Jinyong Jeong

CSCI 4730 (OS)

Professor Lee

README

Makefile command: make all, make clean

Run command: RUN SERVER FIRST

Run server: ./webserver_multi [port number] [# of threads]

Run client: ./client [host ip (ex. vcf0.cs.uga.edu) [port number] [# of threads]

Example: ./webserver_multi 2222 10 (create 10 worker threads on port 2222 to handle requests from client)

./client vcf0.cs.uga.edu 2222 20 (send 20 requests to webserver listening on port 2222 through host vcf0)

Project output - Server:

1. Worker thread: tid and pid
2. Confirmation of listener thread creation
3. Status message (listening on port X)

```
Creating worker thread: [tid #0, pid #3931]
Creating worker thread: [tid #1, pid #3931]
Creating worker thread: [tid #2, pid #3931]
Creating worker thread: [tid #3, pid #3931]
Creating worker thread: [tid #4, pid #3931]
Creating worker thread: [tid #5, pid #3931]
Creating worker thread: [tid #6, pid #3931]
Creating worker thread: [tid #7, pid #3931]
Creating worker thread: [tid #8, pid #3931]
Creating worker thread: [tid #9, pid #3931]
[TEST] Hi listener is created
HTTP server listening on port 2023
```

Project output – Client:

1. Request thread tid
2. Bytes received
3. Cumulative time taken to process
4. Total number of requests

```
09:58 PM~/csci4730/project2$ ./client vcf0.cs.uga.edu 2023 15
Request: GET vcf0.cs.uga.edu:2023//, # of client: 15
[tid 20411] received 1799 bytes (1.057383 sec).
[tid 20409] received 1799 bytes (1.057758 sec).
[tid 20413] received 1799 bytes (1.056863 sec).
[tid 20414] received 1799 bytes (1.055601 sec).
[tid 20408] received 1799 bytes (1.057901 sec).
[tid 20412] received 1799 bytes (1.057680 sec).
[tid 20407] received 1799 bytes (1.057992 sec).
[tid 20410] received 1799 bytes (1.218623 sec).
[tid 20415] received 1799 bytes (1.215500 sec).
[tid 20416] received 1799 bytes (2.008505 sec).
[tid 20418] received 1799 bytes (2.051494 sec).
[tid 20420] received 1799 bytes (2.050817 sec).
[tid 20419] received 1799 bytes (2.051299 sec).
[tid 20417] received 1799 bytes (2.052270 sec).
[tid 20421] received 1799 bytes (2.050544 sec).
Time to handle 15 requests (0 failed): 2.059505 sec
10:16 PM~/csci4730/project2$
```

Server output – post client call:

```
[pid 3931, tid 14] Received a request from 128.192.101.135:3177
[pid 3931, tid 14] (from 128.192.101.135:3177) URL: GET / HTTP/1.0
[pid 3931, tid 14] Reply: SUCCEED
[pid 3931, tid 13] Received a request from 128.192.101.135:3179
[pid 3931, tid 13] (from 128.192.101.135:3179) URL: GET / HTTP/1.0
[pid 3931, tid 13] Reply: SUCCEED
[pid 3931, tid 7] Received a request from 128.192.101.135:3181
[pid 3931, tid 7] (from 128.192.101.135:3181) URL: GET / HTTP/1.0
[pid 3931, tid 7] Reply: SUCCEED
[pid 3931, tid 6] Received a request from 128.192.101.135:3183
[pid 3931, tid 6] (from 128.192.101.135:3183) URL: GET / HTTP/1.0
[pid 3931, tid 8] Received a request from 128.192.101.135:3185
[pid 3931, tid 8] (from 128.192.101.135:3185) URL: GET / HTTP/1.0
[pid 3931, tid 11] Received a request from 128.192.101.135:3191
[pid 3931, tid 11] (from 128.192.101.135:3191) URL: GET / HTTP/1.0
[pid 3931, tid 10] Received a request from 128.192.101.135:3189
[pid 3931, tid 10] (from 128.192.101.135:3189) URL: GET / HTTP/1.0
[pid 3931, tid 12] Received a request from 128.192.101.135:3187
[pid 3931, tid 12] (from 128.192.101.135:3187) URL: GET / HTTP/1.0
[pid 3931, tid 8] Reply: SUCCEED
[pid 3931, tid 10] Reply: SUCCEED
[pid 3931, tid 6] Reply: SUCCEED
[pid 3931, tid 11] Reply: SUCCEED
[pid 3931, tid 12] Reply: SUCCEED
```

1. Worker thread tid and pid
2. SUCCESS / FAILURE message

Time explanation:

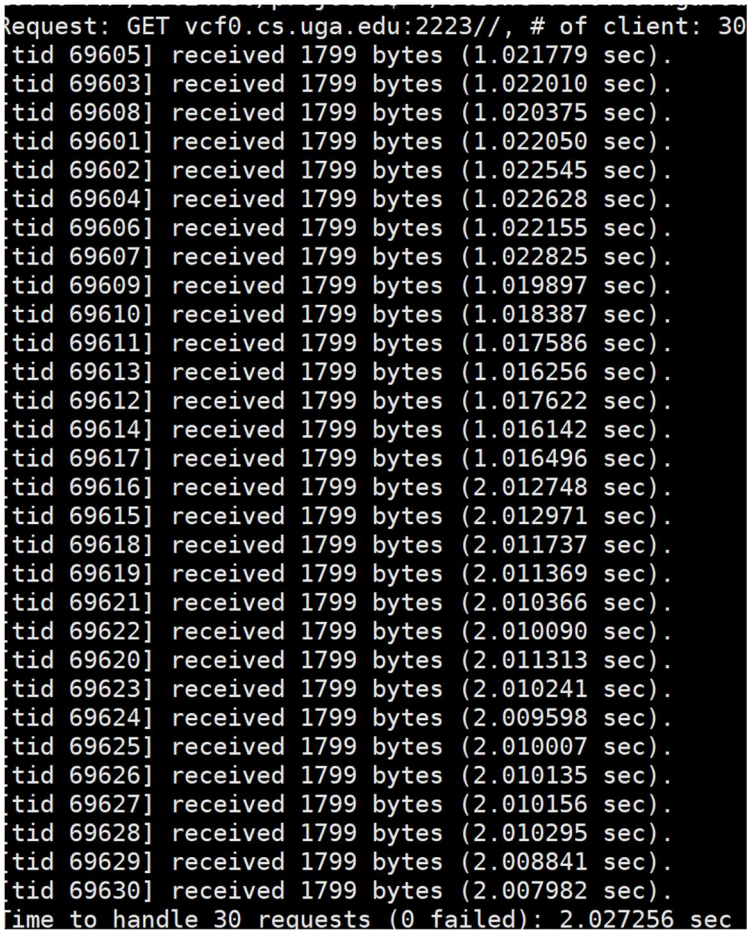
Each request if processed 1 at a time will take 1 second on average. That means that when run **sequentially, t**he webserver **will take 15 seconds to complete 15 requests.**

```
Request: GET vcf0.cs.uga.edu:2223//, # of client: 15
[tid 60368] received 1799 bytes (1.016844 sec).
[tid 60370] received 1799 bytes (2.018176 sec).
[tid 60372] received 1799 bytes (3.019005 sec).
[tid 60373] received 1799 bytes (4.020126 sec).
[tid 60369] received 1799 bytes (5.021420 sec).
[tid 60371] received 1799 bytes (6.022296 sec).
[tid 60374] received 1799 bytes (7.022476 sec).
[tid 60375] received 1799 bytes (8.021408 sec).
[tid 60376] received 1799 bytes (9.020219 sec).
[tid 60379] received 1799 bytes (10.020189 sec).
[tid 60378] received 1799 bytes (11.021228 sec).
[tid 60377] received 1799 bytes (12.022460 sec).
[tid 60380] received 1799 bytes (13.022533 sec).
[tid 60381] received 1799 bytes (14.023374 sec).
[tid 60382] received 1799 bytes (15.024347 sec).
Time to handle 15 requests (0 failed): 15.032269 sec
10:40 PM~/csci4730/project2$
```

Through **parallel** threads, we cut down the time of processing GET requests drastically. Thanks to multithreading, each thread **pool** session takes 1 second.

```
Request: GET vcf0.cs.uga.edu:2223//, # of client: 15
[tid 53591] received 1799 bytes (1.009618 sec).
[tid 53589] received 1799 bytes (1.010013 sec).
[tid 53588] received 1799 bytes (1.010102 sec).
[tid 53592] received 1799 bytes (1.009984 sec).
[tid 53593] received 1799 bytes (1.009925 sec).
[tid 53587] received 1799 bytes (1.010915 sec).
[tid 53590] received 1799 bytes (1.011001 sec).
[tid 53595] received 1799 bytes (1.008433 sec).
[tid 53596] received 1799 bytes (1.007437 sec).
[tid 53597] received 1799 bytes (1.007174 sec).
[tid 53598] received 1799 bytes (1.007255 sec).
[tid 53599] received 1799 bytes (1.007704 sec).
[tid 53601] received 1799 bytes (1.007622 sec).
[tid 53600] received 1799 bytes (1.007871 sec).
[tid 53602] received 1799 bytes (1.008198 sec).
Time to handle 15 requests (0 failed): 1.014219 sec
10:35 PM~/csci4730/project2$
```

