CSci 4061 Spring 2015 | Assignment 5: Image Server

Due

Due on 7th May, 2015 at 11:59PM (Online only, no hard copy)

Purpose

The purpose of this assignment is to understand the use of sockets and simple client-server communication interactions.

Overview

In Assignment 4, you were asked to browse through a particular directory tree, locate and create a catalog of a certain subset of the files. However, this catalog was not accessible over a network. Building on assignment 4, in this assignment, you are tasked with writing two programs, a client and server, which communicate with each other via TCP sockets and facilitate sharing of files over a network. Since files downloaded over a network have a high probability of being corrupted, you will also implement error checking using a widely used hashing algorithm.

Assignment

Your task is to write two programs, a client and a server.

Program Invocation:

The server program will be invoked as follows:

`./server <port> <directory>`

a. The server program when started, should open a TCP socket on the specified port and wait for incoming client connections.
b. The server should browse the directory specified in `<directory>` and prepare a catalog.

The client program will be invoked as follows:

`./client <IP_address> <port> <optional_extension_argument>`

a. `<ip_address>` is an IPv4 address in dotted-decimal format, which specifies the IP address of the machine on which your server program is running.
b. `<port>` indicates the port on which your server is listening for connections.
c. The `<optional_extension_argument>` may or may not be present. The client should run in passive mode if this argument is present, and interactive mode if not present.

Server:

The server first browses through a specified directory and searches for image files of type `png`, `gif` and `tiff`. The server then creates a catalog file, named `catalog.csv`, which contains a record for each image file found (Details on creating this file are specified later in this handout). The server then starts listening for client connections on a specified port.

Client:

The client can be run in 2 modes: Interactive and passive. In both the modes, the client must first attempt to connect to a server. After the socket connection has been successfully set up, the client must download `catalog.csv` from the server. The contents of this file must then be displayed to the user on the screen/console.

Passive mode: In passive mode, the file extension will be provided as one of the command line arguments to the client program. This will be one of `png`, `gif` or `tiff`. The client program must then download all files corresponding to the specified type from the server.
Interactive mode: In the interactive mode, the client must allow the user to choose files for download. You must allow the user to download multiple image files in this mode of execution.
Interactive mode:

```
$ ./client 127.0.0.1 54432
===============================
Dumping contents of catalog.csv
[1] image01.tiff
[2] image02.png
[3] image03.gif
[4] image04.tiff
[5] image05.png
[6] image06.gif
===============================
Enter ID to download (0 to quit): 1
Downloaded image01.tiff
Enter ID to download (0 to quit): 3
Downloaded image03.gif
Enter ID to download (0 to quit): 0
Computing checksums for downloaded files..
Generating HTML file..
Exiting..
$
```

Passive mode:

```
$ ./client 127.0.0.1 54432 png
===============================
Dumping contents of catalog.csv
[1] image01.tiff
[2] image02.png
[3] image03.gif
[4] image04.tiff
[5] image05.png
[6] image06.gif
===============================
Running in passive mode..Downloading ‘png’ files..
Downloaded image02.png
Downloaded image05.png
Computing checksums for downloaded files..
Generating HTML file..
Exiting..
$
```

The text in `red` indicates user input.
The downloaded images must be saved in a subdirectory named `images`. This subdirectory must be created in the current working directory of the client program. After the files have been downloaded, the checksum must be computed for each downloaded file and compared with the

checksum in the downloaded `catalog.csv` file. An HTML file called `download.html` must also be created in the current working directory of the client program.

catalog.csv:

This is a CSV formatted file. You need to use a comma(,) as the delimiter. The header line for this file should be

```
filename, size, checksum
```

Following the header line, each image file found in the directory must have its own record in the file on a separate line. The filename must be stripped down to the last `/` (e.g `image01.png` instead of `/home/grad03/student/hw5/image01.png`). You need to compute the checksum for each image file. You may use the provided API for this. You may refer to `catalog.csv` in the attached tar ball.

download.html:

This is an HTML document which should list the names of all downloaded files with hyperlinks to the full size images. Additionally, “Checksum match” or “Checksum mismatch” must be displayed next to each filename depending on the result of the checksum comparison. (Note: If, for a particular file there is a checksum mismatch, you do not need to provide a link to the full size image, since the image is probably corrupted). You can find a sample `download.html` in the attached tar ball.

