CSC3150 Operating Systems

Project 2 report

Name: Xiao Nan

Student ID: 119010344

Date: 2021/10/23

The Chinese University of Hong Kong, Shen Zhen

0. Introduction:

In homework 2 I implemented the frog crossing river game using the Pthread library in C++. I completed 11 threads with 9 corresponding to the moving logs, one corresponding to printing out the map, as well as one corresponding to controlling the movement of the frog using the curses library. If the player wins, loses or exits, the relevant messages would be printed in the terminal. In the bonus part I implemented the GUI for the game using OpenGL, randomized the number of logs in each row of the river and implemented a slide bar for controlling the speed of the moving log.

1. Design of the program:

In homework 2 I used a two-dimensional array map[][] to record the game state at every timestamp. I defined a global struct to store the position of the frog and update it whenever necessary in other threads. I first initialized the screen by updating the banks to map[][] and randomly generating logs with different lengths (with the length of the logs stored in a global row_log_nums array) and place them on the corresponding places (randomized positions in each row of the river stored in log_start_arr array) in map[][]. I also initialized the frog's position (which is at the center-down of the map). Then I created 11 threads for the game:

I created 9 threads for controlling the logs. In order to ensure isolation of operations, the movement of each log is carried out with a mutex lock being used and the lock is unlocked when the log moving operation is done (to be discussed later). Each thread controls one log moving on a certain row of the river.

The motion of the logs can be divided into two cases: left-running ones and right-running ones. For the left-running logs, if the frog is not on the log (judging whether frog.x coincides with the log's row number), then I simply swap the tail log element (the rightmost log) with the river element next to the head log element (the leftmost log) to finish moving left, and if the frog is on the log, I first swap the frog with the log element next to the frog (on the left-hand side) and then swap the tail log element with the river element next to the head log. For the right-running logs. If the frog is not on the log then I do the same thing as the case for the left running log without frog, and if the frog is on the log then I first swap the frog with the element next to the frog on the right and then do the swapping between the tail log element and the river element next to the head. In both cases the issue of logs crossing the edge is solved by taking the modulus (E.g. curr_head = (curr_head + 1) % (COLUMN-1)) every time when the log is moving, which makes the code more brief and efficient. The moving logs are stopped when the player wins, loses or exits the game.

I created 2 other threads assisting the map manipulation: One thread periodically prints out the map information with the time interval specified by the global variable LATENCY (in miliseconds). The other thread controls the movement of the frog: When the frog is moving to the log I simply replace the destination with the frog element and recover the previous location with either the bank element or the log element. When the frog moves to the river, the "lose" flag would be set. Such movement is also synchronized by usleep(LATENCY) for printing properly. Note that before carrying out frog movement and map

printing, the mutex lock must be set to avoid the situation where the movement or printing is done with the log moving incompletely. Finally when all threads are terminating, I use pthread_join to collect all threads and print out win, lose or exit status messages.

In the bonus problem I implemented the GUI using GLUT and achieved random generalization of logs as well as slide bar control. For the GUI, I used another one thread for initializing the window using glutinit(), binding that to mouse, motion and display functions and starting the main loop of the window display. In the display function I achieved refreshing by calling glutPostRedisplay in it with time interval specified by LATENCY. For random generalization of logs I used srand() to generate random numbers and use rand() to apply them on initializing log_start_arr. For the slide bar, I drew the bar on the right side of the window and control it by binding that to mouse click function (which enables selecting speed by clicking the bar) and mouse drag functions (enabling dragging the bar to adjust the speed dynamically).

2. Runtime environment:

2.1. Linux version:

```
1. namshoo@ubuntu:/$ cat /etc/issue
2. Ubuntu 16.04.5 LTS \n \l
```

2.2. GCC version:

```
1. namshoo@ubuntu:/$ gcc --version
2. gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609
3. Copyright (C) 2015 Free Software Foundation, Inc.
4. This is free software; see the source for copying conditions. T
   here is NO
```

```
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICU
LAR PURPOSE.
```

2.3. OpenGL/GLUT library installation for the bonus problem. To install those libraries, type the following commands in your terminal:

```
namshoo@ubuntu:/$ sudo apt-get install freeglut3-dev
…
namshoo@ubuntu:/$ sudo apt-get install mesa-common-dev
```

