CSC3150 A2 Report

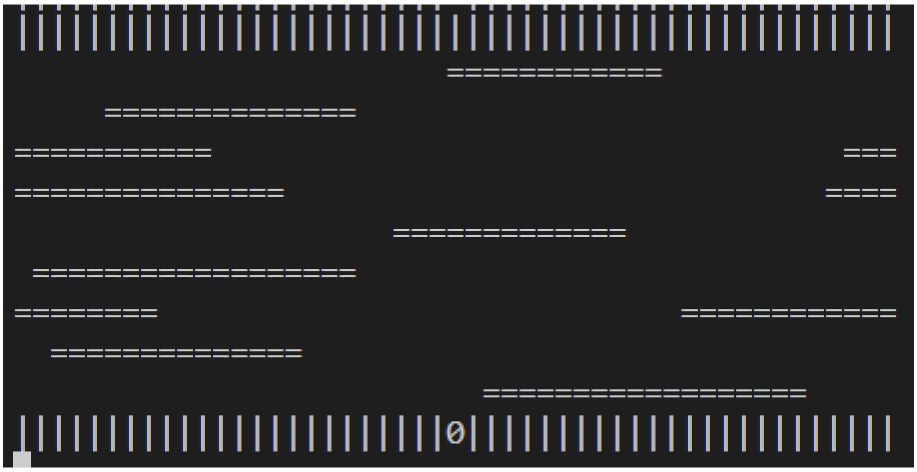
Jiakun Fan

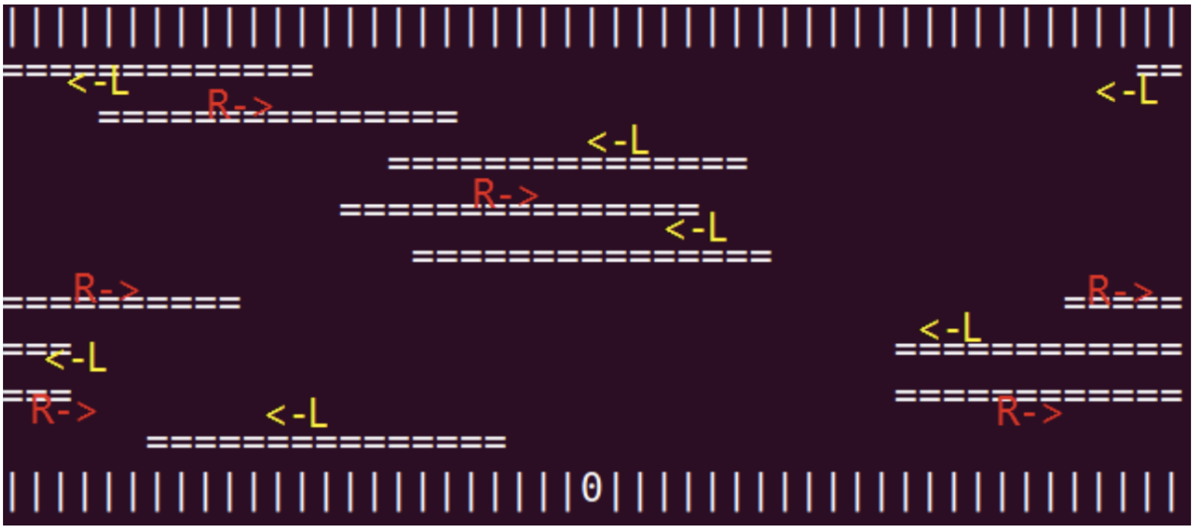
120090316

Part1: Problem Brief

Frog Crosses River

Complete a multithread program to implement the game "frog crosses river".





