Project 1

Short File Transfer (SFT) Client and Server

Due October 7, 11:59pm

This project gives hands on practice with inter-process communication like lab 3. Processes that communicate over a network transfer data via sockets, an abstraction provided by the operating system. This project will utilize your string parsing skills and will give you more practice working with C's standard socket and file API.

SFT Client-Server

In this project, you will build a simple file transfer server that can transfer small files <= ~ 1KB. You will also build a client that sends file requests to the SFT server. The SFT server will respond to the request by sending the file contents to the client, which will write the contents to a new file.

The SFT Protocol

When processes communicate over a network, they follow a protocol so they know how to interact with each other. The SFT server expects to receive the correct protocol when a client makes a file request. When the server receives the protocol it knows to return a file to a client. The protocol the server understands is as follows:

GET filename EOR

For example, the request

GET r1.txt EOR

tells the server to send the contents of the file r1.txt to the client. Notice you must parse out the file name so that your server can open that file and return the contents. The text GET and EOR gives you markers to parse out the file name.

When the server receives this protocol, it will open the file requested and send back the contents. The server does not add a protocol to the response.

Project Set Up

Put sft_client.c, sft_client_main.c, sft_server.c, sft_server_main.c and the header files in the same directory. In that same directory put r1.txt, r2.txt, r3.txt, r4.txt, r5.txt. requests.txt should also go in the same 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. You have been given the design to use for your client and servers.
2. Client – The starting point (main function) of the client is in sft_client_main.c. In the main function you should open up "requests.txt". requests.txt contains 5 file names - r1.txt, r2.txt, r3.txt, r4.txt and r5.txt These are the only file requests the client will send to the server. Send each request one at a time to the function make_request in sft_client.c.

   sft_client.c will make the connection to the server and send the request using the protocol described above. It is also responsible for receiving the response and writing the response to a file in the client_files directory. The file it writes should have the same name as the file name requested. For example, if the client should make the request

   GET r1.txt EOR

   When it receives the file contents from the server, it should write the contents to

   ./client_files/r1.txt

   When you open r1.txt for writing you would use:

```
fopen("./client_files/r1.txt", "w");
```

3. Server – the server starts in sft_server_main. The main function calls the start_server function in sft_server.c.

   sft_server.c is responsible for accepting requests from the client, parsing the request to get a file name, and returning the contents of the file requested to the client.

4. Like the echo server, the SFT server will listen for connections from a client on the localhost at port 1600. The server does not terminate after sending back the file contents. It continues to run and service requests from clients until terminated at the command line with cntl c.

Compilation

To compile the server use:

gcc -o server sft_server.c sft_server_main.c

To compile the client use:

gcc -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 and the header files.

Grading

80 points - Implementation

Each program is worth 40 points.

10 points

The code is in the correct files – sft_client.c, sft_client_main.c, 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 lab3.c

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 lab. Once notified, you will have 3 days to fix the lab to receive partial credit, up to 75%.

EXTRA CREDIT – 20 points

Because the network interface is a shared resource, the OS does not guarantee that calls to write and read will copy the contents of the full range of memory that is specified to the socket. Instead, (for non-blocking sockets) the OS may copy as much of the memory range as it can fit in the network buffer and then let the caller know how much it copied via the return value.

In several ways, this behavior is desirable, not only from the perspective of the operating system which must ensure that resources are properly shared, but also from the user's perspective. For instance, the user may have no need to store in memory all of the data to be transferred. If the goal is to simply save data received over the network to disk, then this can be done a chunk at a time without having to store the whole file in memory.

In fact, it is this strategy that you should employ if you choose to do the extra credit. You are to modify your client to check the return from the read function and continue to read data from the socket until the read function returns < 0 indicating it has read all of the data from the socket. It should write the contents of each read to the file.

Make sure to test your program with very large files and confirm that the entire file has been transferred and your client writes the full file to disk.