

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**

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

Objectives:

1. Implementing mutual exclusion
2. Queuing updates
3. Handling concurrent operations

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 #1 and 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).

This lab has specific submission requirements. Failure to follow these submission requirements will result in your lab not being accepted for a grade.

Lab #1 Infrastructure

You will write a program that will generate a composite directory listing from multiple servers. Your project will consist of a client process and two server processes and function as a command line instruction.

Each server, Server A and Server B, will feature a pre-designated directory named `directory_a` and `directory_b`, respectively. When executed, your client will establish a connection to Server A, which will generate a listing of the contents of `directory_a` (this listing is analogous to the `ls -l` command on Linux/Bash or `dir` on Windows). Server A will then establish a connection to Server B, which will generate a listing of the contents of `directory_b` and return the listing to Server A.

Server A should combine the listing of contents of `directory_a` and `directory_b` into a single list sorted by file name. The list should only include the file name, file size, and either the time the file was created or the time the file was last modified. Server A will return the composite list to the Client, which will print the data to the command line.

Lab #2 Infrastructure

Server A and Server B will autonomously synchronize the contents of `directory_a` and `directory_b` during runtime, including both files and file metadata. During runtime, any change to the contents of a directory on one server, including adding, deleting, or modifying a file, should be applied at the other server. Files will be added, deleted, or modified with the host's native file manager (e.g., Windows Explorer for Windows or Finder on macOS), and for the purposes of this lab assignment neither directory will include subdirectories.

Upon startup, Servers A and B will generate an inventory of the contents of their designated directories and compare contents. Any content discrepancy should be addressed, with duplicated files being made consistent based on the most-recent modified-at time.

Any change to the contents of the directory during runtime should be recognized within five seconds. The user should be notified of the servers' actions in real-time, and the notifications should include which file is being synchronized when applicable. The mechanism by which the directory contents are made consistent is left to the developer's discretion.

Lab #3 Additions

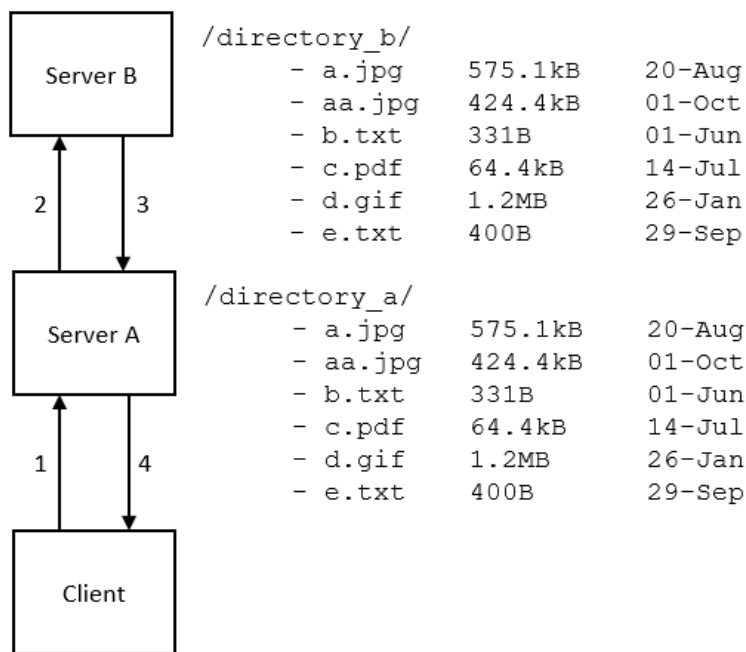
The client will index files 0 to $n - 1$ when listing directory contents to the user. The user will have the ability to lock files at Server A by running a command: `./lab3 -lock -<index>`.

While a file is locked at Server A, any updates to a locked file in `directory_b` will be placed into a FIFO queue at Server A. When a user unlocks a file by executing `./lab3 -unlock -<index>` at the client, updates to that locked file will be applied in the order they were received.

Any file not locked should continue to be updated according to the instructions in Lab #2.

Example:

At time T_1 , the user retrieves an indexed list of directory contents by executing `./lab3` at the client.