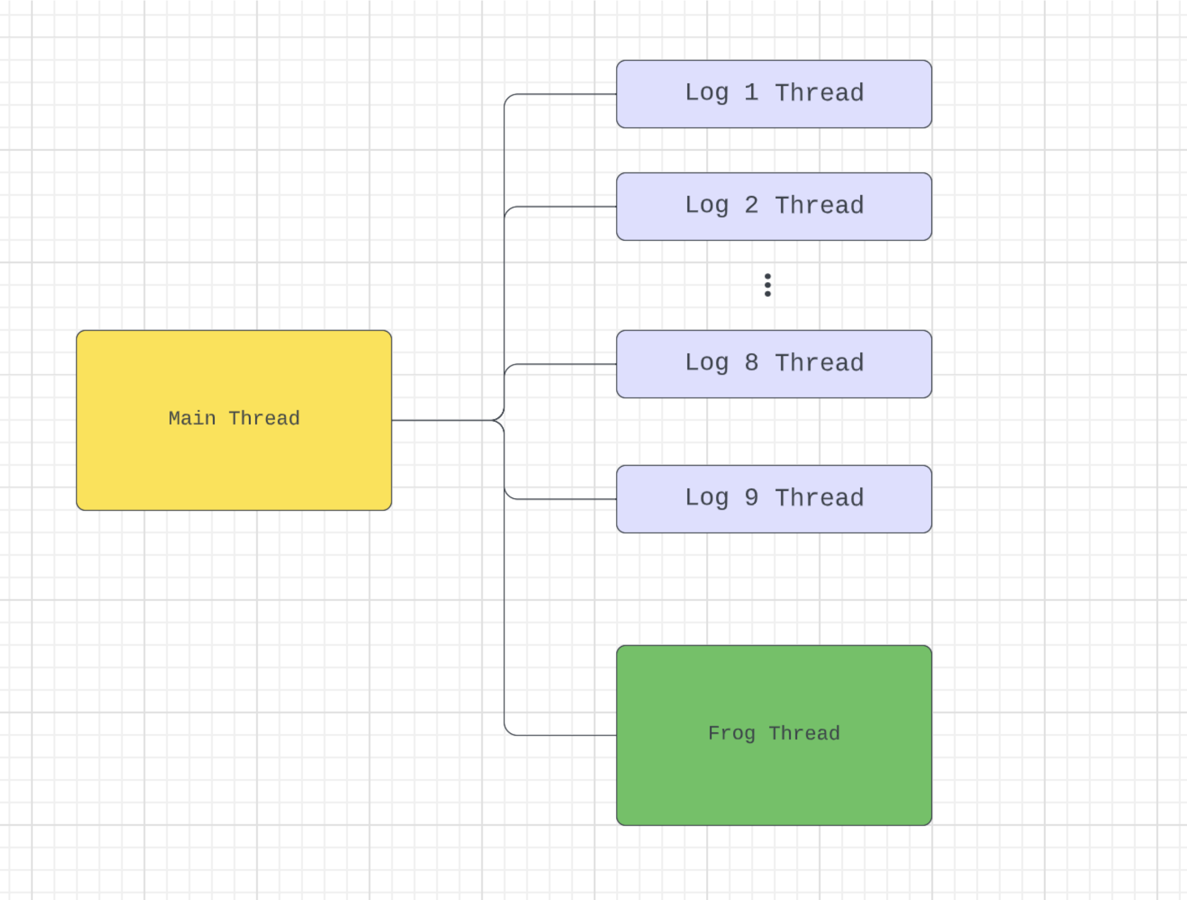
Bonus

- Implement in **async.c** and **async.h**: `void async_init(int num_threads)` and `void async_run(void (*handler)(int), int args)`
- You can use list data structure in **utlist.h**, for example: `DL_APPEND(my_queue->head, my_item);` (adding to queue end) and `DL_DELETE(my_queue->head, my_queue->head);` (popping from queue head)
- When no jobs are coming, your threads created in `async_init` have to go to sleep and is not allowed to do busy waiting like `while(1){sleep(any);}`, and when jobs are coming a sleeping thread in your thread pool must wake up immediately (that is, no `sleep()` call is allowed).

- `async_run` should be asynchronous without further call to `pthread_create`, that is it should return immediately before the job is handled (in the code we give you, async_run runs synchronously, so you need to rewrite the function)

