CSC3150 Assignment 2

Chen Zhixin, 120090222

How I design my program?

Main

There are totally 11 threads in the main program. They loops forever until the frog reached the opposite river bank.

- 9 x `logs_move` Thread
- 1 x `Print_Map` Thread
- 1 x `Controller` Thread

At first, the `controller` thread will go to sleep. Now the loops remains 10 valid threads (9 x `logs_move`, 1 x `Print_Map`).

For each `logs_move` thread, it will analyze one bit of keyboard input:

- If `Q` detected, wake the `controller` up and quit the game.
- If `W` detected: Moves the frog **upward** (If the frog step into the water, quit the game)
- If `A` detected: Moves the frog **leftward** (If the frog step into the water, quit the game)
- If `S` detected: Moves the frog **downward** (If the frog step into the water, quit the game)
- If `D` detected: Moves the frog **rightward** (If the frog step into the water, quit the game)
- If the frog reached the opposite river bank, wake `Controller` Thread up.

Then the corresponding # log moves leftward/rightward for one unit length.

Once if the `Controller` Thread woke up, It will ask all other threads to quit (suicide).

Bonus

Long story short, I implemented thread pool by **producer-consumer pattern**.

- There is a waiting list, all the pending tasks are stored in it.
- Each thread is the **consumer**. They execute the remaining tasks in the waiting list one after another. If **consumer** has no task to execute, it will go to sleep.

- `async_run` is **producer**. It will pass the task into the waiting list. After **producer** pass in a new task, it will randomly wake one **consumer** up.

Environment

OS Version: Ubuntu 20.04

Kernel Version: 5.10.x

Check g++ version > 4.9

```
g++ --version
```

Check kernel version ~5.10.x

```
uname -r
```

(Optional) Install `libncurses5-dev`

```
sudo apt-get install libncurses5-dev
```

