Program 2: Peer-to-Peer Introduction

Due: By 23:59 Sunday, March 9

Introduction

In this program you will accomplish several goals:

- Expand your knowledge of the socket API
- Get familiar with the peer-to-peer (P2P) application paradigm
- Develop an application according to a documented standard
- Design and implement a simple P2P peer application

All programming assignments must be done in pairs.

Requirements

In this program, you will implement part of a peer-to-peer (P2P) file sharing application. There are two roles for applications in a P2P network: the registry and the peer. Within each P2P network there are multiple peers, but only one registry, which keeps track of available peers and the files located at each peer (a centralized index). Peers that wish to share files contact the registry to: join the P2P network, announce the files available at the peer, find available peers, and locate files at other peers.

Your task is to create a peer application that interacts with an existing registry application according to the protocol defined in the next section. Your peer must be able to:

- JOIN the P2P network by sending a JOIN request to the registry.
- PUBLISH the files available at the client to the registry.
- SEARCH for files using the registry.

Your peer must be an interactive application that accepts user *commands* from the terminal. Thus, your application should wait for commands from the user before performing the JOIN, PUBLISH, and SEARCH *actions*. Your application performs an action by sending a *request* message to the registry and, for some actions, receiving a *response* message. The next section provides details on how the user specifies each of these commands and how your application will send or receive messages. Input and output examples are available later in the handout.

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 “EXIT”, close the peer application.

A registry application, `p2_registry`, is provided on `jaguar`, in the same directory as the testing script, for you to use when testing your peer application. The registry application runs 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. Any port number greater than 2000 and less than 65536 will work. Two optional command line arguments may be used with the registry. An optional `-d` argument enables debug output for the registry, which you might find useful. An optional `-t` argument enables testing output and creates entries for several imaginary peers. You may find the `-t` argument useful when working on the PUBLISH command. The registry source code will not be released.

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

1. Start the registry application `p2_registry`
2. Start your peer application
3. Perform any tests you want with your application
4. EXIT your peer application
5. Close the registry (Ctrl-C)

While an application may send multiple pieces of data using several calls to `send`, this incurs needless system call overhead. Collecting all the data in a single place and then making a single call to `send` is a better approach. To gain full credit on this, and future programming assignments, sending a protocol request must occur through a single `send` call. Handling partial sends is permitted, but you should attempt to send the entire request in one `send` call. You are not required to do this for `recv`, but you may wish to try after completing the assignment.

Some functions students commonly use in these programs have better equivalents or are unnecessary. Your program should not use `memset` or equivalent operations. These operations are almost always used due to a student misunderstanding or to cover bugs. If you desire to use `memset`, think carefully about what you really *need* to do. Additionally, `inet_ntoa` performs conversion, but has a much better equivalent in `inet_ntop`, which handles a broader selection of cases. Manually printing the address is even worse, so try not to do that. To earn full points, your program must not use `memset`, or equivalent operations, and it must use `inet_ntop` to convert IP addresses.

P2P Protocol

For this assignment, the P2P protocol consists of three actions: JOIN, PUBLISH, and SEARCH. For each 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 to send back a response message. Response messages do not contain an Action field.

General Request Message Format

1 B	***Variable***
Action	Request Data

JOIN

Purpose

Peers use the JOIN action to notify the registry that they wish to participate in the P2P network. Peers should JOIN the P2P network by sending an appropriate JOIN request to the registry before sending any other request. However, you do not need to enforce this in your program. A peer only needs to send the JOIN request once per session (once per execution) and sending multiple JOIN requests in the same session produces an error at the registry. A peer that closes its socket must send a new JOIN request before sending any other commands; the registry does not remember peers across connections. The registry does not send a response to a JOIN request.

Message Format

1 B	**4 B**
Action = 0	Peer ID

A JOIN request includes a 1 B field of 0 (Action equals 0) and then the 4 B peer ID. The Peer ID must be in network byte order. Each peer must have a unique ID, which is provided as a command line argument for this assignment.

Terminal Input

A user enters the JOIN command by typing the string "JOIN" at the command prompt of the peer application. There are no arguments to the JOIN command.