Part2: Program Design

Frog Crosses River



Main thread

- Main thread create other threads and init maps.
- Main thread prints the map and checks if the frog is out of bound or drops into the river.

```
// Initialize the river map and frog's starting position

memset(map, 0, sizeof(map));
int i, j;
for (i = 1; i < ROW; ++i)
{
    for (j = 0; j < COLUMN - 1; ++j)
        map[i][j] = ' ';
}

for (j = 0; j < COLUMN - 1; ++j)
{
    map[ROW][j] = '|';
    map[0][j] = '|';
}

frog = Node(ROW, (COLUMN - 1) / 2);
old_frog = Node(ROW, (COLUMN - 1) / 2);
```

```
    map[frog.x][frog.y] = '0';

    for (i = 0; i < LOG_NUM; ++i)
    {
        logs_pos[i] = Node(i + 1, logs_init_pos[i]);
        for (j = 0; j < logs_len[i]; j++)
            map[logs_pos[i].x][logs_pos[i].y + j] = '=';
    }

    for (i = 0; i <= ROW; ++i)
        puts(map[i]);
    int indexs[] = {0, 1, 2, 3, 4, 5, 6, 7, 8};

    /*  Create pthreads for wood move and frog control.  */
    pthread_t frog_ctl, logs[LOG_NUM];
    for (i = 0; i < LOG_NUM; i++)
        pthread_create(&logs[i], NULL, logs_move, &indexs[i]);

    pthread_create(&frog_ctl, NULL, frog_move, NULL);
    /*  Display the output for user: win, lose or quit.  */
    // Print the map into screen
    while (true)
    {
        usleep(UPDATE_INTERVAL);
        pthread_mutex_lock(&lock);
        if (is_quit)
        {
            std::cout << "You exit the game." << std::endl;
            break;
        }

        for (j = 0; j < COLUMN - 1; ++j)
        {
            map[0][j] = '|';
            map[ROW][j] = '|';
        }

        if (frog.y < 0 || frog.y >= COLUMN - 1 || frog.x > ROW)
        {
            std::cout << "You lost the game!!" << std::endl;
            break;
        }

        map[frog.x][frog.y] = '0';

        for (i = 0; i <= ROW; ++i)
            puts(map[i]);
        if (frog.x <= 0)
        {
            std::cout << "You win the game!!" << std::endl;
            break;
        }

        if (frog.x != ROW)
        {
            int a = frog.x - 1;
            int pos = frog.y;
            int start = logs_pos[a].y;
            int end = (logs_pos[a].y + logs_len[a] - 1) % (COLUMN - 1);
            if ((logs_pos[a].y + logs_len[a] > 49 && pos < start && pos > end)
|| (logs_pos[a].y + logs_len[a] <= 49 && (pos > end || pos < start)))
            {
                std::cout << "You lose the game!!" << std::endl;
                break;
            }
        };
        pthread_mutex_unlock(&lock);
    };
    pthread_mutex_unlock(&lock);
    is_quit = 1;
    usleep(UPDATE_INTERVAL);
    usleep(UPDATE_INTERVAL);
    return 0;
```

Log Thread

- Control the move of logs

```
void *logs_move(void *index)
{
    int a = *((int *)index);
    while (true)
    {
        usleep(UPDATE_INTERVAL);
        int is_move = 0;

        pthread_mutex_lock(&lock);
        if (logs_pos[a].x % 2 == 0)
        {

            map[logs_pos[a].x][logs_pos[a].y] = ' ';
            if (logs_pos[a].y == frog.y && logs_pos[a].x == frog.x)
                is_move = 1;
            logs_pos[a].y = (logs_pos[a].y + 1) % (COLUMN - 1);
            for (int i = 0; i < logs_len[a]; i++)
            {
                if ((logs_pos[a].y + i) % (COLUMN - 1) == frog.y &&
logs_pos[a].x == frog.x)
                    is_move = 1;
                map[logs_pos[a].x][(logs_pos[a].y + i) % (COLUMN - 1)] = '=';
            }
            if (is_move)
            {
                frog.y++;
            }
        }
        else if ((logs_pos[a].x % 2) != 0)
        {
            map[logs_pos[a].x][(logs_pos[a].y + logs_len[a]) % (COLUMN - 1)] = '
';
            if (logs_pos[a].y == frog.y && logs_pos[a].x == frog.x)
                is_move = 1;
            logs_pos[a].y = (logs_pos[a].y + 48) % (COLUMN - 1);
            for (int i = 0; i < logs_len[a]; i++)
            {
                if ((logs_pos[a].y + i) % (COLUMN - 1) == frog.y &&
logs_pos[a].x == frog.x)
                    is_move = 1;
                map[logs_pos[a].x][(logs_pos[a].y + i) % (COLUMN - 1)] = '=';
            }
            if (is_move)
            {
                frog.y--;
            }
        }
        pthread_mutex_unlock(&lock);
        if (is_quit)
            return NULL;
    }
}
```

