Práctica 2.2. Sistema de Ficheros

Objetivos

En esta práctica se revisan las funciones del sistema básicas para manejar un sistema de ficheros, referentes a la creación de ficheros y directorios, duplicación de descriptores, obtención de información de ficheros o el uso de cerrojos.

Contenidos

Preparación del entorno para la práctica
Creación y atributos de ficheros
Redirecciones y duplicación de descriptores
Cerrojos de ficheros
Directorios

Preparación del entorno para la práctica

La realización de esta práctica únicamente requiere del entorno de desarrollo (compilador, editores y utilidades de depuración). Estas herramientas están disponibles en las máquinas virtuales de la asignatura y en la máquina física de los puestos del laboratorio.

Creación y atributos de ficheros

El inodo de un fichero guarda diferentes atributos de éste, como por ejemplo el propietario, permisos de acceso, tamaño o los tiempos de acceso, modificación y creación. En esta sección veremos las llamadas al sistema más importantes para consultar y fijar estos atributos así como las herramientas del sistema para su gestión.

Ejercicio 1. `ls(1)` muestra el contenido de directorios y los atributos básicos de los ficheros. Consultar la página de manual y estudiar el uso de las opciones `-a -l -d -h -i -R -1 -F` y `--color`. Estudiar el significado de la salida en cada caso.

```
ls - list directory contents

-a, --all
       do not ignore entries starting with .

-l    use a long listing format

-d, --directory
       list directories themselves, not their contents

-h, --human-readable
       with -l, print sizes in human readable format (e.g., 1K 234M 2G)

-i, --inode
       print the index number of each file

-R, --recursive
       list subdirectories recursively

-1    list one file per line
```

```
-F, --classify
        append indicator (one of */=>@|) to entries

--color[=WHEN]
        colorize the output; WHEN can be 'never', 'auto', or 'always'
        (the default); more info below
```

Ejercicio 2. El *modo* de un fichero es `<tipo><rwx_propietario><rwx_grupo><rwx_resto>`:

- `tipo`: `-` fichero ordinario; `d` directorio; `l` enlace; `c` dispositivo carácter; `b` dispositivo bloque; `p` FIFO; `s` socket
- `rwx`: `r` lectura (4); `w` escritura (2); `x` ejecución (1)

Comprobar los permisos de algunos directorios (con `ls -ld`).

```
[cursoredes@localhost ~]$ ls -ld
drwx------. 19 cursoredes cursoredes 4096 Nov  9 12:08 .

[cursoredes@localhost Documents]$ ls -ld
drwxr-xr-x 2 cursoredes cursoredes 6 Sep  9  2018 .
```

Ejercicio 3. Los permisos se pueden otorgar de forma selectiva usando la notación octal o la simbólica. Ejemplo, probar las siguientes órdenes (equivalentes):

- `chmod 540 fichero`
- `chmod u+rx,g+r-wx,o-wxr fichero`

¿Cómo se podrían fijar los permisos `rw-r--r-x`, de las dos formas?

```
ls a.txt -ld

Original:
-rw-rw-r-- 1 cursoredes cursoredes 4 Nov  9 12:17 a.txt

Tras comandos:
-r-xr----- 1 cursoredes cursoredes 4 Nov  9 12:17 a.txt

Para fijar los nuevos permisos:
chmod 645 a.txt
chmod u+rw-x,g+r-wx,o+xr-w a.txt
```

Ejercicio 4. Crear un directorio y quitar los permisos de ejecución para usuario, grupo y otros. Intentar cambiar al directorio.

```
[cursoredes@localhost Documents]$ mkdir dir
[cursoredes@localhost Documents]$ chmod -x dir
[cursoredes@localhost Documents]$ cd dir
bash: cd: dir: Permission denied
```

Ejercicio 5. Escribir un programa que, usando `open(2)`, cree un fichero con los permisos `rw-r--r-x`. Comprobar el resultado y las características del fichero con `ls(1)`.

