Project #1 description

Important Dates

Date Assigned: October 22, 2019
Checkpoint: October 29, 2019 at 11:59pm
Date Due: November 5, 2019 at 11:59pm

Project Overview

Quick Response (QR) codes have become more popular given the rise of smart phones. A QR code can encode a URL or other information in a two-dimensional barcode. With a QR-code reader, a smart phone can use its built-in camera to take a photo of the QR-code and load the URL. This allows a user to access a long URL without having to type it and reducing the risk of typos.

Figure 1: Example QR Code

Mobile devices typically translate the QR code from an image file to the text output locally. In this project, we will have the mobile device offload this functionality to a server that will translate the QR code and provide the client with a URL to retrieve. This is mainly for educational purposes to motivate the use of a client-server architecture. However, this approach shows how computationally intensive processes can be out-sourced from a mobile device to save battery power.

In this project, we will create a QR code server for public use. Additionally, we will create a simple client to send the images and fetch the URLs. The goal is to become familiar with socket programming while learning more about how QR codes are used.

Project Specification

Expected QR code server functionality

Your server must run on the CS Linux machines on a specified port. The operator of this server would invoke it as: "./QRServer [option1, ..., optionN]". Below is a list of possible options. If an option is omitted, the default value should be used.

- PORT <port number>

- RATE <number requests> <number seconds>
- MAX_USERS <number of users>
- TIME_OUT <number of seconds>

We now specify the details of these options:

- **Listening port (PORT):** The server should listen on the specified TCP port to receive QR codes. Possible values: 2000 - 3000. Default setting: 2012.

- **Rate limiting (RATE):** Without reasonable safeguards, public servers could be attacked. To prevent this, the server operator should be able to specify the number of QR codes, specified by <number requests>, that should be allowed in a <number seconds> timeframe. Requests issued in excess of this rate should be discarded, and the occurrence of this condition **logged at the server**. The client should **not** be disconnected for violating rate limiting. Instead, they should receive an error message indicating why their command was not processed. Default: 3 requests per user per 60 seconds.

- **Maximum number of concurrent users (MAX_USERS):** Only a specified number of users should be able to connect to the QR code server at a given time. Above this threshold, connecting users should receive an error message indicating that the server is busy before terminating the connection. Such refused connections should be **logged at the server**. When a user disconnects, a slot should be opened for another connection. Default: 3 users.

- **Time out connections (TIME_OUT):** The server must monitor the connection for inactivity. If more than the specified number of seconds elapse since the last client interaction, the server must 1) report to the user that a time-out occurred using the **return code and server message format specified**, 2) close the TCP connection, and 3) **log the event at the server**. Default: 80 seconds.

Administrative Log

The administrative log reports user activity and system behavior for the server operator. All entries should be prefixed with the time and client IP address in **YYYY-MM-DD HH:mm:ss xxx.xxx.xxx.xxx** format, where 1) "YYYY-MM-DD" is the year, month, and day, 2) "HH:mm:ss" is the time of day in 24 hour format (HH can take values from 00-23), and 3) "xxx.xxx.xxx.xxx" represents the IP address of the client in dotted decimal format. Valid QR code interactions and invalid actions from clients should be logged. The log file must also contain information about server start-ups, connections, and disconnections from users, events as described by the RATE/MAX_USERS instructions, and any other information regarding user behavior or system events.

Expected Client and Server Interaction

Clients should connect to the QR code server using TCP. The server should provide separate and concurrent sessions for each client. This can be accomplished using a new process for each user (via

fork) or via threads. Upon establishing a connection, the client is allowed to perform zero or more interactions.

In an interaction with the client, the client begins by sending the size of the QR code image file, followed by a byte array of the indicated size containing the image file contents. The server then replies with the URL that the QR code represents. The details of these messages follows:

- **Client Message:** <32 bit unsigned integer representing file size><binary byte array of size indicated containing file contents>
- **Server Message:** <32 bit unsigned integer representing return code><32 bit unsigned integer representing URL character array length><character array of size indicated>

The server return codes are used to allow the server to reply to the client to provide additional feedback about the situation. Below are the valid return codes:

- 0 - Success. The URL is being returned as specified below.
- 1 - Failure. Something went wrong and no URL is being returned. The character array length is set to 0 and no character array is transmitted. This condition can be valid if the image uploaded does not represent a valid QR code or the client violates our network security requirements.
- 2 - Timeout. The connection is being closed. A human-readable text message is created and supplied as the character array. The character array length is set to the length of this human readable message.
- 3 - Rate Limit Exceeded. An error message about the rate limit being exceed is set in the character array with the size set to the character array.

Network Security

In a networking environment, you cannot assume that your clients will be acting in good faith. These attackers may be attempting to compromise your server through buffer overflows or denial of service attacks. Your server must protect itself:

1. The server must set a reasonable upper-bound for the image size it is willing to process. In the protocol specified above, the client could specify and upload a file of size (2^32-1 = 4,294,967,295 bytes or roughly 4GB). This upload could be used to reduce the server's available bandwidth while reserving a user connection slot, preventing other clients from connecting. Instead, you must set a lower limit to requests, returning a failure code for requests specifying a larger file size.
2. The server must not process any image file bytes that exceed the size specified by the client. The server must process only the the portion of the file specified by the client, discarding all other bytes that arrive before the server's response.
3. The server must prohibit clients from blocking other clients from connecting by using the timeouts specified above.

In each of these cases, we must resolve client violations within the above protocol. Do not disconnect a client simply for violating the file size upload limits or sending extraneous bytes.

