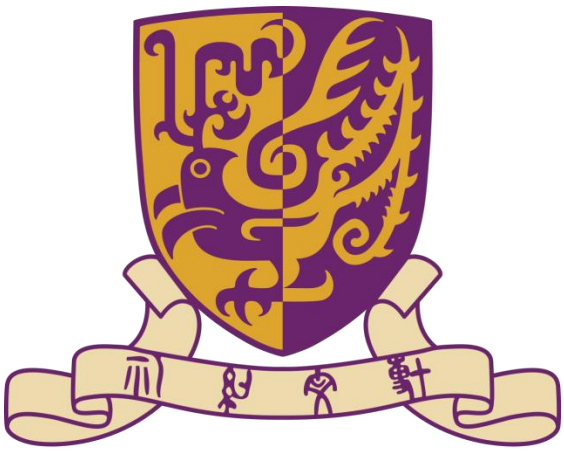


THE CNINESE UNIVERSITY OF HONG KONG, SHENZHEN

CSC3150

Operating Systems

Report for Assignment 2

Author:
Li Jingyu 李璟瑜

Student Number:
118010141

Oct 25, 2020

Contents

1 Design

Assignment 2 requires us to write a game using multithread technique. The game simulates the process that a frog crosses the river by jumping to the floating logs.

For the fundamental function implementation, I will explain my design in terms of **variables definition, functions, threads usage, program flow, and mutex lock**.

(a) **Variables definition:** Since this is a multi-thread program, the definition of the shared variables (global data) is necessary. “Node” (frog) represents the position of the frog (x & y). “Map” is the information of the graph to be printed (for program output). “Stop” represents whether the program reaches the end of the execution. “Sleep_time” sets the frequency for the screen to update. “Mutex” is the mutex lock to be used.

(b) **Functions:** Several functions have been used to encapsulate the certain process. “Print_map” is to print the map to the screen dynamically. “Game_status” is used to judge the game status, which includes win, lose and quit. “Kbhit” is used to accept the keyboard input. “**Logs_move**” is the most important thread functions which control the movement of the log. Since there are 9 logs in total, these functions will have 9 executions representing 9 rows which differ in the direction of the movement or some other properties. “Frog_move” is another important function which controls the movement of the frog, responding and judging the game status according the input. “Thread exception” is used to handle exceptional case. “Init_map” initializes

the map at the beginning of the execution. The main process is written in "main" function.

(c) **Threads usage**: I have used 11 threads in this program. One is for the output, printing the current state of map to the screen dynamically. One is for the frog movement, and it will receive the keyboard input to decide how the frog will move in the next step. It will also judge the game status according to the position of the frog. Player can also quit here. The other 9 threads are used to control the log movement. Each thread is responsible for moving one log. They will change the state of map, moving the log one step left or right. The starting point of the log is randomly set. Special attention is needed that the threads will usleep() for some time (except the frog) to suspend the thread for better user experience, or the program will print the map extremely quickly and makes the screen a mess. I set the usleep() time 80000 in hw2.cpp.

(d) **Program flow**: At first, the map will be initialized, and the frog will stand at one side of the river. Then, initialize the mutex lock, and create the 11 threads (log, frog, print). The threads will run iteratively until the frog reaches the other side of the river or die in the process (go back at the origin, hit the wall, drown into river). Then the threads are joined, and the threads exit one by one. The message for game status will be printed out.

(e) **Mutex lock**: Mutex lock is the primary mean to implement thread synchronization and protect shared date. When a piece of the code is locked, the threads cannot perform the operations simultaneously. Multiple writes

and multiple reads are not allowed. The lock provides the protection of accessing the shared data resource. As a result, it should be added to the processes which deal with the shared data (global variables). In this program, I add two mutex locks. One is in the threads for moving the logs, and it is used to prevent multiple writing to the “map” and ensure safety (during the process the information of the frog in the map is also changed). The second lock is added in the thread for printing the map. It is used to protect the map from changing when showing the map to the screen. Pay attention that the mutex lock should be unlocked before the usleep() or the threads cannot run normally since the time interval of one lock is too large.

For the bonus question, I will explain what functions I have added.

(a) **Color**. Add the green color to the bank and make it seem like the grass. Add the blue color to the logs to make it seem like a river. Change the frog to green color and change the blue log where the frog lands in to green so as to see where the frog is more clearly and make the picture more interesting. (A very simple “GUI”)

(b) **Random length**. Set the length of the logs randomly by using srand() and rand() function. Now the length of the logs will vary from 8 to 18 (original: 15).

(c) **Random speed**. The speed of each log varies slightly now. The difference is 5000 units (usleep(5000)).

(d) **Adjustable speed**. Now the difficulty of the game can be adjusted by increasing or decreasing the speed of the log. There are in total 9 levels. Use ‘j’ or ‘k’ to adjust.

(e) **Hint**. Now some hint messages (name, operations, current speed) are printed to make the interface more informative and user-friendly.

