Assignment 2

Li Dongming, 119020023

Design of Program (Main Task & Bonus)

Main Task

1. Besides the main thread, create another two threads, one for moving the logs and another for listening to keyboard hit and move the frog correspondingly. The main thread also takes care of printing the game status too.
2. [main thread] - every 0.01 second, check the game status (win, lose, quit, or playing) and print the game, i.e. `map`. If the game is ended, i.e. not playing, print corresponding messages. When the game is ended, the main thread exits and other threads will also be terminated automatically.
3. [log_move] - every 0.1 second, move the logs (and the frog on it, and checking whether the frog is out of the game playground) and put them in the global variable `map`.
    - Note: different locks are acquired when operating on `map` and `frog`.
    - Note: **THIS IS A GAME!!!**. So there is some randomness in it. When initializing the game, the positions of the leading logs at each line are randomly initialized to satisfy the constraint that *it is the leading one*. Logs move one space each 0.1 second. The length of the logs at each row are also randomly initialized. In order to make the game looks like the one shown in demo, the spaces between logs at each line are set so that when one is leaving the game playground, the next one is entering with exact to boundaries. I would **STRONGLY DISAPPROVE** to follow the demo exactly because it is against the spirit of creativity and engagingness inherent in games.
4. [player_hit] - every 0.05 second, check whether user hits any keys and perform corresponding action for the frog. It will also check whether the frog is in water every time the frog is moved by user (i.e. lose the game) or is at the other side of the bank (i.e. win the game).

Bonus

1. Some core ideas of my implementation, although not perfect:
    - two queues: one for waiting jobs and the other for waiting thread workers (will appear as worker in the following content of this report).
    - different `cond_signal` for different workers, i.e. 5 workers will have 5 different `cond_signal`. This is designed this way so that we have to choice to schedule the jobs. It would be useful if different workers have different properties, such as workers in different machines in a distributed setting.
2. Workflow:
    - when each new job comes in, first check if there is any worker waiting for jobs. If there is, directly dispatch the incoming job to the available worker (by `pthread_cond_signal` and global variables specific for this thread to store job function and job arguments) and remove that worker from worker queue. If multiple workers are available, choose

the one at the front of worker queue. However, if all workers are busy, add the incoming job to job queue.
    - when a worker finishes its job, it will look at the job queue. If job queue is empty, simply put itself back to worker queue and wait. If there are jobs in the queue, dequeue the first job from the queue and do the job.

Environment

For both the main task and bonus, the tested environments are as follows:

- WSL1.0 on Windows 10: kernel version 4.4.0-19041-Microsoft, gcc version 7.5.0, g++ version 7.5.0, Ubuntu version 18.04.
- Virtualbox VM on Windows 10: kernel version 5.10.146, gcc/g++ version 5.4.0, Ubuntu version 16.04.

Steps to Execute

Main Task

Use the Makefile to compile. Note that you need to install ncurses library first.

```
# in the source code directory
sudo apt-get install libncurses5-dev libncursesw5-dev
make
./hw2
```

Bonus

Use the Makefile to compile.

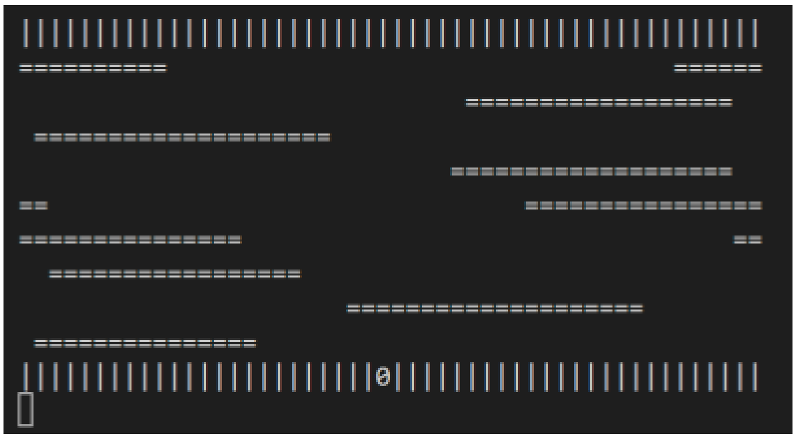
```
# in the thread_poll directory
make
# use proxy
./httpserver --proxy inst.eecs.berkeley.edu:80 --port 8000 --num-threads 5
# use local file
./httpserver --files files/ --port 8000 --num-threads 5
```

Local file is preferred over proxy because the latter may suffer connection problem.