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

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

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

O_RDONLY,  O_WRONLY,  or  O_RDWR
O_CLOEXEC, O_CREAT,  O_DIRECTORY,  O_EXCL,  O_NOCTTY,  O_NOFOLLOW,  O_TRUNC, and
O_TTY_INIT


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

int main() {
        int fd = open("a.txt", O_CREAT, 0645);
        if (fd == -1) {
                printf("Error creating the file\n");
                return 1;
        }
        return 0;
}

[cursoredes@localhost practica2]$ ls a.txt -l
-rw-r--r-x 1 cursoredes cursoredes 0 Nov  9 12:54 a.txt
```

Ejercicio 6. Cuando se crea un fichero, los permisos por defecto se derivan de la máscara de usuario (*umask*). El comando interno de la *shell* `umask` permite consultar y fijar esta máscara. Usando este comando, fijar la máscara de forma que los nuevos ficheros no tengan permiso de escritura para el grupo y no tengan ningún permiso para otros. Comprobar el funcionamiento con `touch(1)`, `mkdir(1)` y `ls(1)`.

```
[cursoredes@localhost ~]$ umask 0027

[cursoredes@localhost ~]$ touch a.txt
[cursoredes@localhost ~]$ ls a.txt -l
-rw-r----- 1 cursoredes cursoredes 0 Nov  9 17:13 a.txt

[cursoredes@localhost ~]$ ls dir -l -d
drwxr-x--- 2 cursoredes cursoredes 6 Nov  9 17:15 dir
```

Ejercicio 7. Modificar el ejercicio 5 para que, antes de crear el fichero, se fije la máscara igual que en el ejercicio 6. Comprobar el resultado con `ls(1)`. Comprobar que la máscara del proceso padre (la *shell*) no cambia.

```
umask - set file mode creation mask

#include <sys/types.h>
#include <sys/stat.h>

mode_t umask(mode_t mask);


#include <stdio.h>
```

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

int main() {
        umask(0027);
        int fd = open("a.txt", O_CREAT, 0645);
        if (fd == -1) {
                printf("Error creating the file\n");
                return 1;
        }
        return 0;
}
```

Ejercicio 8. `ls(1)` puede mostrar el inodo con la opción `-i`. El resto de información del inodo puede obtenerse usando `stat(1)`. Consultar las opciones del comando y comprobar su funcionamiento.

```
stat - display file or file system status

stat [OPTION]... FILE...

-L, --dereference
        follow links

-f, --file-system
        display file system status instead of file status


[cursoredes@localhost practica2]$ ls a.txt -i
918790 a.txt
[cursoredes@localhost practica2]$ stat a.txt
  File: ‘a.txt’
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d    Inode: 918790      Links: 1
Access: (0645/-rw-r--r-x)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2021-11-09 12:54:52.537182774 +0100
Modify: 2021-11-09 12:54:52.537182774 +0100
Change: 2021-11-09 12:54:52.537182774 +0100
 Birth: -
[cursoredes@localhost practica2]$ stat a.txt -f
  File: "a.txt"
    ID: fd0000000000 Namelen: 255     Type: xfs
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 4452864    Free: 3152501    Available: 3152501
Inodes: Total: 8910848    Free: 8680264
```

Ejercicio 9. Escribir un programa que emule el comportamiento de `stat(1)` y muestre:

- El número *major* y *minor* asociado al dispositivo.
- El número de inodo del fichero.
- El tipo de fichero (directorio, enlace simbólico o fichero ordinario).
- La hora en la que se accedió el fichero por última vez. ¿Qué diferencia hay entre `st_mtime` y `st_ctime`?

```
stat, fstat, lstat, fstatat - get file status
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *pathname, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *pathname, struct stat *buf);

struct stat {
        dev_t     st_dev;       /* ID of device containing file */
        ino_t     st_ino;       /* inode number */
        mode_t    st_mode;      /* file type and mode */
        nlink_t   st_nlink;     /* number of hard links */
        uid_t     st_uid;       /* user ID of owner */
        gid_t     st_gid;       /* group ID of owner */
        dev_t     st_rdev;      /* device ID (if special file) */
        off_t     st_size;      /* total size, in bytes */
        blksize_t st_blksize;   /* blocksize for filesystem I/O */
        blkcnt_t  st_blocks;    /* number of 512B blocks allocated */
};

#include <sys/types.h>

dev_t makedev(int maj, int min);

unsigned int major(dev_t dev);
unsigned int minor(dev_t dev);


#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <time.h>

int main(int argc, char *argv[]) {
        if (argc != 2) {
                printf("Incorrect number of arguments\n");
                return 1;
        }

        struct stat stats;
        int i = stat(argv[1], &stats);

        if (i == -1) {
                printf("File error\n");
                return 1;
        }

        printf("Major number: %d\n", major(stats.st_dev));
        printf("Minor number: %d\n", minor(stats.st_dev));

        printf("I-node: %d\n", stats.st_ino);

        mode_t mode = stats.st_mode;
        printf("%i\n",mode);
```

```
        if (S_ISLNK(mode))
                printf("Type: Symbolic link\n");
        else if (S_ISREG(mode))
                printf("Type: Regular file\n");
        else
                printf("Type: Directory\n");

        time_t t = stats.st_atime;
        char *date= ctime(&t);
        printf("Last access: %s\n", date);

        return 0;
}
```

