# Système d'Exploitation Unix
# TP
# Les appels système : gestion de fichiers

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction `main (int argc, char *argv [])`. Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affiches avec `perror`. Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`

## Question 1 Création de fichiers `open ()`

a) Ecrire un programme qui crée un fichier en lecture/écriture au travers de l'appel

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

Si le fichier existe déjà, une erreur doit être retournée.

```
#include<stdio.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>

int main (int nb_args, char *args[]){
int fd, fc;

    if (nb_args != 2){
       fprintf (stderr, "Usage: %s fichier\n",args[0]);
       exit(1);
    }

     //O_CREAT creation du fichier s il n'existe pas
     //O_CREAT|O_EXCL creation du fichier s il n'existe pas
    //                mais erreur s'il existe déjà

     fd = open (args[1], O_CREAT|O_EXCL, 0666);

     if (fd == -1) {
         perror(args[1]);
         exit(1);
     }

     printf("fichier %s créé\n",args[1]);

     return 0;
}
```

b) Quel est le code d'erreur retourné lorsque le fichier existe déjà ?
Le code est EEXIST:pathname existe déjà (cf. man 2 open).

Si l'argument mode spécifié est `755` (`rwxr-xr-x`), est-ce que le fichier est créé avec



exactement ces droits ? Expliquer !
*Les droits obtenus sont le résultat de l'application du umask à mode: chaque bit à 1 du umask interdit le droit correspondant de mode.*

## Question 2 Caractéristique d'un fichier `stat(), fstat(), lstat`

Ecrire un programme qui récupère les caractéristiques de fichiers donnés, au travers des appels des fonctions `int stat(); int fstat();` et `int lstat;`. Pour un fichier donné, afficher les caractéristiques suivantes :

- le numéro d'inode,
- la protection,
- le nombre de liens physiques,
- l'ID du propriétaire,
- l'ID du groupe,
- la taille du fichier,
- la taille de bloc,
- le nombre de blocs,
- l'heure du dernier accès.

L'affichage d'une heure dans un format lisible peut être accompli en utilisant la fonction `ctime()`.

```
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

/* ================================================== */
   char *typeFichier (mode_t *st_mode){
/* ================================================== */
   char *ptr;

        if (S_ISREG(*st_mode)) ptr = "regular";
     else if (S_ISDIR(*st_mode)) ptr = "directory";
     else if (S_ISCHR(*st_mode)) ptr = "character special";
     else if (S_ISBLK(*st_mode)) ptr = "bloc special";
     else if (S_ISFIFO(*st_mode)) ptr = "fifo";
     else if (S_ISLNK(*st_mode)) ptr = "symbolic link";
     else if (S_ISSOCK(*st_mode)) ptr = "socket";
     else ptr = "** unknown mode **";

     return ptr;

} /* typeFichier */

/* ================================================== */
     main (int argc, char *argv[]) {
/* ================================================== */
     struct stat info; char *ptr;

     if (argc != 2){
        fprintf (stderr, "Usage: %s fichier\n",argv[0]);
        exit(1);
     }

   if (stat(argv[1], &info) != 0) {
        perror("Echec stat : ");
        exit(1);
```

```
   }

   printf("- Type\t\t\t%s\n", typeFichier(&info.st_mode));
   printf("- Inode\t\t\t%d\n",info.st_ino);
   printf("- Protection\t\t%lo (octal)\n",(unsigned long) info.st_mode);
   printf("- Lien(s)\t\t%d\n",info.st_nlink);
   printf("- Uid\t\t\t%d\n",info.st_uid);
   printf("- Gid\t\t\t%d\n",info.st_gid);
   printf("- Taille\t\t%d octets\n",info.st_size);
   printf("- Bloc E/S\t\t%d octets\n",info.st_blksize);
   printf("- Bloc(s)\t\t%d (X 512 octets)\n",info.st_blocks);
   printf("- Acces\t\t\t%s\n",ctime(&(info.st_atime)));
   printf("- Modification\t\t%s\n",ctime(&(info.st_mtime)));
   printf("- Etat\t\t\t%s\n",ctime(&(info.st_ctime)));

   exit(0);
}// main
```

