

CSC3150 Assignment 2

Chen Zhixin, 120090222

CSC3150 Assignment 2

How I design my program?

Main

Bonus

Environment

The steps to execute my program

Main

Bonus

Screenshot of the program Output

Main

Bonus

What did I learned from task

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

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

You quit the game.
ubuntu@VM-12-4-ubuntu:~/CSC3150/Assignment2/source$

You lose the game.
ubuntu@VM-12-4-ubuntu:~/CSC3150/Assignment2/source$

You win
ubuntu@VM-12-4-ubuntu:~/CSC3150/Assignment2/source$
```

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:

```
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: 0.863 seconds
Complete requests:  5000
Failed requests:     0
Total transferred:  23460000 bytes
HTML transferred:   23130000 bytes
Requests per second: 5791.14 [#/sec] (mean)
Time per request:    1.727 [ms] (mean)
Time per request:    0.173 [ms] (mean, across all concurrent requests)
Transfer rate:        26535.18 [Kbytes/sec] received

Connection Times (ms)
      min  mean[+/-sd] median  max
Connect:    0     0  0.1      0     1
Processing:  0     2  0.7      2     8
Waiting:    0     2  0.7      2     8
Total:       0     2  0.7      2     8

Percentage of the requests served within a certain time (ms)
 50%    2
 66%    2
 75%    2
 80%    2
 90%    2
 95%    3
 98%    3
 99%    4
100%    8 (longest request)
ubuntu@VM-12-4-ubuntu:/$
```

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.