

# 1 Introduction

The BSLE is intended for you to demonstrate your ability to build a program that follows a specification provided by a senior developer that meets an operational requirement.

## 1.1 Overview

You are tasked with creating a secure file transfer utility. It will consist of two components: client and server. The client and server will maintain two separate connections: one for data and one for control.

- Client: This component will be written in Python 3.8. It will be the primary user interface to the file server.
- Server: This component will be written in C. It will run as a service, interact with one to many clients via a wire protocol that will be specified below, and store files delivered and retrieved by clients.

## 1.2 Administrative

### 1.2.1 Time Considerations

- Candidates will be given 10 business days without interruptions to complete this exam.
- Unavoidable interruptions will be compensated at a rate of one and a half (1.5) times the duration of interruption up to one day.
  - Example 1: A mandatory training event lasting two hours will result in an additional three hours for the candidate.
  - Example 2: A training holiday will result in an additional day for the candidate.

### 1.2.2 Assistance

- Candidates may ask for clarification on the project specification from the exam proctor, the senior developers in their organization, and/or a designated basic developer *if no senior developers are available*.
- Any source code that a candidate does not write must be clearly documented in source code via comments clearly attributing the source. This should occur in very limited cases and should not be excessive. If panel members deem that source code was plagiarized, or that documented sources were excessive, they will make a recommendation as such to the Lead Developer for final arbitration.
- Senior developers may provide limited assistance identifying isolated problems a candidate gets stuck on. The definition of "limited" is a ballpark estimate of approximately 5-10 minutes to spot and troubleshoot.

### 1.2.3 Constraints

For this project everything must be original code with exception to the C and Python standard libraries and POSIX Thread library. No third party libraries can be used in the release binary. Third party unit-testing libraries are authorized for debug and test builds. The *make* and *cmake* utilities are authorized as well as any build system that your proctor authorizes given it is relevant to your organization's build process. The source code shall adhere to the style guides specified in this document.

### 1.2.4 Reporting Exam Issues

If you discover any issues with the examination please create a GitLab issues in your repo. Your exam proctor will resolve your questions in a timely manner. The CSD Talent Manager maintains an open door policy for issues you believe should not be addressed in a GitLab issue.

## 2 Project Environment

### 2.1 Directory Structure

You have been given a Makefile to modify to your needs. Fill out the Makefile so that it compiles the code you place in the *src* directory and outputs a binary named *capstone* in a *bin* directory. The *src* directory is for source files and private header files. You can organize the *src* directory however you see fit. The Makefile must run by running *make* from the root directory of this project. *Do not assume the bin directory exists.*

```
/
├── bin
│   └── example_server
├── build
├── include
├── test
│   ├── server
│   └── client
└── src
```

Alternately you may use CMAKE to replace the provided Makefile, but you must document the change in your User Manual.

### 2.2 Operating System

Ubuntu 20.04 LTS

### 2.3 Dependencies

```
apt update -yq && apt install -y \
    clang-tidy \
    clang \
    cmake \
    libcunit1-dev \
    valgrind \
    gdb \
    strace \
    git \
    python3
```

### 2.4 Coding Standards

This project *shall* adhere to the following style guides. Failure to follow this style guide will be explained in detail in the grading rubric document.

| Language    | Style Guide                        |
|-------------|------------------------------------|
| C           | <a href="#">BARR-C</a>             |
| Python 3.8+ | <a href="#">Google Style Guide</a> |

## 3 Evaluation Criteria

### 3.1 General Criteria

**4.1.1 Grading Rubric** If all implementation priorities are met then the project will be graded according to the provided rubric. Implementation priorities are labeled with the following priorities: A, B, and C.

| Priority | Point Value (Client) | Point Value (Server) |
|----------|----------------------|----------------------|
|----------|----------------------|----------------------|

|   |           |           |
|---|-----------|-----------|
| A | Pass/Fail | Pass/Fail |
| B | 4         | 15        |
| C | 3         | 10        |

- Project must correctly implement ALL of the A tasks. If any are incomplete or not working it is an automatic failure.

**3.1.2 Git Flow** Your account will not have permission to merge to the master branch. A Merge Request to the merge the develop branch into master has been created for you. Only code in the develop branch will be graded via the existing Merge Request. It is encouraged to use the '[GitLab Flow](#)' where your develop branch is treated like master in order to cleanly organize your exam, but the only requirement is that your submission is committed to develop.