Screenshots

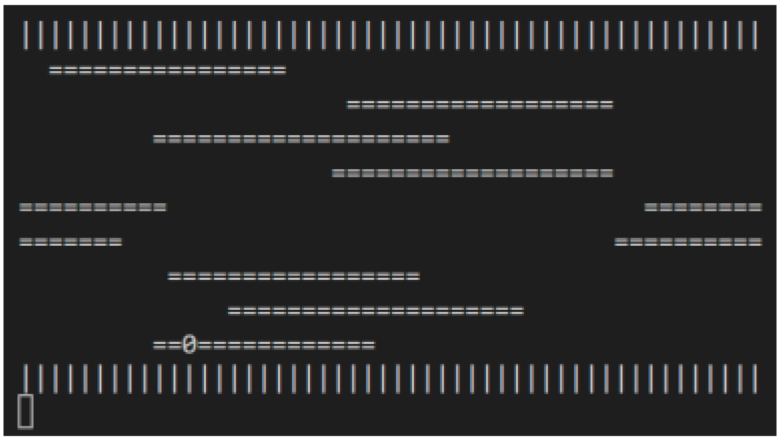
Main Task

Initial state:

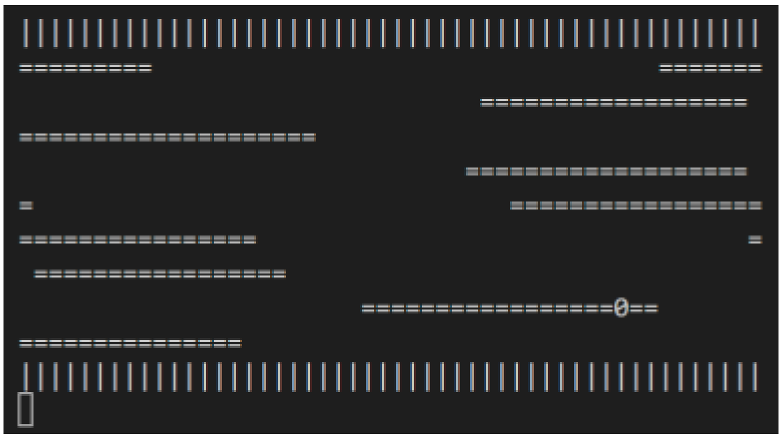


```
||||||||||||||||||||||||||||||||||||||||||||||||||
==========                                   ======
                              ==================
 ====================
                             ===================
==                                ================
===============                                  ==
  =================
                      ====================
 ===============
||||||||||||||||||||||||0|||||||||||||||||||||||||
```

Playing:

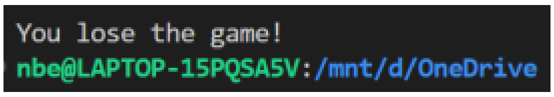


```
||||||||||||||||||||||||||||||||||||||||||||||||||
  ================
                      ==================
         ====================
                     ===================
==========                                ========
=======                                 ==========
          =================
              ====================
         ==0============
||||||||||||||||||||||||||||||||||||||||||||||||||
```

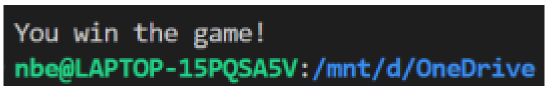


```
||||||||||||||||||||||||||||||||||||||||||||||||||
=========                                   =======
                               ==================
====================
                              ===================
=                                 =================
================                                  =
 =================
                       =================0==
===============
||||||||||||||||||||||||||||||||||||||||||||||||||
```

