CP386: Assignment 1

This is a group assignment designed to practice the concept of system calls and implement basic functionalities of an Operating System. The focus is on teamwork and the effective application of system-level programming concepts.

General Instructions:

- **Compiler:** Test your program thoroughly with the GCC compiler.
- **Compilation:** If your code does not compile, you will score zero. Ensure all syntax errors are removed.
- **Plagiarism:** Submitted code will be checked for plagiarism. By submitting, you confirm that you have not received unauthorized assistance and are aware of course policies regarding submitted work.
- **Requirements:** Marks will be deducted if these instructions are not followed.
- **Submission:** Multiple attempts are allowed; however, the last submission before the deadline will be graded. Points may be deducted for not following directions.

Warning:
Strictly adhere to the specified file names and function names as the assignment will be auto-graded. Non-compliance will result in a grade of zero.

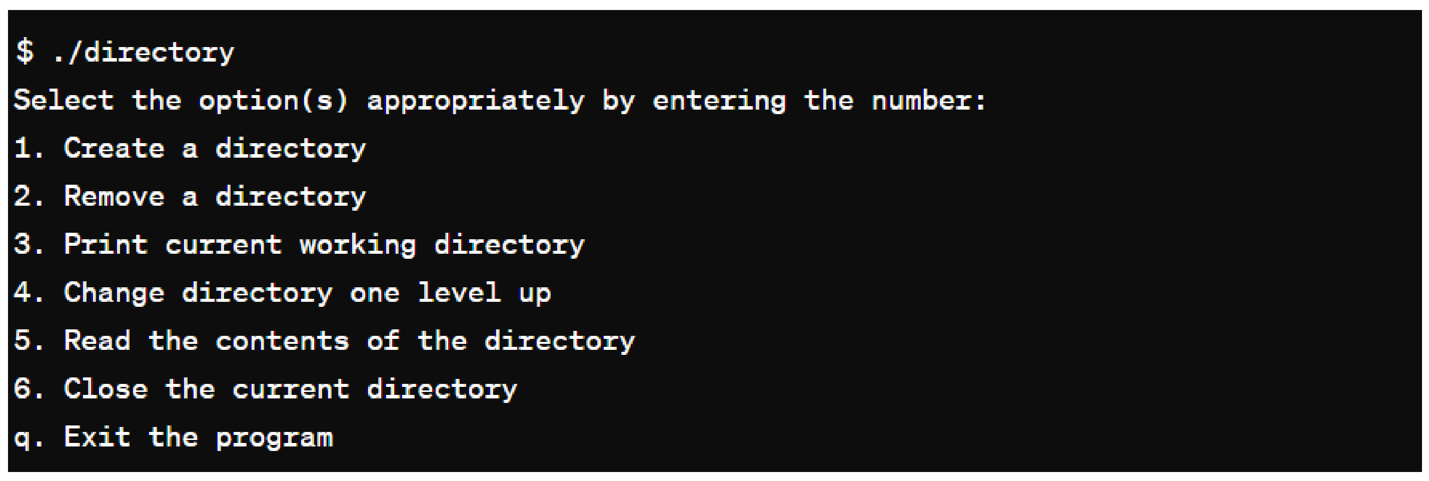
Question 1
Write a C program named **directory.c** that enables various directory operations, enhancing understanding of file system manipulations and system calls. Your program should support the following operations:

1. Creating a directory
2. Removing a directory
3. Getting the current working directory
4. Changing to the parent directory
5. Reading the contents of the current directory
6. Closing the opened directory

Instructions:

- Continuously display a menu with options until the user decides to exit by entering "q".
- Implement each directory operation as a function within your program.
- For creating and removing directories, prompt the user for the directory name.
- Ensure your program handles errors appropriately (e.g., attempting to remove a non-existent directory).
- Include a Makefile for compiling the program.

User Interaction: When (“**make run”** or “**./directory”)** is executed, the program should display the following interaction pattern:



```
$ ./directory
Select the option(s) appropriately by entering the number:
1. Create a directory
2. Remove a directory
3. Print current working directory
4. Change directory one level up
5. Read the contents of the directory
6. Close the current directory
q. Exit the program
```

Output Expectations: The program should clearly print the results of each operation, including any errors encountered. After performing an operation related to the directory change or reading, ensure the output reflects the current state or contents of the directory.

Submission: Submit the “**directory.c”** file and the Makefile in your course’s designated submission area. Ensure your code is well-commented and explain the logic of each operation.

Question 2

Create a C program named **filecopy.c** that copies the contents of one file to another. This will help you understand file handling and system call utilization.

Requirements:

1. **File Names as Input:**
   - The program should start by asking the user for the names of the input and output files.
   - Ensure the program checks for the correct number of input arguments. If the necessary arguments are not supplied, print "Insufficient parameters passed."

2. **Opening Files:**
   - Open the input file for reading. If the file does not exist or cannot be accessed, print an appropriate error message and terminate the program abnormally.
   - Create and open the output file for writing. If an output file with the same name exists, it should be overwritten.

3. **Copying Content:**
   - Read from the input file and write to the output file in a loop until the end of the file is reached.
   - Handle possible read and write errors, such as hardware failures or insufficient disk space.

4. **Error Handling:**
   - Implement comprehensive error handling for each system call, including opening, reading, and writing files.

5. **Use of System Trace Utilities:**
   - Upon completing the program, use **strace** on Linux or **dtruss** on macOS to trace the system calls made by your program. Ensure you have administrative privileges if required.

Output Expectations:

1. **No Input Arguments:**
   - **./filecopy** should output: "Insufficient parameters passed."
2. **Successful Copy:**
   - **./filecopy input.txt output.txt** should output: "The contents of file 'input.txt' have been successfully copied into 'output.txt' file."
3. **System Calls Trace:**
   - Using **strace -c ./filecopy input.txt output.txt** on Linux or the equivalent on macOS to trace and count system calls, providing a summary of the operations performed.

Development Notes: Test your program thoroughly to ensure it handles all specified error conditions gracefully.

Submission: Submit the **filecopy.c** file in your course's designated submission area. Ensure your code is well-commented and explain the logic of each operation.

Usage Instructions for Students:

Make sure to use the provided Makefile for compiling and running your program.

1. **Compile Individual Program**:
   - To compile just the **directory** program, run **make directory**.
   - To compile just the **filecopy** program, run **make filecopy**.

2. **Clean the Build**:
   - Run **make clean** to remove all compiled files and reset the build environment.

Rubric for Grading:

- **Functionality (70%)**: Correct implementation of all specified functionalities.
- **Error Handling (10%)**: Comprehensive and correct error handling.
- **Code Quality (10%)**: Code readability, and use of comments.
- **Submission Compliance (10%)**: Adherence to submission instructions, file naming, and documentation requirements.