PUBLISH

Purpose

A peer uses the PUBLISH action to inform the registry about the files it has available to share. All files in the directory "SharedFiles" relative to the peer application should be PUBLISHed to the registry. You should not publish directories nor should you recurse into subdirectories. The registry updates its index of files available in the P2P network based on the PUBLISH requests it receives from peers. If a peer has multiple files to publish, all files must be listed within a *single* PUBLISH request. The registry does not send a response to a PUBLISH request.

Message Format

1 B	**4 B**	***Variable***
Action = 1	Count	File Names

A PUBLISH request includes a 1 B field containing 1 (Action equals 1), a 4 B file count, and a list of NULL-terminated file names. The PUBLISH request must contain Count file names in total with exactly Count NULL characters. Count must be in network byte order. You may assume each filename is at most 100 B (including NULL). No unused bytes are allowed between file names. A PUBLISH request will be no larger than 1200 B.

For example, if a peer PUBLISHed the two files "a.txt" and "B.pdf" then the raw PUBLISH request in Wireshark would be:

```
0x01 0x00 0x00 0x00 0x02 0x61 0x2e 0x74 0x78 0x74 0x00 0x42 0x2e 0x70 0x64 0x66 0x00
```

Terminal Input

A user enters the PUBLISH command by typing the string "PUBLISH" at the command prompt of the peer application. There are no arguments to the PUBLISH command.

SEARCH

Purpose

A peer uses the SEARCH action to locate another peer that contains a file of interest. Within the SEARCH request, the peer sends the file name the registry should locate. The registry sends a SEARCH response after it receives a SEARCH request from a peer. The SEARCH response indicates another peer that has the desired file. The registry will not locate a file PUBLISHED only by the peer sending the REQUEST.

Message Format

A SEARCH request (sent from peer to registry) has the format:

1 B	***Variable***
Action = 2	File Name

A SEARCH request includes a 1 B field containing 2 (Action equals 2) and a variable length, NULL-terminated file name.

For example, if a peer SEARCHed for the file "me.png" then the raw SEARCH request in Wireshark would be:

```
0x02 0x6d 0x65 0x2e 0x70 0x6e 0x67 0x00
```

A SEARCH response (sent from registry to peer) has the format:

4 B	**4 B**	**2 B**
Peer ID	Peer IPv4 Address	Peer Port Number

A SEARCH response includes the ID, IPv4 address, and port number of the peer that has the requested file. All fields are in network byte order. If the registry is unable to locate the file, then all SEARCH response fields will contain zero. *The registry will not return the info of the peer that sent the SEARCH request.*

For example, if the registry found the requested file at peer 17, which uses the IPv4 address 155.13.30.80 at port 4020, then the raw SEARCH response in Wireshark would be:

```
0x00 0x00 0x00 0x11 0x9b 0x0d 0x1e 0x50 0x0f 0xb4
```

Terminal Input

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

1. "SEARCH"
2. `<Enter>`
3. "p3.pdf"
4. `<Enter>`

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

Additional Requirements

Additional requirements for this program include:

- You may not use any form of sleep function 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/program2_check` on `jaguar` to check your submission. Run the script without arguments or with the argument `help` for directions.
- Submit your program files through Blackboard Learn. 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. Select a positive number less than $2^{32} - 1$ as the ID.
- 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.

Input/Output

Below is an input and output example for the peer. Make your programs match the style and formatting of these examples, but you are free to write your own error messages. Peer application output is colored blue and user input is colored red.

```
$ ./peer my.server 5000 16
Enter a command: JOIN
Enter a command: SEARCH
Enter a file name: found.pdf
File found at
 Peer 34
 201.18.192.56:8225
Enter a command: PUBLISH
Enter a command: SEARCH
Enter a file name: missing.txt
File not indexed by registry
Enter a command: EXIT
$
```

Evaluation

Your program will be evaluated based on:

- 10 points: Connection made to registry
- 10 points: Commands read from user
- 15 points: JOIN command sent correctly
- 20 points: PUBLISH command sent correctly
- 10 points: SEARCH command sent correctly
- 20 points: SEARCH response printed correctly
- 5 points: EXIT command handled correctly
- 5 points: All requests sent as single packets
- 5 points: `memset` or equivalent not used and `inet_ntop` used

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 (`p2_registry`). 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 your peer and copy some files into it.
3. Start your peer using 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.
4. Perform any tests you desire.

Hints

- **You are strongly encouraged** to create a function that performs a SEARCH operation and returns the peer information. This will help you on the next program.
- 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.
- 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/