UNIVERSITY AT ALBANY
State University of New York

COLLEGE OF
NANOTECHNOLOGY, SCIENCES, AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

ICSI 333 SYSTEM FUNDAMENTALS
CQUPT, SPRING 2024
LAB 3.1 – 3.3

The total grade for the assignment (all three labs) is 100 points.

You must follow the programming and documentation guidelines (see file *Programming Assignments Requirements and Recommendations.docx*).

DESCRIPTION

In this project, you must write a C program that performs the system actions – copying or moving a file or a group of files to a specified path. The program acts differently depending on the usage:

- it performs copying if used as

```
copy source1 [source2 ...] destination
```

- it performs moving if used as

```
move source1 [source2 ...] destination
```

The source and destination names are full or relative paths, and your program must be able to extract the file or directory base name when needed. Assume that the space symbol is not allowed in the file and directory names.

Examples:

```
../xxx/yyy  -->   yyy
zzz         -->   zzz
/           -->   /
```

When your program performs moving (not copying), it copies a file to the new location and then deletes the file (unlinks the old path). Physical coping of all bytes takes time; the more efficient way would be to create a new link (the new path linked to the old bytes on a disk). And I suggest you try this way first, but the OS may not let you do so for several reasons. So then go to the first plan.

You may need to study online resources and discuss the project with classmates. But you should not borrow a solution online or from other students. It is better to skip some functionalities than to submit a plagiarized code. Of course, you can always ask your TA and instructor for help.

To implement the required usage, you need file multiple linking, for example,

```
gcc prog.c -o copy; ln copy move
```

After that, the file with the same executable code can be run under two different names, and the program itself decides on coping or moving depending on the name that the user used.

Keep in mind that there are two types of file links – hard and soft (or symbolic).

- **Hard links (each file must have at least one):** Files have names that people use and unique numbers that the OS uses. The unique number (inode + device ID) corresponds to the physical collection of bytes on a disk, and only the kernel manages such a number. File names are created by people. You can give a name and then change it. The OS links your name to a physical location of the collection of bytes. Thus, a file name is linked to a unique number. You may have several file names linked to the physical location (why you may need it is another story).

- **Soft links (not a must-have):** You may want to link a new file name to the existing file name (not to the physical bytes). For example, to have an alias that you can delete later and not worry about deleting the real file bytes. It is a soft link.

Once the required action (copying or moving) is defined, the program must check the destination:

  - only a directory or device can be the destination for copying more than one file and

  - only a directory can be the destination for moving more than one file.

An invalid destination must result in the error message and program termination. For this project, you must understand and use the system call `stat()`. The following information adopted from https://man7.org/linux/man-pages/man7/inode.7.html will be helpful while detecting the file type.

The structure `stat` has the field `stat.st_mode` that contains the file type and mode. POSIX refers to the `stat.st_mode` bits corresponding to the mask `0170000`[1] as the **file type**, the 12 bits corresponding to the mask `07777` as the **file mode bits**, and the least significant 9 bits (the mask `0777`) as the **file permission bits**.

The following mask values are defined for the file type:

S_IFMT	0170000	bit mask for the file type bit field
S_IFSOCK	0140000	socket
S_IFLNK	0120000	symbolic link
S_IFREG	0100000	regular file
S_IFBLK	0060000	block device
S_IFDIR	0040000	directory
S_IFCHR	0020000	character device
S_IFIFO	0010000	FIFO

[1] Note the number system used.

Thus, to test for a regular file (for example), one could write:

```
stat(pathname, &sb);
if ((sb.st_mode & S_IFMT) == S_IFREG) {
        /* Handle regular file */
}
```

Because tests of the above form are common, additional macros are defined by POSIX to allow the test to be written more concisely:

```
S_ISREG(m)  //is it a regular file?
S_ISDIR(m)  //directory?
S_ISCHR(m)  //character device?
S_ISBLK(m)  //block device?
S_ISFIFO(m) //FIFO (named pipe)?
S_ISLNK(m)  //symbolic link?  (Not in POSIX.1-1996.)
S_ISSOCK(m) //socket?  (Not in POSIX.1-1996.)
```

The preceding code snippet could thus be rewritten as:

```
stat(pathname, &sb);
if (S_ISREG(sb.st_mode)) {
/* Handle regular file */
}
```

Then your program must process each file to copy or move. If a source file does not exist, the error message must be generated, and the next file must be tried.

To copy a file, you must use system calls that read and write big blocks (use `BUFSIZ` macro).

- A file should not be copied to itself.
- If the file with such a name already exists in the destination folder, permission for overwriting should be asked.

To move a file, the first try should be linking it to the new path. If it does not work (because the OS may block it or for another reason), you must copy this file and then delete the source.

- You can add this code to check why `link` does not work:

```
if( link( src, dst ) < 0 ){
        printf( "Can't link to directory %s\n", dst );
        perror( "link" );}
```

- Use unlinking to delete a file.

The program must output its actions.

EXAMPLES OF PROGRAM EXECUTION

```
./move
Usage: move source1 [source2 ...] destination


./copy MyFile.c NextFile.c ../backups/
MyFile.c NextFile.c successfully copied to ../backups
```

SUBMISSION, GRADING, AND ACADEMIC INTEGRITY

The project must be submitted on Blackboard. You have three attempts; please read *Programming Assignments Requirements and Recommendations* on Blackboard for suggested use of the attempts and **submission** package.

Please read *Programming Assignments Requirements and Recommendations* on Blackboard for the **grading** rubric.

Please read *Programming Assignments Requirements and Recommendations* on Blackboard for a strong warning on **cheating**.