Program 3: Peer-to-Peer File Download

Due: By 23:59 Sunday, November 9

Introduction

In this program you will accomplish several goals:

- Expand your knowledge of the socket API and peer-to-peer (P2P) application development
- Develop an application according to a documented standard
- Retrieve files from a remote source and save them locally
- Extend your P2P peer application

All programming assignments must be done in pairs.

Requirements

In this program, you will extend your peer-to-peer (P2P) peer application to include file download capabilities. Most P2P applications both upload and download files, but for this assignment your peer will only download files from solution peers that have upload capability.

Your application must still handle the commands from Program 2, so if your program is not working correctly get help immediately. The commands defined in Program 2 (JOIN, PUBLISH, and SEARCH) form the basis of this assignment, but you will now add new commands, as detailed in a following section.

Your peer must be able to:

- JOIN the P2P network by sending a JOIN request to the registry. (Program 2)
- PUBLISH the files available at the client to the registry. (Program 2)
- SEARCH for files using the registry. (Program 2)
- FETCH a file from another peer and save it locally. (*NEW*)

The peer application should follow these general steps:

1. Read user commands from the terminal.
    - If "JOIN", send a JOIN request to the registry.
    - If "PUBLISH", send a PUBLISH request to the registry.
    - If "SEARCH",
        (a) read a file name from the terminal,
        (b) send a SEARCH request to the registry,
        (c) and print the peer info from the SEARCH response or a message if the file was not found.
    - If "FETCH",
        (a) read a file name from the terminal,
        (b) send a SEARCH request to the registry for the file,
        (c) receive the peer information from the registry,
        (d) send a FETCH request to the identified peer,
        (e) and then receive and save the file information from the peer and save it to a file with the same name as the user requested in the local directory of the peer application.
    - If "EXIT", close the peer application.

A registry application (`p3_registry`) and peer application (`p3_peer`) are provided for you to use when testing your peer application. The applications are available on `jaguar` in the same directory as the testing script, `/user/home/kkredo/public_bin`. You can copy these files to your local directory if you wish to run them manually for testing. The applications run on `jaguar` and on the computers in OCNL 340; these applications might run on other Linux-based systems, but use it on these systems at your own risk. Provide the listening port number as the only required command line argument to the registry application. The provided peer application has the same arguments as your peer application. Two optional command line arguments may be used with the applications. An optional `-d` argument enables debug output, which you might find useful. An optional `-t` argument enables testing output and creates entries for several imaginary peers in the registry. You may find the `-t` argument useful when working on the PUBLISH command. *Note that the imaginary peers do not have any files.* The source code for these applications will not be released.

In order to test the functionality of your program, you will need to run several programs. You should create separate directories for each of these programs and run them from these separate directories. When running tests for your program, follow these steps:

1. Start the registry
2. Start the `p3_peer` application
3. Run the REGISTER command in `p3_peer`
4. Start your peer application
5. Perform any tests you want with your application
6. EXIT your peer application
7. EXIT the `p3_peer` application
8. Close the registry (Ctrl-C)

P2P Protocol

For this assignment, the P2P protocol extends the protocol defined for Program 2 by introducing the actions FETCH and REGISTER. For each new action, this section defines the purpose of each action, the message formats exchanged between a peer and the registry, and the command entered by the user running the peer.

Every P2P message starts with a 1 B Action field, which indicates the specific action included in that message. Some messages will cause the registry or a peer to send back a response message. Response messages do not contain an Action field.

FETCH

Purpose

Peers use the FETCH action to request a file from another peer in the P2P network. Peers must JOIN (or REGISTER to) the P2P network before sending a FETCH request. A peer that receives a FETCH request responds with a FETCH reply that indicates if the peer is able to send the file data. **Note that since your application only downloads files, it only sends FETCH requests. It does not need to receive and handle FETCH requests.**

Message Format

1 B	***Variable***
Action = 3	File Name

A FETCH request includes a 1 B field of 3 (Action equals 3) and then a variable-length, NULL-terminated file name. You may assume each filename is at most 100 B (including NULL).

A FETCH response (sent from the other peer to your peer) has the format:

1 B	***Variable***
Response	File data, if no error

A FETCH response includes a response code indicating if the peer is able to send the file. A response of **0 indicates success** and **any other value indicates error**. If the peer indicates success (response code equals 0), then the file data will immediately follow the response code. If the peer indicates error, it will send no further data. After sending a FETCH response, the peer will close the socket. **Make no assumption about the type of data nor the amount of data you will receive for a file.** You must be able to handle files of arbitrary size (i.e., you must use multiple calls to `recv`).

Terminal Input

A user enters the FETCH command by typing the string “FETCH” at the command prompt of the peer application. Next, on a separate line, the user enters the file name for the FETCH command. Thus, a complete FETCH command for the file “p3.pdf” would be:

1. “FETCH”
2. `<Enter>`
3. “p3.pdf”
4. `<Enter>`

where `<Enter>` is pressing the Enter key and quotes are removed.

