**THERE IS A ZERO TOLERANCE CHEATING POLICY**

**ANY HONOR CODE VIOLATION – NO MATTER HOW SLIGHT – WILL RESULT IN AN F IN THE COURSE AND REFERRAL TO THE OFFICE OF STUDENT CONDUCT**

**CSE 5306** **Lab # 3** **Fall 2020**

**Objectives**

1. Exposure to sequence-dependent operations.
2. Further experience with thread management.

**Project Specification**

This lab is intended to be built upon Lab #2. While it is not a requirement that this lab utilize your code from Lab #2, all of the functionality from Lab #2 must be included in Lab #3.

These will be **individual** projects. You may write the program in any language that is supported under any Integrated Development Environment (IDE). Keep in mind that available controls, objects, libraries, et cetera, may make some of these tasks easier in one language than in another. Finally, because of the lack of restrictions on IDEs, you will have to have that IDE available to demo to the TA (e.g., you will demo the program on your own laptop).

All components should be managed with a *simple* GUI. The GUI should provide a way to kill the process without using the 'exit' button on the window.

**Lab #1 and #2 Infrastructure**

You will write a centralized directory management system consisting of a server and three client processes. Each client process will connect to the server over a socket connection and register a username at the server. The server should be able to handle all three clients simultaneously and display the names of the connected clients in real time.

Two or more clients may not use the same username simultaneously. Should the server detect a concurrent conflict in username, the client's connection should be rejected, and the client's user should be prompted to input a different name.

Upon connection to the server, the server will check its local disk for a directory matching the client's username. If a directory matching the client's username does not exist, the server will create one. If a directory matching the client's username already exists, the server will utilize the existing directory. This directory will be designated the client's 'home directory'.

Inside the home directory, the client will have the ability to:

- Create directories;
- Delete directories;
- Move directories;
- Rename directories; and,
- List the contents of directories.

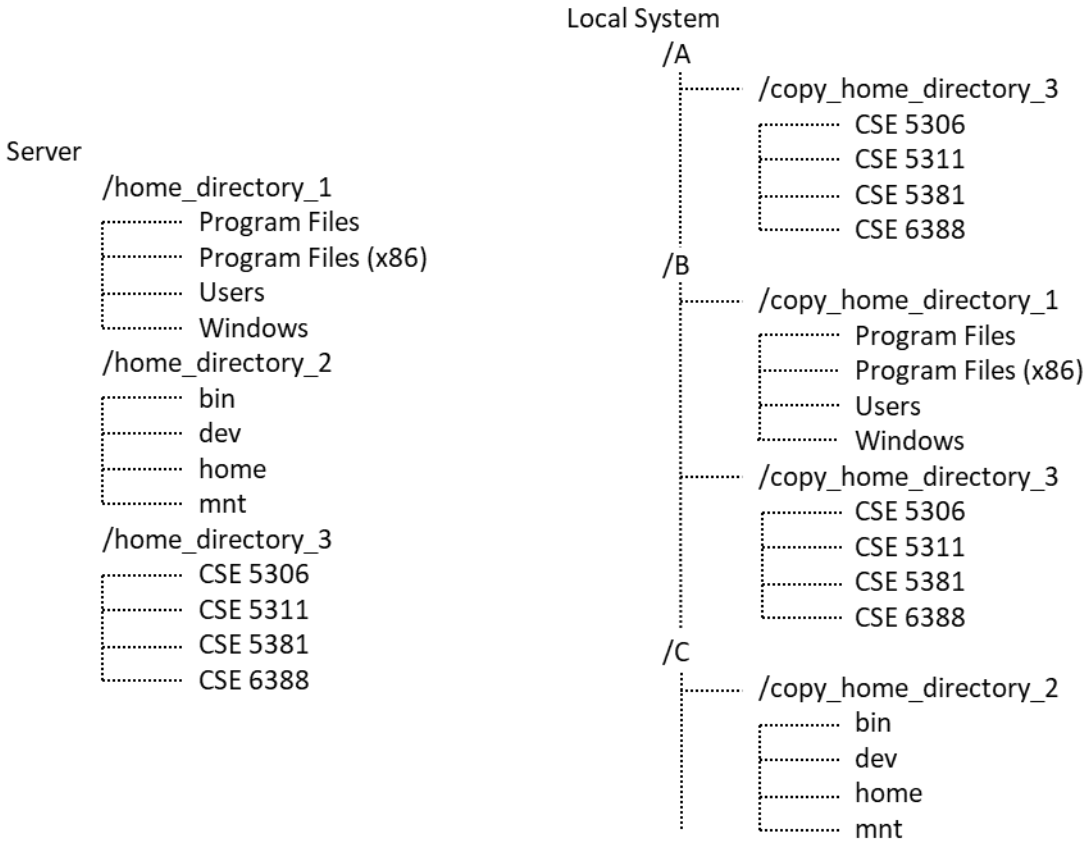
Each client will be explicitly confined to its own home directory – no client should be able to navigate to, list the contents of, or modify the contents of any parent directory. The user may input instructions to the client as conventional text commands, but these commands must be accepted via the GUI, not the command line.