## 3.2 Documentation

**3.2.1 Comments** Code is expected to be well documented. Comments should be succinct and meaningful - i.e. do not explain *what* the code is doing, but rather explain *why* you implemented it the way you did.

**3.2.2 User Manual** Candidate must provide a User Manual. It should be a detailed document containing instructions on how to compile and run the project. At a minimum it should include the following:

- Project compilation
- How to start the service
- How to connect
- How to interact with client
- Known issues

## 3.3 Programming

### 3.3.1 General Guidance

- Data validation and error checking must be performed where appropriate interfaces exist.
- Clear concise naming conventions used.
- Proper memory management

### 3.3.2 Server

- Must be written in C.
- Uses user-based permissions for server operations
  - READ: can get files from server
  - READ/WRITE: can get and put files from server
  - ADMIN: can get and put files from server and delete existing users
- Supports registering new users and deletion of existing users.
  - Users should only be able to create other users at their own permission level and below
- Support listening for multiple clients at a time.
- Ensure client operations are restricted to within a specified directory.

- Management of sessions (unique session IDs, enforcement of session timeout, etc)
- Must create a default user with admin permissions with username: “admin”, and password : “password” (for the purposes of the panel’s testing script)
- Server must accept the following settings as command line options
  - -t : timeout (in seconds) of user Session IDs
  - -d : folder to server; files/directories can be created inside of this directory, but not outside of it
  - -p : port server will listen on

### 3.3.3 Client

- Must be written in Python3
- Filenames containing "." and ".." (assuming the server is not in the top level directory) should function as they do in Linux.
- Connect to the server at a specified PORT and IP address (Both should be command line arguments)
- All functionality available through CLI arguments and an alternate fully interactive terminal.
- Support the following operations:

| Command             | Description   |
|---------------------|---|
| get [src] [dst]     | Gets a file from server [src] path and copies it into the client [dst] path |
| put [src] [dst]     | Sends a file from client [src] path to be placed in the server [dst] path   |
| help [opt: command] | Print available commands, optionally describes given command                |
| quit / exit         | Closes client   |
| delete [path]       | Deletes file at server [path]   |
| l_delete [path]     | Deletes file at local [path]  |
| ls                  | Lists remote directory contents   |
| l_ls                | Lists local directory contents  |
| mkdir [path]        | Makes directory at server [path]  |
| l_mdmdir [path]     | Makes directory at client [path]  |

- NOTE: All [path] values sent to server should be RELATIVE to the server home directory. Putting a file at path ‘/’ should place that file in the server’s home directory.

### 3.4 Presentation

Following completion of the project, the developer candidate must present and discuss the completed project with a panel of graders. The presentation will be an opportunity for the developer candidate to explain and defend the design and implementation of the project.

## 4 Detailed Capability Requirements

### 4.1 Wire Protocol and Sequence Diagrams

The message size used by both the client and server will be limited to 1024 bytes, and the maximum file size that can be sent is 1016 bytes.

Any multi-byte integers described in the wire protocol MUST be transmitted in network byte order (big-endian).

Client/Server operations and responses defined in the Wire Protocol are each associated with single byte opcode/return code. These are outlined in the tables below:

#### Operation Codes

|                       |      |
|-----------------------|------|
| User Operation        | 0x01 |
| Delete remote file    | 0x02 |
| List remote directory | 0x03 |
| Get remote file       | 0x04 |
| Make remote directory | 0x05 |
| Put remote file       | 0x06 |

#### Return Codes

| Code             | Value | Description   |
|------------------|-------|---|
| Success          | 0x01  | Server action was successful  |
| Session error    | 0x02  | Provided Session ID was invalid or expired  |
| Permission error | 0x03  | User associated with provided Session ID had insufficient permissions to perform the action |
| User exists      | 0x04  | User could not be created because it already exists   |
| File exists      | 0x05  | File could not be created because it already exists   |
| Failure          | 0xff  | Server action failed  |

