CSC3150 Project2 Report

Liang MingRui 120090723

2022.10.20

Bonus has been finished.

My way to design the program:

Frog crosses river:

I divide the whole game into three parts. And every part is grasped by one thread.

Firstly, I write the main function as the master thread to master all the things. At the beginning of the master thread, I just randomly put those logs on my map. And then I print all the things on the screen. I then create and initialize a mutex as a lock. Besides, I create two threads to execute my two functions, logs_move and frog_move. After that, I ask my master to wait until other two threads ends with help of the function pthread_join(). Lastly, I use the signal sent by the two threads to show the result of my game. And then I end the master thread using pthread_exit().

Second, I write the function logs_move(), which is used in another thread. In this thread, the function will decide which log moves to which direction. I use a while loop to execute the movement of logs when the signal is correct. Also, I control the speed of logs using usleep() to make the game suitable for playing. By the way, the way I implement the movement is to clean the screen and print all the things to the screen again.

To suit the rule of the game, I ask the thread to change the signal to lose when the frog collide with the boundary.

Third, I write the function frog_move(), which is also use in another thread. In this thread, the function will help the player to control the frog to move. The function kbhit() can allow the program to know whether a keyboard be pressed. If it is pressed, the function will return 1. With help of the function, I also use a while loop to check if there exists an I/O and use usleep to avoid the program continuously check the I/O. If the input is one of a(A), s(S), d(D), w(W) and q(Q), the program will execute according to different inputs (w/W control the frogs move forward, s/S control the frog moves backward, a/A control the frog moves left, d/D control the frog moves right and q/Q control the program to exit). Others input will be ignored. To suit the rule of the game, when the frog collides with the boundary or leave the log, the thread will change the signal to tell the game lose. Only when the frog arrives the other side of the bank, the signal can be win.

Then the game is done.

Tips: I design the signal as two-bit binary number, 0b00 means the game is going on, 0b01 means you lose the game, 0b10 means you exit the game and 0b11 means you win the game.

Bonus:

Firstly, I complete the struct of my_item to allow to receive the values of hanlder and args. Then, I creates threads pool in the async_init function, in which I use these threads to execute async_process to deal with the request. Lastly, I finish async_run to receive request and add it into queue. Moreover, in the async_process, I use pthread_cond_wait to allow the threads to sleep until some requests come, which avoid busy waiting. After, a request come, the item in the queue will be pop out and be dealt. Considering the case that no signal comes but there are still some requests in queue, I also create a thread in async_init to handle the case. When a thread finish the request, it will unlock and trylock again. If no thread are locking now, this thread will signal the wait in function wake to reboardcast all threads to finish the rest items in the queue.

Development environment:

The environment I use is the Linux kernel 5.10.10. And the gcc version is 5.40.

The step to execute the program:

Frog crosses river:

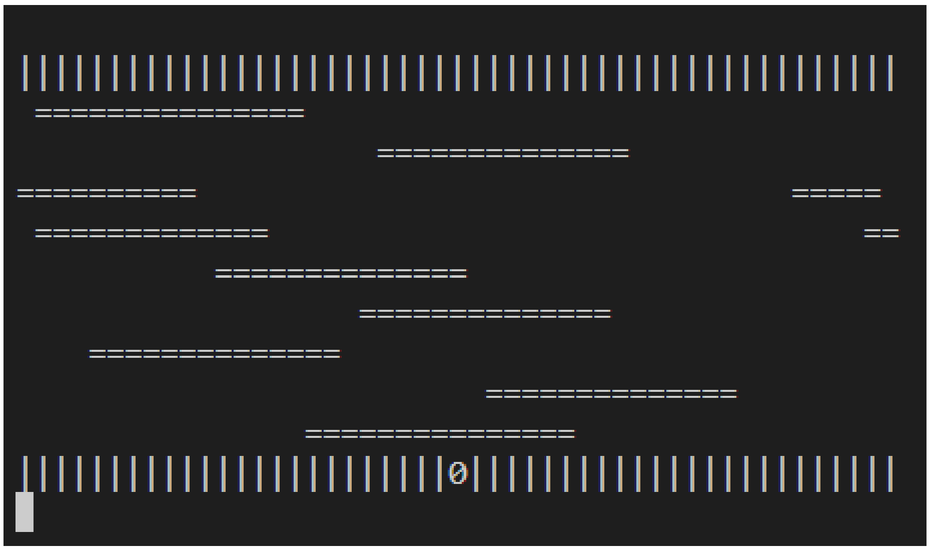
I write a makefile. In the makefile, I use command g++ -std=c++11 hw2.cpp -o a -lpthread to generate file a. and write rm -f a to remove file a. Then, I type make and use "./a" to execute the game.