st_mtime
This is the file's last modification timestamp. It is changed by file modifications, for example, by mknod(2), truncate(2), utime(2), and write(2) (of more than zero bytes). Moreover, st_mtime of a directory is changed by the creation or deletion of files in that directory. The st_mtime field is not changed for changes in owner, group, hard link count, or mode.

st_ctime
This is the file's last status change timestamp. It is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).

La diferencia que tienen es que ctime es más general, se actualizará con más frecuencia ya que engloba mtime + otras modificaciones.

Ejercicio 10. Los enlaces se crean con `ln(1)`:

- Con la opción `-s`, se crea un enlace simbólico. Crear un enlace simbólico a un fichero ordinario y otro a un directorio. Comprobar el resultado con `ls -l` y `ls -i`. Determinar el inodo de cada fichero.
- Repetir el apartado anterior con enlaces rígidos. Determinar los inodos de los ficheros y las propiedades con `stat` (observar el atributo número de enlaces).
- ¿Qué sucede cuando se borra uno de los enlaces rígidos? ¿Qué sucede si se borra uno de los enlaces simbólicos? ¿Y si se borra el fichero original?

```
ln - make links between files

-s, --symbolic
        make symbolic links instead of hard links

[cursoredes@localhost practica2]$ ln -s aOrig aLink
[cursoredes@localhost practica2]$ ln -s dirOrig dirLink
[cursoredes@localhost practica2]$ ls -l
total 52
lrwxrwxrwx 1 cursoredes cursoredes    5 Nov 12 18:08 aLink -> aOrig
-rw-rw-r-- 1 cursoredes cursoredes    2 Nov 12 18:07 aOrig
lrwxrwxrwx 1 cursoredes cursoredes    7 Nov 12 18:08 dirLink -> dirOrig
drwxrwxr-x 2 cursoredes cursoredes    6 Nov 12 18:07 dirOrig
...
[cursoredes@localhost practica2]$ ls -i
3576999 aLink  3577000 dirLink    918790 aOrig  3508947 dirOrig …
```

```
[cursoredes@localhost practica2]$ stat aLink
  File: ‘aLink’ -> ‘aOrig’
  Size: 5               Blocks: 0          IO Block: 4096   symbolic link
Device: fd00h/64768d    Inode: 3576999     Links: 1
Access: (0777/lrwxrwxrwx)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2021-11-12 18:08:06.274805897 +0100
Modify: 2021-11-12 18:08:06.274805897 +0100
Change: 2021-11-12 18:08:06.274805897 +0100
 Birth: -
i-Nodos distintos para cada uno

[cursoredes@localhost dirOrig]$ ln aOrig aLink
[cursoredes@localhost dirOrig]$ ln dirOrig dirLink
ln: ‘dirOrig’: hard link not allowed for directory
[cursoredes@localhost dirOrig]$ ls -l
total 8
-rw-rw-r-- 2 cursoredes cursoredes 2 Nov 12 18:12 aLink
-rw-rw-r-- 2 cursoredes cursoredes 2 Nov 12 18:12 aOrig
drwxrwxr-x 2 cursoredes cursoredes 6 Nov 12 18:13 dirOrig
[cursoredes@localhost dirOrig]$ ls -i
 3577001 aLink   3577001 aOrig  18828155 dirOrig
[cursoredes@localhost dirOrig]$ stat aOrig
  File: ‘aOrig’
  Size: 2               Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 3577001     Links: 2
Access: (0664/-rw-rw-r--)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2021-11-12 18:12:51.678668981 +0100
Modify: 2021-11-12 18:12:51.680669094 +0100
Change: 2021-11-12 18:13:14.978994505 +0100
 Birth: -
i-Nodos iguales

Cuando se borra el enlace duro simplemente se reduce el contador de links a 1 del i-Nodo
correspondiente
Cuando se borra el enlace simbólico no pasa nada ya que está en un i-Nodo independiente
Cuando se borra el fichero original del enlace duro no pasa simplemente se reduce el contador de
links a 1 del i-Nodo correspondiente
Cuando se borra el fichero original del enlace simbólico este se corrompe ya que comienza a
apuntar a un fichero (i-Nodo) que no existe
```

