TP4 Gestion de fichiers - Système d'exploitation

Sereysethy Touch

30 octobre 2018

Nota :
— Que les exercices marqués * sont obligatoires, vous devez rendre vos travaux avant le **06/11/2018**.
— Pour chaque exercice, il faut toujours un fichier Makefile qui permet de compiler vos programmes et éventuellement un fichier `README.md` indiquant comment exécuter votre programme.
— S'il s'agit de documenter un programme, il faut simplement un fichier `README.md`.

Lecture, écriture d'un fichiers

Exercice 1*

Écrivez un programme qui se comporte comme le programme `cat`, c'est-à-dire qu'il lit un ou plusieurs fichiers passés en argument et puis affiche leurs contenus sur l'écran.

Exercice 2*

Écrivez un programme C qui copie un fichier source vers un fichier destination. Le fichier destination hérite les permissions de fichier original. Le programme lit les deux noms de fichiers passés en arguments.

Exercice 3*

Écrivez un programme qui ouvre un fichier texte dont le chemin est passé en argument et puis crée un processus fils. Le processus père lit 5 caractères et les affiche sur l'écran. Le processus fils lit 10 caractères et les affiche sur l'écran. Finalement répondez aux questions suivantes :

1. Qu'observez-vous les comportements de ces deux processus ?
2. Comment on peut résoudre le problème observé ?

Le fichier texte contient des messages suivants :

```
Bonjour tout le monde. La programmation système est intéressante.
```

Exercice 4*

Écrivez un programme qui, à partir de deux fichiers, crée un nouveau fichier. Le but de ce programme est d'entrelacer les lignes des deux fichiers donnés en arguments.

Exemple : `$ >./entrelacer fichier1 fichier2 fichier3`

Exercice 5* - dup et dup2

Écrivez un programme `lancer` qui crée un processus fils. Le processus fils exécute le programme qui est passé en argument au programme `lancer`. Le résultat de l'exécution du programme ou commande est stocké dans un fichier qui est aussi passé en argument.

Le programme `lancer` a un format suivant :

```
lancer commande [arg1, arg2, ...] fichier_sortie

Exemple:
$ ./lancer ls . /etc resultat
$ ./lancer cp resultat resultat2 sortie
```

Exercice 6* - manipulation des répertoires

Ecrire un programme qui dessine un arbre montrant la hiérarchie des relations parent-fils de **tous** les processus dans le système, jusqu'aux processus racines par exemple **init**. Pour chaque processus, le programme pourra afficher le PID, PPID et la commande étant exécutée. L'affichage peut être similaire à celui de la commande **pstree(1)**, mais il peut être plus simple.

Le parent de chaque processus peut être trouvé en inspectant la ligne PPID des fichiers **/proc/PID/status** dans le système.

Utilisez les fonctions `opendir`, `readdir`, `closedir`, etc. qui sont définies dans le fichier en-tête `<dirent.h>`.

Exemple :

```
sethy@dev:~$ pstree -A 1
systemd-+-acpid
        |-agetty
        |-apache2---5*[apache2]
        |-atd
        |-cron
        |-dbus-daemon
        |-dhclient
        |-dockerd-+-docker-containe---6*[{docker-containe}]
        |         `-9*[{dockerd}]
        |-exim4
        |-memcached---5*[{memcached}]
        |-mysqld_safe---mysqld---17*[{mysqld}]
        |-nmbd
        |-ntpd---ntpd
        |-proftpd
        |-redis-server---2*[{redis-server}]
        |-rpc.idmapd
        |-rpc.statd
```

```
|-rpcbind
|-rsyslogd-+-{in:imklog}
|          |-{in:imuxsock}
|          `-{rs:main Q:Reg}
|-smbd---smbd
|-sshd---sshd---sshd---bash---pstree
|-supervisord
|-systemd---(sd-pam)
|-systemd-journal
|-systemd-logind
`-systemd-udevd
```

Exercice 7 - optionnel

En reprenant l'exercice de dictionnaire du TP2, on veut ajouter deux autres fonctions :

— une fonction qui permet de sauvegarder des mots et des définitions dans un fichier. Le programme demande à l'utilisateur le chemin où il veut sauvegarder des données.

— une fonction pour reconstruire la liste des mots et des définitions à partir d'un fichier. Le programme demande à l'utilisateur le chemin où se localise le fichier.