Project 2

Multi-threaded Short File Transfer (SFT) Server

Due November 15, 11:59pm

This project will build on project 1 by making the SFT server multi-threaded. You will get experience building multi-threaded programs and gain experience with the boss-worker thread design pattern.

Note: The SFT server should use a thread pool and follow the boss-worker thread pattern. This will be graded. You will have points removed if you don't follow this instruction.

So far, the SFT server can only handle a single request at a time. To overcome this limitation, you will make your SFT server multi-threaded by updating sft_server.c to create worker threads to handle incoming requests. The main, i.e. boss, thread will continue listening to the socket and accepting new connection requests. When new requests come in, the boss thread should add those requests to a global queue. These connections will be handled by worker threads who are waiting at the queue for incoming requests. The pool of worker threads should be initialized to 10.

Project Set Up

Project 2 builds on project 1 so you will be using your files from project 1 to complete project 2. If you had errors in project 1, you will need to fix those before beginning project 2.

Project 1 set up refresher – Put your sft_client.c, sft_client_main.c, sft_server.c, sft_server_main.c, and the client and server header files from project 1 in the same directory. In addition, put r1.txt, r2.txt, r3.txt, r4.txt, r5.txt and requests.txt in the directory. Finally, add queue.c and queue.h to the directory.

Off of that directory create a new directory called **client_files**. When a request for a file comes in, the server retrieves the file and sends the contents to the client. After receiving the response from the server, the client will write the file contents to the **client_files** directory.

Instructions

1. Server – the server starts in sft_server_main and should remain unchanged. The main function calls the start_server function in sft_server.c.

   sft_server.c should first create a pool of 10 worker threads. When a connection and request is received, it should put the details of the request in a workdetails struct, which is then placed in the global queue. The definition of the workdetails struct is included in a new sft_server.h found on Blackboard. Worker threads are circling this queue waiting to handle requests. Since the workorder queue is a global and shared variable your code needs to include proper synchronization of the queue.
2. Client- the client code from project 1 does not change.

Compilation

To compile the server use:

```
gcc -g -o server voidQqueue.c sft_server.c sft_server_main.c -pthread
```

To compile the client use:

```
gcc -g -o client sft_client.c sft_client_main.c
```

Important Notes: You may assume that neither the message to the server nor the response will be longer than 1600 bytes. You can statically allocate memory for the request and file contents like so:

```
buffer[1601];
```

Because these files are small, you may assume that the full file content is sent or received in every write or read call.

Submission

Submit sft_client.c, sft_client_main.c, sft_server.c, sft_server_main.c. You do not need to submit the header files, unless you have modified them.

Grading

80 points - Implementation

40 points – the program runs without runtime errors and the client correctly writes the 5 files to client_files directory.

40 points- correct implementation of the boss-worker thread design pattern.

**I have given you a binary of a multi-threaded client. If your server works with the given client, and you correctly implemented the boss-worker design, you can expect full credit. If your multi-threaded server works with your own single-threaded client, and you've implemented the boss-worker thread design correctly, you can expect close to full credit.*

You should design and test with your own client before moving onto testing with the multi-threaded client.

10 points

The code is in the correct files – sft_server.c, sft_server_main.c and the header files.

10 points

Your name: Add your name in the comments section at the top of the project

Readability: Make sure your code is indented and neatly commented. Do not leave commented out code in your submission. Comments describing what your code does is ok. Don't leave old code in the source file.

Compilation: If your code does not compile, you will receive a 0 on the project. Once notified, you will have 3 days to fix the lab to receive partial credit, up to 75%.

Late Policy

There will be no extensions given on the project except under extreme circumstances. A 3 point per day late penalty will be instituted.

EXTRA CREDIT – 10 points

Similarly, on the client side, the SFT client sends one request per file at a time. For extra credit, modify the client so that for each file request, 3 new threads are created, which each sends a request for that same file. The client does not need to follow the boss-worker thread pattern or worry about synchronization.