## Question 3 **Utilisation d'un fichier** `open(), read(), write()`

a) Ecrire un programme qui recopie un fichier source, `fichier_source`, dans un fichier destinataire, `fichier_destinataire`. Le programme doit vérifier que le fichier source est un fichier régulier (utiliser la macro `S_ISREG(m)`, cf. `int stat()`). Le programme doit également vérifier qu'il n'existe pas déjà de fichier de même nom que le fichier destinataire.

```
#include<stdio.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>

main (int argc, char *argv[]) {
     struct stat info;
     char *buffer;
     int
          nb_read,
          nb_write,
          fd_source,
          fd_destination,
          taille_buffer,
          total_copie,
          total_ES;

     if (argc != 4){
        fprintf (stderr, "Usage: %s fichier_source fichier_destination
taille_buffer\n",argv[0]);
        exit(EXIT_FAILURE);
     }

     if (stat(argv[1], &info) != 0) {
        perror("Echec stat : ");
        exit(EXIT_FAILURE);
     }

     if (!S_ISREG(info.st_mode)){
```



```
          fprintf (stderr, "Usage: %s - %s n'est pas un fichier
regulier\n",argv[0], argv[1]);
          exit(EXIT_FAILURE);
     }

     fd_source=open(argv[1],O_RDONLY);
     if (fd_source == -1) {
          perror("Echec open fichier source : ");
          exit(EXIT_FAILURE);
     }

     fd_destination=open(argv[2],O_WRONLY|O_CREAT|O_EXCL, 0644);
     if (fd_destination == -1) {
          perror(argv[2]);
         exit(EXIT_FAILURE);
     }

     taille_buffer=atoi (argv[3]);
     buffer=malloc (taille_buffer);
     if (buffer == NULL) {
          perror ("malloc ");
          exit(EXIT_FAILURE);
     }

     for (nb_read=read(fd_source, buffer, taille_buffer), total_copie=0,
total_ES=0;
               (nb_read!=0)&&(nb_read!=-1);
                    nb_read=read(fd_source, buffer, taille_buffer)){

          if (nb_read>0)
               nb_write=write(fd_destination, buffer, nb_read);
        if (nb_write == -1) {
               perror(argv[2]);
               exit(EXIT_FAILURE);
          }
          total_copie=total_copie+nb_write;
          total_ES++;
     }// for
     if (nb_read == -1) {
          perror(argv[1]);
          exit(EXIT_FAILURE);
     }
     printf ("copié:\t\t%d octets\n", total_copie);
     printf ("buffer:\t\t%d octets\n", taille_buffer);
     printf ("nombre d'E/S:\t%d\n", total_ES);

     exit(EXIT_SUCCESS);
}// main
```

b) Quels sont les temps d'exécution (utiliser `/bin/time`) respectifs si la taille du buffer utilisé dans la fonction `read()` est de 1024 octets puis un octet ? Expliquer !

*Le temps d'exécution diminue si la taille du buffer augmente. Plus la taille du buffer augmente plus le nombre d'appels à `read` et `write` diminue. Cette diminution des appels contribue à diminuer le temps d'exécution. Le temps d'exécution est aussi fonction du nombre de lecture/écriture physiques qui est lui dépendant de la taille du bloc sur le disque.*

## Question 4 Duplication des descripteurs de fichier `dup(), dup2()`

a) Ecrire un programme qui redirige la sortie d'erreur standard vers un fichier, `fichier_erreur`, préalablement créé. C'est à dire, toute écriture de la forme `write(2, …)` doit se faire dans le fichier `fichier_erreur`. Le descripteur de valeur 2 étant au départ celui de la sortie d'erreur standard.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <fcntl.h>

#define STD_ERR   2

/*=====================================================================*/
  main(int argc, char *argv[]){
/*=====================================================================*/
int FdErr;

   if (argc!=2){fprintf (stderr,"nombre de parametres incorrect\n"); exit (1);}


   /* Ouverture du fichier de redirection choisi par l utilisateur */
   FdErr=open (argv[1], O_WRONLY|O_CREAT|O_TRUNC, 0644);
   if (FdErr==-1) {perror ("pb ouverture fichier\n"); exit (EXIT_FAILURE);}

   /* ================================================================== */
   /* Redirection des erreurs vers le fichier choisi par l utilisateur   */
   /* ================================================================== */