Ejercicio 11. `link(2)` y `symlink(2)` crean enlaces rígidos y simbólicos, respectivamente. Escribir un programa que reciba una ruta a un fichero como argumento. Si la ruta es un fichero regular, creará un enlace simbólico y rígido con el mismo nombre terminado en `.sym` y `.hard`, respectivamente. Comprobar el resultado con `ls(1)`.

```
symlink - make a new name for a file

#include <unistd.h>
int symlink(const char *oldpath, const char *newpath);

link - make a new name for a file

#include <unistd.h>
int link(const char *oldpath, const char *newpath);
```

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

int main(int argc, char *argv[]) {
	if (argc != 2) {
		printf("Incorrect number of arguments\n");
		return 1;
	}

	struct stat stats;
	int i = stat(argv[1], &stats);

	if (i == -1) {
		printf("File error\n");
		return 1;
	}

	if (!S_ISREG(stats.st_mode)) {
		printf("The file isn't a regular file\n");
		return 1;
	}

	printf("Type: Regular file\n");

	char* hard = malloc(sizeof(char)*(5 + strlen(argv[1])));
	strcpy(hard, argv[1]);
	hard = strcat(hard, ".hard");
	int lnk = link(argv[1], hard);
	if (lnk == -1){
		printf("Error: %d %s\n", errno, strerror(errno));
		return 1;
	}
	printf("Hard link created\n");

	char* sym = malloc(sizeof(char)*(5 + strlen(argv[1])));
	strcpy(sym, argv[1]);
	sym = strcat(sym, ".sym");
	int symlnk = symlink(argv[1], sym);
	if (symlnk == -1){
		printf("Error: %d %s\n", errno, strerror(errno));
		return 1;
	}
	printf("Symbolic link created\n");

	return 0;
}

[cursoredes@localhost practica2]$ ls -l
total 72
-rw-rw-r-- 2 cursoredes cursoredes    4 Nov 12 18:46 a
-rw-rw-r-- 2 cursoredes cursoredes    4 Nov 12 18:46 a.hard
lrwxrwxrwx 1 cursoredes cursoredes    1 Nov 12 19:05 a.sym -> a
```

```
...
[cursoredes@localhost practica2]$ ls -i
3577013 a  3577013 a.hard  3577017 a.sym ...
```

Redirecciones y duplicación de descriptores

La *shell* proporciona operadores (>, >&, >>) que permiten redirigir un fichero a otro, ver los ejercicios propuestos en la práctica opcional. Esta funcionalidad se implementa mediante `dup(2)` y `dup2(2)`.

Ejercicio 12. Escribir un programa que redirija la salida estándar a un fichero cuya ruta se pasa como primer argumento. Probar haciendo que el programa escriba varias cadenas en la salida estándar.

