EE488A

Advanced Programming Techniques for Electrical Engineering, Fall 2020

HW3 (300 points)

Due date: 12/18/2020 (11:59:59pm)

1. Introduction

In this assignment, you need to design and implement a web server which is not only running with multi-threads but also uses I/O multiplexing to handle concurrent requests from clients. Your ultimate goal is to build a high-performance web server that can process about 100,000 requests per second. There are two required programs;

(1) an event-based server with I/O multiplexing to handle requests, and
- Bootstrapping: Your server should make an inverted index of words in all documents in the folder. This should be done when the server is started, and the server should store the generated indices in the memory. (It is the same with HW1)
- Searching: Your server program should be able to handle a large number of request with multi-thread, and it should respond to relevant documents and lines by the queried words.

(2) 2 client programs that send multiple requests to a Web server.

You will implement a multi-thread text search server to handle a large number of search requests from multiple clients. The implementation may contain two main parts.

2. Server

(1) Bootstrapping

1. Your server should get an argument that specify the running folder when it started:
   [your_server_name] [absolute_path_to_target_folder] [port number]
2. The server should parse all files in the folder into inverted index and store it in its own memory. (refer to "lecture 5" on KLMS) Because we will only check text files, you don't have to consider binary files.

Notes:

- You can only use pointer arithmetics when parsing. In other words, you should not use string library.
- Generated inverted index should store line number with the file name.

- When parsing, you should deal with only alphabetic words. All white spaces and special characters should be considered as delimiter of words.
- It is the same with HW1. You can just use what you implemented.

Example)

Text	Parsed Words
abc bcd	[abc, bcd]
abc"b cd"	[abc, b, cd]
abc[bcd]	[abc, bcd]
abc "bcd"	[abc, bcd]

(2) Event-based server with I/O multiplexing to handle requests

1. In the server, there is a listener thread which receives requests from the client.
2. Listener thread manages multiple socket descriptors. Thus, you should use the "select()" function to pick pending inputs.
3. When the listener thread received a request, it delivers that request to a specific thread in a thread pool. Listener thread makes the connection with the client and receives a request from the client.
4. Idle thread in the thread pool should process request, and returns the search result to the client.
5. The tasks should be distributed to the *10 threads* in the thread pool.
6. The protocol should be follow the section *4. protocol*.

Notes:

- You have to use memory allocation when you make packet. If you use tc malloc that you implemented in HW2, you will get **extra** 60 points. (Which means, maximum points you can get is 360 points)
- The program should return same result with HW1. You can just use what you implemented.

3. Implementation

(1) Client1

1. The client should send multiple requests.

2. To generate simultaneous requests, we will use simple multiple threads using POSIX Threads (Pthreads) Interface.
3. After execution, the main process generates threads as the number of r equest from the arguments.
4. Then, each thread creates a new socket, connects the server.
5. You should receive two parameters for client code: number of threads & number of requests per thread.
6. Thus, the command line should be *"./client1 [server_ip] [server_port] [number_of_threads] [number_of_requests_per_thread] [word_to_searc h]"*, (e.g., ./client 127.0.0.1 8080 100 1000 abcd)

(2) Client2

1. You should print "(your_program_name)>" to identify CLI just like HW1. In this part, you will implement CLI client program for implemented ser ver.
2. Your input for searching word should follow below form exactly.
   search [word]
3. Your output for corresponded input should follow below form exactly.
   [filename1]: line #[line number1]
   [filename1]: line #[line number2]
   [filename2]: line #[line number3]
   [filename3]: line #[line number4]
4. The command line should be "./client2 [server_ip] [server_port]"

4. Protocol

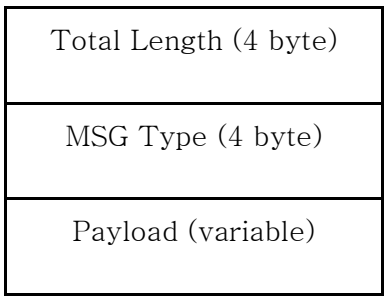

Figure 1. Protocol layout

Figure 1 denotes the protocol for this application, and server program should understand this protocol.

- Total Length: all payload length (application data including this header)

- MSG type
  - 0x00000010: REQ from client to server
  - 0x00000011: RESP from server to client
  - 0x00000020: ERROR from server to client, when the file is not in server.
- Payload: Text that contains the input or output of HW1.
  [filename1]: line #[line number1]
  [filename1]: line #[line number2]
  [filename2]: line #[line number3]
  [filename3]: line #[line number4]

5. Submission instructions

Use KAIST KLMS to submit your homework. Your submission should be one gzipped tar file whose name is YourStudentID_hw3.tar.gz. For example, if your student ID is 20201234, and it is for homework #3, please name the file as 20201234_hw3.tar.gz. If you do not follow this zip file name, you will lose some points.

Your zip file should contain **three things**:

1. *One PDF* file for document which explains your codes (hw3.pdf). This PDF file should show explanations for code in detail.
2. Your C *files and header files.*
3. *Makefile to compile your program*

*Do not include Korean letters in any file name or directory name when you submit.
Important note: Files not following the above format will be heavily punished! (You will get 0p)

Test environment : Ubuntu 18.04 build-essential.

Late submission policy : **We will not accept assignment submitted after deadline.**

Plagiarism

Discussions with other people are permitted and encouraged. However, when the time comes to write your solution, such discussions (except with course staff members) are no longer appropriate: you must write down your own solutions independently. If you received any help, you must specify on the top of your written homework any individuals from whom you received help, and the nature of the help that you received. Do not, under any circumstances, copy another person's solution. We check all submissions for plagiarism and take any violations seriously.

**** Plagiarism will get severely penalized if detected, 0 points for all assignments (both providers and consumers)**