Marion ARMENGO & Arsène LAPOSTOLET

Système d'Exploitation : TP4 : IPC

A propos

Nous utilisons CMake pour compiler notre projet. Nous avons organisé notre code en créant à l'aide de ce dernier un fichier exécutable pour chaque exercice.

Vous pouvez compiler le projet grace à cmake en utilisant le fichier `CMakeList.txt` fourni.

Question 1

En faisant appel à `fileno` sur `stdin`, `stdout` et `stderr` nous avons obtenu les numéros de descripteur de fichier suivants :

Les numéros des flux standards sont :

- (Input) Stdin: 0
- (Output) Stdout: 1
- (Error) Stderr: 2

Question 2

La commande shell équivalente pour la redirection est :

```
entree > fichier_de_redirection
```

Question 3

La recherche qui suit à été effectuée grâce aux pages du manuel linux.

Open

```
int open(const char *pathname, int flags);
```

Un appel à `open()` créé une liaison entre un fichier et un descripteur de fichier. Un descripteur de fichier est une référence à l'une des entrées du fichier, cette dernière n'est pas affectée si le chemin du fichier ouvert est ultérieurement supprimé ou modifié pour se référer à un fichier différent. Le paramètre flags doit inclure l'un des mode d'accès suivants : `O_RDONLY`,`O_WRONLY` ou `O_RDWR`. Ceux-ci réclament respectivement l'ouverture du fichier en lecture seule, écriture seule, ou lecture-écriture. `open()` renvoie le nouveau descripteur de fichier s'ils réussissent,ou -1 s'il échoue, auquel cas errno contient le code d'erreur.

Read

```
ssize_t read(int fd, void *buf, size_t count);
```

`read()` lit jusqu'au nombre count d'octets depuis le descripteur de fichier fd dans le tampon pointé par buf. Si count vaut zéro, `read()` renvoie zéro. Si count est supérieur à `SSIZE_MAX`, le résultat est indéfini. `read()` renvoie -1 s'il échoue, auquel cas errno contient le code d'erreur, et la position de la tête de lecture est indéfinie. Sinon, `read()` renvoie le nombre d'octets lus (0 en fin de fichier), et avance la tête de lecture de ce nombre. `ssize_t` est identique à `size_t`, mais correspond à un type signé. ssize_t peut représenter le nombre -1, qui est renvoyé par plusieurs appels système et fonctions de bibliothèque pour indiquer une erreur.

Write

```
ssize_t write(int fd, const void *buf, size_t count);
```

`write()` écrit dans le fichier associé au descripteur fd depuis le tampon pointé par buf le nombre count d'octets. `write()` renvoie le nombre d'octets écrits (0 signifiant aucune écriture), ou -1 s'il échoue, auquel cas errno contient le code d'erreur. Si count vaut zéro, et si fd est associé à un fichier normal, `write()` peut renvoyer un code d'erreur si l'une des erreurs ci-dessous est détectée. S'il n'y a aucune erreur, 0 sera renvoyé. Si count vaut zéro, et si fd est associé à autre chose qu'un fichier ordinaire, les résultats sont indéfinis.

Close

```
int close(int fd);
```

`close()` ferme le descripteur de fichier, de manière à ce qu'il ne référence plus aucun fichier, et puisse être réutilisé. S'il réussit, `close()` renvoie zéro. S'il échoue, il renvoie -1 et errno contient le code d'erreur.

Question 4

Utilisation de `pipe()` pour la communication entre les deux processus pères et fils. Dans notre programme, chaque processus ferme l'extrémité du descripteur de fichier (notre tube) inutilisée selon si le processus utilise le pipe en lecture ou en écriture.

Nous utilisons deux tubes :

- Un tube permettant au processus fils de communiquer ses hypothèses de prix au processus père.
- Un tube permettant au processus fils de communiquer au processus fils si il a gagné la partie ou pas.

Question 5

Notre programme prend en argument en ligne de commande, ce argument est passé lettre par lettre par un processus père à un processus fils par l'intermédiaire d'un pipe.

Question 6

Dans ce programme, nous avons utilisé la fonction `dup2` de la librairie unistd.h afin de pouvoir dupliquer les flux d'entrée et de sorties (selon notre besoin) pour pouvoir communiquer (via un tube) le résultat d'une commande réalisée par le processus père vers le processus fils afin que ce dernier puisse exploiter ce résultat.

Question 7

Nous avons tenté de résoudre ce problème : nous avons essayé de réaliser un code qui permettrait de faire en sorte que chaque processus fils puisse s'exécuter dans le bon ordre afin de pouvoir exploiter le résultat du précédent au moment d'exécuter sa commande. Pour ce faire nous avons essaué d'utiliser un tableau de N tubes qui permettrait à chaque fils de transmettre les résultats de sa commande au suivant, la lecture sur le tube étant bloquante.

Nous ne somme cependant pas parvenu à faire fonctionner un tel programme.

Nous avons essayé un certain nombre de démarches mais nous ne sommes pas parvenu à un résultat concluant. Nous avons fourni notre ébauche la plus avancée.