**Department of Computing**

# 

**CS 330: Operating Systems**

**BSCS: 10ABC**

**Final Lab Exam: Distributed File Management System**

**Date: 22 Dec 2022 & 23 Dec 2022**

**Time: 10:00 AM-01:00 PM & 02:00 PM-05:00 PM**

**Instructor: Dr. Farzana Jabeen (AB) & Dr. Sana Qadir (C)**

**Final Lab Exam**

**Instructions**

* You are allowed to form groups of 2 students.
* The code should be developed within the team and any form of plagiarism will automatically result in zero for both the groups involved without any questions asked.
* Each student is responsible to understand the code being submitted under his or her name. Division of work is not explicitly required, however, each group member will be given average viva marks of the group. This implies that if a group member gets zero in viva and another gets full marks then both will score 50%.
* The submission deadline is midnight before next lab. Assessment will be done based on presentation and live demo.
* Any questions and comments on the lab must reach the lab engineer no later than a day before the deadline through email or in person meeting. No questions will be answered later.

**Introduction**

The purpose of this activity is for you to:

(1) Build the **server for the file management component of distributed file management system** you built in previous labs. The goal is to build the structure for file management to provide access to user(s) to create, delete, update and query files in the system remotely by using threads. You are also free to make the design choices as discussed in book or a combination thereof as long as the requirements for the system is met.

(2) Convert the above client-server system into a **multi-user system with proper synchronization** applied to it. In this task we will protect the files from **reader writer problem**. The description of reader-writer problem is described in section 7.1.2 of your textbook, Operating System Concepts, 10th Edition by Avi Silberschatz et al. You may use monitors, locks or semaphore libraries provided by the language APIs to implement the tasks.

You are allowed to build the system in the language of your choice and in the operating system of your choice as long as the objective of the lab are met.

**Objectives**

By the end of this exercise you will be able to build a server to provide file structure and understand the operations on how to make a server by using threads and socket programming. Also you will learn practical uses of synchronization and implement it to solve reader-writer problem.

**Tasks**

**Task 1**

Before you start your main task, go through the links below to learn how to make a server. You can use code from the following links to understand what a server is and how it works. The codes (in different languages) as a sample to implement socket programming for sending and receiving simple messages is given below.

* Socket programming in Python <https://www.journaldev.com/15906/python-socket-programming-server-client>
* Socket programming in Java <https://www.journaldev.com/741/java-socket-programming-server-client>
* Socket programming in C++ <https://www.educba.com/socket-programming-in-c-plus-plus/>

This task has two further subtasks which are interrelated.

1. **Design a protocol** on top of your previous work in i.e., File Management and Threaded File Management System. A protocol is a pre-defined system of rules to exchange information. Here the client will be calling functions at the server. That means that **server is the file server you implemented and the client will be remotely executing the tasks on the server machine**. A protocol may mean that you assign numbers to each of the function and use some sort of symbols to separate the parameters or some other scheme. You may read some of the common protocols such as POP3 or SMTP to understand how a protocol work.
2. **Implement this protocol through server and provide a client program to the user**. The **server program** will receive the message through socket, execute the task and return the response. The **client side program** will provide an interface to the user where different operations can be executed and results shown.

**Task 2**

This task has two subtask as well however you may choose and implement **any one of the two.** The task here is to implement mutual exclusion of file access. Multiple threads can attempt to access your files but synchronization can be achieved by

1. **Implement a queue for readers**. Multiple threads can attempt to read a file and some can request a write.
2. Your task is to make sure that while the file is being read, no writer is allowed while maintaining the order of writes.
3. Any request to open the file in write mode is considered to be in active writing till the user closes the file.
4. Similarly any request to open the file in read mode is considered to be in active read till the user closes the file.
5. Multiple users can read the file concurrently but writes will be mutually exclusive.
6. **Implement a limit on the user name to access a file.**
7. Multiple users can access your system.
8. Though it is not required that you implement any type of security, but any user may not be able to access more than 5 files at one time.
9. If more than 5 requests are placed then the requesting thread must wait.
10. Each file can only be accessed by 3 users, be it for read or write.

**Requirements**

* + - 1. You must provide two programs, a server and a client.
      2. The client must allow the user to specify the IP address of the server at the time of connection.
      3. The client will allow user to first specify the user name at the connection setup time and all communication should display it.
      4. The client must provide an interface to apply the operations developed in the previous labs.
      5. The client must give errors when the server is not available.
      6. The client must display the response of the actions performed.
      7. The server must respond to multiple requests at the same time (this will require threads)
      8. The server must bind to port 95 whereas the client port can be any number higher than 1024.
      9. The server and clients can run on different machines.

**Deliverables**

**Submit**

1. Complete code
2. Sample data file (sample.dat) consisting of files and directories to show your output and memory map.
3. Presentation slides
   1. Brief about your implementation of the previous labs i.e., File Management and Threaded File Management System.
   2. Development of protocol and establishment of client-server system
   3. Synchronization primitives used and implementation details and reason of your choice.
   4. Testing and debugging

**Evaluation** inNext Lab

Presentation 5 mins

Live Demo: 5 mins