Frog Thread

- Listen to the keyboard hit and control the move of frog

```
void *frog_move(void *)
{
    while (1)
    {
        if (kbhit())
        {
            pthread_mutex_lock(&lock);
            char ch = (char)getchar();
            switch (ch)
            {
            case 'w':
            case 'W':
                frog.x--;
                break;
            case 'a':
            case 'A':
                frog.y--;
                break;
            case 's':
            case 'S':
                frog.x++;
                break;
            case 'd':
            case 'D':
                frog.y++;
                break;
            case 'q':
            case 'Q':
                is_quit = 1;
                break;
            default:
                break;
            }
        }
        pthread_mutex_unlock(&lock);
        if (is_quit)
            return NULL;
    }
}
```

Mutex

- We initialize a mutex to ensure that only one thread to access the map at a time.
- If some thread want to access the map, it must grab the lock.

Bonus

Use producer-consumer pattern to solve the problem.

```
// worker:
    pthread_mutex_lock(&mutex);
    while (!condition)
        pthread_cond_wait(&cond, &mutex);
    /* do something that requires holding the mutex and condition is true */
    // grab a task from queue
    pthread_mutex_unlock(&mutex);
    // handle the task

// async_run():
    pthread_mutex_lock(&mutex);
    /* do something that might make condition true */
    // enqueue a task
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);
```

- Init worker threads, lock and condition variable.

```
void async_init(int num_threads)
{
    pthread_t threads[num_threads];
    task_queue = (my_queue_t *)malloc(sizeof(my_queue_t));
    task_queue->head = NULL;
    task_queue->size = 0;
    for (int i = 0; i < num_threads; i++)
    {
        pthread_create(&threads[i], NULL, wait_to_wakeup, NULL);
    }
    return;
}
```

- Worker threads wait for handling tasks

```
void *wait_to_wakeup(void *args)
{

    int arg;
    my_item_t *item_ptr;
    void (*handler)(int);
    for (;;)
    {
        pthread_mutex_lock(&lock);
        while (task_queue->size == 0)
            pthread_cond_wait(&cond, &lock);
        task_queue->size--;
        handler = task_queue->head->handler_ptr;
        arg = task_queue->head->args;

        item_ptr = task_queue->head;

        if ((task_queue->head)->prev == (task_queue->head))
        {
            (task_queue->head) = NULL;
        }
        else
        {
            task_queue->head = task_queue->head->prev;
            task_queue->head->next = NULL;
        };

        pthread_mutex_unlock(&lock);

        (*handler)(arg);
        free(item_ptr);
    }
}
```

- Task publisher enqueue tasks

```
void async_run(void (*hanlder)(int), int args)
{
    my_item_t *item_ptr = (my_item_t *)malloc(sizeof(my_item_t));
    item_ptr->args = args;
    item_ptr->handler_ptr = hanlder;

    pthread_mutex_lock(&lock);
    if (task_queue->head != NULL)
    {
        item_ptr->prev = task_queue->head;
        task_queue->head->next = item_ptr;
        task_queue->head = task_queue->head->next;
        (task_queue->head)->next = NULL;
    }
    else
    {
        (task_queue->head) = (item_ptr);
        (task_queue->head)->prev = (task_queue->head);
        (task_queue->head)->next = NULL;
    }

    task_queue->size++;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&lock);
    return;
}
```

Part3: Environment

- Virtual machine application: virtual box 6.1.32
- The program is run on a Ubuntu 16.04 LTS operation system, with kernel version 4.4.0-210-generic.
- Compiler: gcc version 5.4.0

