Friday, 3 November 2017

EE 324, Programming Assignment #3

Implementing a web server that can handle large-scale requests

*** Update History**

- We changed the client command to *"./client [server_ip] [port] [number_of_threads] [number_of_requests_per_thread]"*
- ~~The thread_number should be assigned from 1 to the number_of_request (e.g., assuming that number_of_request is 20, then the first thread_number is 1, and the next is 2,.., and so on).~~ → ***in 3. Client, ignore it***

1. Overview

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 HTTP 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 HTTP requests, and (2) a client that sends multiple HTTP requests to a Web server. Figure 1 shows an overview of a workflow.

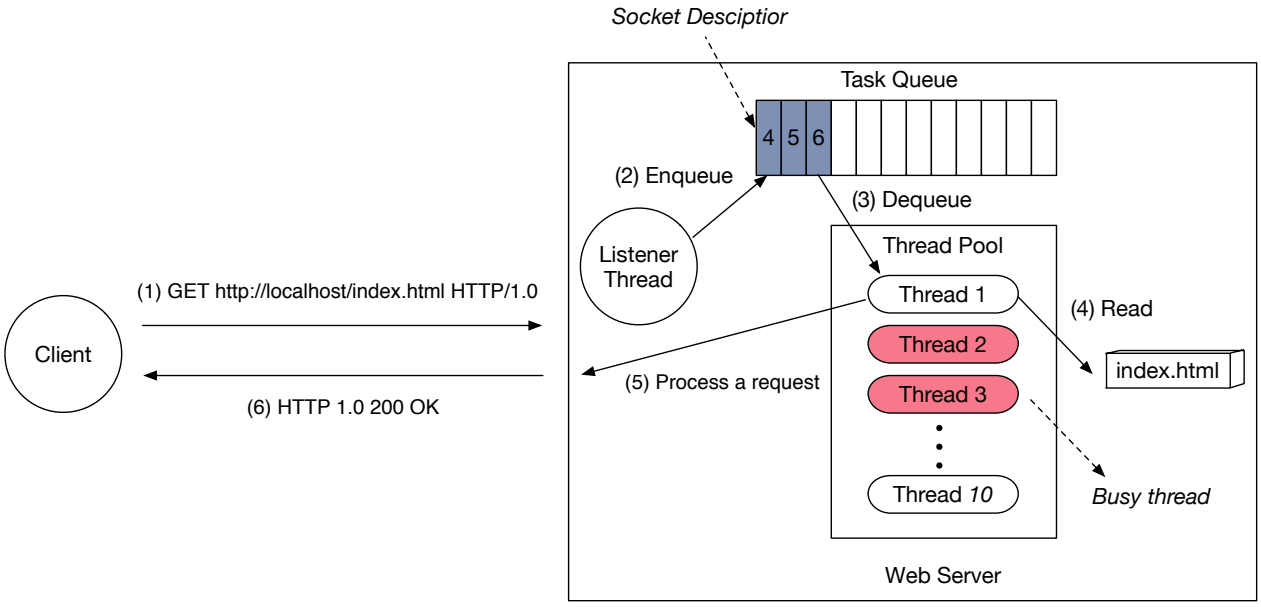

Figure 1 An overview of a workflow

2. Web server (event-based server with I/O multiplexing)

- Web server port should be 8080.

- In the web server, there is a listener thread which receives HTTP GET requests from the client.

- Listener thread manages multiple socket descriptors. Thus, you should use the “select()” function to pick pending inputs.

- When the listener thread received a request, it delivers that request to a specific thread in a thread pool.

- More clearly, listener thread makes the connection with the client and receives an HTTP GET request from the client. Then it enqueues socket descriptor to the task queue.

- Socket descriptors in the Task Queue are dequeued one by one, then an idle thread in the thread pool should process that task (HTTP request), and returns the result to the client.

- The tasks should be evenly distributed to the ***ten*** threads in the thread pool. For example, if a client sends 100 HTTP GET requests, each thread should process ten requests.

- When threads in thread pool process requests, they read “index.html” and return the HTTP response.

- Also, each thread should write a log.

  File Format: [thread_number].log (e.g., thread_1.log, thread_2.log …)

  Content: [timestamp] [client IP:port] [payload]

  * the timestamp format should be “*HH:mm:ss*”

  ex) thread_1.log

  17:02:21 143.248.111.14:12345 GET /index.html

  17:02:22 143.248.111.17:54321 GET /index.html

- The binary must be named as “web_server”

NOTE: index.html file for HTTP request will be uploaded soon.