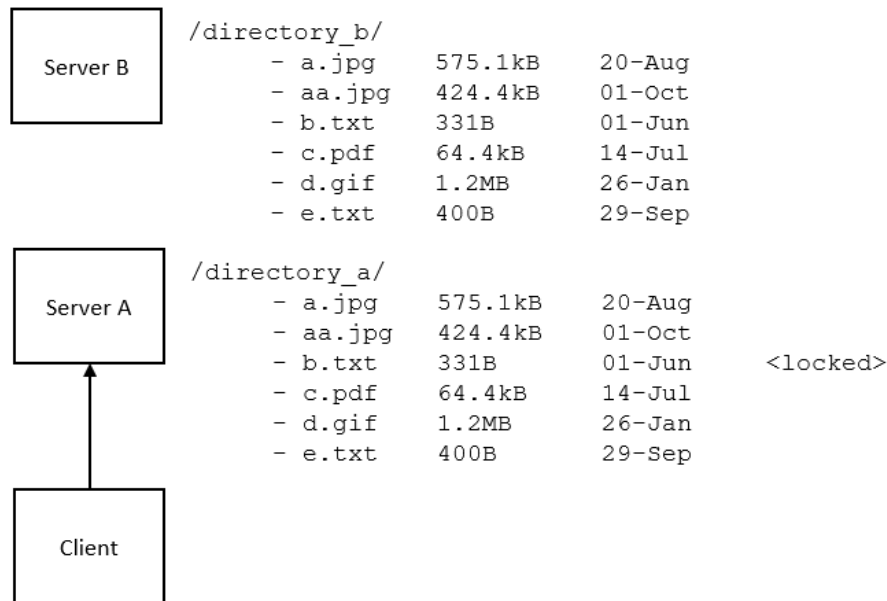


T_1 : Client Query

```

user@Client:~$ ./lab3
[0] a.jpg      575.1kB    20-Aug
[1] aa.jpg     424.4kB    01-Oct
[2] b.txt      331B      01-Jun
[3] c.pdf      64.4kB    14-Jul
[4] d.gif      1.2MB     26-Jan
[5] e.txt      400B      29-Sep
  
```

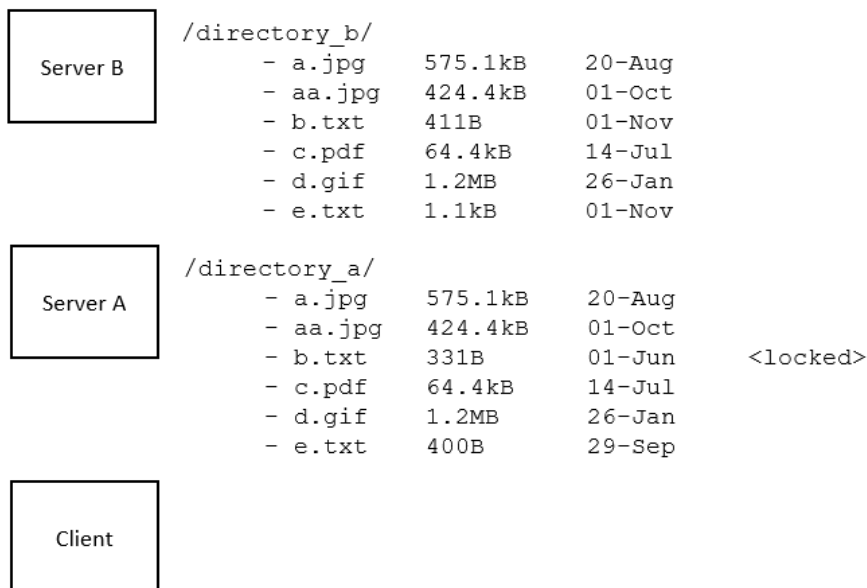
At time T_2 , the user locks `b.txt` at Server A by executing `./lab3 -lock -2`.



T_2 : Client Locks File

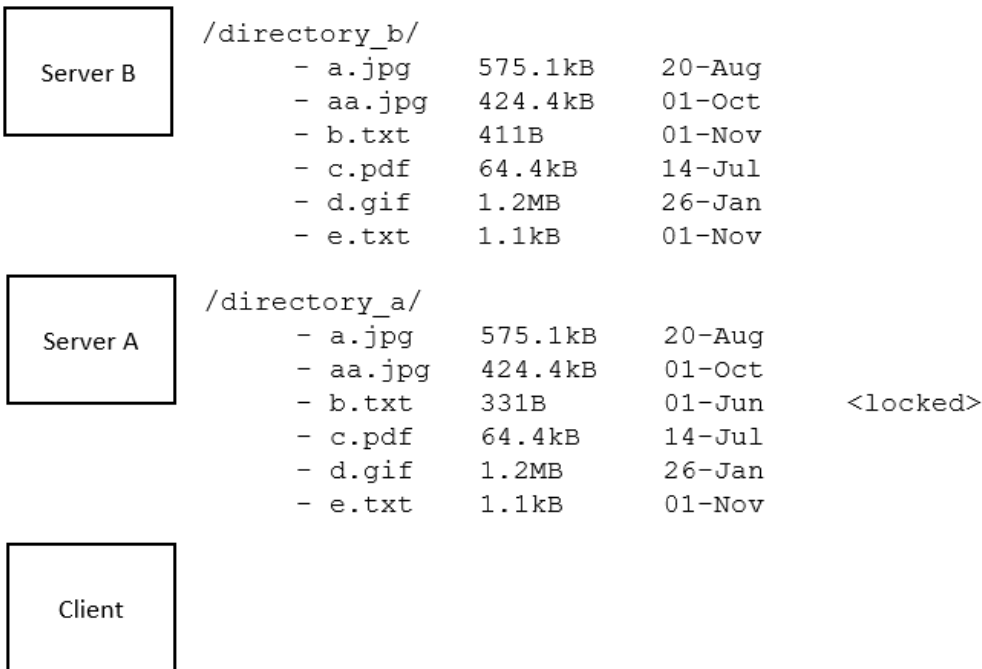
```
user@Client:~$ ./lab3 -lock -2
```

At time T_3 , `b.txt` and `e.txt` are updated in `directory_b`.