Then, ensure that those packages are installed:

```
namshoo@ubuntu:/$ dpkg -l | grep freeglut3
ii  freeglut3:amd64                          2.8.1-2
amd64        OpenGL Utility Toolkit
ii  freeglut3-dev:amd64                      2.8.1-2
amd64        OpenGL Utility Toolkit development files
namshoo@ubuntu:/$ dpkg -l | grep mesa-common
ii  mesa-common-dev:amd64                    18.0.5-
0ubuntu0~16.04.1                        amd64        Developer
documentation for Mesa
```

3. Steps to execute my program:

In command line, you need to first unzip the .zip file. Make sure that the .zip file is under the current directory:

```
namshoo@ubuntu:~$ unzip Assignment_2_119010344.zip
```

3.1 Normal game:

"cd" to the "source" execution directory, and type "make hw2" to compile the source code. Make sure that the hw2.cpp is under the current directory:

```
namshoo@ubuntu:~$ cd Assignment_2_119010344/source/
namshoo@ubuntu:~/Assignment_2_119010344/source/$ make hw2
```

Then, type "./a.out" to execute the program.

```
1. namshoo@ubuntu:~/Assignment_2_119010344/source$ ./a.out # use
   w, a, s, d to control the movement of the frog. After the game
   terminates, the outputs would be shown on the terminal.
```

3.2 Bonus program:

"cd" to the "source" execution directory. Type "make bonus" to compile the bonus program (bonus.cpp). Make sure that the libraries specified in the "Runtime Environment" for the bonus problem had been installed (if not, go to that section for installation guides):

```
1. namshoo@ubuntu:~$ cd Assignment_2_119010344/source/
2. namshoo@ubuntu:~/Assignment_2_119010344/source$ make bonus
```

Then execute the bonus program:

```
1. namshoo@ubuntu:~/Assignment_2_119010344/source $ ./bonus.out
```

4. Program output screenshots:

4.1 Homework 2:

The results for winning, losing and exiting the game is shown in figure 4.1, 4.2, 4.3. The state where the frog is on the bank is shown in figure 4.4:

Figure 4.1: Winning state

Figure 4.2: Losing state

```
You exit the game!!
namshoo@ubuntu:~/Desktop/Assignment_2_119010344/source$
```

Figure 4.3: Exiting state

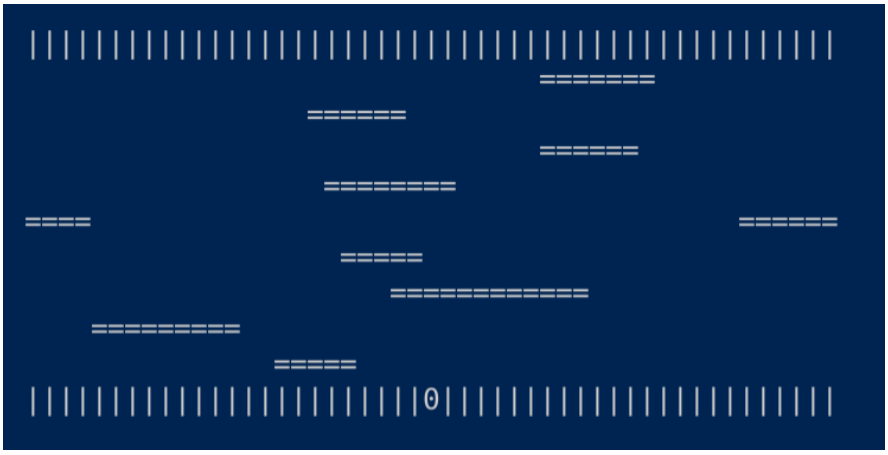

Figure 4.4: The state on the bank

4.2 Bonus:

Normal state is shown in figure 4.5. Winning, losing and exiting states are shown in figure 4.6, 4.7, 4.8, respectively. The video is attached in the submitted files (named "bonus_video.mp4"):