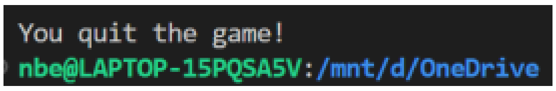
Game ended:



```
You lose the game!
nbe@LAPTOP-15PQSA5V:/mnt/d/OneDrive
```



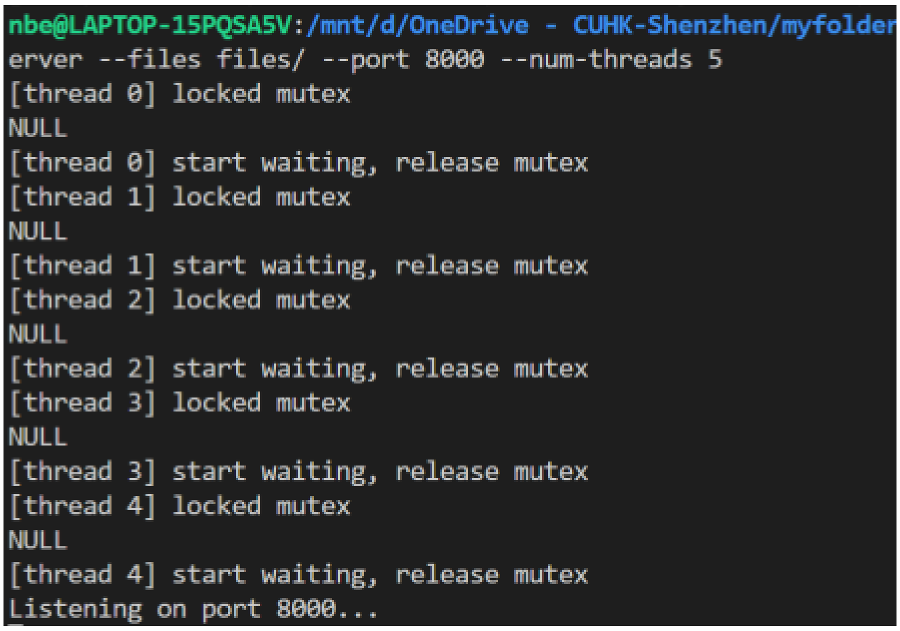
```
You win the game!
nbe@LAPTOP-15PQSA5V:/mnt/d/OneDrive
```



```
You quit the game!
nbe@LAPTOP-15PQSA5V:/mnt/d/OneDrive
```