2 Problems and Solutions

In this part I will discuss some problems I met and how I solved them when writing the assignment.

(a) **Problem**: C-string. When seeing the template, I was a little bit confused for some sentences and I think the codes were not so beautiful, so I changed it. However, after I modified the template, the program just printed the map in a mess. Multiple lines are combined in one row and printed many times.

Solution: For C-string (char*), a ‘\0’ must be added at the end to symbolize the termination of the string. So when initializing the map (char[][]), one extra space is needed to store the zero. I change the initialization into char[ROW+1][COLUMN+1] and iterate in for normally (range:0-ROW/COLUMN-1).

(b) **Problem**: Threads. At first I didn’t know how many threads I should use and what they should be responsible for. Also the threads run abnormally.

Solution: Create 9 threads for log movement, 1 thread for frog and 1 thread for printing. I think I separate them reasonably. For the second problem, remember to exit the thread.

(c) **Problem**: Print. The map is printed too fast and it makes the interface a mess.

Solution: Use a thread for printing the map specifically. Add usleep() too the thread and make it roughly the same as that of the log_move.

(d) **Problem**: Compilation. Fail to compile. Unable to see the error of compilation.

Solution: For multi-thread program, special command associated with threads needs to be used (g++ hw2.cpp –lpthread ./a.out). Do not input the compiling and executing command in one line (use ;). Separate them. Step by step.

(e) **Problem**: Fail to update the screen.

Solution: Use printf("\033[H\033[2J"); to clear the current screen.

(f) **Problem**: Bugs.

Solution: For example, change "==" to "=".

3 Environment and Execution

Please run in the virtual machine (**Virtualbox 6.1.14**)

OS version: **Linux Ubuntu 16.04**

Kernel version: **4.10.14**

Open the **terminal**, cd to the target directory. Enter the command:

"**g++ hw2.cpp -lpthread**" (for the bonus, change hw2.cpp to hw2-bonus.cpp)

Enter: "**./a.out**"

```
[10/25/20]seed@VM:~/.../source$ g++ hw2.cpp -lpthread
[10/25/20]seed@VM:~/.../source$ ./a.out
```

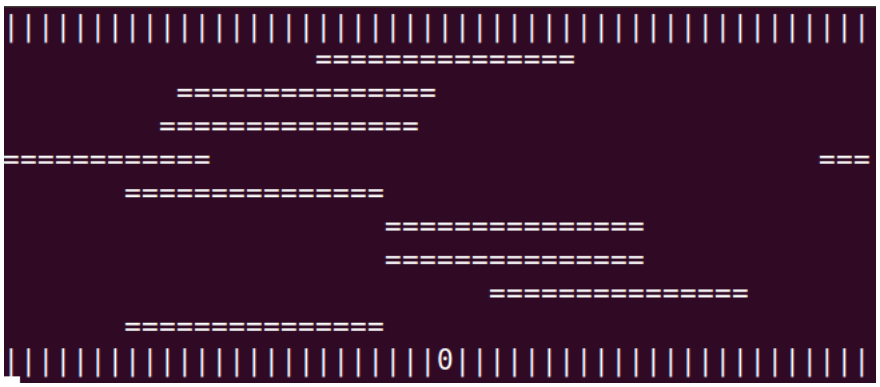
Then you can start to play the game!

4 Output

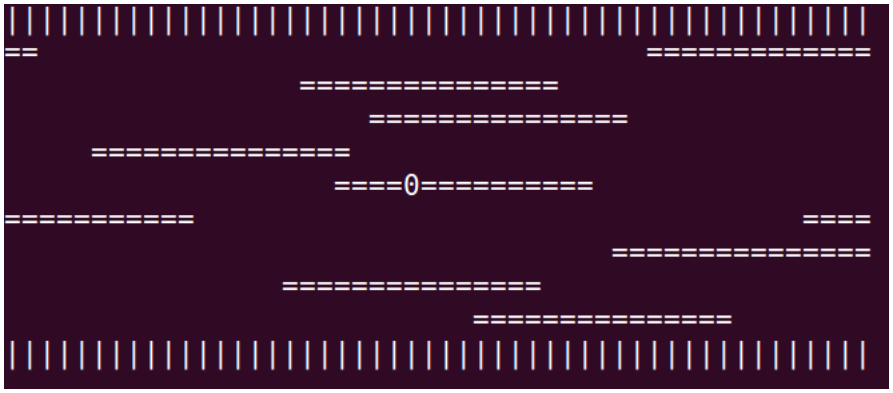
In this part the relevant program outputs will be shown.