#### User Operation

| User Operation (Client Request) |            |              |          |   |                           |   |                 |   |
|---------------------------------|------------|--------------|----------|---|---------------------------|---|-----------------|---|
|                                 | 0          | 1            | 2        | 3 | 4                         | 5 | 6               | 7 |
| +0                              | OPCODE     | USER<br>FLAG | RESERVED |   | USERNAME LENGTH           |   | PASSWORD LENGTH |   |
| +8                              | SESSION ID |              |          |   | USERNAME... + PASSWORD... |   |                 |   |

| User Operation (Server Response) |             |          |            |   |   |   |     |
|----------------------------------|-------------|----------|------------|---|---|---|-----|
|                                  | 0           | 1        | 2          | 3 | 4 | 5 | 6 7 |
| +0                               | RETURN CODE | RESERVED | SESSION ID |   |   |   |     |

| User Flag Codes  |       |  |
|------------------|-------|--|
| Code             | Value | Description  |
| Login            | 0x00  | A standard login message<br>(No Session ID required)                       |
| Create Read Only | 0x01  | Create the given user with Read permissions<br>(Valid Session ID required) |



| Delete Remote File (Server Response) |                |   |   |   |   |   |   |   |
|--------------------------------------|----------------|---|---|---|---|---|---|---|
|                                      | 0              | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| +0                                   | RETURN<br>CODE |   |   |   |   |   |   |   |

Client will send the appropriate OPCODE, the length of the filename as an 2-byte unsigned integer, and a valid Session ID, followed by the Filename to be deleted.

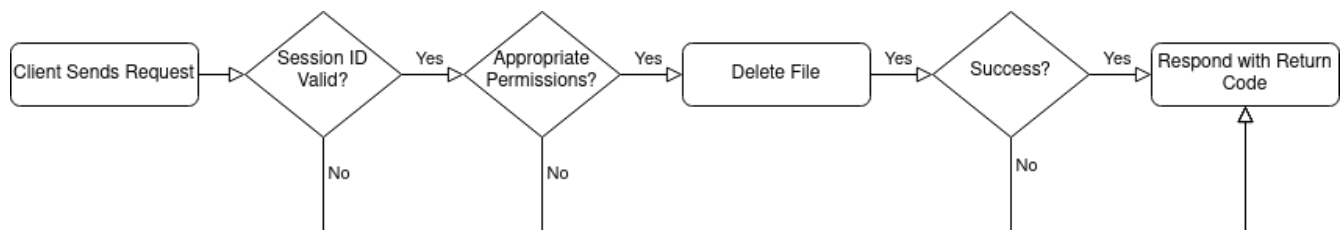
**NOTE:** Filenames should be the path relative to the server's home directory.

**The server will check if the Session ID is valid, returning SESSION\_ERROR if it is not.**

If the session is valid, the Server will verify that the user associated with that Session ID has the appropriate permissions to delete files and return PERM\_ERROR if not.

If the user has permission to delete files, the server will find the file indicated by the Filename, and attempt to delete it.

**If the action is successful, the server will return SUCCESS. Otherwise, it will return FAILURE.**

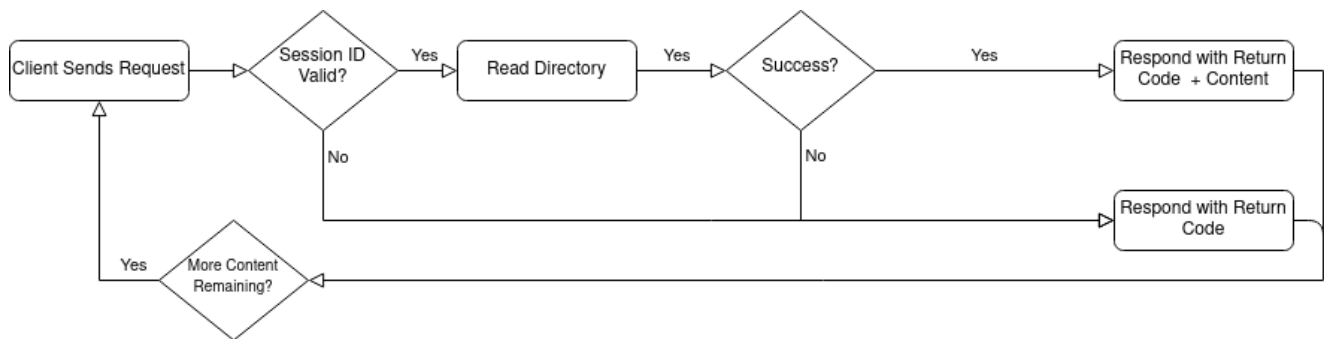


| List Remote Directory (Client Request) |   |          |                          |   |                   |   |   |   |
|--|---|----------|--------------------------|---|-------------------|---|---|---|
|  | 0   | 1        | 2                        | 3 | 4                 | 5 | 6 | 7 |
| +0                                     | OPCODE  | RESERVED | DIRECTORY NAME<br>LENGTH |   | SESSION ID        |   |   |   |
| +8                                     | CURRENT POSITION<br>(NUMBER OF BYTES ALREADY<br>RECEIVED) |          |                          |   | DIRECTORY NAME... |   |   |   |