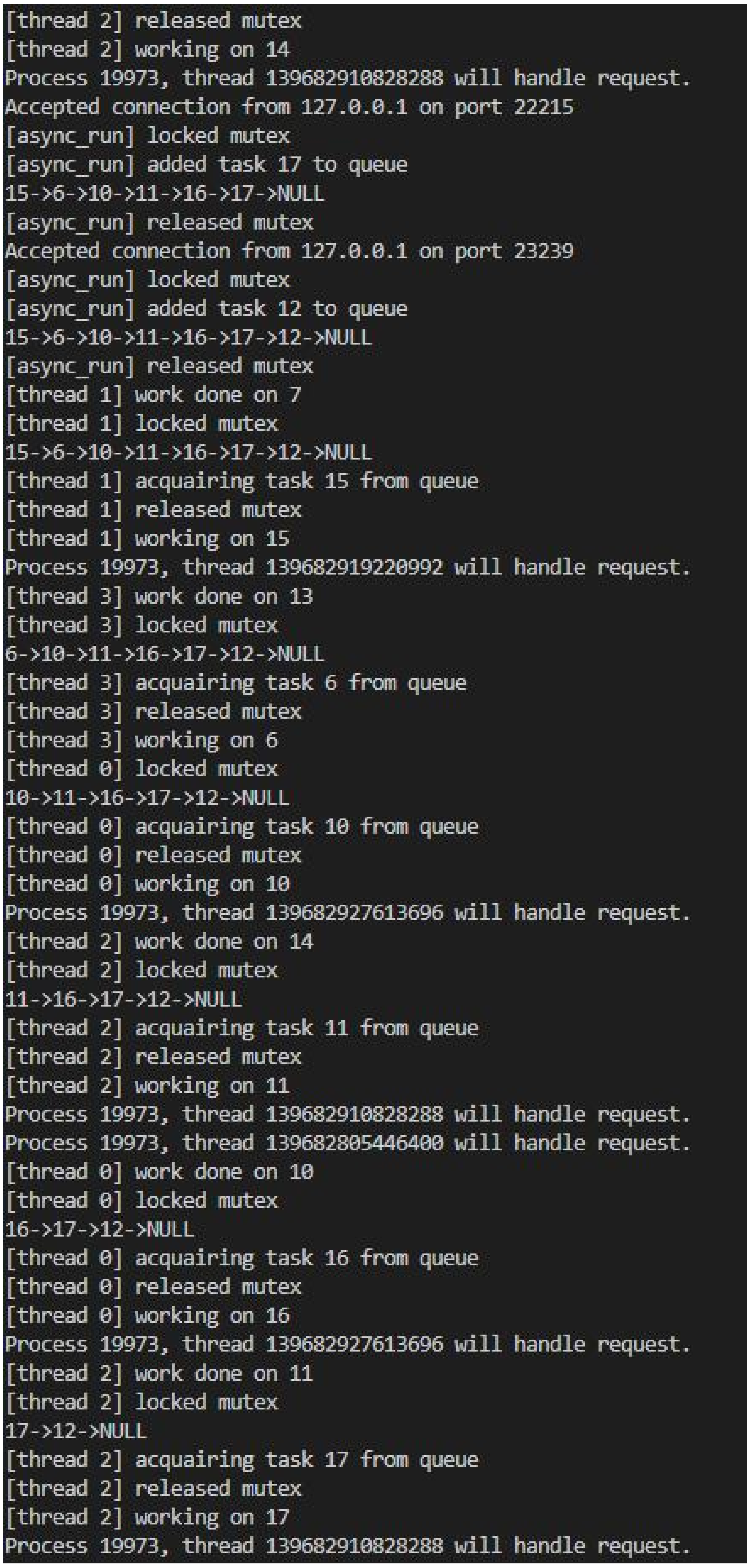
Bonus

`async_init`, no incoming jobs:

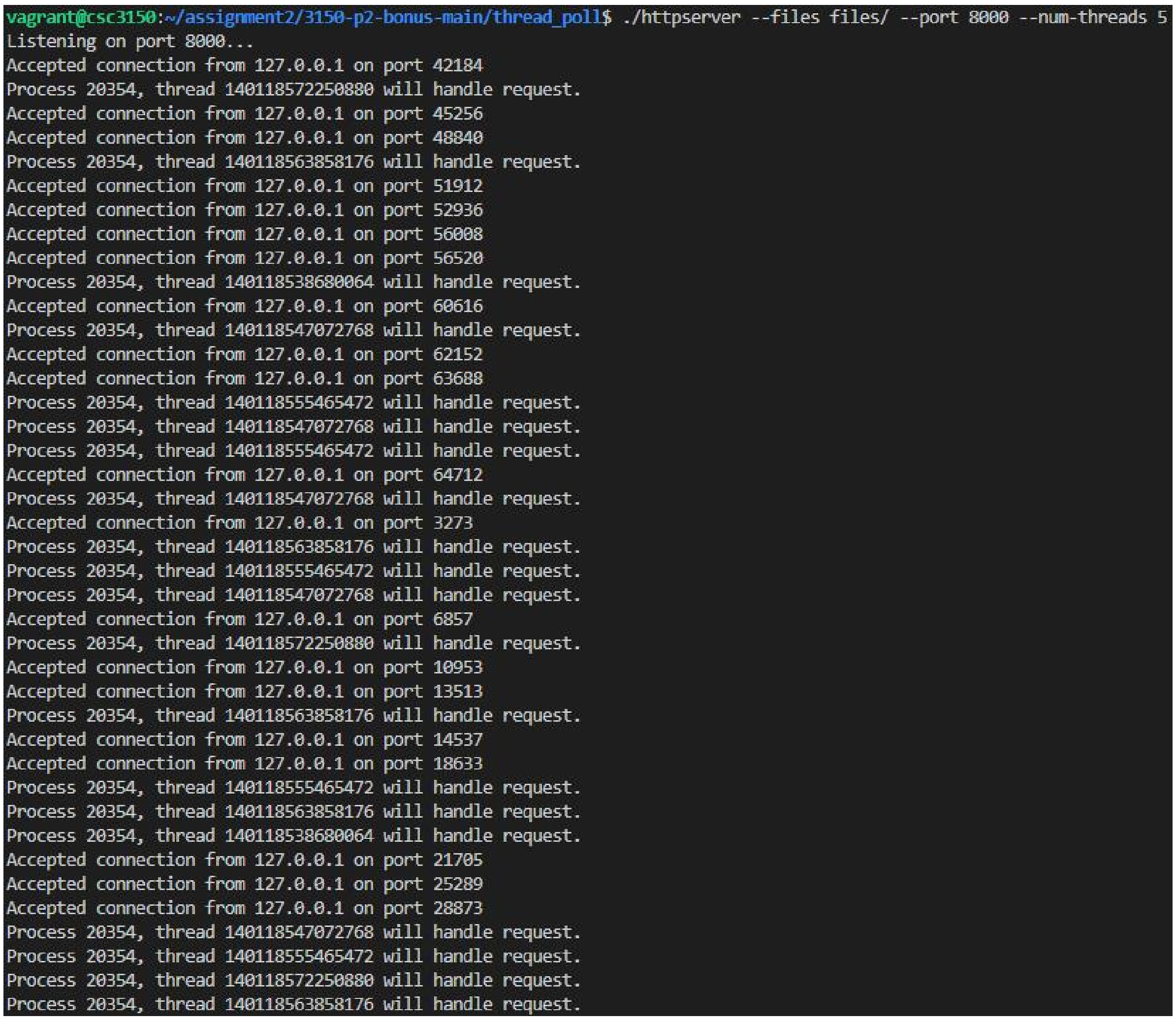


```
nbe@LAPTOP-15PQSA5V:/mnt/d/OneDrive - CUHK-Shenzhen/myfolder
erver --files files/ --port 8000 --num-threads 5
[thread 0] locked mutex
NULL
[thread 0] start waiting, release mutex
[thread 1] locked mutex
NULL
[thread 1] start waiting, release mutex
[thread 2] locked mutex
NULL
[thread 2] start waiting, release mutex
[thread 3] locked mutex
NULL
[thread 3] start waiting, release mutex
[thread 4] locked mutex
NULL
[thread 4] start waiting, release mutex
Listening on port 8000...
```

In job execution, with detailed output of which thread is doing what (this is an option that can be disabled by defining macro):

