CSC3150 Assignment2 Report

Ziqi Gao 高梓骐

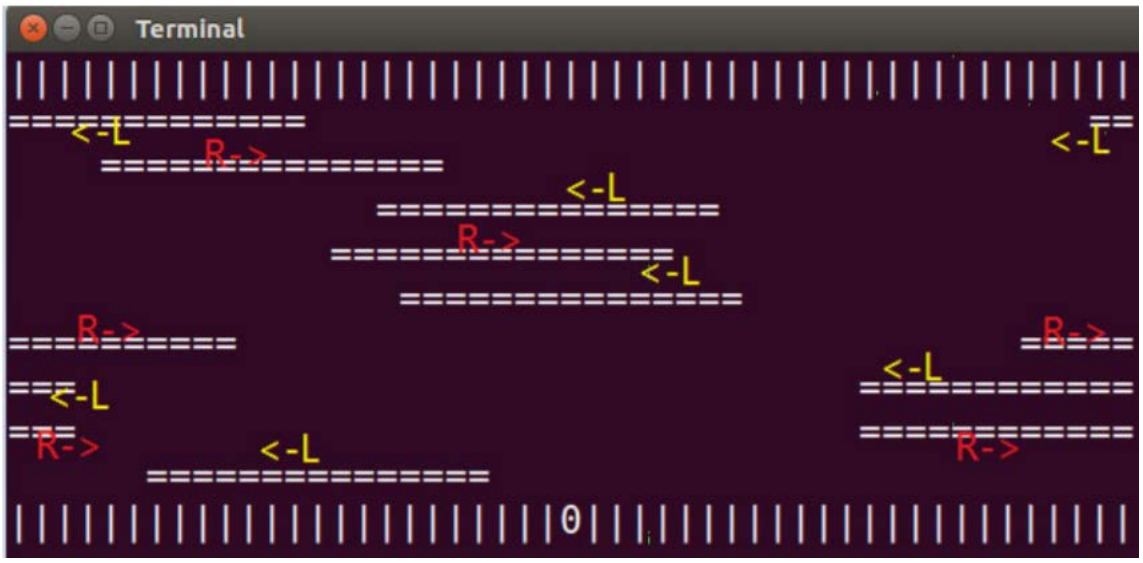
118010077

1. Introduction

Versions of OS: Ubuntu 20.04.1 LTS based on VMware Workstation 15

Kernel-Version: 5.4.0-52-generic

In this task, a program ("hw2.cpp") will implement a simple game where the player can drive the frog to get to the opposite of a river through some logs inside the river. The starting point of each log is random, while the speed (10 characters per second), length (15 characters) and the motion direction of each log are fixed. The directions and the lengths of the logs (shown as '======') are the same as the image shows:



Here are some rules of this game.

The player can use 'w', 'a', 's', and 'd' to move the frog, and 'q' to quit.

- 'w' or 'W': Move up by one unit.
- 'a' or 'A': Move left by one unit.
- 's' or 'S': Move down by one unit.
- 'd' or 'D': Move right by one unit.
- 'q' or 'Q': Quit the game.

How to win this game:

The '=' in this program is a part of the log. The frog can jump up to the log ('=') and move with the log together. If the frog can reach the opposite (topmost line) with the help of those logs, the player wins this game.

Loss Conditions:

However, if the frog jump into the river unfortunately (blank space except for the bottom and the top lines), the player loses this game.

Also, if the frog reaches the leftmost or the rightmost point of the game with the moving log, the player loses this game, either.

2. Basic Design

This program uses the API given by Pthreads to implement a multithread program. This program totally has 11 children threads:

- Each log (9 logs in total) is implemented by one thread. Those threads receive three parameters: the starting point, move direction, and the order of the log. It will firstly lock the thread and update the map's data to move the log and the frog if the frog is on the log. After the updating, it unlocks the process.
- One thread checks the user's input and moves the frog. This thread will check the user's keyboard input using the function kbhit(). According to the input, it will lock the thread and update the map's data and the frog's coordinate.
- One thread works as an umpire to monitor the game status. That is, it can check whether the player loses, wins, or quits the game. This thread will check the status according to the frog's next coordinate to judge whether the player wins, the frog jumps onto a log, or the frog jumps into the river and fails.
- The parent thread will print out the map with the help of the mutex lock.

