Zhenyu Yan
CSCI4730

Project 3 Multi-threaded Web Server report

In this project , I tackle multi-threading. Your job design & implement an efficient multithreaded webserver given working serial (single-threading) webserver.

Implementation of thread control:

In the single-thread webserver, the listener calls the method process(s) to do the request one by one. As requires, we need to have a thread pool model to deal with multi-thread webserver. We need to have N worker threads and One listener thread.

Here is the thread control method to create and handle threads.

```
void thread_control()
{
  int i;
  // file descriptor max is 1
  sem_init(&empty, 0, 1);
  sem_init(&full, 0, 0);
  // mutax to lock the insert and pop request
  sem_init(&mutex, 0, 1);

  // create worker's thread
  pthread_t threadPool[numThread];
  for(i = 0; i < numThread; i++){
    pthread_create(&threadPool[i], NULL, workerThread, NULL);
  }

  // listerner thread
  pthread_t listenerThread;
  pthread_create(&listenerThread, NULL, (void*) &listener,  NULL);

  //wait on the listener end which it never end
  pthread_join(listenerThread, NULL);
}
```

Modified listener code:

```
while( 1 ) {
      int s;
      s = accept(sock, NULL, NULL);
      if (s < 0) break;
      // add s to the queue
      enqueue_request(s);
}
```

Workers code:

```
void * workerThread(){
  while(1){
    process(dequeue_request());
  }
}
```

Implementation of synchronization:

For synchronization, listener will have a enqueue_request() function to push the request into a queue. Then the workers will have a dequeue_request() function to pop a request and process it. The most important part is that two semaphores empty and full to solve the race condition for workers thread to access the same request. The mutex semaphores is to ensure avoid the deadlock.

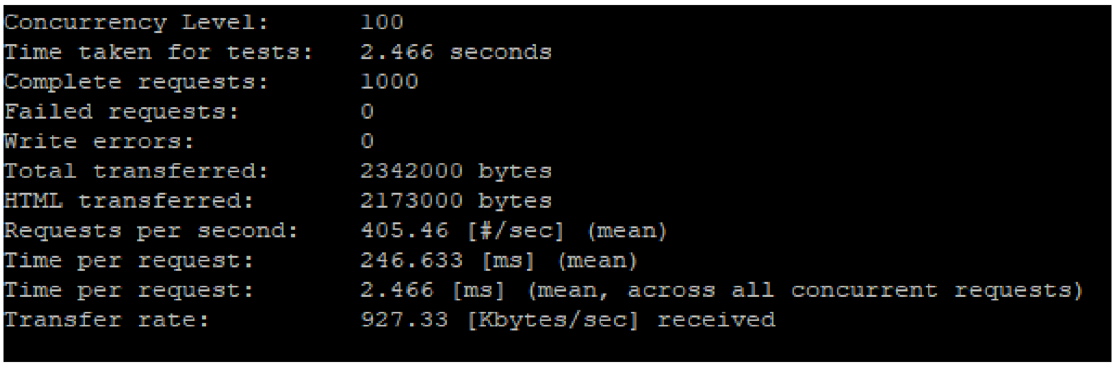
```
int enqueue_request(int s){
  sem_wait(&empty);
  sem_wait(&mutex);
  /* critical section*/
  f1->value = s;
  push(f1);
  /* critical section*/
  sem_post(&mutex);
  sem_post(&full);
}

int dequeue_request(){
  int value;
  sem_wait(&full);
  sem_wait(&mutex);
  /* critical section*/
  queue *f;
  if ((f = pop())!= NULL) {
    value = f->value;
  }
  /* critical section*/
  sem_post(&mutex);
  sem_post(&empty);
  return value;
}
```

Result:

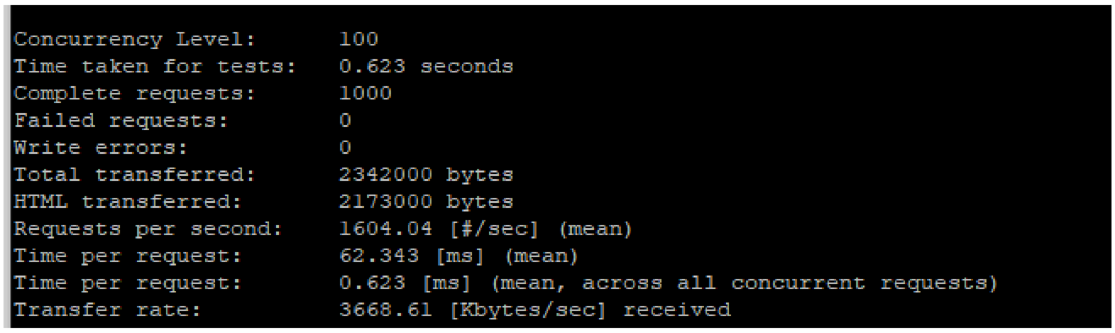
Below is the result of 1, 3, 5, 7, 10, 25, 50 threads result with 1000 request and 100 concurrency level.

- **1 thread**



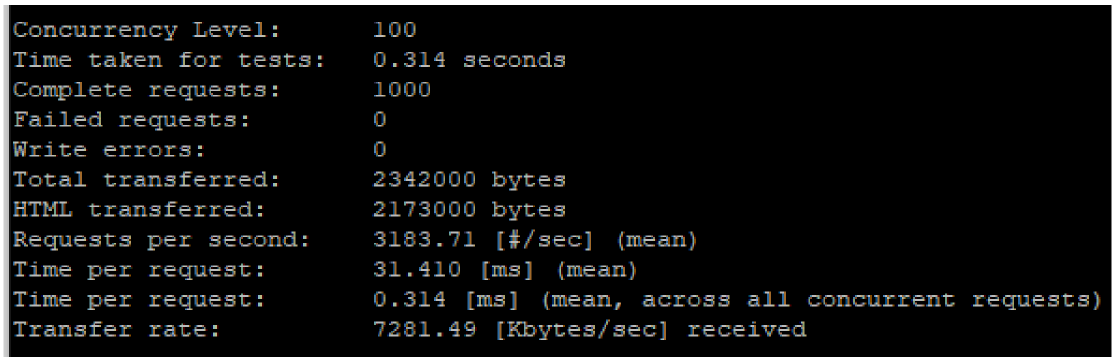
```
Concurrency Level:      100
Time taken for tests:   2.466 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2342000 bytes
HTML transferred:       2173000 bytes
Requests per second:    405.46 [#/sec] (mean)
Time per request:       246.633 [ms] (mean)
Time per request:       2.466 [ms] (mean, across all concurrent requests)
Transfer rate:          927.33 [Kbytes/sec] received
```

- **3 threads**



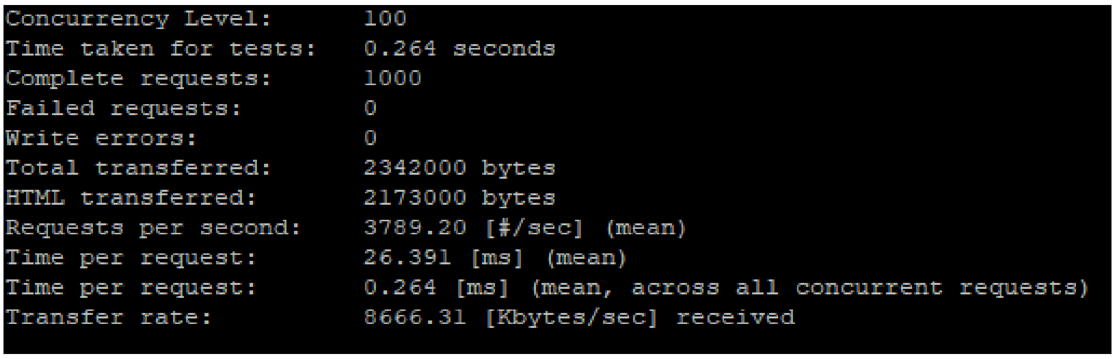
```
Concurrency Level:      100
Time taken for tests:   0.623 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2342000 bytes
HTML transferred:       2173000 bytes
Requests per second:    1604.04 [#/sec] (mean)
Time per request:       62.343 [ms] (mean)
Time per request:       0.623 [ms] (mean, across all concurrent requests)
Transfer rate:          3668.61 [Kbytes/sec] received
```

