Lab 6

Basic Linux Commands:

Low Level File Operations

The UNIX operating system kernel keeps track of open files for each process by using “file descriptors". File descriptors are non-negative integers. It is customary to have the first three file descriptors reserved for special purposes.

Descriptor	Purpose
0	Standard Input (keyboard by default)
1	Standard Output (screen by default)
2	Standard Error (screen by default)

A C programmer can perform low-level file operations using file descriptors. This approach is necessary for some of the exercises that will be performed below.

File Permissions

“File Permissions" indicate what operations a user is allowed to do with a file. There are three (3) different operations that are permissible:

- r (read)
- w (write)
- x (execute)

UNIX operating systems have separate file permissions for:

- User (i.e. owner of the file)
- Group (i.e. a group of users)
- Other (i.e. any other user than the above)

When you use the `ls` command to view the contents of a directory, it is possible to see the file permissions associated with each file.

Activity 1

- Open a shell window
- Enter the command `ls -l  [ENTER]`

You will notice that in the first column, the file permissions are displayed. Ignoring the first character, there are 3 groups of 3 characters. The first three represent the read, write and execute permissions for the User. The next three are for the Group and the last three are for Others.

Creating a File Using `creat()`

The `creat()` function is used to create a new file. This function takes two (2) parameters.The first is the name of the file to be created. The second in an integer that specifies the file permissions that will be used to create the file. The return value of `creat()` is an integer representing the file descriptor for the file created.

The permissions integer is usually specified as a 3-digit octal number, which is constructed as follows. The leftmost digit represents permissions for the User. The middle digit represents permissions for the Group. And the rightmost digit represents permissions for Others.

To understand how an octal digit can represent file permissions, the digit must first be written as a 3-digit binary number. When written as such, the bits represent the read, write and execute permission, where a 1 means \set" and a 0 means \not set".

Octal Digit	3-Digit Binary	Permissions
0	000	— — —
1	001	— — x
2	010	— w —
3	011	— w x
4	100	r — —
5	101	r — x
6	110	r w —
7	111	r w x

So for example if the second parameter to `creat()` is the octal number 0777, this would indicate that the file to be create should allow reading, writing and executing for users, group members and all others

Activity 2

- Create a new C program called `testcreat.c`.
- At the top of the source code, include the header file `fcntl.h`.
- Make a call to `creat()` to create a file called `"file.dat"`, with `rwx` permissions for the user, and `r--` permissions for the group and others.
- Capture the return value of the above function call in an integer called `fd` (short for file descriptor).
- Compile and run `testcreat.c`
- At the shell prompt, use the `ls -l` command to view the files. Search for the file you just created.

Closing a File Using `close()`

The `close()` function is used to close a new file. The only argument to this function is an integer representing the file descriptor of the file to be closed.

- Add a call to `close()` in your above code to close the file before exiting.

Opening a File Using `open()`

The open() function is used to open a file. There is more than one version of this function, each version takes a different number of arguments. Here, we are interested in the version that takes 2 arguments. The signature is:

```
int open(char *filename, int oflags)
```

The first parameter is the name of the file to be opened. The second is an integer specifying the mode in which to open the file. The most basic constants are:

- `O_RDONLY` - Open for reading only.
- `O_WRONLY` - Open for writing only.
- `O_RDWR` - Open for reading and writing.

The return value is a file descriptor.

Writing to a File Using `write()`

The `write()` function is used to write data to a file. Its signature is

```
int write(int fd, void *buff, int num_bytes)
```

The first argument `fd` is a file descriptor corresponding to the file where the data will be written. The pointer `buff` is a pointer to the place in memory where the data to be written is stored, and `num_bytes` is the number of bytes to write.

Activity 3

- Save a copy of your above code as `testwrite.c`
- Declare an integer `x` and initialize it to the value 5.
- After the call to `creat()`, add a `write()` statement to write the contents of the variable `x` to the file.
- Compile and run the program.
- Use the command `ls -l` to determine the size of the file you created. Is the size what you expect?

Reading from a File Using `read()`

The read() function is used to read data from a file. Its signature is

```
int read(int fd, void *buff, int num_bytes)
```

`fd` is a file descriptor for the file from which data will be read. `buff` is a pointer to a place in memory where the data will be stored. `num_bytes` is the number of bytes to read from the file.

Activity 4

- Save a copy of your above code as `testread.c`.
- Remove the call to `creat()`.
- In its place call `open()` to open the `file.dat` for reading.
- Declare an integer `y`, but do not initialize it.
- After the `open()` statement, use `read()` to read an integer from the file into the variable `y`.
- Print the value of `y`.
- Compile and run `testread.c`.

Re-directing a Process' Output Stream

By default any given process has three “streams" associated with it. These are, an input stream, an output stream and an error stream. The input stream is by default the keyboard. The output stream is by default the screen, and the error stream is also by default the screen.

As noted before, these streams are identified by the file descriptors 0, 1 and 2 respectively.

Thus reading from the keyboard is actually conceptualized as reading from a “file". Similarly, writing to the screen is conceptualized as writing to a “file”. It is possible to change these streams by manipulating either process' file descriptors.