NOTE: We recommend you to read

https://www.joinc.co.kr/w/Site/Thread/Advanced/ThreadPool (Korean)

https://docs.oracle.com/cd/E19253-01/816-5137/ggedn/index.html (English)

for the thread pool implementation.

3. Client

- The client should send multiple HTTP requests. However, in this assignment, we will only use HTTP GET request.
- To generate simultaneous requests, we will use simple multiple threads using POSIX Threads (Pthreads) Interface.
- And, the command line should be "./client [server_ip] [number_of_threads] [number_of_requests_per_thread]", e.g., ./client 127.0.0.1 8080 100 1000).
- After execution, the main process generates threads as the number of request from the arguments
- Then, each thread creates a new socket, connects the server
- Each request is HTTP GET request. The required field is Host field only.

  (e.g. GET /index.html/ HTTP/1.1

  Host: index.html)
- ~~The thread_number should be assigned from 1 to the number_of_request (e.g., assuming that number_of_request is 20, then the first thread_number is 1, and the next is 2,.., and so on).~~
- The binary must be named as **"client"**

NOTE: You can only use POSIX library for thread functions. Other third-party libraries should not be used.

4. Instruction for Submission

- You will be submitting one taball file to the KLMS website.
- Create a tarball (tar.gz) with all the source codes, README, and executables.
  - Tarball structure: (for program assignment 3)
    - Create a folder called "p3"
    - Put your source code in the folder named "p3/src"
    - Put your executable(s) in the folder named "p3/bin"
    - Put your "Makefile" in the root path - "p3/"
      - Make sure that your Makefile correctly complies your source code
    - readme.txt file goes to the root path - "p3/"

- No README, No points; let us know how to run your program

- install.sh goes to the root path as well - "p3/" (optional)

  - But, if your program needs a third party library, you should provide a script “install.sh”

- Then, compress the entire "p3" folder into a single tarball; the name will be **“P3_yourIDnumber.tar.gz”**

- Write the concise comments in the source code to understand your program.

! **IMPORTANT 1: Please strictly follow the above structure and name policy; if not, TAs will not evaluate your program.**

! **IMPORTANT 2: Please make sure that there is no compile and execution error. TAs will not give you any score in case of the problems.**

5. Test Case

1) Functionality – 100 points

I. Does web server use select() to pick pending connections? – 10 points

II. Can the web server process HTTP GET requests from the client? – 10 points

III. Can the web server send the result of the requests to the client? – 10 points

IIII. Can the web server process 100 requests from the client using thread pool? – 50 points

IV. Do threads in the thread pool process requests evenly? – 20 points

2) Performance – 50 points

V. We will send maximum 100,000 requests at one second using 100 threads (each thread sends 1,000 requests). If your server can handle all these requests well, you get full points. If your server can handle below the maximum, you get partial points.

6. Test Environment

- Language: C or C++

- Test O/S: **Ubuntu 14.04.5 LTS (Trusty Tahr) 64bit**

  - http://releases.ubuntu.com/14.04/

- If you need a VM, download using the following link.

  - http://nss.kaist.ac.kr/ee324.ova
  - username: ee324, password: ee324
  - Import the VM using VMware or VirtualBox
- **NOTE: We will not consider your compilation and execution problems due to the different OS versions.**

7. Due Date

- **11:59 PM, Nov. 19, 2017 (Sunday)**
- The website automatically marks the time of the last submission/modification. (The website often becomes unavailable, so please make sure that you submit your assignment early)
- **10% late penalty per day**
- Hard deadline: **11:59 PM, Nov. 24. 2017 (Friday)**
  - We will not receive additional submissions after the hard deadline.
  - Exceptions: documented medical/personal emergency.
- Please DO NOT e-mail Prof. Shin or TAs to submit your assignments.

8. Plagiarism

- You can discuss with your colleagues, but you should turn in your own programs

✓ Copy and Paste

- Will run plagiarism detection on source code
- “Copy and paste” codes will get severely penalized
- If detected, 0 points for all assignments (both providers and consumers)
- But you will have a chance to defend yourself

9. Questions?

- Please use KLMS Q&A board to ask any questions.
- Or you can email TAs to ask questions.

- Junsik Seo (js0780@kaist.ac.kr)
- Jinwoo Kim (jinwoo.kim@kaist.ac.kr)