Basic:

Starting point ~



In the middle of the river ~



Win the game ~

```
You win the game!!
[10/25/20]seed@VM:~/.../source$
```

Lose the game ~

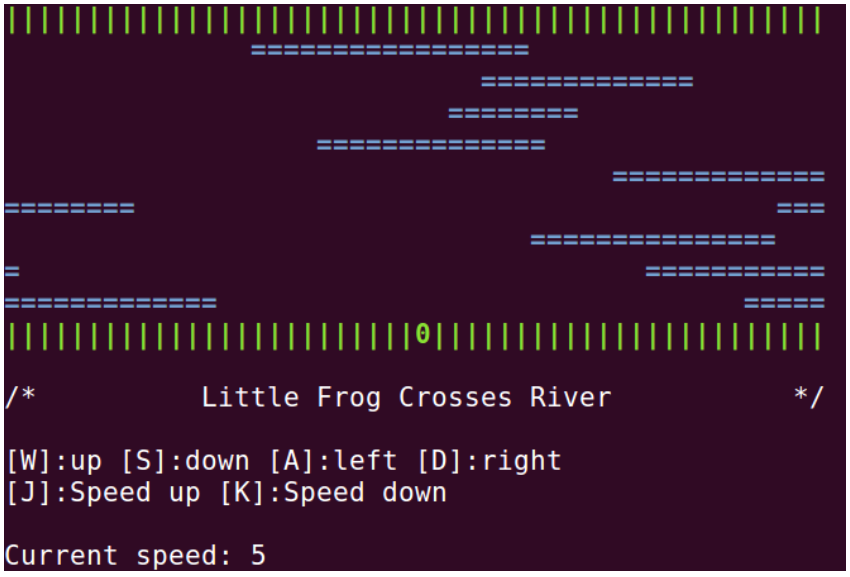
```
You lose the game!!
[10/25/20]seed@VM:~/.../source$
```

Quit the game ~

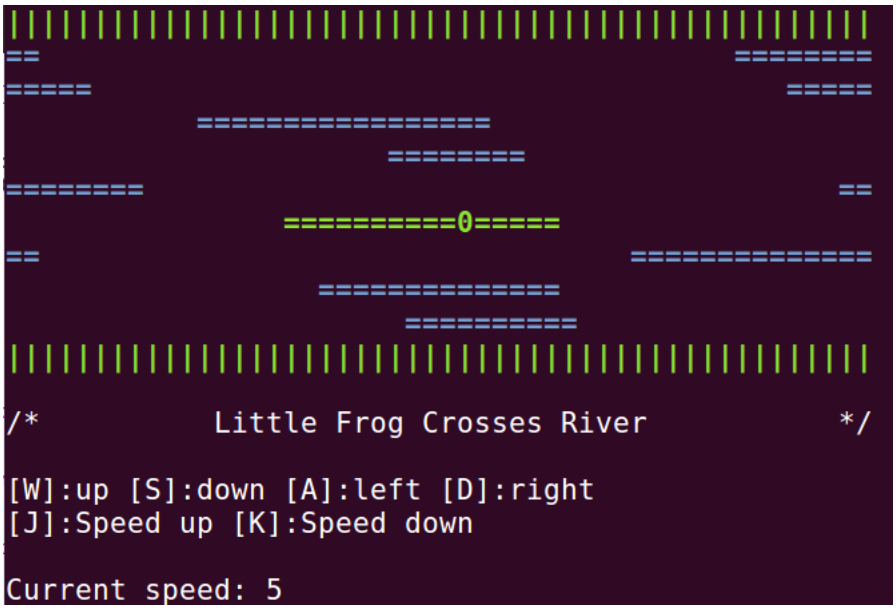
```
You exit the game.
[10/25/20]seed@VM:~/.../source$
```

Bonus:

Interface ~



In the middle of the river ~



5 Feeling

Assignment 2 is not as hard as I imagined at first. I spent less time than assignment 1 finishing the coding. Many of my classmates finished it even several days in advance. The aim of the project is to lead us to try multi-thread programming and comprehend the gist of parallel process. What I have learned in this assignment:

(a) Multi-thread programs. I have just known a little bit about multithreads before. After the tutorial and coding experience, I get to know more about its idea and become more familiar with the creation, termination, join, signal conditions and mutex lock of the threads. The project makes me more interested in multi-thread programming.

(b) Divide and conquer. The core of multi-thread is to divide the task into different parts and solve them separately. In the program I distribute 9 threads

for movement and 1 thread for frog control and 1 thread for printing. It makes me feel like I am the commander who can finish the task efficiently.

(c) Some practical knowledge / skills. The macro definition “#define” can make the code more beautiful and clear. The escape character “\033+…” can clear the screen or move the cursor or add color to the outputs or perform some magical functions. The use of while and usleep() can make the interface dynamic, just like animation.

(d) Improvement of programming ability. Use of function to encapsulate the certain process. Debug and remember some common mistakes (forget the \0 after C-string, some typos). Use Linux to compile C/C++ programs. To design and make some simple games.

(e) ……

I also feel that the tutorials are important and highly related to the project, I think I will pay more attention to it. Besides, I think TA is interesting sometimes. He will share us with his study abroad experience in Europe. He will also play psychological warfare to make the cheating students admit their plagiarism behaviors.

That’s all.