The above example used a 15 threadpool – So, for 15 requests, we only need **1 pool session** to run process all requests – resulting in a 1 second handling time



The above picture uses the same **15 thread pool** to handle **30 GET requests.** The process will take 2 pool sessions to complete. Thus, it takes **2 seconds through parallel processin**g. If we had run everything sequentially, the **process would have taken 30 seconds.**

Logic Explanation:

The server utilizes 2 threads to process requests – the listenerthread and the workerthread. The workerthread is initialized as a threadpool set by the user. When the client sends a request, the listener stores the request in an array **buffer.** The threadpool traverses through the buffer and takes out the socket value sent by the client, then sends them through the process.

The project utilizes 2 semaphores (empty, full), and a semaphore functioning as a lock (mutex). The mutex lock prevents race conditions and deadlock by preventing items in the buffer from being changed at the same time process is called.

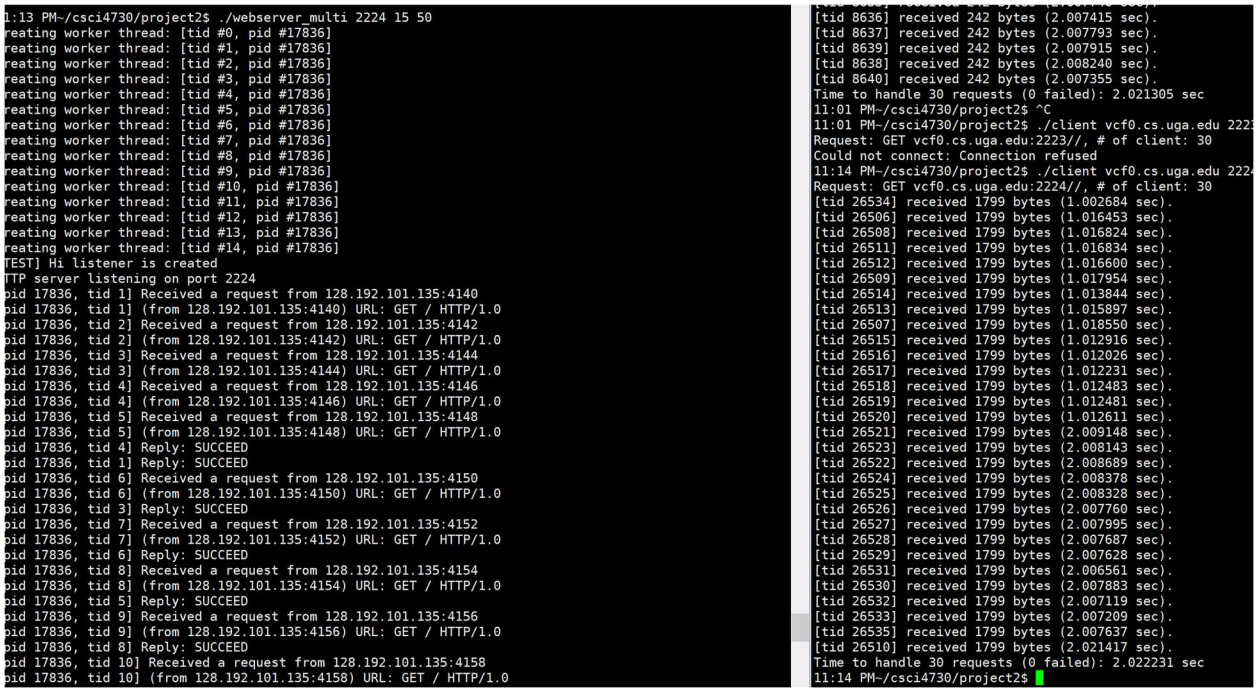
```
sem_wait(&semEmpty); //empty buffer count -1
sem_wait(&semMutex);

buffer[buffermarker] = s;
buffermarker = buffermarker+1;

sem_post(&semMutex);
sem_post(&semFull); //full buffer count +1;
```

We use two separate integers to mark our places in the buffer- ‘buffermarker’ for the listener, and ‘bufferat’ for the worker string. The two work independently of each other to ensure a parallel process.

Crash handling:

Crash handling should be able to be called via ./webserver_multi [port number] [thread pool number] [crash rate].

However, even when using the parameter crash rate, I was unable to trigger a crash. The client reported all bytes received satisfactorily.



Even at crash rate 50, the server navigated all requests successfully.