The steps to execute my program

Main

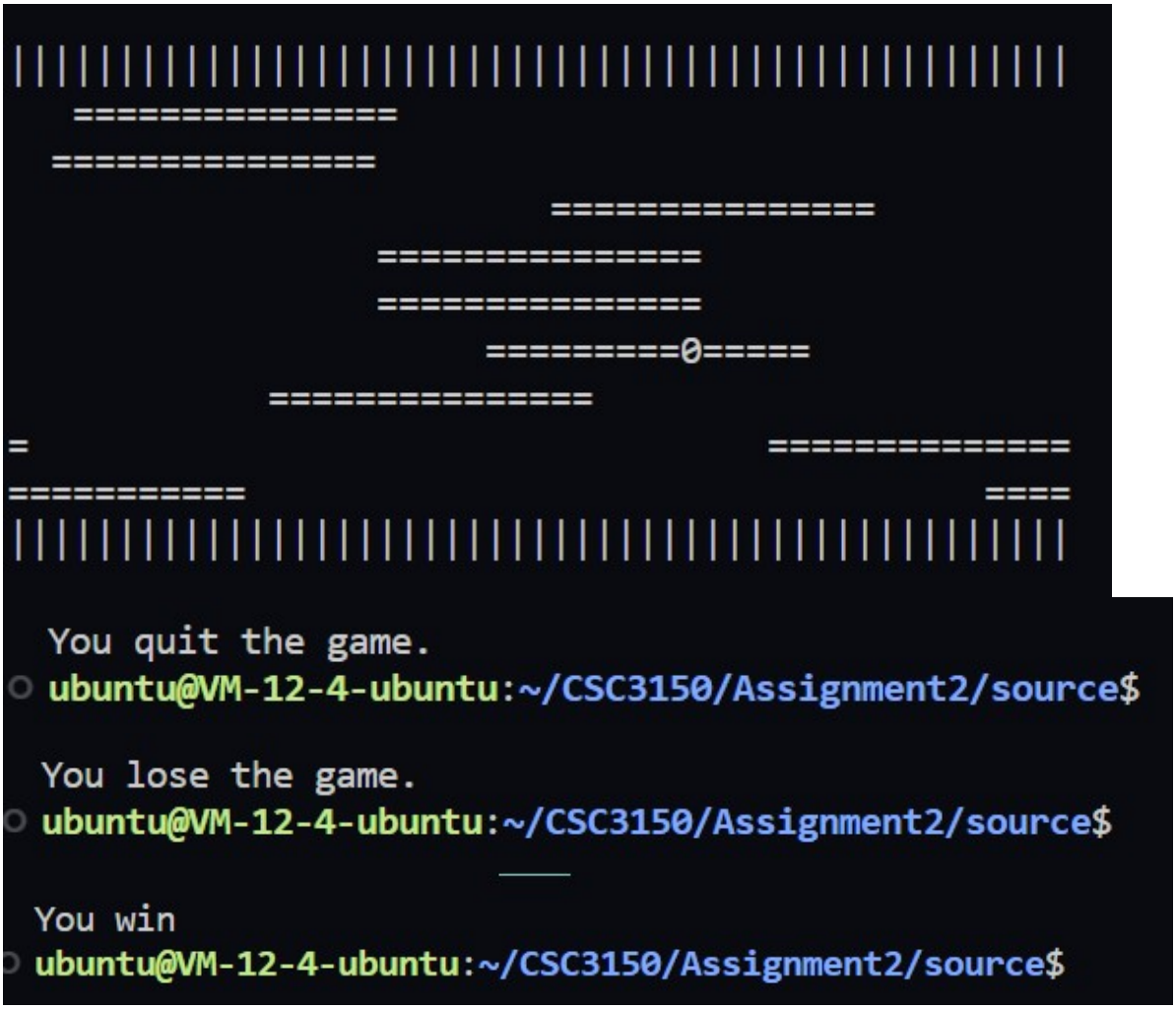
```
cd ~/source
make
./a.out
```

Bonus

```
cd ~/3150-p2-bonus-main/thread_poll/
make
./httpserver --files files/ --port 8000 --num-threads 10
ab -n 5000 -c 10 http://localhost:8000/
```

Screenshot of the program Output

Main



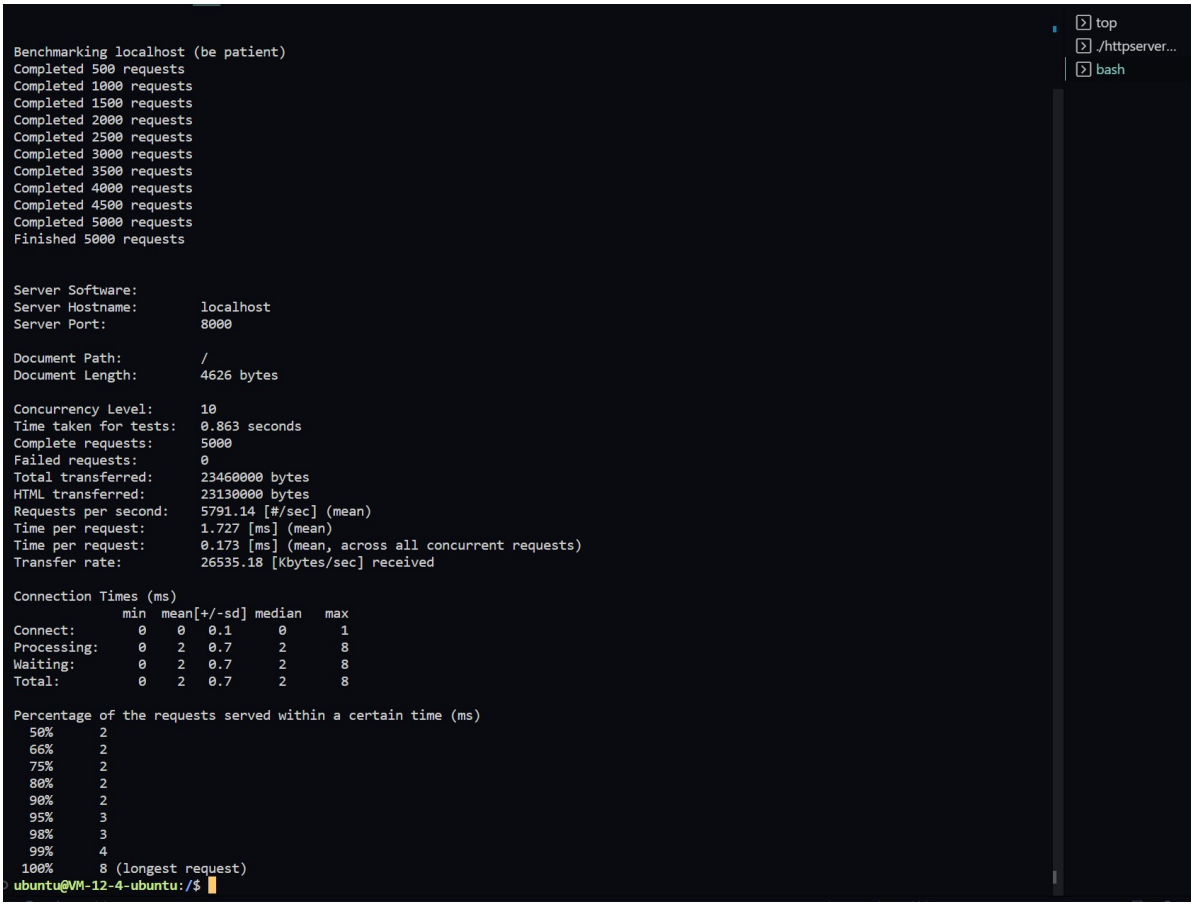
If you wanna see the ***Video for DEMO***, click **HERE**

> Video may get stuck due to video server bandwidth limitations.

Bonus

With AB benchmark test

10 Threads, 5000 requests:



What did I learned from task

Concurrency is a very important topic in OS. In this task, I implemented a lot of features relative to mutual exclusive lock, conditional variable and so on. Hence I deeply understand how important it is to handle the concurrency conflict in parallel calculation among multiple cores. Also, during a lot of debugging work, I learned new techniques of figuring out where exactly the bug is by some automatic tools like `gdb` or `objdump`. Also, I can analyze the program flow with state map, which helps me to understand the process, and find out the logical fault of the program.