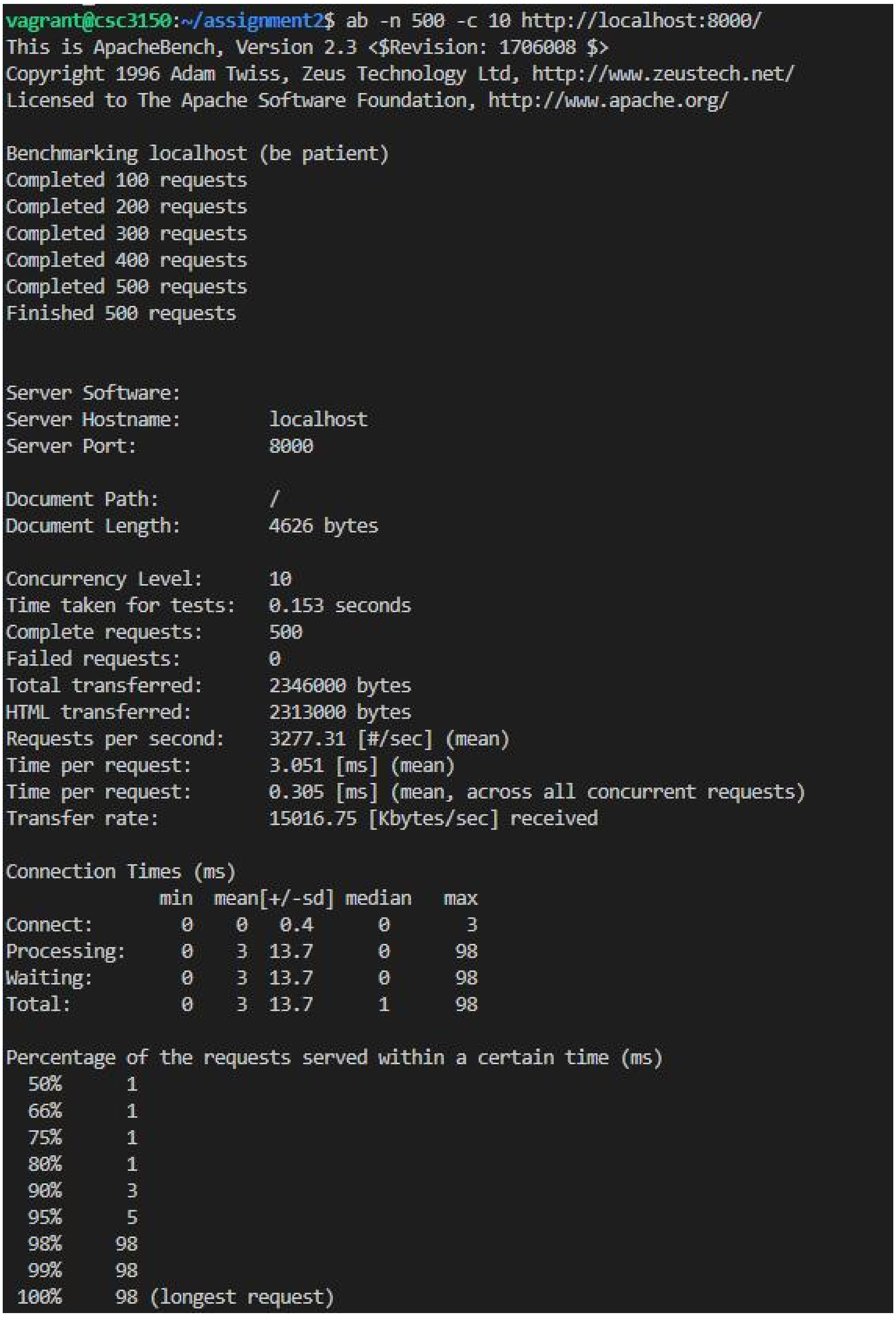

```
[thread 2] released mutex
[thread 2] working on 14
Process 19973, thread 139682910828288 will handle request.
Accepted connection from 127.0.0.1 on port 22215
[async_run] locked mutex
[async_run] added task 17 to queue
15->6->10->11->16->17->NULL
[async_run] released mutex
Accepted connection from 127.0.0.1 on port 23239
[async_run] locked mutex
[async_run] added task 12 to queue
15->6->10->11->16->17->12->NULL
[async_run] released mutex
[thread 1] work done on 7
[thread 1] locked mutex
15->6->10->11->16->17->12->NULL
[thread 1] acquairing task 15 from queue
[thread 1] released mutex
[thread 1] working on 15
Process 19973, thread 139682919220992 will handle request.
[thread 3] work done on 13
[thread 3] locked mutex
6->10->11->16->17->12->NULL
[thread 3] acquairing task 6 from queue
[thread 3] released mutex
[thread 3] working on 6
[thread 0] locked mutex
10->11->16->17->12->NULL
[thread 0] acquairing task 10 from queue
[thread 0] released mutex
[thread 0] working on 10
Process 19973, thread 139682927613696 will handle request.
[thread 2] work done on 14
[thread 2] locked mutex
11->16->17->12->NULL
[thread 2] acquairing task 11 from queue
[thread 2] released mutex
[thread 2] working on 11
Process 19973, thread 139682910828288 will handle request.
Process 19973, thread 139682805446400 will handle request.
[thread 0] work done on 10
[thread 0] locked mutex
16->17->12->NULL
[thread 0] acquairing task 16 from queue
[thread 0] released mutex
[thread 0] working on 16
Process 19973, thread 139682927613696 will handle request.
[thread 2] work done on 11
[thread 2] locked mutex
17->12->NULL
[thread 2] acquairing task 17 from queue
[thread 2] released mutex
[thread 2] working on 17
Process 19973, thread 139682910828288 will handle request.
```

