Course COMP-8567
Project: Distributed File System (using socket programming)
Summer 2025
Due Date: Apr/13/2025, 11PM EDT
100 Marks
Plagiarism Detection Tool: MOSS

Associated Learning Outcomes:

- Apply OS concepts to design algorithms to solve systems programming problems in a variety of different systems, such as Unix/Linux/Android environments.
- Correctly define systems programming problems and identify and apply appropriate solutions approaches.
- Design and implement solutions that use the hardware and/or kernel services to solve systems programming problems involving the latest computing technologies.
- Interpret informal written descriptions of systems programming problems, and create clear, formal design specifications from them
- Write reports and software documentations for problems and solutions to be used by others
- Recognize and identify potential growth areas in operating systems' use and propose original ideas to create future applications.

Note: Please check the following link for the **complete list** of learning outcomes for COMP 8567
https://ctl2.uwindsor.ca/cuma/public/courses/pdf/ee1b450a-23a6-4635-b0c6-40a47a21331f

Instructions

- The project work can be carried out alone or in teams of two students.
- Only students from the same section can form a team.
- In case of a team, each team member is expected to contribute evenly (in reasonable terms) towards the development of the project.
- Along with the file submission, the working of the project must be demonstrated during the scheduled slot (TBA) which will be followed by a **viva**.
    - In case of a team, the working of the project must be demonstrated individually by team members as per the stipulated schedule.
    - Demo slots can be scheduled anytime on **Apr 14th ,15th and 16th** and will be announced suitably ahead of time.

Introduction

In this project, you are required to implement to a **distributed file system** through socket programming.

The distributed file system has four servers:

- **S1**
- **S2**
- **S3**
- **S4**

and can support multiple client connections.

Section A – Servers : S1, S2 , S3 and S4

Clients are allowed to upload/store four file types (.c,.pdf,.txt and .zip) onto **S1**, however **S1** only stores .c files locally and **transfers** all **.pdf files** to the **S2 sever**, **.txt files** to the **S3 sever** and **.zip files** to **S4** server (all in the background). Clients are not aware of this operation and assume all files are stored at **S1**.

All clients communicate with **S1 only** and are **not aware** of the presence of **S2**,**S3 and S4**.

- Upon receiving a connection request from a client, **S1** forks a child process that services the client request exclusively in a function called prcclient() and (**S1**) returns to listening to requests from other clients.
  - The prcclient() function enters an infinite loop waiting for the client to send a command
  - Upon the receipt of a command from the client, prclient() performs the action required to process the command as per the requirements listed **in section B** and returns the result to the client
- **S2,S3 and S4** act as servers to **S1** and service its requests based on the commands entered in **w25clients (Section B)**

Note:

- The servers **S1, S2, S3, S4** and w25clients process/es must run on different machines/terminals and must communicate using sockets only.
- Files in **S1** must be saved under ~/S1
- Files in **S2** must be saved under ~/S2
- Files in **S3** must be saved under ~/S3
- Files in **S4** must be saved under ~/S4

Section B (w25clients)

The client process runs an infinite loop waiting for the user to enter one of the commands.

Note: The commands **<u>are not</u>** Linux commands and are defined (in this project) to denote the action to be performed by the **S1**.

Once the command is entered, the client verifies the **syntax of the command** and if it is okay, sends the command to **S1**, else it prints an appropriate error message.

Client Commands : (5 commands) Done

uploadf *filename destination_path*

Transfers (uploads) *filename* from the PWD of the client to S1

- filename : valid filename (.c /.pdf/ .txt/.zip) in client's PWD
- *destination_path*: A path **in S1** //must belong to ~/S1 of the main server
    - if destination path is not already present in the main server, it must be newly created
    - Only .c files are stored in the main server (but the user is not aware of it)
    - **pdf files are transferred from S1 to S2** and are stored in the corresponding folders in the S2 server (replace S1 with S2)
    - **.txt files are transferred from S1 to S3** and are stored in the corresponding folders in the S3 server (replace S1 with S3)
    - **.zip files are transferred from S1 to S4** and are stored in the corresponding folders in the S4 server (replace S1 with S4)
    - **<u>Note:</u>** All files <u>non .C</u> files are **deleted** in S1 after transferring them to S2/S3/S4

Examples:

Note: In all the examples below, the client should <u>initially transfer</u> the specified file **to S1** and S1 takes further action as indicated in the comments

