

Encore des manipulations avec les fichiers

Durée prévue : 1 séance (1h30)

4.1 Implémenter la commande cp en C/C++

EXERCICE 1 ► Un programme mycp

Le but de cet exercice, est d'expérimenter avec les appels systèmes `read()` et `write()`. On va écrire un programme similaire à la commande `cp` pour copier des fichiers réguliers.

Contrairement à ce que l'on a fait au TP précédent, on ne va pas lire et écrire les octets (`char`) un par un, mais on va utiliser un buffer (tampon en Français) de `LEN` octets (`LEN` est une macro entière définie dans le code source à l'aide de `#define LEN 20`).

Les fichiers `mycp.cpp` et `in.pdf` sont disponibles dans l'archive fournie.

Expérimentons : On écrit un premier programme permettant de copier un morceau de fichier, que l'on améliorera par la suite.

- 1) On vous fournit le code source `mycp.cpp`; utilisez `g++ -o mycp mycp.cpp -Wall` pour obtenir l'exécutable `mycp`. À la compilation, vous allez avoir des *warnings* et **c'est tout à fait normal** : dans le code, il y a des variables déclarées que vous n'utiliserez que plus tard.
- 2) Commencez par vous faire un `Makefile` simple pour compiler le programme (une seule règle `all`), et effacer l'exécutable (règle `clean`).
- 3) Intéressez-vous au source `mycp.cpp`. À quoi vont servir les variables `fdin`, `fdout` et `buf`? Expliquez le comportement du programme (`man open` peut vous aider).
- 4) Donnez une première version de la copie dans laquelle :
 - 1 - vous tentez de lire `LEN` octets du fichier source pour les ranger dans le buffer,
 - 2 - vous écrivez le contenu du buffer dans le fichier de destination.

Utilisez un unique appel à `read()`, suivi d'un unique appel à `write()`, sans vous préoccuper des cas d'erreurs dans lesquels ces appels retournent `-1`. Dans cette question, on suppose que le fichier d'entrée comporte plus de `LEN` caractères : expérimentez avec un fichier `in.txt` qui contient une phrase d'au moins 20 caractères.

- 5) Consultez la page de manuel de `read()` pour vérifier qu'un appel à cette primitive retourne `-1` en cas d'erreur. Faites en sorte que, si jamais votre appel à `read()` retourne `-1`, alors votre programme se termine après avoir affiché le code d'erreur contenu dans la variable globale `errno` (à l'aide de `strerror()`, en prenant exemple sur le code déjà fourni). Que se passe-t-il si le fichier contient strictement moins de `LEN` caractères? Est-ce un comportement désirable? Expérimentez puis corrigez le cas échéant!

Copie d'un plus gros fichier :

1. Vous devez avoir dans votre répertoire de travail le fichier `in.pdf`, qui va nous servir de fichier de test pour le programme `mycp`. Quelle est la taille de ce fichier en octets (`ls -l`)? Jetez un œil au contenu du fichier également (avec `evince in.pdf` ou `xpdf in.pdf` par exemple). Testez `./mycp in.pdf out.pdf` : quel est l'effet produit sur `out.pdf` (`ls -l` et `evince`)?
2. Consultez la page de manuel de `read` : que signifie une valeur de retour non nulle de `read()`¹? Dans quel cas sait-on que l'on a atteint la fin du fichier? Dans quel cas doit-on considérer qu'il y a une erreur, et qu'il faut abandonner?
3. Consultez la page de manuel de `write()` : doit-on forcément considérer qu'un appel va permettre d'écrire le nombre d'octets passé en paramètre? Dans quel cas doit-on considérer qu'il y a une erreur, et qu'il faut définitivement abandonner?
4. Modifiez votre programme `mycp.cpp` de façon à copier tout le contenu du fichier d'entrée dans le fichier de destination. Pour cela, inspirez vous du « patron » suivant :

```
char buf[LEN]; // Buffer à utiliser pour les lectures/écritures.
char *p;      // Pointeur pour avancer dans le tampon buf.
int nbrd;     // Nombre d'octets lus sur fdin à chaque appel de read().
int nbwr;     // Nombre d'octets que l'on arrive à écrire sur fdout à chaque appel de write().
int nbrem;    // Nombre d'octets dans le buffer restants à écrire sur fdout.
do {
    // Lire avec read() au plus LEN caractères sur fdin en les rangeant dans buf.
    // Soit nbrd le nombre de caractères lus par read dans fdin.
```

1. Oui, c'est la même question que plus haut, la répétition est la mère de l'apprentissage

```
// En cas d'erreur lors du write(), terminer le programme en retournant EXIT_FAILURE.
nbrem = nbrd; // Nombre d'octets qu'il va falloir écrire sur fdout.
p = buf;      // On place le pointeur au début du buffer.
while(nbrem > 0) { // On reprend tant qu'il reste des octets du buffer à écrire
    // On tente d'écrire avec write() les nbrem octets restants (à partir de l'adresse p) vers fdout.
    // Soit nbwr le nombre d'octets que l'on a réussi à écrire.
    // En cas d'erreur lors du write(), terminer le programme en retournant EXIT_FAILURE.
    // Mettre à jour nbrem en fonction de nbwr.
    // Mettre à jour p pour avancer dans le tampon.
}
} while(nbrd > 0);
```

Utilisez `man read` et `man write` pour bien vérifier les paramètres de `read` et de `write`. N'oubliez pas de bien gérer les cas d'erreurs de `read`, et d'afficher les messages correspondant au code présent dans `errno`. Tous les messages d'erreurs doivent être envoyés vers la sortie d'erreur standard (utilisez `cerr`).

5. Testez votre programme avec `./mycp in.pdf out.pdf`; assurez-vous que le fichier copié fait bien la même taille en octets que le fichier initial (`ls -l`). Assurez-vous également que les deux fichiers contiennent bien le même document avec un lecteur de pdf (`evince out.pdf` par exemple) : si tel n'est pas le cas, déboguez!
6. Justifiez l'utilisation d'un *buffer* `buf` au lieu d'une copie octet par octet qui est clairement plus simple.

4.2 Pour aller plus loin : implémenter tee

EXERCICE 2 ► Commande tee

En utilisant `tee` (`man tee`) :

1. Écrire le résultat de la commande `ls` sur la sortie standard et dans un fichier en même temps.
2. Écrire le résultat de la commande `ls` dans trois fichiers de noms différents.
3. Écrire le résultat de la commande `ls` et dans le même temps, chercher un fichier précis dans la liste rendue par `ls`.

EXERCICE 3 ► Programme mytee

Sur le modèle de l'exercice précédent :

1. Programmer en C un clone du programme `tee` qui lit sur l'entrée standard et écrit sur la sortie standard, qui se lancera avec `./mytee`.
2. Ajouter l'écriture dans un fichier passé en paramètre.
3. Implémenter l'option `-a` de `tee`.

N'oubliez pas de tester les valeurs de retour des appels système pour vérifier que tout s'est bien passé!