Activity 5

- Download the source code `redirect_output.c`.
- Compile and run the program.
- Use the following command to view the output of the program, which would have been redirected to the file `output.txt`

```
cat output.txt [ENTER]
```

You will notice that the code for the last program made use of a function called `dup2()`. This function is used to duplicate file descriptors. It takes two arguments. The second is the file descriptor to be duplicated, and the first is another file descriptor into which the duplicate will be made. The statement `dup2(fd1, fd2);` closes `fd2`, and duplicates `fd1` into its place.

In the source code of `redirect_output.c`, the statement `dup2(fd, 1);` is used. This closes file descriptor 1 (i.e. the screen output stream) and duplicates the file descriptor `fd` into its place. Thus anything that the program prints that would normally go to the screen would instead go to the file associated with `fd`.

Re-directing a Child's Output

When a process spawns a child process, both the parent and the child share the same input, output and error streams. Thus, output produced by the parent and child can become intermixed and hence confusing to read.

This problem can be solved by redirecting the output of the child, or the parent, or both.

Activity 6

- Download the source code `redirect_parent_child_output.c.`
- Compile and run the program.
- View the output of the child and the parent in their respective files.

Using Pipes for Inter-Process Communication

One of the methods available for inter-process communication in UNIX is the pipe. A pipe can be thought of as a communication link. There are two “ends" on a pipe, the “read end” and the “write end”.

One process can put data into the pipe at the write end, and another process can read data from the pipe from the read end.

The `pipe()` function is used to create a pipe. It takes a single argument which should be a size-2 integer array. Once the call to `pipe()` has been made, index 0 of the integer array contains a file descriptor corresponding to the read end of the pipe and index 1 contains a file descriptor corresponding to the write end of the pipe.

Activity 7

- Download the source code `parent_child_pipe.c`.
- Compile and run the program.
- Modify the program so that the `for` loop runs up to 100000. Save your program as `parent_child_pipe_2.c`
- Note that you will need to modify the number of bytes read and written in each step.
- Compile and run `parent_child_pipe_2.c`
- Modify the code further so that the parent and child output to files called `output1.txt` and `output2.txt` respectively.

Additional Basic Linux File Operations

- Create a new file: `cat > list.txt`
  ```
  1.   Roald Dahl Author
  2.   Penguin Books Publisher
  ```
  Press <Ctrl + D> to save the file and exit the command

- Create a new directory: `mkdir D1`
  This creates a new directory called `D1`

- Remove a file or directory: `rm -r D1`
  This removes the directory called `D1`

- Copy a file or directory (1): `cp list.txt D2`
  Copies the file `list.txt` to the folder `D2`. The created file has the name `list.txt`

- Copy a file or directory (2): `cp list.txt D2/list2.txt`
  Copies the file `list.txt` to the folder `D2`. The created file has the name `list2.txt`

- Copy a file or directory (3): `cp -r D1/* D2`
  Copies the files, directories and subdirectories in the folder `D1` to the folder `D2`. You do not see a folder named `D1` in the folder `D2`.

- Copy a file or directory (4): `cp -r D1 D2`
  Copies the folder `D1` to the folder `D2`. You see a folder named `D1` in the folder `D2`.

- Move or rename a file (1): `mv test1.txt test2.txt`
  Changes the name of the file `test1.txt` to a new name `test2.txt`.

- Move or rename a file (2): `mv student.txt D1`
  Moves the file `student.txt` to the folder `D1` without changing the file name.

- Move or rename a file (3): `mv student.txt D1/student2.txt`
  Moves the file `student.txt` to the folder `D1` with a new name.

- Move or rename a file (4): `mv D1/student.txt /home/comp3100`
  Moves the file `student.txt` in the folder `D1` back to the parent folder `comp3100`..

- Display a file on screen (1): `cat list.txt`
  Displays the file content on the screen.

- Display a file on screen (2): `head list.txt`
  Displays the first ten lines of the file content on screen.

- Display a file on screen (3): `tail list.txt`
  Displays the last ten lines of the file content on screen.

- Display a file on screen (4): `more list.txt`
  Displays screen-sized portion of the file content on screen. Press spacebar key to display the next portion.

- Sort the lines of text in a file: `sort -k2 list.txt`
  ```
  2.   Penguin Books Publisher
  1.   Roald Dahl Author
  ```
  Sorts on the second key (i.e., column 2). The sorted result is displayed on screen.

- Output the sorted lines of text in a file: `sort -k2 list.txt > sList.txt`
  ```
  2.   Penguin Books Publisher
  1.   Roald Dahl Author
  ```
  Sorts on the second key (i.e., column 2). The sorted result is stored in the file `sList.txt`

- Print lines matching a pattern: `grep "Roald" list.txt`
  ```
  1.   Roald Dahl Author
  ```
  Searches for the string "Roald" in the file `list.txt`.

- Display the location of a file.: `whereis gcc`
  ```
  /usr/bin/gcc
  ```
  Shows that the C compiler named `gcc` is located in folder `/usr/bin`.