The program uses two mutex locks to protect the global variable:

- One lock "mutex" keeps that only one thread is reading or writing the map data structure.

- One lock "mutex_frog" keeps that only one thread is reading or changing the frog's location.

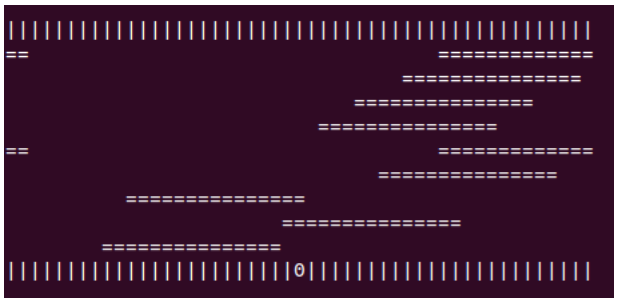
It also has one conditional signal to control the threads. Whether the frog thread will update the frog's location depends on the umpire thread's signal.

3. Execution

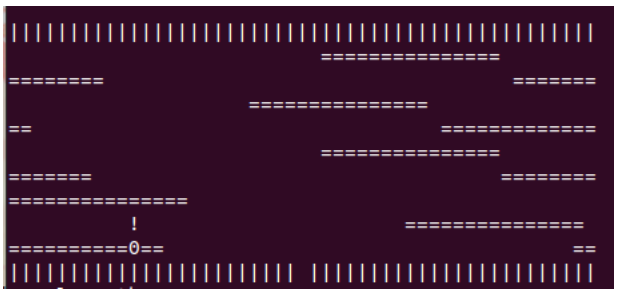
Since this program only has one source file, I did not write the Makefile. Please use g++ directly to compile this program, and do not forget to link the Pthread library.

```
andy@ubuntu:~/EXP-3150$ g++ -o hw2 hw2.cpp -lpthread
```

Finally, please enter "./hw2" in the terminal to run the program. You will see the output, and you can use 'w', 'a', 's', or 'd' to move the frog:

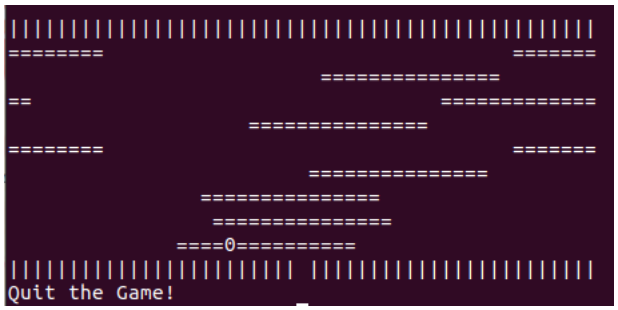


If the frog slip into the water or hit the border accidently, you will lose the game. The '!' shows the last location of the frog.



If you successfully reach the opposite, you will see the output like this:

You can quit this game anytime you like to exit by entering 'q':



4. What I Learned

Writing this program is my first time to code a multithread program. It is also my first time to write a game. This is definitely a precious experience. When I was coding, I did meet lots of problems. For example, in the beginning, I used a for loop to define a variable and then used this variable as arguments to create the thread. However, the loop did not work as I wished. Because of the memory allocation, some thread actually uses the variable created in the last loop.

Also, it is really hard to debug a multithread problem. I did not find any debug tool to help me like what I did when I was coding single thread programs. What I did is to trace the code and used the print function to debug.

Multithread programming is really interesting. Also, a multithread program is really powerful. I will read more materials or books to try to finish more complex projects.