All three clients should have the ability to make changes and navigate through their respective home directories in parallel. All operations performed by the server should be displayed on the server's GUI. Any directory operation errors generated by the server's host operating system must be conveyed to the client and displayed to the user. Directory operation errors should not result in the client disconnecting from the server.

**ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE**

Each client process will be given an identifier of A, B, or C. This client identifier is separate from the username the client provides to the server. Each client will be assigned a local directory (LD) corresponding to its identifier: e.g., client A will always use LD 'A', client B will always use LD 'B', client C will always use LD 'C'. How the process is mapped to the client identifier is left to the developer's discretion (i.e., the mapping could be assigned statically in the source code, the user could be presented with the option of selecting A, B, or C on startup, et cetera), but only one process may use an identifier at a time.

Users will be given the option of locally synchronizing any home directory available at the server. Upon connection to the server, users will be presented with a list of available home directories located on the server. A user may select one or more home directories of which to create a local copy. The local copy will be maintained in the client's LD and the user's selection of directories to synchronize, as well as the contents of the LD itself, will be persistent between sessions.




*Example Directory Layout*

Once a copy of the remote directory has been created locally, any changes to the server-based directory should be reflected in the local copy. Any update to the local copy will only be initiated at the server: a client will not make modifications to its local copy directly.

Any synchronized directory may also be desynchronized. Upon desynchronization, the copy of the directory should be removed from the local system. It is the developer's discretion as to how a user should specify a local copy to be desynchronized, but this option must be accessible via the client's GUI at least once per process instance.

**Lab #3 Additions**

**ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE**

The server will maintain a log of all directory operations. This log will be persistent and presented on the server's GUI. The user will be provided with the option to 'Undo' any logged operation at the server. When a user chooses to 'Undo' an operation, that operation should be rolled back across every applicable directory in the distributed system (e.g., across the server and all clients).

Operations which are selected for 'Undo' should be removed from the log once all applicable systems reach a consistent state. Any sequence-dependent operations must also be rolled back and removed from the log. For example, assume the following operations:

- Operation 1: Create parent directory;
- Operation 2: Create child directory.

A user choosing to 'Undo' Operation 1 would necessarily 'Undo' Operation 2. Both the parent directory and child directory should be removed from all applicable systems and both Operation 1 and Operation 2 should be removed from the log.

The developer should **not** assume sequence-dependent operations will be contiguous. Only one operation may be selected for 'Undo' at a time.

The functionality for the clients and the server is summarized as follows.

## Client

### Startup

1. Prompt the user to input a username.
2. Screen the username to ensure it does not contain illegal characters. How to address illegal characters is left to the developer's discretion.
3. Connect to the server over a socket and register the username.
    a. When the client is connected, the user should be notified of the active connection.
    b. If the provided username is already in use, the client should disconnect and prompt the user to input another username.
4. Display a list of the available server directories and allow the user to select one or more server-based home directories to synchronize to the local system.

### Synchronization

1. The client will automatically monitor the server for changes to the server-based home directory.
2. Upon detecting changes to the server-based directory, the client will update the copy of the home directory in its LD to match the server-based home directory.
3. Return to Step 1 of 'Synchronization' until manually terminated by the user.

### Directory Commands

1. Proceed to accept directory commands from the user via the GUI.
2. Display the output of those commands to the user via the GUI.
3. Return to Step 1 of 'Directory Commands' until manually terminated by the user.

## Server

The server should support three concurrently connected clients. The server should indicate which of those clients are presently connected on its GUI.

### Startup

1. Listen for incoming connections.
2. Print that a client has connected, and:
    a. If the client's username is available (e.g., not currently being used by another client), fork a thread to handle that client; or,
    b. If the username is in use, reject the connection from that client.
3. Provide a home directory to the client (e.g., create a directory if necessary or use an extant directory).
4. Proceed to accept and execute directory commands received from the client.
5. Return to Step 1 of 'Startup' until manually killed by the user.

**Logging**

1. The server will log all directory commands executed on behalf of clients.
2. Logged commands will be presented to users. Users will have the ability to select any logged command and choose 'Undo'.
3. The effects of the command chosen for 'Undo', as well as the effects of any sequence-dependent operations, will be rolled back across all applicable systems.
4. The command chosen for 'Undo', as well as any sequence-dependent commands, will be removed from the log.
5. Return to Step 1 of 'Logging' until manually killed by the user.