Part4: How to Run

Frogs Crosses River

Under `/source` directory,

```
make
./hw2
```

Clean：

```
make clean
```

Bonus

Under `/3150-p2-bonus-main/thread_poll` directory,

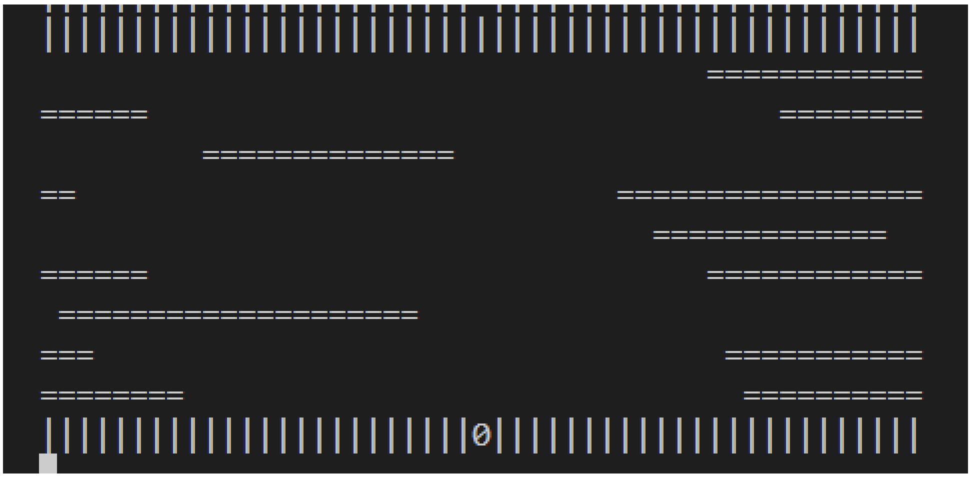
```
make
// test
./httpserver --files files/ --port 8000 --num-threads 10

```

In another terminal,

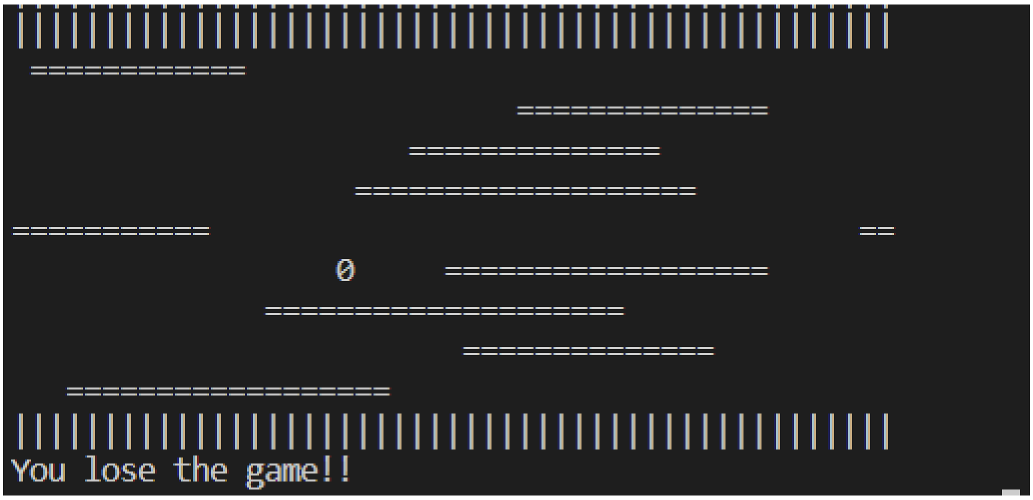
```
ab -n 5000 -c 10  http://localhost:8000/
```