- **w25clients$ uploadf sample.c ~S1/folder1/folder2** //should store **sample.c** in the specified folder on the **S1** server
- **w25clients$ uploadf sample.txt ~S1/folder1/folder2** // **S1** transfers sample.txt to the **S3** server and the **S3** server in turn stores **sample.txt** in **~S3/folder1/folder2** //User assumes sample.txt is stored in **S1**, but all text

files must actually be stored in the S3 server in the corresponding path (replace ~S1 with ~S3)

- w25clients$ **uploadf sample.pdf ~S1/folder1/folder2** // S1 transfers sample.pdf to the S2 server and the S2 server in turn stores **sample.pdf** in **~S2/folder1/folder2** //User assumes sample.pdf is stored in S1, but all pdf files must actually be stored in the S2 server in the corresponding path (replace ~S1 with ~S2)
- w25clients$ **uploadf xyz.zip ~S1/folder1/folder2** // S1 transfers xyz.zip to the S4 server and the S4 server in turn stores **xyz.zip** in **~S4/folder1/folder2** //User assumes xyz.zip is stored in S1, but all .zip files must actually be stored in the S4 server in the corresponding path (replace ~S1 with ~S4)

- **Note: Clients can directly communicate with S1 only and are not aware of the presence of S2 and S3 servers**

downlf filename

✓ [illegible]

Transfers (downloads) *filename* from S1 to the PWD of the client

- filename : valid path of a file in S1 (.c /.pdf/ .txt files only)
  - If the request is for a .c file, S1 processes the request (locally) and sends the corresponding file to the client
  - If the request is for a .pdf file, S1 obtains the file from S2 and then sends the corresponding file to the client
  - If the request is for a .txt file, S1 obtains the file from S3 and then sends the corresponding file to the client
  - If the request is for a .zip file, S1 obtains the file from S4 and then sends the corresponding file to the client

Examples:

- w25clients$ **downlf ~S1/folder1/folder2/sample.c** // S1 processes the request (locally) and sends sample.c to the client
- w25clients$ **downlf ~S1/folder1/folder2/sample.pdf** // S1 obtains sample.pdf from the corresponding directory in S2 and then sends sample.pdf to the client

- w25clients$ **downlf ~S1/folder1/folder2/sample.txt** // S1 obtains sample.txt from the corresponding directory in S3 and then sends sample.txt to the client
- w25clients$ **downlf ~S1/folder1/folder2/xyz.pdf** // S1 obtains xyz.pdf from the corresponding directory in S3 and then sends xyz.pdf to the client

✓ removef filename ✓ done

Removes (deletes) *filename* from S1 to the PWD of the client

- filename : valid path of a file in S1 (.c /.pdf/ .txt files only)
  - If the request is for a .c file, S1 processes the request (locally) and deletes the corresponding file
  - If the request is for a .txt file, S1 sends a request to S3 to delete the text file in the corresponding directory.
  - If the request is for a .txt file, S1 sends a request to S3 to delete the text file in the corresponding directory.

Example:

w25clients$ **removef ~S1/folder1/folder2/sample.pdf** // S1 requests S2 to delete sample.pdf in the corresponding directory

✓ downltar filetype ✓ done

Creates a tar file of the specified file type and transfers (downloads) the tar file from S1 to the PWD of the client

- **Filetype: .c/.txt/.pdf (Does not include .zip)**
  - If the filetype is .c , S1 creates a **tar file (cfiles.tar) of all .c files present in the directory subtree** rooted at ~/S1 and sends the tar file to the client
  - If the filetype is .pdf , S1 requests and obtains pdf.tar **of all .pdf files present in the directory subtree** rooted at ~/S2 from the S2 server and sends pdf.tar to the client

    - If the filetype is .txt , S1 requests and obtains text.tar **of all .txt files present in the directory subtree** rooted at ~/S3 from the S3 server and sends pdf.tar to the client

dispfnames *pathname*

Displays the names (only) of <u>all files</u> that belong to *pathname* in S1 to the PWD of the client

- pathname : valid path of a directory in S1 that belongs to ~/S1
    - S1 obtains the list of all .pdf, .txt and .zip files (if any) from the <u>corresponding</u> directories in **S2**,**S3 and S4**
    - S1 then combines the list obtained in the previous step with the list of .c files present locally in *pathname* and transfers the consolidated list of .c,.pdf, .txt and .zip files (in that order) to the client
        - Additionally, files within a file type group must be listed alphabetically
        - //Please Note: only the names of files along with their extensions are transferred to the client and not the actual files

Submission Instructions:

- Comments must be included to explain the working of the program
- The program must reasonably handle error conditions based on the requirements

Plagiarism Detection Tool: MOSS

You are required to **<u>submit 5 files</u>**.

1. S1.c
2. S2.c
3. S3.c
4. S4.c
5. w25clients.c