In execution of 20 paralleled jobs, without detailed output:



```
vagrant@csc3150:~/assignment2/3150-p2-bonus-main/thread_poll$ ./httpserver --files files/ --port 8000 --num-threads 5
Listening on port 8000...
Accepted connection from 127.0.0.1 on port 42184
Process 20354, thread 140118572250880 will handle request.
Accepted connection from 127.0.0.1 on port 45256
Accepted connection from 127.0.0.1 on port 48840
Process 20354, thread 140118563858176 will handle request.
Accepted connection from 127.0.0.1 on port 51912
Accepted connection from 127.0.0.1 on port 52936
Accepted connection from 127.0.0.1 on port 56008
Accepted connection from 127.0.0.1 on port 56520
Process 20354, thread 140118538680064 will handle request.
Accepted connection from 127.0.0.1 on port 60616
Process 20354, thread 140118547072768 will handle request.
Accepted connection from 127.0.0.1 on port 62152
Accepted connection from 127.0.0.1 on port 63688
Process 20354, thread 140118555465472 will handle request.
Process 20354, thread 140118547072768 will handle request.
Process 20354, thread 140118555465472 will handle request.
Accepted connection from 127.0.0.1 on port 64712
Process 20354, thread 140118547072768 will handle request.
Accepted connection from 127.0.0.1 on port 3273
Process 20354, thread 140118563858176 will handle request.
Process 20354, thread 140118555465472 will handle request.
Process 20354, thread 140118547072768 will handle request.
Accepted connection from 127.0.0.1 on port 6857
Process 20354, thread 140118572250880 will handle request.
Accepted connection from 127.0.0.1 on port 10953
Accepted connection from 127.0.0.1 on port 13513
Process 20354, thread 140118563858176 will handle request.
Accepted connection from 127.0.0.1 on port 14537
Accepted connection from 127.0.0.1 on port 18633
Process 20354, thread 140118555465472 will handle request.
Process 20354, thread 140118563858176 will handle request.
Process 20354, thread 140118538680064 will handle request.
Accepted connection from 127.0.0.1 on port 21705
Accepted connection from 127.0.0.1 on port 25289
Accepted connection from 127.0.0.1 on port 28873
Process 20354, thread 140118547072768 will handle request.
Process 20354, thread 140118555465472 will handle request.
Process 20354, thread 140118572250880 will handle request.
Process 20354, thread 140118563858176 will handle request.
```

Systematic testing:



```
vagrant@csc3150:~/assignment2$ ab -n 500 -c 10 http://localhost:8000/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:
Server Hostname:        localhost
Server Port:            8000

Document Path:          /
Document Length:        4626 bytes

Concurrency Level:      10
Time taken for tests:   0.153 seconds
Complete requests:      500
Failed requests:        0
Total transferred:      2346000 bytes
HTML transferred:       2313000 bytes
Requests per second:    3277.31 [#/sec] (mean)
Time per request:       3.051 [ms] (mean)
Time per request:       0.305 [ms] (mean, across all concurrent requests)
Transfer rate:          15016.75 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4      0       3
Processing:     0    3  13.7      0      98
Waiting:        0    3  13.7      0      98
Total:          0    3  13.7      1      98

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      1
  75%      1
  80%      1
  90%      3
  95%      5
  98%     98
  99%     98
 100%     98 (longest request)
```

What Did I Learn

1. mutex lock
2. working with blocking and non-blocking functions
3. debugging in multithread applications
4. understanding of deadlocks and starvation
5. how to refresh screens