REGISTER

Purpose

You are not expected to send or receive REGISTER messages. The information here is for your reference, since the provided peer uses the REGISTER command. An uploading peer (not your program) uses REGISTER to share its network information with the registry so that other peers can connect to it. An uploading peer must send a REGISTER request to the registry before any other peer send it a FETCH request. A peer may send a REGISTER request instead of a JOIN request. The registry does not send a REGISTER response.

Message Format

1 B	**4 B**	**4 B**	**2 B**
Action = 4	Peer ID	Peer IPv4 Address	Peer Port Number

The REGISTER request includes a 1 B Action field equal to 1, a 4 B peer id, an IPv4 address, and a 2 B port number. All fields are in network byte order.

Terminal Input

A user enters the REGISTER command by typing the string “REGISTER” at the command prompt of the peer application. There are no arguments to the REGISTER command.

Additional Requirements

Additional requirements for this program include:

- You may not use any form of sleep function, or equivalent, in this assignment and you may not use any socket flags (i.e., no `MSG_WAITALL`). You may not use `ioctl(2)` or equivalent functions.
- All submissions must pass compilation checks before they will be graded. Run the script `/user/home/kkredo/public_bin/program3_check` on `jaguar` to check your submission. Run the script without arguments or with the argument `help` for directions.
- Submit your program files through Canvas. Do not submit archive (zip/tar) files.
- You must include a Makefile with your submission, which builds the program by default and removes all generated files with the target `clean`. The peer executable must be named `peer`.
- The peer should accept three command line arguments. The first is the registry location, which may be provided as an IP address or a hostname. The second argument is the registry port number. The third is the peer's ID for this execution.
- Your program must compile cleanly on `jaguar` or the machines in OCNL 340 and you must use the gcc/g++ argument `-Wall`. You may not use `-w` to disable warnings.
- Check all function calls for errors. Points will be deducted for poor code.
- Put both group member names, the class, and the semester in the comments of all files you submit.
- To earn full credit, you must print addresses using `inet_ntop` and you may not use `memcpy` beyond copying single fields into our out of a packet.

Input/Output

User input and output is similar to Program 2, with the addition of the FETCH command. FETCH has user prompts similar to SEARCH.

Evaluation

Your program will be evaluated based on:

- 20 points: JOIN, PUBLISH, SEARCH, and EXIT commands work according to Program 2
- 15 points: Connection made to peer for FETCH request
- 20 points: FETCH request sent correctly
- 10 points: Text files transferred correctly
- 20 points: Binary files transferred correctly
- 10 points: All requests sent as single packets
- 5 points: Professional practices
  - `memset` or equivalent not used on more than one byte
  - Data copied only one time
  - `inet_ntop` used for IP address conversion
  - No socket flags or `ioctl` or similar functions
  - No `sleep` or delay functions

Creating the P2P Network for Testing

In order to test your implementation you will need to have several programs running that your peer can communicate with. **You should run these programs from separate directories if they all execute on the same machine.** You can create a test setup by following these steps:

1. Start the registry (`p3_registry`), which has the same arguments as Program 2. Note the IP address or hostname for the computer running the registry as well as the port number you provide as a command line argument. You are encouraged to use the `-d` flag for debugging purposes.
2. Create a `SharedFiles` directory in the directory of each peer and copy some files into each of those `SharedFiles` directories.
3. Start one or more solution peers (`p3_peer`), which have the same arguments as your peer in Program 2. Use the hostname or IP address and the port number you noted when you started the registry. **If you run the peers and registry all on jaguar, use `localhost` or `127.0.0.1`**, not jaguar's full hostname. While each peer will use the same hostname or IP address and port number, each peer must have a unique ID. You are encouraged to use the `-d` flag for debugging purposes.
4. On each solution peer, enter the `REGISTER` and `PUBLISH` commands.
5. Finally, start your peer using the hostname or IP address and port number you used for the solution peers, but use a unique ID. Perform any tests you desire.

Hints

- Beej's Guide to Network Programming[1] is a valuable resource.
- If the registry produces an "Address in use" error, then pick a different port number or wait for a couple minutes and try again.
- To help identify files within a directory, you can use opendir(3) and readdir(3) when using C or std::filesystem::directory_iterator when using C++. However, other methods are available as well.
- Ensure you pay attention to byte ordering. The byteorder(3) man page will be useful.
- Use Wireshark to help debug!
- Data type sizes vary across machines and operating systems in C and C++. Consider using defined size types, such as `uint32_t` for unsigned integer of 32 bits.
- Make no assumptions about the file contents. Any bit pattern may appear at any point in the file. **Do not use string functions on file data!**
- Test and debug in steps. Start with a subset of the program requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts. In general, the Evaluation requirements are ordered in a way to ease this implementation process.
- You may have an optional command line argument that enables debugging output. The debug argument must be optional (meaning that your program must run when it is not specified) and the debug output in your code must be cleanly organized.

[1] https://beej.us/guide/bgnet/