Part5: Screenshot of Output

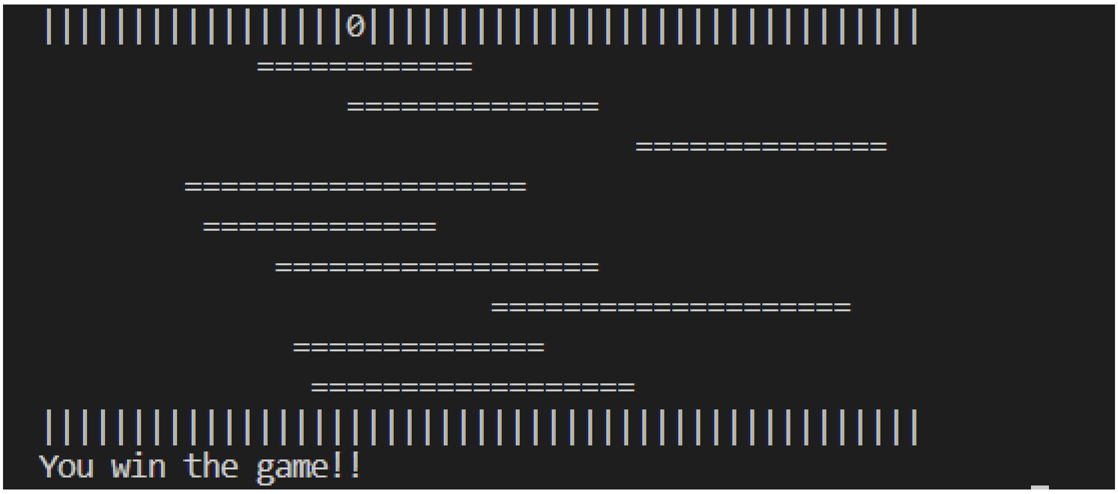
Frog Crosses River



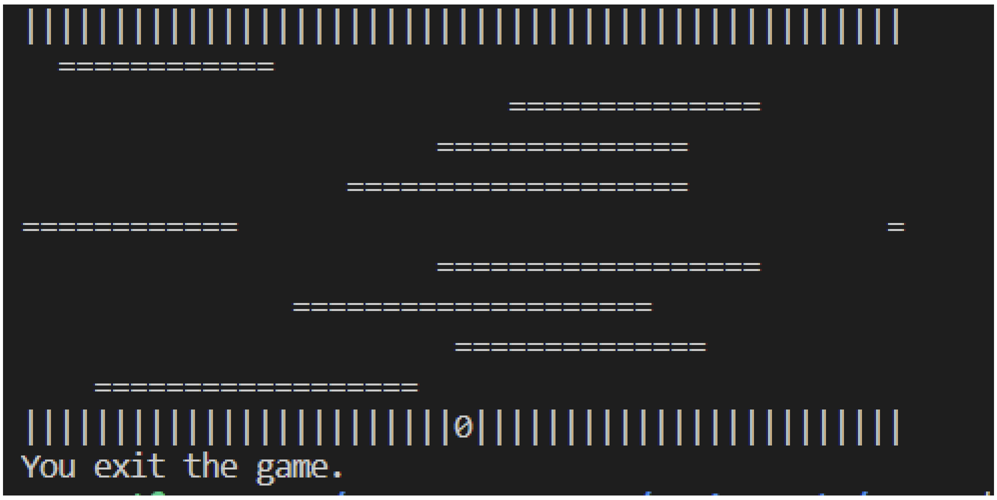
```
|||||||||||||||||||||||||||||||||||||||||||||||||
                                     ============

======                                   ========

         ==============

==                              =================

                                  =============

======                               ============

 ====================

===                                   ===========

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



```
|||||||||||||||||||||||||||||||||||||||||||||||||
 ============
                            ==============
                      ==============
                   ===================
===========                                    ==
                  0     ==================
              ====================
                         ==============
   ==================
|||||||||||||||||||||||||||||||||||||||||||||||||
You lose the game!!
```

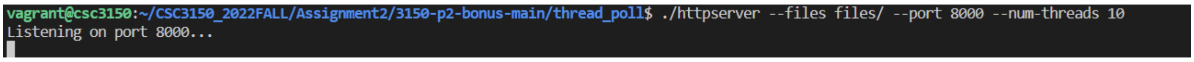