```
dup, dup2, dup3 - duplicate a file descriptor

#include <unistd.h>
int dup(int oldfd);
int dup2(int oldfd, int newfd);

These system calls create a copy of the file descriptor oldfd.
dup() uses the lowest-numbered unused descriptor for the new descriptor.
dup2() makes newfd be the copy of oldfd, closing newfd first if necessary


#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char *argv[]){
        if(argc != 2){
                printf("Incorrect number of arguments\n");
                return 1;
        }

        int fd = open(argv[1], O_CREAT | O_RDWR, 0777);
        if(fd == -1){
                perror("Error creating the file\n");
                return 1;
        }

        dup2(fd, STDOUT_FILENO);
        printf("Enviando cadena 1\n");
        printf("Enviando cadena 2\n");
        return 0;
}
```

Ejercicio 13. Modificar el programa anterior para que también redirija la salida estándar de error al fichero. Comprobar el funcionamiento incluyendo varias sentencias que impriman en ambos flujos. ¿Hay diferencia si las redirecciones se hacen en diferente orden? ¿Por qué `ls > dirlist 2>&1` es diferente a `ls 2>&1 > dirlist`?

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char *argv[]){
	if(argc != 2){
		printf("Incorrect number of arguments\n");
		return 1;
	}

	int fd = open(argv[1], O_CREAT, 0777);
	if(fd == -1){
		perror("Error creating the file\n");
		return 1;
	}

	dup2(fd, STDOUT_FILENO);
	dup2(fd, STDERR_FILENO);

	printf("Enviando cadena 1 correcta\n");
	fprintf("Enviando cadena 1 error\n");
	printf("Enviando cadena 2 correcta\n");
	fprintf("Enviando cadena 2 error\n");
	return 0;
}
```

No cambia si las redirecciones se ejecutan en distinto orden, las cadenas de error siempre se escriben antes que las normales.

Es distinto porque el orden importa, esto es debido a que cuando se hace 2>&1 en el segundo comando stdout todavía no apunta al archivo entonces el error al archivo sino que se mantiene en la consola.

Cerrojos de ficheros

El sistema de ficheros ofrece cerrojos de ficheros consultivos.

Ejercicio 14. El estado y cerrojos de fichero en uso en el sistema se pueden consultar en el fichero `/proc/locks`. Estudiar el contenido de este fichero.

```
1: POSIX  ADVISORY  WRITE 1576 fd:00:53013121 0 EOF
2: POSIX  ADVISORY  WRITE 1569 fd:00:53013120 0 EOF
3: POSIX  ADVISORY  WRITE 1563 fd:00:53012991 0 EOF
4: POSIX  ADVISORY  WRITE 1553 fd:00:53012987 0 EOF
5: POSIX  ADVISORY  WRITE 1346 00:13:21281 0 EOF
6: FLOCK  ADVISORY  WRITE 1149 fd:00:17458434 0 EOF
7: FLOCK  ADVISORY  WRITE 1149 fd:00:30831 0 EOF
8: POSIX  ADVISORY  WRITE 1039 00:13:19890 0 EOF
9: FLOCK  ADVISORY  WRITE 1038 00:13:19877 0 EOF
10: POSIX  ADVISORY  WRITE 787 00:13:18349 0 EOF
12: POSIX  ADVISORY  WRITE 483 00:13:12830 0 EOF
```

It is a file in the procfs virtual file system:

1. The first column is a sequence number.
2. The second field indicates the class of the lock used, such as FLOCK (from flock system

call) or POSIX (from the lockf, fcntl system call).
3. This column is for the type of lock. It can have two values: ADVISORY or MANDATORY.
4. The fourth field reveals if the lock is a WRITE or READ lock.
5. Then we have the ID of the process holding the lock.
6. This field contains a colon-separated-values string, showing the id of the locked file in the format of “major-device:minor-device:inode“.
7. This column, together with the last one, shows the start and end of the locked region of the file being locked. In this example row, the entire file is locked.

Ejercicio 15. Escribir un programa que consulte y muestre en pantalla el estado del cerrojo sobre un fichero usando `lockf(3)`. El programa mostrará el estado del cerrojo (bloqueado o desbloqueado). Además:

- Si está desbloqueado, fijará un cerrojo y escribirá la hora actual. Después suspenderá su ejecución durante 30 segundos (con `sleep(3)`) y a continuación liberará el cerrojo.
- Si está bloqueado, terminará el programa.

```
#include <unistd.h>
int lockf(int fd, int cmd, off_t len);

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

int main(int argc, char *argv[]) {
	if (argc != 2) {
		printf("Incorrect number of arguments\n");
		return 1;
	}

	int fd = open(argv[1], O_RDWR | O_CREAT);

	if (fd == -1) {
		printf("File error\n");
		return 1;
	}

	int n = lockf(fd, F_TLOCK, 0);

	if (n == -1) {
		printf("Error: %d %s\n", errno, strerror(errno));
		printf("Cerrojo bloqueado\n");
		return 1;
	}

	time_t tiempo;
	time(&tiempo);
	printf("Hour: %s", asctime(localtime(&tiempo)));

	sleep(30);
```

```
        lockf(fd, F_ULOCK, 0);

        return 0;
}
```