| List Remote Directory (Server Response) |   |          |   |   |   |   |   |   |
|---|---|----------|---|---|---|---|---|---|
|   | 0   | 1        | 2 | 3 | 4   | 5 | 6 | 7 |
| +0                                      | RETURN<br>CODE  | RESERVED |   |   | TOTAL CONTENT LENGTH<br>(TOTAL NUMBER OF BYTES OF ALL<br>CONTENT)             |   |   |   |
| +8                                      | MESSAGE CONTENT LENGTH<br>(NUMBER OF BYTES INCLUDED IN THIS<br>MESSAGE) |          |   |   | CURRENT POSITION<br>(POSITION WITHIN TOTAL CONTENT<br>LENGTH MESSAGE ENDS AT) |   |   |   |

| Server Response Content Format                  |               |  |              |   |               |
|---|---------------|--|--------------|---|---------------|
| 1-BYTE<br>0x01 (if<br>file)<br>0x02 (if<br>dir) | FILE/DIR NAME |  | NULL<br>BYTE | 1-BYTE<br>0x01 (if<br>file)<br>0x02 (if<br>dir) | FILE/DIR NAME |
|   |               |  |              |   | NULL<br>BYTE  |

- Client will send the appropriate OPCODE, a 2-byte unsigned integer of the length of the directory name, a valid Session ID, a 4-byte unsigned integer of the number of bytes already received (if it is a continuation), and the directory name.  
**NOTE:** Directory names should be the path relative to the server's home directory.
- The server will validate the Session ID and respond with the return code, followed by the 4-byte unsigned integer total content length, a 4-byte unsigned integer indicating the number of bytes of content included in message, a 4-byte unsigned integer of the current position at the end of content length, and the content in the above Server Response Content Format
- Repeat as necessary



### Get Remote File

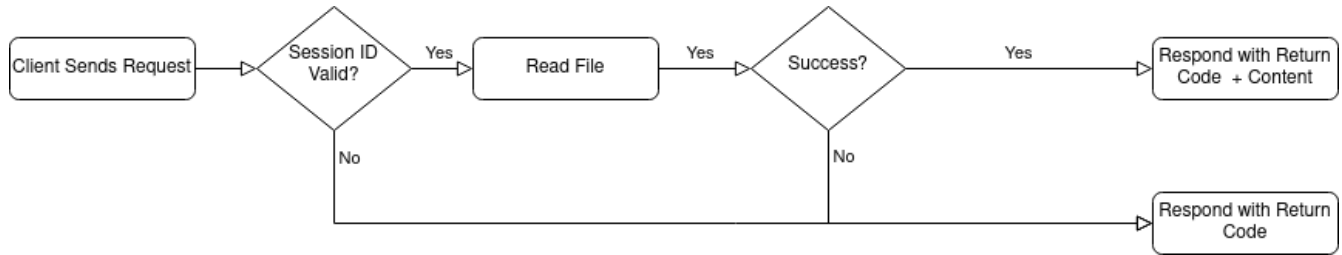
| Get Remote File (Client Request) |             |          |                 |   |            |   |   |   |
|----------------------------------|-------------|----------|-----------------|---|------------|---|---|---|
|                                  | 0           | 1        | 2               | 3 | 4          | 5 | 6 | 7 |
| +0                               | OPCODE      | RESERVED | FILENAME LENGTH |   | SESSION ID |   |   |   |
| +8                               | FILENAME... |          |                 |   |            |   |   |   |

| Get Remote File (Server Response) |                |          |             |   |   |   |            |   |
|-----------------------------------|----------------|----------|-------------|---|---|---|------------|---|
|                                   | 0              | 1        | 2           | 3 | 4 | 5 | 6          | 7 |
| +0                                | RETURN<br>CODE | RESERVED | CONTENT_LEN |   |   |   | CONTENT... |   |

- Client will send the appropriate OPCODE, length of the file name as an 2-byte unsigned integer, and a valid Session ID, and the name of the file to be downloaded.  
**NOTE:** Filenames should be the path relative to the server's home directory.



