CS3.304 – Advanced Operating Systems

Assignment 1

Deadline: August 14, 2025, 11:55 PM

Q1: File Reversal with Multiple Modes

Overview

Given a file, you need to reverse its contents using different reversal strategies and store the result in a new file in the directory named `Assignment1`.

General Requirements

- The percentage of the file written should be printed on the console during file writing and **overwritten** each time (i.e., it should not be written multiple times).
- The directory created should have **read, write, and execute permissions** for the user who created it.
- The new file created should have **read and write permissions** for the user who created it.
- The program will be tested on **LARGE (>1GB) files** which may be larger than the RAM size.

Command Line Usage

```
$ gcc Q1.c / g++ Q1.cpp
$ ./a.out <input file name> <flag> [additional arguments]
```

Flag Operations

Flag 0: Block-wise Reversal

- Reverses the file in fixed-size blocks (**block size must be specified as the 3rd argument**).
- Each block is reversed individually, but blocks remain in original order.
- **Usage:** `./a.out <input_file> 0 <block_size>`
- **Output file:** `Assignment1/0_<input_file_name>`

- **Example:** If `A.txt` contains `ABCDEFGH` and block size 4, output is `DCBAHGFE`.
- **Complexity:** Simplest reversal pattern with sequential block processing

Flag 1: Full File Reversal

- Reverses the entire file contents globally.
- First byte becomes last, last byte becomes first, etc.
- **Usage:** `./a.out <input_file> 1`
- **Output file:** `Assignment1/1_<input_file_name>`
- **Example:** `Hello, World!` becomes `!dlroW ,olleH`.
- **Complexity:** Requires efficient memory management and seeking for large files

Flag 2: Partial Range Reversal

- Requires start and end indices as command-line arguments.
- Reverses two parts: from beginning to start index, and from end index to end of file.
- The portion between start and end indices remains unchanged.
- Indices are zero-based; assume the first character is at index 0.
- **Usage:** `./a.out <input_file> 2 <start_index> <end_index>`
- **Output file:** `Assignment1/2_<input_file_name>`
- **Example:** Input `Hello, World!`, `2 2 7` outputs `eHllo, W!dlro`.
- **Complexity:** Most challenging - requires precise index handling, multiple seek operations, and complex memory management for three separate regions

Summary of Usage:

- `./a.out <input_file> 0 <block_size>`
- `./a.out <input_file> 1`
- `./a.out <input_file> 2 <start_index> <end_index>`

Q2: File and Permission Verification

Purpose

Write a program to:

1. Check the permissions for the two files and the directory.
2. Check whether the content in the new file follows the expected reversal pattern **based on the flag used in Q1**.

Input

Paths for the new file, old file, directory, flag used, and additional arguments (block size or indices) should be passed as command-line arguments.

```
./a.out <newfilepath> <oldfilepath> <directory> <flag> [<blockSize>|<start> <end>]
```

- For flag 0: 5 arguments (`flag 0, block_size`)
- For flag 1: 4 arguments
- For flag 2: 6 arguments (`flag 2, start, end`)

Verification Logic by Flag

- **Flag 0 (Block-wise Reversal):**
  - Read both files in blocks of the specified size.
  - For each block, check that the new file's block is the reverse of the old file's block; blocks must appear in the same order.
- **Flag 1 (Full File Reversal):**
  - Read from the start of the original and from the end of the new file in chunks.
  - Compare each chunk from the original with the corresponding reversed chunk from the new file.
- **Flag 2 (Partial Range Reversal):**
  - Check that bytes [0, start_index-1] in the new file are the reverse of [0, start_index-1] in the old file.
  - Check that bytes [start_index, end_index] are identical in both files.
  - Check that bytes [end_index+1, EOF] in the new file are the reverse of the corresponding bytes in the old file.

Expected Permission Codes

The program should verify that files and directories have the following specific permission codes:

- **Directory (Assignment1/)**: `700`
  - User: read(4) + write(2) + execute(1) = 7
  - Group: no permissions = 0
  - Others: no permissions = 0
- **New file (Assignment1/X_input.txt)**: `600`
  - User: read(4) + write(2) + no execute(0) = 6
  - Group: no permissions = 0

  - Others: no permissions = 0

- **Original input file**: `644` (typical default)
  - User: read(4) + write(2) + no execute(0) = 6
  - Group: read(4) + no write(0) + no execute(0) = 4
  - Others: read(4) + no write(0) + no execute(0) = 4

Output Format

```
Directory is created: Yes
Whether file contents are correctly processed: Yes
Both Files Sizes are Same: Yes
User has read permissions on newfile: Yes
User has write permission on newfile: Yes
User has execute permission on newfile: No
Group has read permissions on newfile: No
Group has write permission on newfile: No
Group has execute permission on newfile: No
Others has read permissions on newfile: No
Others has write permission on newfile: No
Others has execute permission on newfile: No
```

For each: new file, old file, and directory (total 30 lines).

Usage Examples

- For flag 0: `./a.out Assignment1/0_input.txt input.txt Assignment1 0 1024`
- For flag 1: `./a.out Assignment1/1_input.txt input.txt Assignment1 1`
- For flag 2: `./a.out Assignment1/2_input.txt input.txt Assignment1 2 5 10`

Technical Guidelines

1. Assignment must be coded in **C++ only**.
2. All programs must use **system calls only**.
3. Useful system calls: `read`, `write`, `lseek`, `stat`, `fflush`, `perror`, `mkdir`, `open`, `close`.
4. Use **man pages exclusively** for system call documentation.
5. Use of system commands like `ls`, `cp`, `mv`, `mkdir`, etc., is **NOT allowed**. Implement using system calls only.
6. Modularize and indent your code properly; add comments for readability.
7. Handle error cases as appropriate wherever required.
8. Add a **README.md** file (compulsory) containing execution instructions and explanation of the code workflow.

Academic Integrity

ZERO tolerance towards plagiarism of any kind.

- **Peer plagiarism is strictly prohibited.** Do not copy from or share your code with classmates or others. Even sharing a few lines will result in a **zero for both parties**.
- **AI-generated code and online sources are not permitted.** Do not copy code from AI tools or websites like GitHub, etc. Submissions found using such sources will be penalized.
- **Each submission must reflect your own original work.**
- All submissions will be run through plagiarism detection tools (including peer and AI-based checks).

Submission Format

Directory Structure

```
RollNumber_A1/
     RollNumber_A1_Q1.cpp
     RollNumber_A1_Q2.cpp
     README.md
```

Replace "RollNumber" with your actual roll number.

How to Submit

- Create a zip file named `RollNumber_A1.zip` containing the `RollNumber_A1` folder.
- Submit this zip file on Moodle.
- **Strict deadline:** Any submissions after the deadline will receive **0 marks**.
- **Format compliance:** Submissions not following the specified format will receive **0 marks**.
- Submission method: Only Moodle submissions accepted (no email/teams).

Generation of Test Files

A cpp file for generating random string test files will be provided separately with instructions for creating large files.

Note

This assignment tests your understanding of low-level file operations, handling large files, permission management, and robust system-level programming. Plan your approach carefully!

Marks Distribution Summary

- **Q1 Total: 60 Marks**
  - Error Handling: 5 marks
  - Progress & Verbosity: 5 marks
  - Flag 0 (Block-wise): 12 marks
  - Flag 1 (Full File): 18 marks
  - Flag 2 (Partial Range): 20 marks
- **Q2 Total: 40 Marks**
  - Logic Implementation: 25 marks
  - Permission Verification: 10 marks
  - Output Formatting: 5 marks