UNIVERSITÉ PARIS SUD
IUT D'ORSAY

DUT Informatique
M3101 Système S3
2015 / 2016

Travaux Pratiques n° 2 : Fichiers et tubes

Nom(s) :
Groupe :
Date :

Objectifs : maîtriser les primitives de manipulation de fichiers Unix. Savoir utiliser les tubes (pipes) Unix pour la communication entre les processus.

Exercice 1 : Sens dessus dessous. Dans cet exercice on écrit un programme C qui reçoit un nom de fichier par la ligne de commande, et affiche sur la sortie standard son contenu, en changeant toute lettre minuscule en majuscule et vice versa. Utiliser les fonctions `isupper()`, `islower()`, `toupper()` et `tolower()`. Voir les pages de `man` correspondantes pour les informations sur ces fonctions.

Exercice 2 : Palimpseste. Modifiez le programme de l'exercice précédent pour écrire le texte transformé dans le fichier traité, en écrasant les caractères originaux. Pensez à préparer plusieurs fichiers texte pour faire des tests : appliquer ce programme à son propre code source (ou à tout autre fichier que vous voulez garder) peut être une mauvaise idée.

Notes de cours : Exécution de commandes. La primitive système

```
#include <unistd.h>
int execl(const char *chemin, const char *arg0, ..., NULL);
```

exécute le fichier avec le nom `chemin` en lui passant les paramètres suivants (`arg0`, ...) comme sa ligne de commande. La liste de paramètres doit se terminer par le pointeur `NULL`. Par exemple, pour lancer la commande « `ls -l /etc` », on peut écrire `execl("/bin/ls","ls","-l","/etc",NULL)`. Afin de ne pas spécifier le chemin complet de l'exécutable, vous pouvez utiliser la primitive `execlp()` qui cherchera le fichier dans les répertoires énumérés dans la variable d'environnement `PATH`, comme le fait shell : `execlp("ls","ls","-l","/etc",NULL)`.

Les primitives `execl()` et `execlp()` (ainsi que les autres primitives de la famille `exec()`, voir la page `man 3 exec`), exécutent le nouveau programme *dans le processus courant*. Le code actuel du processus et le contenu de sa mémoire sont remplacés par le code et la mémoire du programme exécuté. Pour lancer un programme en parallèle avec le processus courant, il faut appeler `fork()` et exécuter le programme dans le processus fils :

```
if (fork() == 0) { // code du fils
  execlp("xterm", "xterm", "-T", "nouveau terminal", NULL);
  perror("error exec()");
  return 1;
} // le père continue
```

Les primitives `exec()` ne retournent dans le programme appelant qu'en cas d'erreur.

Par défaut, les primitives `exec()` préservent les fichiers ouverts. En particulier, les fichiers standards d'entrée, de sortie et d'erreur seront les mêmes pour le nouveau programme que pour le processus appelant.

Exercice 3 : Exécution simple. Écrire un programme qui reçoit un nom de fichier par la ligne de commande, et affiche le nombre de lignes dans ce fichier. Utiliser la commande `wc` avec l'option `-l` pour compter les lignes.

Exercice 4 : Redirection simple. Modifiez le programme de l'exercice précédent pour écrire le résultat dans le fichier `result.txt` dans le répertoire courant. Si ce fichier n'existe pas, il doit être créé. Si ce fichier existe, son contenu doit être écrasé. Pour rediriger la sortie de la commande `wc` vers `result.txt`, utilisez le fait que `open()` attribue le plus petit numéro de descripteur disponible au nouveau fichier ouvert.

Exercice 5 : Estimation de π. Le but de ce problème est de construire une application client-serveur utilisant les tubes comme moyens de communication pour trouver une approximation du nombre π.

La méthode utilisée pour calculer une valeur approchée de π, appelée méthode de Monte-Carlo, est d'effectuer des tirages aléatoires de nombres réels x et y tels qu'ils soient compris entre 0 et 1. Si N est le nombre de tirages de couples x et y, et M le nombre de tirages tels que $x^2 + y^2 \leq 1$, alors on a $\pi \simeq (4M)/N$.

Pour ce problème, un processus appelé *pi* fera des appels à un autre processus appelé *tirage* pour lui demander un nombre aléatoire entre 0 et 1. Pour cela, *pi* enverra un message (un octet quelconque) à *tirage* au travers d'un tube. À la réception de ce message, *tirage* calculera le nombre et l'enverra au travers d'un deuxième tube. Quand *pi* aura fait une demande et reçu les réponses pour x et pour y, il calculera si $x^2 + y^2 \leq 1$. Il affichera après chaque demande de x et y son évaluation de π qui devrait être de plus en plus précise.

Dans ce schéma, qui est le client et qui est le serveur ?

Dans un premier temps on réalisera une application simplifiée où le processus fils *pi* enverra toutes les secondes à son processus père *tirage* une demande (un octet quelconque). Le processus père réagira en affichant un nombre réel entre 0 et 1 tiré aléatoirement. Rappels :

- Pour qu'un processus s'endorme une seconde, on fait appel à `sleep(1);`
- L'instruction `aleatoire = ((float)random()/RAND_MAX);` place dans la variable de type `float` `aleatoire` un nombre réel entre 0 et 1 choisi aléatoirement. Pensez à réinitialiser le générateur de nombres aléatoires avec un appel à `srandom()` (voir la feuille de TP1).
- Lorsqu'un processus n'utilise pas ou plus un des descripteurs d'un tube (lecture ou écriture), fermer ce descripteur est de la bonne programmation.

Réalisez cette première application, **vous joindrez le code source au compte-rendu**.

Complétez cette première application afin de réaliser le programme complet : créez un autre tube pour que *tirage* puisse communiquer son résultat à *pi* et implémentez dans *pi* les calculs nécessaires à l'estimation du nombre π. Affichez le résultat après chaque estimation.

Réalisez cette deuxième application, **vous joindrez le code source au compte-rendu**.