// (1) Fermeture du fichier standard des erreurs (erreur: stderr)
   if (close (STD_ERR)==-1) {perror ("pb close STD_ERR \n"); exit (1);}

// (2) duplication du descripteur du fichier de redirection dans la 1ere entree
//     de libre de la u_ofile. Ici ce sera l'ex-entree du fichier standard des
//     erreurs grace au close (STD_ERR) qui a précédé
   if (dup (FdErr)==-1){perror ("pb dup STD_ERR \n"); exit (1);}

   fprintf (stderr,"le fichier des erreurs redirigé vers %s\n",argv[1]);
} /* main() */
```

b) Quelle est la propriété de la fonction `dup()` qui est exploitée pour ainsi rediriger les E/S standards ?
*Cette fonction utilise la première entrée libre de la u_ofile.*

c) Modifier le code du mini shell afin qu'il prenne en charge la redirection des entrées (**<**) et des sorties (**>**).

## Question 5 Verrouillage des fichiers `fcntl()`

Ecrire un programme `lock_file` qui illustre la mise en oeuvre et le fonctionnement des verrous sur les fichiers en utilisant la primitive fcntl.
Lancement du programme:



```
./lock_file nom_fichier type_verrou debut_zone longueur_zone
```

Exemple:
- pour un verrou partagé

```
./lock_file test s 10 5
```

- pour un verrou exclusif

```
./lock_file test x 10 5
```

NB: le fichier test doit exister

```
#include<stdio.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
#include<string.h>
#include<stdlib.h>

#define EVER (;;)

int fd,
    mode,
    ret_fcntl;

char
     mode_voulu[10],
     mode_existant[10];

struct flock zone_lock;

int main (int nb_args, char *args[]){

    if (nb_args != 5){
       fprintf (stderr, "Usage: %s fichier verrou (s: partage, x:exclusif) debut
longueur\n",args[0]);
       exit(EXIT_FAILURE);
    }

    fd = open (args[1], O_RDWR);
    if (fd < 0) {
        perror(args[1]);
        exit(EXIT_FAILURE);
    }
 // ------------------------------------------------------------------
 // Quel type de verrou l'utilisateur veut-il poser ?
 // ------------------------------------------------------------------
    switch (*args[2]) {
         case 's' :  mode = F_RDLCK; // on desire un verrou partagé
                     strcpy (mode_voulu, "partage");
                     break;
         case 'x' :  mode = F_WRLCK; // on desire un verrou exclusif
                     strcpy (mode_voulu, "exclusif");
                     break;
         default :
                     fprintf (stderr, "%s mode invalide\n",args[2]);
                     exit (EXIT_FAILURE);
    }// switch
```

```
 // ------------------------------------------------------------------
 // Un verrou a-t-il déjà été posé et de quel type ?
 // Cette partie n'est pas néccessaire pour la pose d'un verrou. Elle
 // là uniquement pour produire une execution explicative de ce qui
 // se passe
 // ------------------------------------------------------------------
    zone_lock.l_type = mode;
    zone_lock.l_whence = SEEK_SET;    // origine:début du fichier
    zone_lock.l_start = atoi(args[3]);// debut de la zone à verrouiller par
rapport à origine
    zone_lock.l_len = atoi(args[4]);  // longueur de la zone à verrouiller

 // ------------------------------------------------------------------
 // F_GETLK: pour indiquer à fcntl qu'on désire connaitre l'état
 // de verrouillage de la zone concernée sur le fichier
 // ------------------------------------------------------------------
    ret_fcntl=fcntl(fd, F_GETLK, &zone_lock);
    //printf("ret_fcntl=%d\n",ret_fcntl);
    if (ret_fcntl<0){
          perror("Erreur lors du test du verrou");
          exit(EXIT_FAILURE);
    }

   if (zone_lock.l_type != F_UNLCK) {
       // il existe DEJA un verrou
       switch (zone_lock.l_type) {// quel verrou existe déjà à quel endroit
                                  // et posé par qui ?
           case F_RDLCK : strcpy (mode_existant, "partage"); break;
           case F_WRLCK : strcpy (mode_existant, "exclusif"); break;
       }// switch

       printf("\n%d:\tIl existe deja un verrou %s\n", getpid(), mode_existant);
       printf("\tposé par %d sur l'intervalle [%d,%d[\n",
              zone_lock.l_pid, zone_lock.l_start,
             zone_lock.l_start+zone_lock.l_len);
    }// if