***Ejercicio 16* (Opcional).** `flock(1)` proporciona funcionalidad de cerrojos antiguos BSD en guiones *shell*. Consultar la página de manual y el funcionamiento del comando.

```
flock - manage locks from shell scripts

flock [options] <file|directory> <command> [command args]
flock [options] <file|directory> -c <command>
flock [options] <file descriptor number>
```

Directorios

Ejercicio 17. Escribir un programa que cumpla las siguientes especificaciones:

- El programa tiene un único argumento que es la ruta a un directorio. El programa debe comprobar la corrección del argumento.
- El programa recorrerá las entradas del directorio de forma que:
    - Si es un fichero normal, escribirá el nombre.
    - Si es un directorio, escribirá el nombre seguido del carácter `'/'`.
    - Si es un enlace simbólico, escribirá su nombre seguido de `'->'` y el nombre del fichero enlazado. Usar `readlink(2)` y dimensionar adecuadamente el *buffer*.
    - Si el fichero es ejecutable, escribirá el nombre seguido del carácter `'*'`.
- Al final de la lista el programa escribirá el tamaño total que ocupan los ficheros (no directorios) en kilobytes.

```
opendir, fdopendir - open a directory

#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name);
DIR *fdopendir(int fd);

readdir, readdir_r - read a directory

#include <dirent.h>
struct dirent *readdir(DIR *dirp);

struct dirent {
        ino_t          d_ino;   /* inode number */
        off_t          d_off;   /* not an offset; see NOTES */
        unsigned short d_reclen;        /* length of this record */
        unsigned char  d_type;  /* type of file; not supported
                                   by all file system types */
        char      d_name[256]; /* filename */
};

readlink - read value of a symbolic link

#include <unistd.h>
```

```
ssize_t readlink(const char *path, char *buf, size_t bufsiz);


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

int main(int argc, char *argv[]) {
        if (argc != 2) {
        printf("Incorrect number of arguments\n");
        return 1;
        }

        DIR *directory = opendir(argv[1]);

        if (directory == NULL) {
        printf("Incorrect argument\n");
        return 1;
        }

  struct dirent *dir = readdir(directory);

        if (dir == NULL) {
        printf("Error: %d %s\n", errno, strerror(errno));
        return 1;
        }

  int size = 0;
  int dirSize = sizeof(char)*strlen(argv[1]);

  while (dir != NULL) {
    char *path = malloc(dirSize + sizeof(char)*(strlen(dir->d_name) + 1));
    strcpy(path, argv[1]);
    strcat(path, "/");
    strcat(path, dir->d_name);

    struct stat stats;
    int i = stat(path, &stats);

    if (i == -1) {
      printf("File error\n");
      free(path);
      closedir(directory);
      return 1;
    }

    if (S_ISREG(stats.st_mode))
      printf("Type: Regular file\nName: %s", dir->d_name);
    else if (S_ISDIR(stats.st_mode))
      printf("Type: Directory\nName: %s\\", dir->d_name);
```

```
      else if (S_ISLNK(stats.st_mode)) {
         char *buf = malloc(stats.st_size + 1);
         int e = readlink(path, buf, stats.st_size + 1);

         if (e == -1){
            printf("Error: %d %s", errno, strerror(errno));
            free(path);
            free(buf);
            closedir(directory);
                  return 1;
         }
         printf("Type: Symbolic link\nName: %s->%s", dir->d_name, buf);
         free(buf);
      }
      if (stats.st_mode & S_IXUSR)
         printf("*\n\n");
      else
         printf("\n\n");

      if (!S_ISDIR(stats.st_mode))
         size += stats.st_blocks/2;
      free(path);
      dir = readdir(directory);
   }
   closedir(directory);

   printf("Total size of files: %d KB\n", size);
            return 0;
}
```