T_3 : File Update at `directory_b`

At time T_4 , the file update to the unlocked file is applied at `directory_a` and the file update to the locked file is queued.

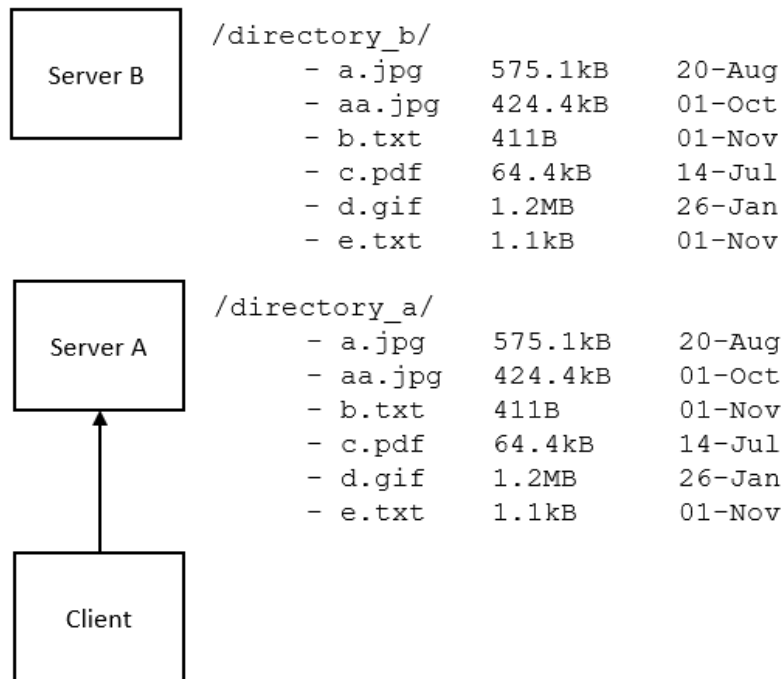


T_4 : Update at directory_a

At time T_5 , the user queries Server A for the present status of directory_a.

```
user@Client:~$ ./lab3
[0] a.jpg          575.1kB          20-Aug
[1] aa.jpg         424.4kB          01-Oct
[2] b.txt          331B           01-Jun    <locked>
[3] c.pdf          64.4kB          14-Jul
[4] d.gif          1.2MB          26-Jan
[5] e.txt          1.1kB          29-Sep
```

At time T_6 , the user unlocks b.txt. The file is unlocked in directory_a and Server A applies the queued updates to b.txt.



T₆: Client Unlocks File

```
user@Client:~$ ./lab3 -unlock -2
```

The system should continue to function as described above until Server A and Server B are manually killed by the user.

Submission Requirements:

In addition to the Submission Guidelines listed below, you will create a single directory in your zip file that will contain nothing other than source code that you wrote or modified for this assignment. That is, if you wrote your program in Java, this directory should have nothing other than .java files that you have personally modified. This directory should be named your loginID and have no subdirectories.

Notes:

- Locked files at `directory_a` will not be manipulated via the host's native file manager during testing.
- Locked files will not have their index changed when testing (e.g., only files indexed "below" a locked file will be deleted).
- There is no upper bound on the number of operations that might be queued or the number of files that might be locked.
- All processes may run on the same physical machine.
- Server A and Server B may be run from different command line instances.
- The IP address and port number of Server A and Server B may be hardcoded.
- The formats of the dates and file sizes are developer's discretion.
- Files may be added, deleted, or modified at any time.
- The contents of the designated directories will be verified with the host's native file manager.
- The program must operate independently of a browser engine.

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

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, simply listing “StackOverflow.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 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.

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:**

- 20 - Lab #2 Infrastructure functioning correctly
- 30 - Server A concurrently handles lock requests and file updates
- 10 - Client successfully locks file
- 10 - Client successfully unlocks file
- 25 - Queued operations applied after file unlocked
- 05 - Client correctly indicates file is locked on query

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

Deductions for failing to follow directions:

- 25 – Late submission per day.
- 05 – Including absolute/ binary/ executable module in submission.
- 10 – 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 lab#_lastname_loginID.zip.
- 05 – Submission is not in zip format.
- 30 – Submitting a complete installation of a virtual machine.
- 10 – Per instance of superfluous citation.

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.

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