Bonus:

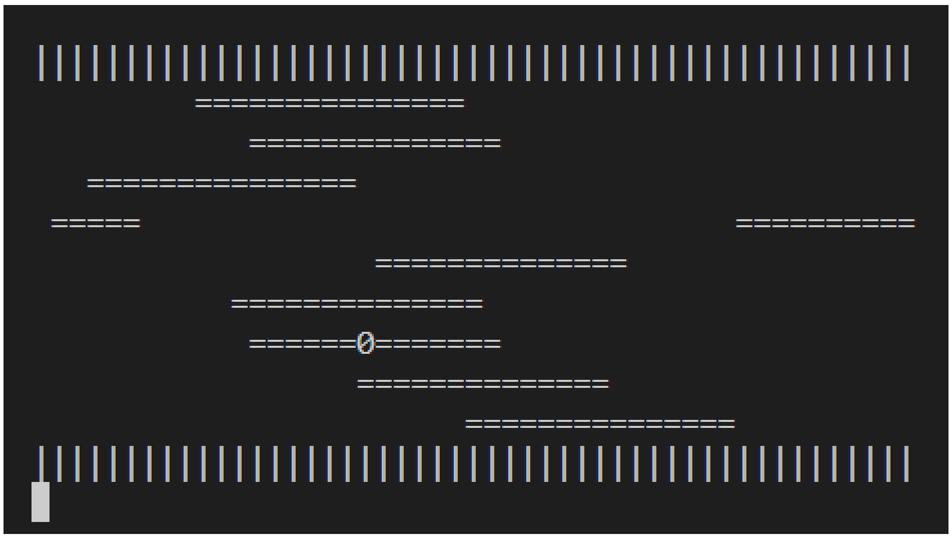
I directly use the makefile given. After using make to compile those files. I use ./httpserver –files files/ --port 8000 –num-threads 10 and ab -n 5000 -c 10 http://localhost:8000/ to test.

Screenshot:

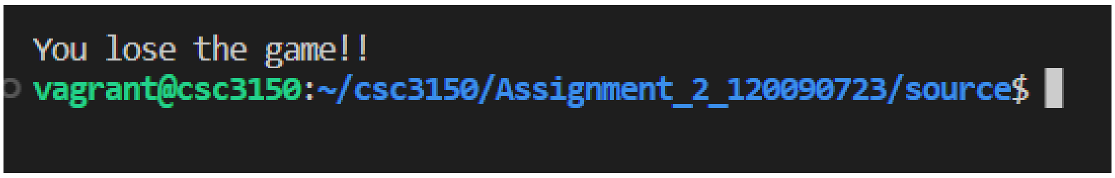
Frog crosses river:



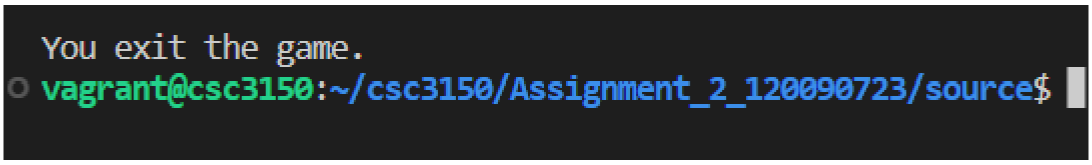
```
|||||||||||||||||||||||||||||||||||||||||||||||||
 ===============
                    ===============
==========                                 =====
 =============                                 ==
           ===============
                   ===============
    ===============
            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



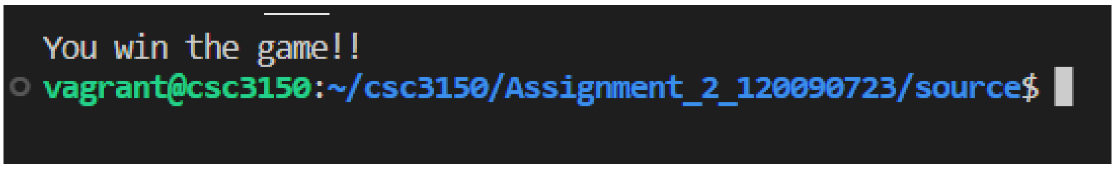
```
|||||||||||||||||||||||||||||||||||||||||||||||||
         ===============
            ===============
   ===============
 =====                                 ==========
                   ===============
           ===============
            ======0========
                  ===============
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



```
You lose the game!!
vagrant@csc3150:~/csc3150/Assignment_2_120090723/source$
```

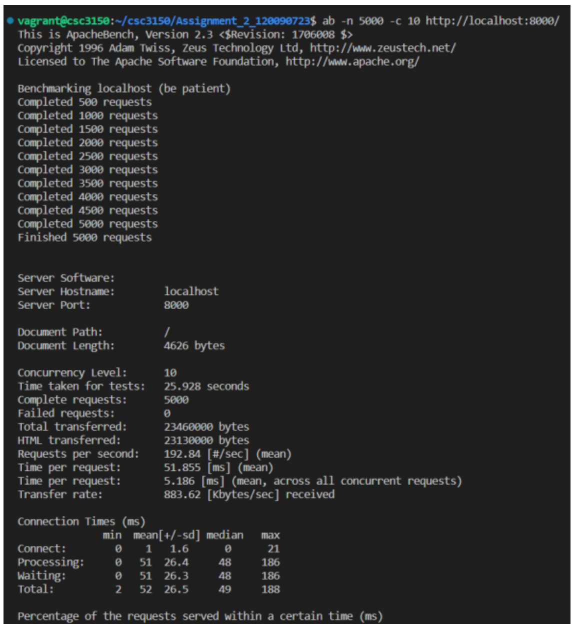


```
You exit the game.
vagrant@csc3150:~/csc3150/Assignment_2_120090723/source$
```

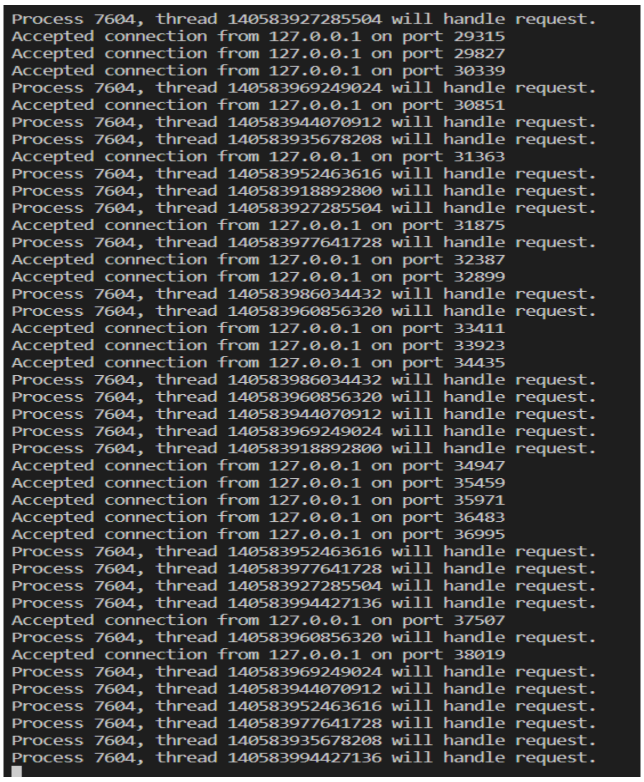


```
You win the game!!
vagrant@csc3150:~/csc3150/Assignment_2_120090723/source$
```