**Notes:**

- All three clients and the server may run on the same machine.
- The server must correctly handle an unexpected client disconnection without crashing.
- When a client disconnects from the server, the server GUI must indicate this to the user in real time.
- The program must operate independently of a browser.

**Citations**:

You may use open source code found on the Internet in your program. When citing this code:

# YOU MUST CITE THE EXACT URL TO THE CODE IN THE METHOD / FUNCTION / SUBROUTINE HEADER WHERE THE CODE IS UTILIZED.

**Failure to cite the exact URL will result in a twenty (20) point deduction on the grade of your lab**

A full list of your source URLs should be included in your writeup file. Including generic citations (for instance, simple listing "StackOverlow.com" without additional details) will result in a ten (10) point deduction in your lab grade, per instance.

**Submission Guidelines:**

**FAILURE TO FOLLOW ANY OF THESE DIRECTIONS WILL RESULT IN DEDUCTION OF SCORES**

Submit your assignment via the submission link on Canvas. You should zip your source files and other necessary items like project definitions, classes, special controls, DLLs, et cetera and your writeup into a single zip file. No other format other than zip will be accepted. The name of this file should be **lab#_lastname_loginID.zip**. Example: If your name is

**ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE**

**CSE 5306** **Lab # 3** **Fall 2020**

John Doe and your login ID is jxd1234, your submission file name must be "lab#_doe_jxd1234.zip" where # is the number of the lab assignment.

Be sure that you include everything necessary to unzip this file on another machine and compile and run it. This might include forms, modules, classes, configuration files, et cetera. DO NOT INCLUDE ANY RUNNABLE EXECUTABLE (binary) program. The first two lines of any file you submit must contain your name and student ID.

You may resubmit the project at any time. Late submissions will be accepted at a penalty of 10 points per day. If your program is not working by the deadline, send it anyway and review it with the TA for partial credit. Do not take a zero or excessive late penalties just because it isn't working yet. We will make an effort to grade you on the work you have done.

**Writeup:**

Your write-up should include instructions on how to compile and run your program. Ideally it should be complete enough that the TA can test your program without your being there. Your writeup should include any known bugs and limitations in your programs. If you made any assumptions, you should document what you decided and why. This writeup can be in a **docx** or **pdf** format and should be submitted along with your code.

**Grading:**

**Points – element:**

10 – Client process works correctly
10 – Server process works correctly
15 – Infrastructure from Lab 1 and Lab 2 works correctly
10 – Server logs operations
10 – User provided with the ability to 'Undo' logged operations at the server
10 – The effects of logged operations selected for 'Undo' are rolled back across all applicable systems
10 – Logged operations selected for 'Undo' are removed from the log
10 – The effects of sequence-dependent operations are rolled back across all applicable systems
10 – Sequence-dependent operations are removed from the log
05 – Comments in code

**Deductions for failing to follow directions:**

-10 – Late submission per day.
-05 – Including absolute/ binary/ executable module in submission.
-02 – Submitted file doesn't have student name and student ID in the first two lines.
-05 – Submitted file has a name other than student's lastname_loginID.zip.
-05 – Submission is not in zip format.
-05 – Submitting a complete installation of the Java Virtual Machine.
**-10 – Per instance of superfluous citation.**
-20 – Server and/or clients run exclusively from a command line.

To receive full credit for comments in the code you should have headers at the start of every module / function / subroutine explaining the inputs, outputs, and purpose of the module. You should have a comment on every data item explaining what it is about. (Almost) every line of code should have a comment explaining what is going on. A comment such as /* Add 1 to counter */ will not be sufficient; the comment should explain what is being counted.

**ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE**

**Important Note:**

**You may discuss the problem definition and tools with other students. You may discuss the lab requirements. You may discuss or share project designs. All coding work must be your own. You may use any book or web reference as long as you cite that reference in the comments. If we detect that portions of your program match portions of any other student's program, it will be presumed that you have colluded unless you both cite a specific source for the code.**

**You must not violate University of Texas at Arlington regulations, laws of the State of Texas or the United States, or professional ethics. Any violations, however small, will not be tolerated.**

**DO NOT POST YOUR CODE ON PUBLICLY ACCESSIBLE SECTIONS OF WEBSITES UNTIL AFTER THE COURSE CONCLUDES. SHOULD YOU DO SO, THIS WILL BE CONSIDERED COLLUSION, AND YOU WILL BE REFERRED TO THE OFFICE OF STUDENT CONDUCT AND RECEIVE A FAILING GRADE IN THE COURSE**

**ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE**