Net Calc

Cyber Solutions Development - Tactical

September 28, 2021

Abstract

Your task is to use the same parser you built for FileCalc, but this time you will accept files over a Network Socket connection, enqueue the data, and process work in the queue with a threadpool.

1 Requirements

In this assignment, you will build an application that will perform the same operations as FileCalc, but will utilize sockets, data structures, and threading. You will also write a Client capable of operating to the given specification that will be able to operate with your Server.

1.1 Basic Requirements

1.1.1 Server

1. Written in C
2. Everything FileCalc does
3. Uses network sockets to listen for and accept multiple incoming connections
4. Read incoming work from the client
5. Utilize a queue to store work
6. Utilize a threadpool to handle clients and process work in the queue
7. Return completed work to the Client over the same network socket

1.1.2 Client

1. Written in Python
2. Given a single Equ File or Folder of Equ Files, send all files to the Server
3. Provide usage documentation and a help output
4. Gracefully handle malformed/invalid input
5. Receive completed work from the Server and save it appropriately

1.2 Specific Requirements

1.2.1 Server

1. Properly use Linux TCP networking protocols and functions
2. Utilize the Queue from DataStructures1
3. Utilize a Threadpool to process work from the queue
4. Correctly use Endian Conversion libraries
5. Use a signal handler to gracefully terminate on a SIGINT (CTRL-C)
6. Utilize getopt to process the following arguments

   -p PORT (port to listen on; defaults to 31337)

   -n NUM (number of threads in the pool; defaults to 4)

   -h (print help menu and exit)

1.2.2 Client

1. Properly use Python networking libraries
2. Handle serialization and endianness of data
3. Graceful and Appropriate Error Handling
4. Proper Python File and Network IO
5. Utilize argparse (or similar) to process the following arguments

   -s IP (IP address of server; defaults to 127.0.0.1)

   -p PORT (port to connect to; defaults to 31337)

   -i IN_FOLDER (folder of unsolved EQU; mandatory w/ no default)

   -o OUT_FOLDER (folder of solved EQU; mandatory w/ no default)

   -h (print help and exit)
6. Prompt for missing mandatory arguments

1.2.3 Data Structures

Trainee dependent. At this point, you should have all the tools you need to solve this in a logical and straightforward way.

1.2.4 Memory Management

However you chose to solve it, your program should not be leaking memory. Your program should show no memory leaks with:

```
        valgrind --leak-check=full ./netcalc <Appropriate Arguments>
```

1.2.5 Thread Management

You must impliment a well designed and fully functional threadpool. Your program should not have deadlocks, race conditions, or odd failure states. Your program should close appropriately with a CTRL-C, and not hang waiting for IO

1.2.6 Assumptions

1. All numbers in EQU spec are little endian
2. All numbers in NET spec are big endian

   It is not save to assume the signedness of the numbers; refer to the Spec
3. All valid files will have a valid Magic Number
4. All Equation IDs are unique (they are unique enough)
5. DO NOT assume the user is providing valid length/size data; ensure these are checked for logal consistency

1.2.7 FileFormat

1. Please see the attached FileSpec.pdf file (No Changes from FileCalc).

1.2.8 NetFormat

1. Please see the attached NetSpec.pdf file.

   This is simply a header that gets stacked onto the FileSpec from FileCalc

Be sure to pay attention to the implementation of File and Net formats. Your server will be tested against the reference Client, and unexpected behavior or formatting will cause the tests to fail.

1.3 Work Receiving Guidelines

1. If you encounter Errors in the Net Header (ie invalid sizes):

   Log an Error to stderr

   Return an Error Response (see NetSpec)

   Resume waiting for Work

2. If you encounter Errors in the Equ Header (ie invalid magic):

   Log an Error to stderr

   Return an Error Response (see NetSpec)

   Resume waiting for Work

3. If you encounter Errors in Individual Equations:

   Use the Error field to indicate it

   Continue Parsing other Equations

1.4 Work Parsing Guidelines

1. If you encounter Errors in the Equ Header (ie invalid magic):

   Log an Error to stderr

   Move the file to a BadFiles directory

   Continue Parsing other files

2. If you encounter Errors in Individual Equations:

   Use the Error field to indicate it

   Continue Parsing other Equations

2 Deliverables

Your code should have the following file structure:

```
4_NetCalc
├── src
│   └── source-files
├── hdrs
│   └── header-files
├── references
│   └── documentation
├── client
│   ├── client.py
│   └── <other python files as needed>
└── CMakeLists.txt
```

Your code should build and compile with the following shell script ran from the NetCalc directory:

```
// build.sh
mkdir build
cd build
cmake ..
```

```
make
```

The build script has the same requirements as the SimpleCalc build script.

3 Resources

Below is a list of resources you may find useful:

1. man getaddrinfo
2. man select
3. man select_tut

4 Notes to grader

The purpose of this project is to demonstrate understanding of the client/server model, use of thread pools, and all previously demonstrated areas. Upon a successful submission, the trainee should be ready to take the BLSE.

Although you ultimately control the trainees JQR completion status, allow them to complete the projects using whichever means they want (within reason). Engineers need to be able to break down problems, and iterate on solutions. If we say ”you will do X and Y and Z to solve this” we risk putting the trainees in a box where they can’t fully flex their problem solving muscles.

/pagebreak

5 JQR Sections Potentially Covered - Mentor/Trainee dependent

NOTE: These projects are designed to give each trainee room to think creatively. Although the reference spec may target specific JQR requirements, it is up to the trainee to pick which JQR items they want to solve, and it should be indicated somewhere in the trainees documentation. It is up to the mentor to determine which JQRs have been satisfied and which need more work.

By this point, your trainee should absolutely be able to point to specific JQR line items, and where they are in the codebase