- The Server will respond (assuming no errors) with a SUCCESS, an 8-byte unsigned integer of number of files, and then a buffer containing the file contents. If an error occurred, respond with the appropriate return code, the length of the file as a 4-byte unsigned integer, and then the file contents.

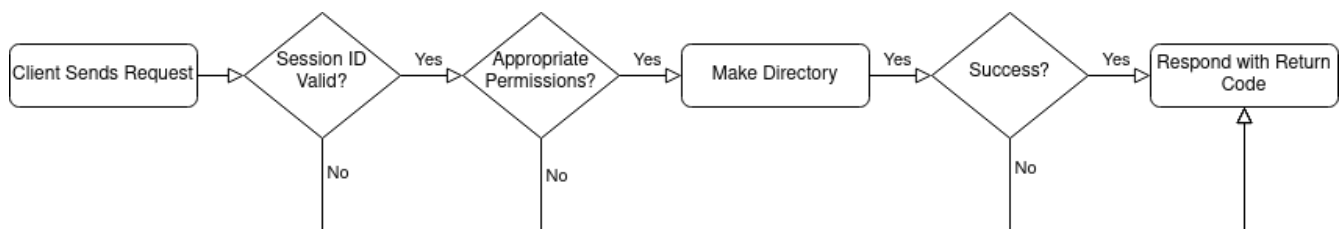


### Make Remote Directory

| Make Remote Directory (Client Request) |          |          |                          |   |                   |   |   |   |
|--|----------|----------|--------------------------|---|-------------------|---|---|---|
|  | 0        | 1        | 2                        | 3 | 4                 | 5 | 6 | 7 |
| +0                                     | OPCODE   | RESERVED | DIRECTORY NAME<br>LENGTH |   | SESSION ID        |   |   |   |
| +8                                     | RESERVED |          |                          |   | DIRECTORY NAME... |   |   |   |

| Make Remote Directory (Server Response) |                |   |   |   |   |   |   |   |
|---|----------------|---|---|---|---|---|---|---|
|   | 0              | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| +0                                      | RETURN<br>CODE |   |   |   |   |   |   |   |

- Client will send the appropriate OPCODE, length of the directory name as an 2-byte unsigned integer, and a valid Session ID, followed by the path of the directory to be created.  
**NOTE:** Directory names should be the path relative to the server's home directory.
- The Server will validate the session ID and verify that the user associated with it has Read/Write or higher permissions, then create the directory respond (assuming no errors) with a SUCCESS. If an error occurred, respond appropriately.



## Put Remote File

| Put Remote File (Client Request) |                     |                     |                 |   |            |   |   |   |
|----------------------------------|---------------------|---------------------|-----------------|---|------------|---|---|---|
|                                  | 0                   | 1                   | 2               | 3 | 4          | 5 | 6 | 7 |
| +0                               | OPCODE              | OVERWR-<br>ITE FLAG | FILENAME LENGTH |   | SESSION ID |   |   |   |
| +8                               | FILE CONTENT LENGTH |                     |                 |   | CONTENT... |   |   |   |

| Put Remote File (Server Response) |             |   |   |   |   |   |   |   |
|-----------------------------------|-------------|---|---|---|---|---|---|---|
|                                   | 0           | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| +0                                | RETURN CODE |   |   |   |   |   |   |   |

| Overwrite Flag Codes |       |                                |
|----------------------|-------|--------------------------------|
| Code                 | Value | Description                    |
| No Overwrite         | 0x00  | Do not overwrite existing file |
| Overwrite            | 0x01  | Overwrite existing file        |

- Client will send the appropriate OPCODE, 1-byte unsigned overwrite flag, 2-byte unsigned integer representing the filename length, a valid Session ID, and the length of the file contents as an 4-byte unsigned integer, followed by the contents of the file to be uploaded.  
**NOTE:** Filenames should be the path relative to the server's home directory.
- The Server will validate the session ID and verify that the user associated with it has Read/Write or higher permissions, then create the directory respond (assuming no errors) with a SUCCESS. If an error occurred, respond appropriately. If the file trying to be uploaded already exists, and the Overwrite flag is not set, server will respond with FILE\_EXISTS.

