TP Système
Mémo

La fonction open

Pour ouvrir un nouveau fichier, on utilise la fonction **`open`**. Voici un court résumé de sa documentation, disponible en tapant par exemple
`man 2 open`

On remarque en particulier qu'il y a deux signatures possibles :

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

Le premier argument est toujours le chemin du fichier à ouvrir (soit le chemin absolu, soit le chemin relatif, à partir de là où l'on se trouve), et la fonction retourne toujours un entier : le descripteur de fichier qui « pointe » vers le nouveau fichier ouvert (ou -1 s'il y a eu un problème). C'est ce descripteur qu'on utilisera ensuite, par exemple dans les fonctions read et write.

Ensuite, il y a des « flags » à spécifier, qui vont déterminer le mode d'ouverture de ce fichier. Selon les « flags » utilisés, il faudra utiliser l'une des deux signatures. Les flags sont en fait simplement des entiers, mais pour connaître à quoi correspond tel ou tel entier, on peut utiliser des constantes déjà définies, et les combiner :

- on doit obligatoirement spécifier un (et un seul) flag parmi l'un des trois suivants :
    - **`O_RDONLY`** : si on souhaite ouvrir le fichier en lecture seulement
    - **`O_WRONLY`** : si on souhaite ouvrir le fichier en écriture seulement
    - **`O_RDWR`** : si on souhaite ouvrir le fichier en lecture et écriture

Si l'ouverture réussit, le fichier est ouvert, et dispose de ce qu'on appelle une « tête de lecture » qui pointe au début du fichier. Si on lit, on lit à partir de cette tête de lecture, et celle-ci avance octet par octet, et idem si on écrit. Si le fichier contenait déjà des données, on écrit donc par dessus. En ajoutant d'autres flags, il est possible de modifier la position de la tête de lecture à l'ouverture, ou de spécifier d'autres choses :

- **`O_APPEND`** : la tête de lecture est positionnée à la fin du fichier (donc, si on écrit sur un fichier qui contient des données, on écrit à la suite, et la taille du fichier augmente)
- **`O_TRUNC`** : toutes les données du fichier sont effacées s'il y en avait. C'est différent du mode par défaut, où, par exemple, si le fichier contient 100 octets et qu'on écrit 10 octets dedans, le fichier comportera toujours 100 octets : on aura simplement remplacé les 10 premiers octets (si on avait utilisé **`O_TRUNC`** et écrit 10 octets, le fichier n'aurait comporté que ces 10 octets à la fin).
- **`O_CREAT`** : sans ce flag, si on essaie d'ouvrir un fichier qui n'existe pas, alors la fonction **`open`** retourne une erreur. Avec ce flag, un nouveau fichier est créé. Dans ce cas, il faut alors utiliser la seconde signature de la fonction **`open`** qui permet de spécifier les droits du nouveau fichier créé, en notation octale.

Par exemple, si je veux ouvrir le fichier **`toto.txt`** (on suppose qu'il existe et est dans le dossier courant) en écriture seulement, et que je veux écrire mes données à la suite du fichier, on peut faire :

```
int fd = open("toto.txt", O_WRONLY | O_APPEND)
```

Ainsi les flags se combinent grâce à l'opérateur « **`|`** » (on peut en combiner plus que 2). On utilisera ensuite par exemple l'entier **`fd`** comme premier paramètre de la fonction **`write`**.

Je veux ouvrir le fichier « **`tata.txt`** » en lecture-écriture. Je ne suis pas certain que ce fichier existe, et je veux le créer si ce n'est pas le cas, je peux alors écrire :

```
int fd = open("tata.txt", O_RDWR | O_CREAT, 0744)
```

Les droits 0744 signifient (rappels de cours de Linux) :

- le premier 0 précise que ce qui suit est en notation octale (et n'est pas l'entier 744)
- 7 : le propriétaire (celui qui a lancé le programme : moi par défaut) a droit de lecture, écriture et exécution sur le fichier
- 4 : les utilisateurs du groupe du propriétaire ont le droit de lecture seulement
- 4 les autres utilisateurs ont le droit le lecture seulement

(chaque chiffre est un entier entre 0 et 7, son écriture binaire vous indique si les utilisateurs concernés ont le droit de lecture, écriture et exécution)