TD3 – NSY103 – CNAM
« Fichiers – SGF »

Ex1 : Un processus crée 2 flots d'entrée/sortie sur le fichier « `./exemple` ». Le processus imprime la valeur du pointeur de lecture/écriture dans les 2 flots, lit 100 octets en utilisant un des flots, puis imprime la nouvelle valeur du pointeur de lecture/écriture dans les 2 flots.

Vous utiliserez les primitives open(), read().

position = lseek(desc, deplacement, mode)

mode : 1-> courant, 0 -> début, 2 -> fin

Ex2 : Un processus crée un flot d'E/S sur le fichier ./exemple, puis crée un nouveau nom local. Le processus imprime la valeur du pointeur L/E du flot en utilisant les 2 noms locaux, lit 100 octets en désignant le flot d'E/S par son premier nom, imprime la nouvelle valeur du pointeur de L/E (en utilisant les 2 noms), lit 50 octets en désignant le flot par son second nom, puis imprime la position finale du pointeur en utilisant encore une fois les 2 noms locaux.

Vous utiliserez les primitives open(), lseek(), read(), dup().

Ex3 : Copie de fichiers.

Question 1. Réécrire la commande Linux `cp` en utilisant les primitives d'E/S open, `creat`, `read` et `write`. On utilisera un tampon d'écriture de 512 caractères où seront placés les caractères lus dans le fichier source pour être ensuite écrits dans le fichier destination. Les messages d'erreurs (nombre de paramètres d'entrée, pb d'ouverture de fichier source, pb de création de fichier destination) sont envoyés sur la sortie standard d'erreur (`stderr`, descripteur 2) et pour chaque erreur correspond une valeur de retour différente du processus.

Question 2. Réécrire la commande Linux `cp` en utilisant les fonctions de la bibliothèque d'E/S standard.

Vous utiliserez pour cela :

```
FILE *fopen(char *ref, char *mode);
int fprintf(FILE *f, char *format, <liste_valeurs>);
int getc(FILE *f);
int putc(FILE *f);
```

Ex4 : Héritage père-fils

Question 1. Ecrivez un programme dans lequel un processus ouvre un fichier en lecture, ce fichier contenant la phrase « Bonjour les auditeurs de NSY103 ». Puis le processus crée un processus fils. Chacun des processus lit alors le contenu du fichier à raison de 4 caractères à la fois et affiche sur la sortie standard les 4 caractères lus. Le fils doit-il ouvrir le fichier à son tour ? Que montrent les traces d'exécution que vous obtenez ?

Question 2. Le processus fils recouvre à présent le code hérité de son père par le contenu de l'exécutable « lire_fichier ». On souhaite obtenir exactement le même comportement que pour la question 1. crivez le programme correspondant au processus père et celui du programme « lire_fichier.c ».

Ex5 : Attribut d'un fichier. Ecrivez un programme qui prend en entrée un nom de fichier et qui affiche la liste de ses attributs. Par exemple : nom de fichier, numéro d'inode, mode du fichier, nombre de liens, propriétaire, groupe et taille du fichier. Pour extraire ces informations vous utiliserez la primitive

```
int stat(char * ref, struct stat * buf);
```

qui donne accès à la structure :

```
struct stat {
        mode_t  st_mode;    //File mode (see mknod(2))
        ino_t   st_ino;     //Inode number

        dev_t   st_dev;             //ID of device containing
                                    //a directory entry for this file
        dev_t   st_rdev;    //ID of device
                                    //This entry is defined only for
                                    //char special or block files

        nlink_t  st_nlink;  //Number of links
        uid_t   st_uid;     //User ID of the file's owner
        gid_t   st_gid;     //Group ID of the file's group
        off_t   st_size;//File size in bytes
        time_t  st_atime;   //Time of last access
        time_t  st_mtime;   //Time of last data modification
        time_t  st_ctime;   //Time of last file status change
                                    //Times measured in seconds since
                                    //00:00:00 UTC, Jan. 1, 1970
}
```