Project 5: Parallel Downloads

Due: Sunday, April 17th, 2015 at 11:59pm

Description

Peer-to-peer (P2P) networks and technologies have completely changed how data is currently distributed through the Internet. When you visit a website, for instance, your web browser (the client) would contact and download a file from a web server that made the file available. This could often result in the overloading of the server if many clients tried to download the same file at the same time (e.g. flash crowds or the Slashdot effect). P2P networks don't have this problem because they rely on the clients themselves to help distribute a file to other clients in the network. This means that when you download a file over a P2P network, you are downloading different parts of it from different systems in parallel.

Downloading a file in parallel requires that parts of the file can be downloaded in any order and reassembled by the client. This requires that a file be divided into multiple blocks of data called chunks, which can be independently downloaded in any order. Effectively, a file is treated like a large array, with each chunk being one entry in that array. This allows a P2P client to download any chunk in the file and know exactly where in the array (file) the data should be placed, because it simply copies it to the location according to its index.

Requirements

In this project you will be creating one half of a P2P application named `client.c` that will download a single file from multiple remote hosts in parallel. We will provide you with the other half that provides the data to your client. In order to download the file in parallel you will need to spawn multiple threads that will each connect to a different file provider to request a specific chunk. Once a thread downloads a chunk it will have to write it out to the file `output.txt`. Note that because multiple threads are downloading chunks in parallel you will need to provide some synchronization to ensure the file is not corrupted.

Requirements

Your program will take as command line arguments a list of IP addresses and port numbers, like so:

```
./client <ipaddr1> <port1> <ipaddr2> <port2> ...
```

Your program will read the set of IP addresses and port numbers and create one thread for each IP address/port. This thread will then connect to the given host, and request a chunk from the file. After the chunk is received, it will request a new chunk until the complete file has been downloaded.

We will provide you with `serv.c`. This implements the server that will provide file chunks to your client.

Its usage is:

```
./serv -p <port> <filename>
```

`port` is the port number it will listen for connections on and `filename` is the name of the file it will provide to any client connections. When you connect to the server, you first have to send it the index number of the chunk you would like to request (as a string). The server will then send the chunk corresponding to the requested index.

You will be able to determine you have reached the last chunk in the file when the server closes the connection before a chunk is transferred. Note that when this happens the server might have sent a partial chunk or no data at all.

Server

You can get the server by copying it into your directory with the command:

```
cp /u/SysLab/shared/serv.c .
```

Please remember the dot at the end as it represents the current directory.

Example

We can launch as many servers to test against as you have ports. In this example, we'll open two servers on ports 1000 and 1001 and run them in the background by using the ampersand. We can then launch our client with the two IP/port combinations and our client will request chunks from the two files until the entire file has been transferred. We can then kill the two servers by sending their process id SIGTERM.

```
./serv -p 1000 file.txt &
[1] 19039
./serv -p 1001 file.txt &
[2] 19040
./client 127.0.0.1 1000 127.0.0.1 1001
kill 19039
kill 19040
```

Ports and Addresses

For this project we will be working on `thot.cs.pitt.edu`. On the class website there is a list of usernames and your designated, personal port numbers. Please use these ports and only these ports. For the address of the machine, we will simply refer to it as localhost, or localhost's reserved IP address: 127.0.0.1

Testing

thot.cs.pitt.edu is firewalled from the outside world, meaning that you will only be able to connect to your server from thot itself. In order to do this, your best bet for testing is to use a few programs:

- `telnet localhost PORT`
- `nc localhost PORT`

Hints/Notes

- For this project the chunk size will be **1024 bytes**
- The `serv` program takes an ASCII string as input, so you can telnet to it and type in the chunk index you want to download. If you ran `serv` with a text file you should be able to read and verify the output.
- It will be easier to write a non-parallel version first, that simply downloads the file using consecutive chunks, and then add in multiple threads once you get that working.
- You will need some way to coordinate which thread downloads which chunk. The easiest method is to assign each thread an index value and then only requests chunks that are a multiple of that threads index.
- You will need to write to multiple locations in the file from multiple threads. The fseek() function allows you to move around in a file, and change the location at which fwrite/fread access data. An example of its usage can be found in the serv.c file.

  ```
  fseek(stream, offset, SEEK_SET);
  ```

  This will move the location in the file to the byte position specified by 'offset'
- You will also need to coordinate access to the output file from multiple client threads. Because files are a shared resource you will need to use a mutex in order to protect the file from concurrent client operations.

Submission

You need to email the well-commented client program's source code to the TA.