// ------------------------------------------------------------------
// Pose du verrou:  F_SETLK. Avec F_SETLKW l'appel est bloquant
//                           l'attente active est inutile
// ------------------------------------------------------------------
    zone_lock.l_type = mode;
    zone_lock.l_whence = SEEK_SET;    // origine:début du fichier
    zone_lock.l_start = atoi(args[3]);// debut de la zone à verrouiller
                                      //par rapport à origine
    zone_lock.l_len = atoi(args[4]);  // longueur de la zone à verrouiller


  // ------------------------------------------------------------------
  // Si la pose du verrou est conflictuelle, fcntl retourne -1 et
  // errno contient EAGAIN ou EACCES (cf. le man de fcntl)
  // ------------------------------------------------------------------
   for(ret_fcntl=fcntl(fd, F_SETLK, &zone_lock);
          ret_fcntl<0;
              ret_fcntl=fcntl(fd, F_SETLK, &zone_lock)){

    // ----------------------------------
    // ATTENTE que le verrou soit levé
    // ----------------------------------
       if (errno==EAGAIN || errno==EACCES)
```



```
          sleep (5);
//          continue;
//          perror("J'attends\n");
    }// for

    if (ret_fcntl<0){
        perror("Pose du verrou: ");
        exit(EXIT_FAILURE);
    }
 // ------------------------------------------------------------------
 // Le verrou est posé avec succès: le processus fait ce qu'il a
 // à faire. Ici Une boucle infinie....
 // ------------------------------------------------------------------
    printf("\n%d:\tpose du verrou %s effectuée\n", getpid(), mode_voulu);
    for EVER sleep (10); // boucle infinie

    return 0;
}// main
```

## Question 6 Déplacement de la tête de lecture/écriture des fichiers `lseek()`

a) Ecrire un programme qui crée un fichier vide. Positionner la tête de lecture/écriture sur le 10 000ème octet à partir du début du fichier. Ecrire un caractère à cette position.

b) Quelle doit être la taille du fichier ? Est-ce cette taille qui est retournée par la commande `ls -l` ? Est-ce que les blocs correspondant au trou de 9 999 caractères ont été alloués (utiliser df) ?
La primitive `lseek()` permet de déplacer l'offset courant d'un fichier et retourne le nouvel `offset`.

c) Ecrire un programme qui, après un certain nombre de lecture/écriture sur un fichier, retourne la valeur de l'`offset` courant.
L'`offset` est associé à un fichier et non pas à un descripteur. Si deux descripteurs référencent un même fichier, la modification (par lecture/écriture ou `lseek()`) de l'offset du fichier via un descripteur est "visible" via l'autre descripteur.

6.d) Vérifier l'affirmation précédente.

## Question 7 Manipulation de répertoires `opendir(),readdir(), mkdir()`

La primitive `DIR *opendir(const char *pathname)` permet d'ouvrir en lecture le répertoire référencé par pathname.

La primitive `struct dirent *readdir(DIR *dp)` permet de lire l'entrée suivante du répertoire identifié par `dp`.

a) Ecrire un programme qui ouvre un répertoire (/tmp par exemple) et qui affiche tous les fichiers qui y sont contenus ainsi que le type de chacun (champ d_type de la strucure dirent).

La primitive `int mkdir(const char *pathname, mode_t mode)` permet de

créer le répertoire de nom référencé par pathname.

b) Modifier le programme précédent en y ajoutant à la fin la création (dans le répertoire /tmp) d'un répertoire ayant pour nom votre propre nom.

c) Vérifier que le répertoire a bien été créé.

La primitive `void rewinddir(DIR *dp)` permet de positionner le pointeur de lecture dans un répertoire sur la première entrée de ce répertoire.

a) Modifier le programme précédent en y ajoutant à la fin 1) le positionnement en début du pointeur de lecture du répertoire et 2) l'affichage à nouveau du contenu du répertoire (/tmp).

Avant chaque exécution, supprimer du répertoire /tmp le répertoire que vous avez créé.

b) Modifier le programme précédent en y ajoutant à la fin la suppression du répertoire créé.

c) Vérifier que le répertoire a bien été supprimé.