- **5 threads**



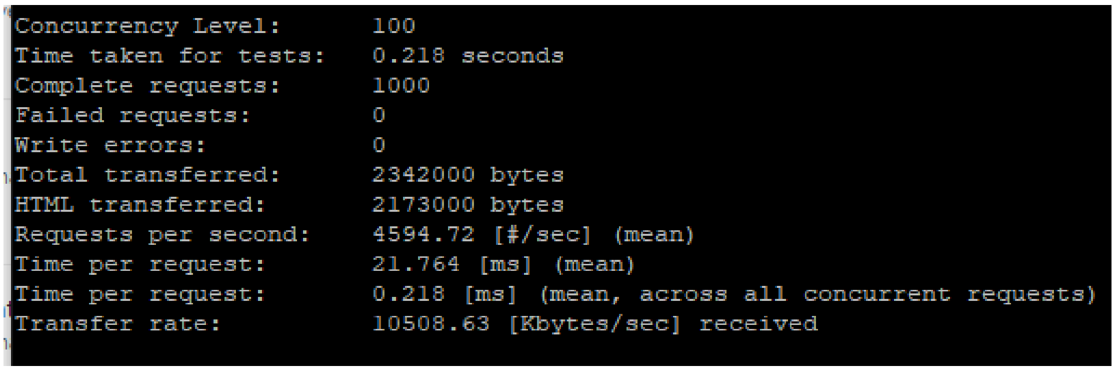
```
Concurrency Level:      100
Time taken for tests:   0.314 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2342000 bytes
HTML transferred:       2173000 bytes
Requests per second:    3183.71 [#/sec] (mean)
Time per request:       31.410 [ms] (mean)
Time per request:       0.314 [ms] (mean, across all concurrent requests)
Transfer rate:          7281.49 [Kbytes/sec] received
```

- **7 threads**



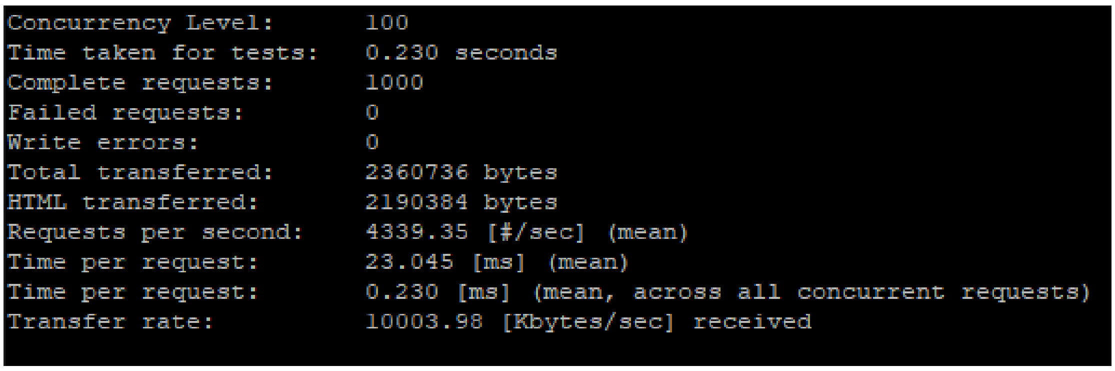
```
Concurrency Level:      100
Time taken for tests:   0.264 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2342000 bytes
HTML transferred:       2173000 bytes
Requests per second:    3789.20 [#/sec] (mean)
Time per request:       26.391 [ms] (mean)
Time per request:       0.264 [ms] (mean, across all concurrent requests)
Transfer rate:          8666.31 [Kbytes/sec] received
```

