**CST–321 File System**

**Activity Directions:**

In this assignment, you will do research on file systems and the Linux system file I/O functions. The following are the tasks you need to complete for this assignment:

1. Explore the Linux File System in your VirtualBox Ubuntu installation. Document the purpose (2–3 sentences) for each of the following directories.

|  |  |
| --- | --- |
| **Directory** | **Purpose** |
| / |  |
| /bin |  |
| /dev |  |
| /etc |  |
| /lib |  |
| /boot |  |
| /home |  |
| /mnt |  |
| /proc |  |
| /tmp |  |
| /usr |  |
| /var |  |
| /sbin |  |

1. Using the Ubuntu *Files* (i.e., file explorer) application, go to your Home directory. For each top-level directory, list the name of the directory and document its purpose (2–3 sentences).

|  |  |
| --- | --- |
| **Directory** | **Purpose** |
|  |  |

1. Given the following scenario, draw a flowchart of the flow and tasks to support the scenario. Your flow and tasks need to clearly identify the Linux system I/O calls that would be used to support the tasks in your flow by showing example code snippets for each of the Linux system I/O calls. Your tasks must document the complete functionality. For example, if you are reading a file, you must show the decisions and associated tasks to read the file completely (i.e., a loop in a flowchart), along with the parameters required for each system I/O call. You must indicate the mode by which your file is being opened and you must close all files when you are done accessing them. No code is required to be developed in this assignment, but your code snippets should be complete and include thorough details to demonstrate a complete understanding for each of the Linux system I/O calls.

**Scenario:** Simulate the tasks required to read a C program and compile the program. The C program name, input as a program argument, will be passed to a compiler 'gcx' to compile the input program. Assume the 'gcx' compiler is in your environment path. The 'gcx' compiler will read one text line (up to a CR/LF or EOF in the file) at a time from the input text-based C program, parse the input text program into compiled tokens, and write the binary tokens to a 'tmp' directory in a file called 'tokens.o'. The 'tmp' directory will be relative to the directory of the C program. All parsed tokens from the C program will be written to a 'tokens.o' binary output file. Once all the tokens are written to its binary output file, the 'gcl' linker will be invoked to assemble the 'tokens.o' file, along with a dependent library, 'system.out', located in the '/lib' directory, to produce a final binary executable file. Assume the 'gcl' linker is in your environment path. The output binary executable will be created with the same name as the C program except with an 'out' extension. Linking is done by reading each .o file one at a time using a processing buffer of 1024 bytes and concatenating the contents of one file to the end of the previous file to produce the output binary file. Once all of the binary data has been written to the output file, a checksum will be calculated and put in the header of the file. The checksum will be written as a 64-bit binary number at the beginning of the output file to produce the final output file. The final output file will be written relative to the directory of the C input text program. All files in the 'tmp' directory will be removed once the linker has finished producing the final output file.

The following are the Linux/Unix system file and directory I/O calls that need to be used in your solution:

Creat: int creat(char \*pathname, mode\_t mode);

o Create a new file and assign a file descriptor

Open: int open(char \*pathname, int flags, mode\_t mode);

o Open the file pathname and return a file descriptor

Close: int close(int fd);

o Close a file descriptor fd

Read: int read(int fd, void \*buf, int count);

o Read up to count bytes from fd, into the buffer at buf

Write: int write(int fd, void \*buf, int count);

o Writes up to count bytes into fd, from the buffer at buf

Lseek: int lseek(int fd, int offset, int whence);

o Assigns the file pointer to a new value by applying an offset.

int mkdir(const char \*pathname, mode\_t mode);

o Creates a directory pathname.

DIR \*opendir(const char \*name);

o Opens a directory stream corresponding to the directory name.

int rmdir(const char \*pathname);

o Removes a directory pathname.

int closedir(DIR \*dirp);

o Closes and releases resources to an open directory.

char \*getcwd(char \*buf, size\_t size);

o Copies an absolute path name of the current working directory.

You will want to refer to your textbook readings in Chapter 4, internet resources, and the in-class activities as resources to support this assignment. If necessary, Google "good flowchart tutorials" to discover online tutorials and best practices for producing flow charts.

Deliverables:

1. Cover sheet with your name, the name of this assignment, and the date.
2. Table from Exercise 1.
3. Table from Exercise 2.
4. Flowchart for the simulated C compiler.
5. Package all of the above as a single Microsoft Word document and upload it to the digital classroom.