Entrée / Sortie

Exercice 1.1
Écrire une commande `mycat` qui écrive caractère par caractère le contenu de son entrée standard sur sa sortie standard. Pour effectuer les entrées-sorties on utilisera le couple de fonctions `read / write`.
Testez votre programme en lui faisant afficher le code source, comme ceci :

```
./mycat < mycat.c
```

Modifiez votre programme pour lire/écrire N caractères à la fois (en définissant une constante dans le source).

Exercice 1.2
Écrire un programme `generer-entiers` qui prennent en argument un entier et qui crée un fichier contenant les entiers (32 bits non signés, sous forme binaire) successifs de 0 à $argument - 1$. Le nom du fichier pourra être codé en dur dans le source.
Écrire un programme `afficher-entiers` qui ouvre le fichier et qui affiche sur la sortie standard la valeur des entiers qui s'y trouvent.

Exercice 1.3 Il s'agit d'écrire deux commandes pour lire et enregistrer un entier de 32 bits non signé à une position donnée d'un fichier donné :

```
enregister-entier fichier position valeur
 lire-entier fichier position
```

1. Tester ces commandes en utilisant un fichier membre d'un répertoire de votre compte.
2. Utiliser la commande `hexdump` pour visualiser le contenu du fichier.
3. Que constate-t-on si l'on cherche lire / écrire après la fin du fichier ?

Exercice 1.4 Écrire la commande `reverse input-file-name output-file-name` qui écrit le miroir du premier fichier dans le second. On utilisera la fonction `lseek()` pour déterminer la taille du fichier source.

Exercice 1.5
On dispose d'une commande `ecrire` dont le programme source est listé ci-dessous :

```
int main(int argc, char *argv[])
{
  int i, fd;

  if(argc != 2) {
    fprintf(stderr, "Usage: ecrire <position_depart>\n");
    exit(1);
  }

  fd = open("fich", O_WRONLY | O_CREAT, 0666);

  for(i = atoi(argv[1]); i < 10; i += 2) {
    lseek(fd, i*sizeof(int), SEEK_SET);
    write(fd, &i, sizeof(int));
    sleep(1);
  }

  close(fd);
  return 0;
}
```

1. Décrivez précisément le contenu du fichier `fich` après l'exécution de la commande shell suivante :

   ```
   [toto@jaguar.u-bordeaux1.fr] ecrire 0
   ```

2. Même question lorsque l'on exécute consécutivement ces deux commandes :

   ```
   [toto@jaguar.u-bordeaux1.fr] ecrire 0
   [toto@jaguar.u-bordeaux1.fr] ecrire 1
   ```

3. Que se passe-t-il si l'on exécute les deux commandes `ecrire 0` et `ecrire 1` en parallèle (en les lançant simultanément dans deux terminaux) ? Que pouvez-vous dire à propos du fichier `fich` ?

Exercice 1.6 Interpréter les résultats produit par l'exécution du code suivant :

```
int main(){
printf("hello ");
write(1,"world", 5);
return 1;
}
```

1. Que se passe-t-il si l'on remplace la première chaîne par `"hello\n"` ? Tester aussi lors d'une redirection dans un fichier.
2. Que se passe-t-il si l'on remplace la sortie standard par la sortie standard d'erreur ?
3. Proposer une clarification du code.

Exercice 1.7 Écrire la commande `mycat-avec-buffer` qui fonctionne sur le même principe que la commande précédente `mycat`, mais en utilisant les fonctions `fread / fwrite`.
Comparer les temps de duplication d'un gros fichiers entre `mycat` et `mycat-avec-buffer` en utilisant la commande unix `time` pour mesurer le temps d'exécution de chaque programme.

Exercice 1.8 Il s'agit de mettre en valeur l'importance des tampons (buffers, en anglais) pour réaliser des entrées/sorties. Pour cela on va mesurer l'influence de la taille des buffers sur des opérations d'entrée-sortie.

1. Compléter le code source `copy.c` de la commande à trois paramètres

   `copy input-file-name output-file-name size`

   où le dernier paramètre correspond au logarithme en base 2 de la taille du buffer à utiliser (si le paramètre vaut n le buffer aura une taille de 2^n). Les fonctions `open()` et `close()` seront utiles.
2. Utiliser la commande `dd if=/dev/zero of=/tmp/toto count=10000` pour créer un fichier `/tmp/toto` de 10 000 blocs de 512 octets (tous nuls).
3. Déterminer expérimentalement la taille optimale pour minimiser le temps de copie d'un gros fichier.
4. Reproduire l'expérience en ouvrant le fichier destination en utilisant le mode `O_SYNC` de linux.

Exercice 1.9 À l'aide de l'appel système `dup2()`, faire en sorte que la commande `lire-entier` écrive ses messages d'erreurs dans le fichier `ERREURS-LIRE.log`.