- **10 threads**



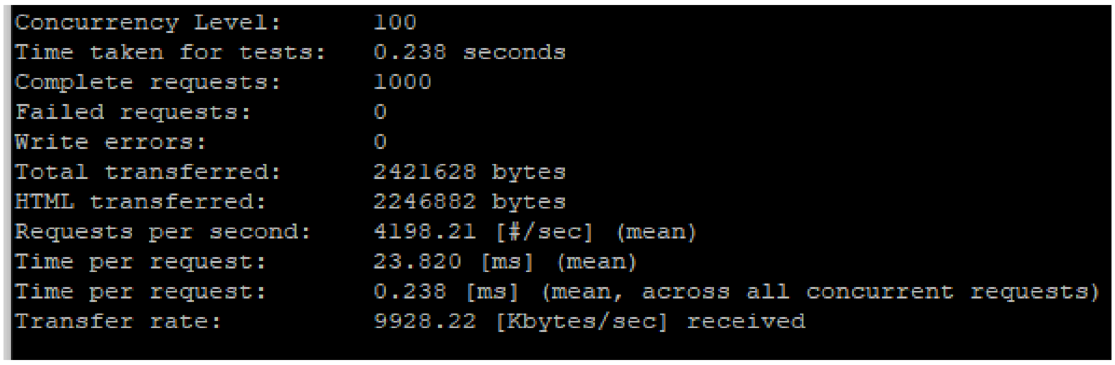
```
Concurrency Level:      100
Time taken for tests:   0.218 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2342000 bytes
HTML transferred:       2173000 bytes
Requests per second:    4594.72 [#/sec] (mean)
Time per request:       21.764 [ms] (mean)
Time per request:       0.218 [ms] (mean, across all concurrent requests)
Transfer rate:          10508.63 [Kbytes/sec] received
```

- **25 threads**



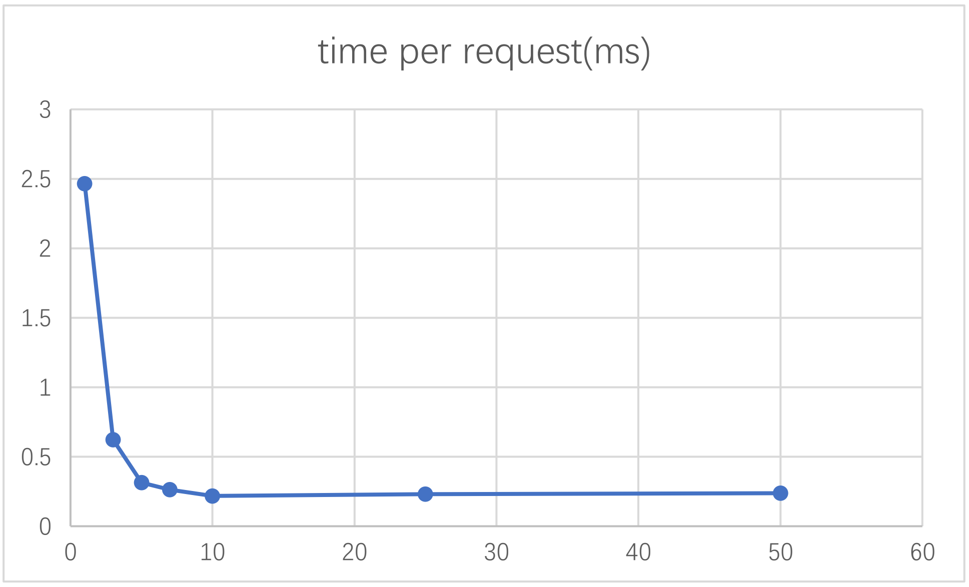
```
Concurrency Level:      100
Time taken for tests:   0.230 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2360736 bytes
HTML transferred:       2190384 bytes
Requests per second:    4339.35 [#/sec] (mean)
Time per request:       23.045 [ms] (mean)
Time per request:       0.230 [ms] (mean, across all concurrent requests)
Transfer rate:          10003.98 [Kbytes/sec] received
```

- **50 threads**



```
Concurrency Level:      100
Time taken for tests:   0.238 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2421628 bytes
HTML transferred:       2246882 bytes
Requests per second:    4198.21 [#/sec] (mean)
Time per request:       23.820 [ms] (mean)
Time per request:       0.238 [ms] (mean, across all concurrent requests)
Transfer rate:          9928.22 [Kbytes/sec] received
```

And here is a graph of the result.



Form above, as the numbers of thread increase, the time per request become faster because there are more workers to work on it. However after 10 threads, the speed is close to 0.230 ms. We can conclude, performance increase as the numbers of threads increase till certain threshold, then the speed will remain close to a constant.

Video demo link will be in the README file.