Erick Gallesio, Stéphane Lavirotte, Jean-Paul Rigault, Hélène Collavizza
Polytech'Nice – Sophia
SI3
2021-2022

TD n°5
Primitives de base sur les fichiers/répertoires

Ces exercices utilisent les primitives Posix de manipulation de fichiers et de répertoires vues en cours, plus éventuellement quelques autres dûment signalées. N'hésitez pas à invoquer la commande `man` pour obtenir des informations d'utilisation des fonctions et commandes Shell citées. En cas d'ambiguïté, souvenez-vous que les primitives Posix sont dans les sections 2 et 3 du manuel (`man 2` … ou bien `man 3` …).

Pour réaliser ce TD, nous vous fournissons une partie du code pour vous permettre de vous concentrer sur les parties importantes. Vous pouvez récupérer l'archive à l'adresse suivante :

https://lms.univ-cotedazur.fr/mod/resource/view.php?id=177634

1 Comprendre les descripteurs de fichiers

Exercice n°1:

A l'aide du programme `fd_simple`, qui vous est fourni, tentez de comprendre et d'expliquer le mécanisme des descripteurs de fichiers. Vous veillerez tout d'abord à utiliser le programme tel que proposé (avec les commentaires) puis à décommenter au fur et à mesure les instructions pour comprendre la numérotation des descripteurs de fichiers et ce à quoi ils correspondent.

Attention, dans un premier temps, lancez l'exécution de ce programme depuis un vrai terminal et non depuis le terminal de l'IDE Visual Studio Code.

Et vous lancez le programme depuis Visual Studio Code, quels numéros de descripteur de fichier obtenez-vous ? Que pouvez-vous en conclure ?

2 Lister dossiers et fichiers : la commande `ls`

Exercice n°2:

Écrire le programme `lsrec` qui affiche les informations des fichiers ou répertoires qui lui sont passés en paramètres. Plus précisément, votre programme devra produire (à peu près) les mêmes sorties que la commande `ls` avec les options `-aR`). Si aucun argument n'est passé à la commande `lsrec`, celle-ci listera le contenu du répertoire courant. Vous noterez que vous disposez dans l'archive d'un script Shell de test pour le programme `lsrec` que vous écrirez (`tst_lsrec.sh`). Vous veillerez à faire un `exit(1)` dans les cas nécessaires. De plus, vous pourrez utiliser le fichier `util.c` pour vous faciliter la programmation sur quelques tests de base.

> **Suggestion** : Pour écrire ce programme, vous aurez besoin de plusieurs fonctions de bibliothèque, comme par exemple :
>
> - les primitives Posix de manipulation de répertoires (`opendir()`, `readdir()`,`rewinddir()`,`closedir()`) ;
> - la primitive Posix permettant d'obtenir les attributs d'un fichier (`stat()`).
>
> Vous serez aussi amené à concaténer des chaînes de caractères (pour créer le chemin sur lequel faire l'appel récursif…). Vous pourrez utiliser la fonction `int snprintf(char *buf, size_t size, const char *format, ...);`
> Cette fonction prend en paramètres : un pointeur sur une zone mémoire allouée `buf`, de taille maximum `size`, et on y stocke une représentation guidée par un `format` et les paramètres optionnels qui suivent, à la manière de `printf()`.

Pour vous faire gagner du temps sur l'écriture de ce programme, voici un squelette que nous vous invitons à suivre pour arriver rapidement au résultat.

TD n°5
Primitives de base sur les fichiers/répertoires

Faire une fonction `list` qui prendra en paramètre une chaîne de caractère (qui sera soit un nom de fichier soit un dossier) et que vous appellerez successivement avec chacun des arguments passés à votre `main`.

- Commencez par tester si c'est un répertoire
  - Si oui
    - Ouvrir celui-ci et imprimer son nom
    - Parcourir l'ensemble des entrées du répertoire et afficher les infos sur le fichier (`print_fileinfo`)
    - Rembobiner le répertoire
    - Reparcourir celui-ci et appeler récursivement `list` si l'entrée est bien un répertoire qui n'est ni `.` ni `..` (pour éviter de boucler indéfiniment). Attention à passer le bon chemin en paramètre de l'appel récursif…
    - Fermer le répertoire
  - Si non
    - Afficher les infos sur le ficher (`print_fileinfo`)

Vous pourrez utiliser les fonctions fournies dans `util.c` pour gagner du temps sur certains tests (par exemple si le répertoire est le répertoire `.` ou `..` par exemple).

3 Copie de Fichiers

Exercice n°3:

Réalisez le programme `mycp1` qui réalise la copie d'un fichier dans un autre fichier, ou d'un ensemble de fichiers dans un répertoire. La syntaxe d'invocation de ce programme doit être :

```
mycp1 fic1 fic2
```

ou bien

```
mycp1 fic1 fic2 ... dir
```

Dans le premier cas le fichier `fic1`, qui doit exister, est copié dans le fichier `fic2` ; si ce dernier existe, il est silencieusement écrasé, sinon il est créé. Dans le second cas, les fichiers `fic1`, `fic2`, … sont copiés dans le répertoire `dir` ; les fichiers `fic1`, `fic2`, … doivent être des fichiers ordinaires et doivent déjà exister ; `dir` doit aussi exister avant l'exécution de la commande ; si `dir` contient déjà des fichiers de mêmes noms que ceux qui sont copiés, les premiers sont silencieusement écrasés.

Vous noterez que vous disposez d'un script Shell de test du programme que vous écrirez (`tst_mycp1.sh`).

> **Suggestion** : Pour écrire ce programme, vous aurez besoin d'assez peu de fonctions de bibliothèque, principalement :
> - les fonctions Posix de gestion élémentaire des E/S (`open()`, `close()`, `read()`, `write()`)
> - la primitive Posix permettant d'obtenir les attributs d'un fichier (`stat()`).

4 Héritage des descripteurs de fichiers entre père et fils

Exercice n°4:

Écrire le programme `fd_herit` permettant de montrer l'héritage des descripteurs de fichiers et répertoires ouverts par un processus après un appel-système à `fork()`, en particulier :

- un processus hérite du descripteur des fichiers ouverts par son père, et partage le pointeur d'E/S ;
- un processus fils hérite aussi des répertoires ouverts par son père ;

Erick Gallesio, Stéphane Lavirotte, Jean-Paul Rigault, Hélène Collavizza
Polytech'Nice – Sophia
SI3
2021-2022

TD n°5
Primitives de base sur les fichiers/répertoires

Pour aller plus loin

Exercice n°5:

Etendez votre programme `lsrec` pour qu'il se comporte comme la commande `ls` d'Unix avec les options `-laR`.

> **Suggestion** : Pour écrire ce programme, vous aurez besoin de plusieurs fonctions de bibliothèque, comme par exemple :
> - les fonctions Posix de formatage des dates (`man strftime`)
> - la fonction Posix permettant d'obtenir des informations sur un utilisateur à partir de son uid (`getpwuid()`) ;

Exercice n°6:

Complétez le programme `mycp1`, pour obtenir un nouvel exécutable, `mycp2`, qui prenne en compte les options `-v` (verbose), `-r` (recursive) et `-i` (interactive) de la même manière que la commande `cp` (lisez les pages de man de cette commande et faites vos propres essais pour identifier son comportement avec les options considérées).

Suggestion : Pour écrire ce programme, vous aurez besoin des fonctions indiquées ci-dessus, plus quelques autres :

- les manipulations de chaînes de caractères en C (`man string`) ;
- les primitives Posix de manipulation de répertoires (`opendir()`, `readdir()`, `closedir()`).