Concurrency

Each connecting client must have a separate process or thread to handle its interactions. No possible execution paths of these concurrent threads or processes should lead to an error. Specifically, ensure that you protect all functions and library calls explicitly unless they are noted to be thread safe. As an example, when updating the administrative log file, make sure that events corresponding to multiple clients are recorded correctly. All memory should be freed when your server program exits. Specifically, there should not be any memory leaks or zombie threads or processes. Do not rely on the OS to clean up terminated threads or processes.

Resources and Restrictions

You are required to use either C or C++ for this assignment. You may use the C++ STL and in general any standard feature of C++17. You may use the native Linux/POSIX socket system calls, but you may **not** use any other socket libraries. No credit will be given to solutions that do not adhere to these requirements, so contact the instructor or the TAs for any clarification before using outside resources. These restrictions are being made so you become familiar with the details of lower-level socket programming.

While QR codes make a good motivational example of socket usage, understanding how they are created or parsed is not the goal for this project. Accordingly, students should use the **ZXing** library for generating and decoding QR codes. To simplify invocation, simply download the **supplied jar files** (https://canvas.wpi.edu/files/2200624) , which will act as a decoder for QR codes. You can invoke the decoder by typing " java -cp javase.jar:core.jar com.google.zxing.client.j2se.CommandLineRunner [image_file]" in the directory where the files are decompressed. In your server program, you will invoke this utility using system calls such as **system** or **exec**.

For testing purposes, you may use the above QR code which decodes to the URL for the course Web page. You can generate additional examples by using the Web application at **http://createqrcode.appspot.com/** (http://createqrcode.appspot.com/) and providing a URL as input.

As always, you are encouraged to avail yourself to Internet resources and Linux manual pages when completing the assignment. Socket tutorials, such as **http://beej.us/guide/bgnet/html/single/bgnet.html** (http://beej.us/guide/bgnet/html/single/bgnet.html) , will be helpful in understanding socket programming. However, you **must ensure that you write your own code** and **explicitly mention any resources you use including this web tutorial, the textbook,** and discussions with individuals outside of your group. In case of any questions about making use of a specific Web resource for the project, seek clarification in advance.

The following system calls will likely be useful in completing the assignment: **listen**, **accept**, **bind**, **recv**, **send**, **setsockopt**, **socket**, **close**, **gethostbyname**, **getprotobyname**, **htons**, **ntohs**, **fork**, **exec**, **dup2**, **waitpid**, **system**. Before getting started, you should consult the manual pages for each of these system calls. Additionally, you would need functions to perform thread or process synchronization.

Road Map

Like all programming projects, you should build your program incrementally. Below are a reasonable set of milestones in completing the project:

1. Create a single threaded server that accepts text from a client in a file, displays the text and disconnects the client.
2. Add support in the server for decoding QR codes.
3. Add support for accepting binary transmissions (this should be a straight-forward change from Step 1).
4. Support concurrent clients.
5. Add error checking and logging functionality.
6. Add security features.

Checkpoint Contributions

Students must submit work that demonstrates substantial progress towards completing the project on the checkpoint date. Substantial progress is judged at the discretion of the grader to allow students flexibility in prioritizing their efforts. **Projects that fail to submit a checkpoint demonstrating significant progress will incur a 10% penalty during final project grading.**

For this project, students can demonstrate substantial progress by submitting a version of the project that implements a single-threaded server that displays content to the client.

Deliverables and Grading

Students should use the following checklist in turning in their assignments:

1. Find a project partner (by end of week #1),
2. Submit the project code and documentation using the appropriate assignment on Canvas,
3. Make sure the project archive includes the **statement of autorship (https://web.cs.wpi.edu/~ldecarli/teaching/cs35161920B/syllabus.html#org0aa5a20)** ,
4. Schedule the project demonstration **[instructions coming soon]**.

When uploading to Canvas, please archive all project materials in a .zip, .tar.gz, or .tar.bz2 archive.

A **grading rubric (https://canvas.wpi.edu/files/2200642)** is available to give you a guide for how the project will be graded. No points can be earned for a task that has a prerequisite unless that prerequisite is working well enough to support the dependent task. For example, credit for max users requires the project to be able to support multiple users, otherwise the functionality is moot.

Groups **must** schedule an appointment to demonstrate their project to the teaching assistants. Groups that fail to demonstrate their project will not receive credit for the project. If a group member fails to

attend his or her scheduled demonstration time slot, he or she will receive a 10 point reduction on his or her project grade.

During the demonstrations, the TAs will be evaluating the contributions of group members. We will use this evaluation, along with partner evaluations, to determine contributions. If contributions are not equal, under-contributing students may be penalized.

Points of Clarification

- The command line arguments should be readable in any order. You are free to use libraries like getopt if you'd like. However, note that getopt does not let you specify two parameters to an option, thus making the RATE parameter challenging. You may instead use two options RATE_MSGS and RATE_TIME to specify that the limit is RATE_MSGS every RATE_TIME seconds.
- The "all memory should be freed when your server program exits" requirement is largely to catch when you fork off server processes. You should do proper garbage collecting rather than relying on the kernel to clean up for you. This should be true for your main process as well, but that's mostly just to encourage good coding practices.
- You need to create the client portion of this protocol as well and submit it for grading. While the specifications focused on the server portion, the client should provide what the server expects.
- "Error checking" in the project rubric refers to proper handling of clients that send files that are too large for the server or that send files that are not valid QR codes (e.g., the Java library cannot find a URL).
- Students should submit the .jar files and their tested images with their submissions.
- **Q: one of the system calls mentioned in the project description is deprecated on the OS I am using. What should I do?** A: you can go ahead and still use it, or use a more recent (non-deprecated) version if available.