Computing checksums:

You have been provided with an API which computes the MD5 hash for a given file. You need to compile `md5sum.c` and link it with your server and client programs. You may use the provided `md5_test.c` and `Makefile` as a reference. (This program is identical to the `md5sum` command in Linux). Do NOT modify the `md5sum.[ch]` files. You will lose significant points if you do so.

Location of server/client:

Although it is possible to run both the server and the client on the same machine, for this assignment you need to ensure that the server and the client both run on different machines. For instance, you can run the server on one of the machines in Lind40 and the client on one of the machines in Keller 4250. (This is how your assignment will be graded, by running your server and client programs on physically different machines, so make sure that it works).

What you need to turn in

1. The source files of your programs (`*.[ch]` files)
2. A `README.txt` which contains the name and student ID of the team members. This file must also contain instructions about the way your programs must be run.
3. A `Makefile` for compiling the two programs. The two executables must be named `server` and `client` respectively.
4. Please do not include `download.html`, `catalog.csv`, `images/` and other image files in your submissions.
5. Submitting your assignment: All the above files must be in a single directory and the directory name must be the student IDs of the team members. This directory must be compressed and named `studentid1_studentid2.tar.gz`. The directory structure should look like this:
   ```
   studentid1_studentid2/Makefile
   studentid1_studentid2/README.txt
   studentid1_studentid2/*.[ch]
   ```

6. Failure to follow these instructions will result in a significant reduction in points.

Grading

- Set-up a TCP socket between the client and server programs
  - Both programs run on the same machine **[5 points]**
  - Both programs are run on different machines on the network **[15 points]**
- Preparing the catalog file on the server as specified **[20 points]**
- Downloading requested files from the server
  - Interactive mode **[10 points]**
  - Passive mode **[10 points]**
- Generating the HTML file on the client as specified **[20 points]**
- Syntax, documentation and readability **[10 points]**
- Compliance and Makefile **[10 points]**

Assumptions

Server-side:

1. The directory path argument to the server can be absolute or relative.
2. The specified directory can contain files other than the image files.
3. The specified directory can contain any number of subdirectories. You need to go through all the subdirectories and check for image files while preparing the catalogue.
4. The filenames encountered in the directory are guaranteed to be unique.
5. If `catalog.csv` exists, you must truncate/delete it and create a new file.

Client-side:

1. The `images` subdirectory may or may not exist initially. If it exists, it may contain some files. However, these files will not be found on the server (i.e. There will not be any filename conflicts during the download phase)
2. You can place all downloaded images in the top-level images subdirectory. There is no need to recreate the directory structure as per what is present on the server.
3. If `download.html` exists, you must truncate/delete the file and create a new file.
4. You may safely assume that the server program is always started before the client program starts.

Resources and Hints

- Socket programming under UNIX.
- Test your code with plaintext files initially, and then switchover to image files.
- Since you need to transfer multiple files over the sockets, figure out a way to specify the end of a particular file/message and the start of a new file/message: This can be a little tricky but not very hard.
- Your code will be graded on the CSELabs machines. Please make sure that it runs on these machines. Failure to do so will result in a significant reduction in points.
- Feel free to post any additional questions you might have on the moodle project discussion forum.

Sample server & client:

You have been provided with 2 executables, `server` and `client` which implement a TCP server and a TCP client respectively. You can first write a server and test it against the provided client or first write a client and test it against the provided server. You can then add the other modules as required for the assignment.

The messages printed on the screen are self explanatory, and should give you a clear idea of what is going on behind-the-scenes. Please be aware of this when testing your client/server with the provided server/client.

To start the server on a port (say 56789) on a machine with IP address 212.35.65.123, execute:

```
$ ./server 56789
```

To start the client and connect to the server, execute:

```
$ ./client 212.35.65.123 56789
```

Image copyrights: