entiers.

En cas de succès, le tableau contiendra deux nouveaux descripteurs de fichier à utiliser pour le tube.

## Mise en application

#include <stdio.h>  
#include <unistd.h>  
#include <sys/types.h>  
main()  
{  
int fd[2];  
pipe(fd);  
//...  
}

Une fois le tube créé nous allons lancer un processus fils avec fork().

Pour rappel, le processus fils hérite des descripteurs des fichiers déjà ouverts par le père.

## Mise en application

#include <stdio.h>  
#include <unistd.h>  
#include <sys/types.h>  
main()  
{  
int fd[2];  
pid\_t childpid;  
pipe(fd);  
  
if((childpid = fork()) == -1)  
{  
perror("fork");  
exit(1);  
}  
}  
//...

Si le processus parent veut recevoir des données du processus fils, il doit fermer fd[1] et le fils doit fermer fd[0].

Si le parent veut envoyer des données au fils, il doit fermer fd[0] et le fils doit fermer fd[1].

Il est important de bien fermer les bonnes extrémités pour donner une direction au flux de données du tube. Autrement, le EOF ne sera jamais renvoyé si les descripteurs inutilisables du tube ne sont pas fermés.

## Mise en application

#include <stdio.h>  
#include <unistd.h>  
#include <sys/types.h>  
main()  
{  
  
int fd[2];  
pid\_t childpid;  
pipe(fd);  
if((childpid = fork()) == -1)  
{  
 perror("fork");  
 exit(1);  
}  
  
if(childpid == 0)  
{  
/\* fermer le 0 \*/  
 close(fd[0]);  
}  
else  
{  
 /\* Fermer le 1 \*/  
 close(fd[1]);  
}  
}

Une fois le tube correctement créé, les descripteurs de fichier sont considérés comme des descripteurs de fichiers normaux.

#include <stdio.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <string.h>  
int main(void)  
{  
int childpid, pid\_t, fd[2], nbytes;  
char string[] = "Hello, world!\n";  
char readbuffer[80];  
  
pipe(fd);  
  
if((childpid = fork()) == -1)  
{  
perror("fork");  
exit(1);  
}  
  
if(childpid == 0)  
{  
 close(fd[0]);  
 write(fd[1], string, (strlen(string)+1));  
 exit(0);  
}  
else  
{  
 close(fd[1]);  
 nbytes = read(fd[0], readbuffer, sizeof(readbuffer));  
 printf("Message recu: %s", readbuffer);  
}  
return(0);  
}

# Programmation avancée

Il est possible de dupliquer les descripteurs du processus fils sur l’entrée ou la sortie standard.

Le processus fils peut alors faire un exec() et lancer un autre programme, qui va hériter de ces flux standards.

Pour rappel, voici la description de l’appel système de dup2():

int dup2(int anciendesc, int nouveaudesc )

La fonction dup2() ferme anciendesc et affecte nouveaudesc comme descripteur au même fichier.

## Mise en application

#include <stdio.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <string.h>  
#include <stdlib.h>  
  
int main(void)  
{  
int fd[2], nbytes, pid\_t, childpid;  
char string[] = "4\n5\n3\n2\n1\n"; //observer l ordre des lignes  
  
pipe(fd);  
  
if((childpid = fork()) == -1)  
{  
perror("fork");  
exit(1);  
}  
  
if(childpid == 0)  
{  
close(fd[1]);  
dup2(fd[0],0);  
execlp("sort", "sort", NULL);  
exit(0);  
}  
else  
{  
close(fd[0]);  
nbytes = write(fd[1], string, strlen(string));  
}  
return(0);  
}