Bonus:



```
vagrant@csc3150:~/csc3150/Assignment_2_120090723$ 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:   25.928 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      23460000 bytes
HTML transferred:       23130000 bytes
Requests per second:    192.84 [#/sec] (mean)
Time per request:       51.855 [ms] (mean)
Time per request:       5.186 [ms] (mean, across all concurrent requests)
Transfer rate:          883.62 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.6      0      21
Processing:     0   51  26.4     48     186
Waiting:        0   51  26.3     48     186
Total:          2   52  26.5     49     188

Percentage of the requests served within a certain time (ms)
```



```
Process 7604, thread 140583927285504 will handle request.
Accepted connection from 127.0.0.1 on port 29315
Accepted connection from 127.0.0.1 on port 29827
Accepted connection from 127.0.0.1 on port 30339
Process 7604, thread 140583969249024 will handle request.
Accepted connection from 127.0.0.1 on port 30851
Process 7604, thread 140583944070912 will handle request.
Process 7604, thread 140583935678208 will handle request.
Accepted connection from 127.0.0.1 on port 31363
Process 7604, thread 140583952463616 will handle request.
Process 7604, thread 140583918892800 will handle request.
Process 7604, thread 140583927285504 will handle request.
Accepted connection from 127.0.0.1 on port 31875
Process 7604, thread 140583977641728 will handle request.
Accepted connection from 127.0.0.1 on port 32387
Accepted connection from 127.0.0.1 on port 32899
Process 7604, thread 140583986034432 will handle request.
Process 7604, thread 140583960856320 will handle request.
Accepted connection from 127.0.0.1 on port 33411
Accepted connection from 127.0.0.1 on port 33923
Accepted connection from 127.0.0.1 on port 34435
Process 7604, thread 140583986034432 will handle request.
Process 7604, thread 140583960856320 will handle request.
Process 7604, thread 140583944070912 will handle request.
Process 7604, thread 140583969249024 will handle request.
Process 7604, thread 140583918892800 will handle request.
Accepted connection from 127.0.0.1 on port 34947
Accepted connection from 127.0.0.1 on port 35459
Accepted connection from 127.0.0.1 on port 35971
Accepted connection from 127.0.0.1 on port 36483
Accepted connection from 127.0.0.1 on port 36995
Process 7604, thread 140583952463616 will handle request.
Process 7604, thread 140583977641728 will handle request.
Process 7604, thread 140583927285504 will handle request.
Process 7604, thread 140583994427136 will handle request.
Accepted connection from 127.0.0.1 on port 37507
Process 7604, thread 140583960856320 will handle request.
Accepted connection from 127.0.0.1 on port 38019
Process 7604, thread 140583969249024 will handle request.
Process 7604, thread 140583944070912 will handle request.
Process 7604, thread 140583952463616 will handle request.
Process 7604, thread 140583977641728 will handle request.
Process 7604, thread 140583935678208 will handle request.
Process 7604, thread 140583994427136 will handle request.
```

What did I learn from the tasks:

1. I learned how to clean the screen.
2. I learned how to make the program sleep for a very short time.
3. I learned how to create a new thread in a process and end it.
4. I learned how to use mutex lock to avoid one thread be interrupted by another thread.
5. I learned how to allow the master thread to wait until the threads it creates ends.
6. I learned how to use signals to control the thread pools to work or sleep.
7. All in all, I learned how to do a multithreads programming.