```
|||||||||||||||||0|||||||||||||||||||||||||||||||
            ============
                 ==============
                                 ==============
        ===================
         =============
             ==================
                         ====================
              ==============
               ==================
|||||||||||||||||||||||||||||||||||||||||||||||||
You win the game!!
```

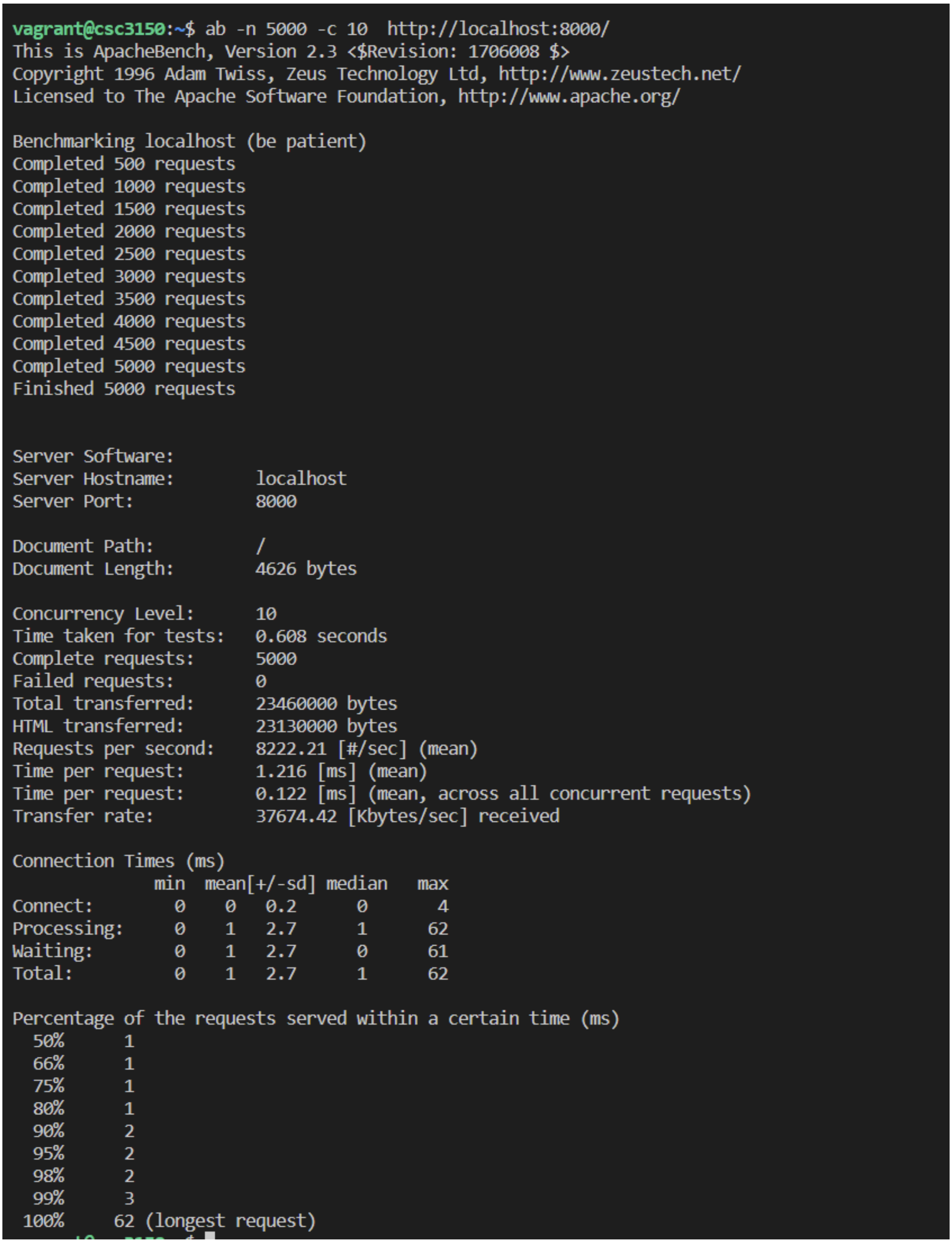


```
|||||||||||||||||||||||||||||||||||||||||||||||||
  ============
                           ==============
                       ==============
                  ===================
============                                    =
                       ==================
               ====================
                        ==============
    ==================
||||||||||||||||||||||||0||||||||||||||||||||||||
You exit the game.
```

Bonus



```
vagrant@csc3150:~/CSC3150_2022FALL/Assignment2/3150-p2-bonus-main/thread_poll$ ./httpserver --files files/ --port 8000 --num-threads 10
Listening on port 8000...
```



```
vagrant@csc3150:~$ ab -n 5000 -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 500 requests
Completed 1000 requests
Completed 1500 requests
Completed 2000 requests
Completed 2500 requests
Completed 3000 requests
Completed 3500 requests
Completed 4000 requests
Completed 4500 requests
Completed 5000 requests
Finished 5000 requests


Server Software:
Server Hostname:        localhost
Server Port:            8000

Document Path:          /
Document Length:        4626 bytes

Concurrency Level:      10
Time taken for tests:   0.608 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      23460000 bytes
HTML transferred:       23130000 bytes
Requests per second:    8222.21 [#/sec] (mean)
Time per request:       1.216 [ms] (mean)
Time per request:       0.122 [ms] (mean, across all concurrent requests)
Transfer rate:          37674.42 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       4
Processing:     0    1   2.7      1      62
Waiting:        0    1   2.7      0      61
Total:          0    1   2.7      1      62

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      1
  75%      1
  80%      1
  90%      2
  95%      2
  98%      2
  99%      3
 100%     62 (longest request)
```

Part6: What I learn

Multithreading program, mutex, condition variable.