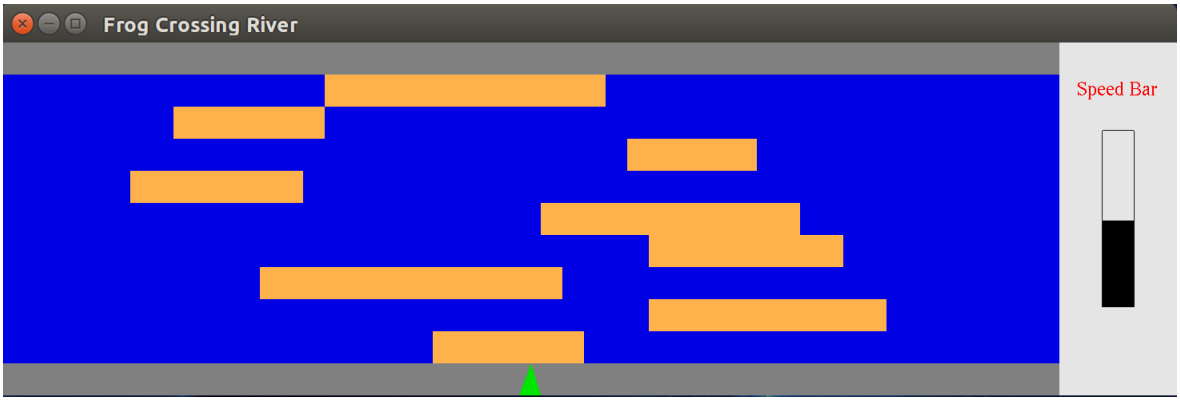

Figure 4.5: Normal state

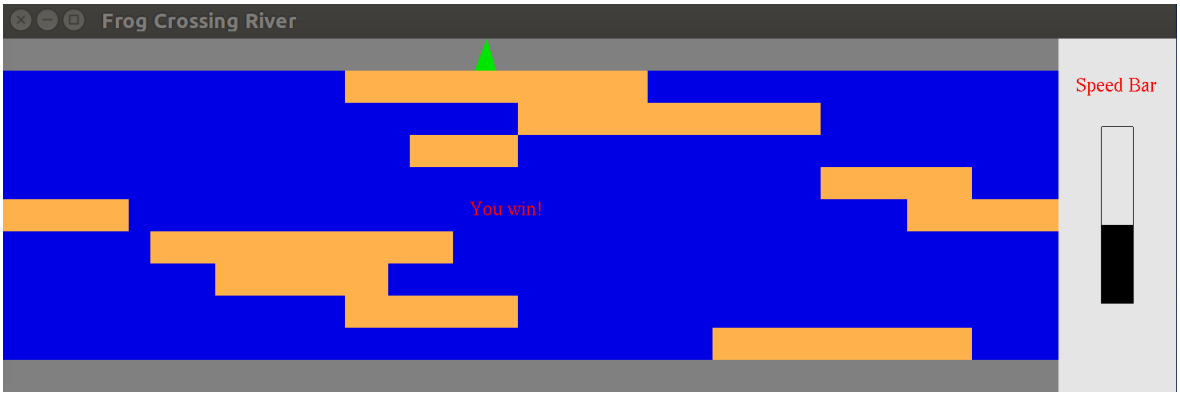

Figure 4.6: Winning state

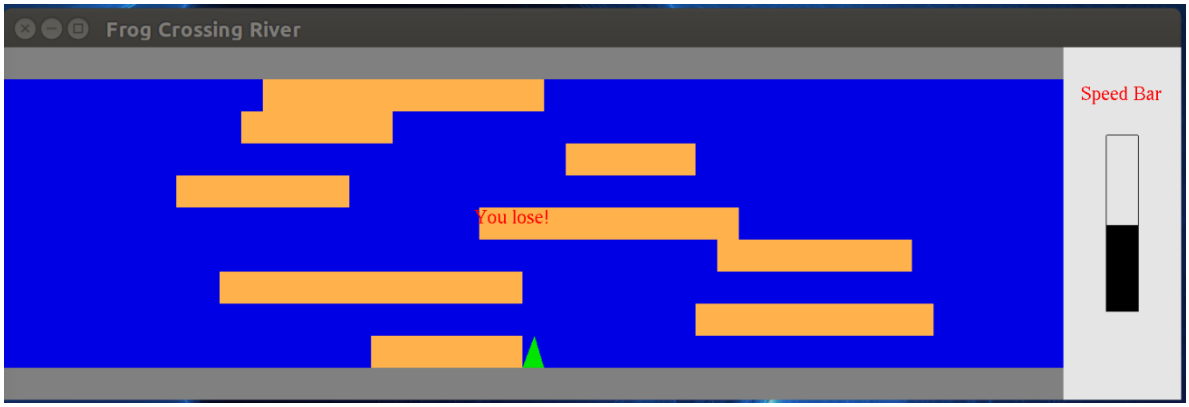

Figure 4.7: Losing state

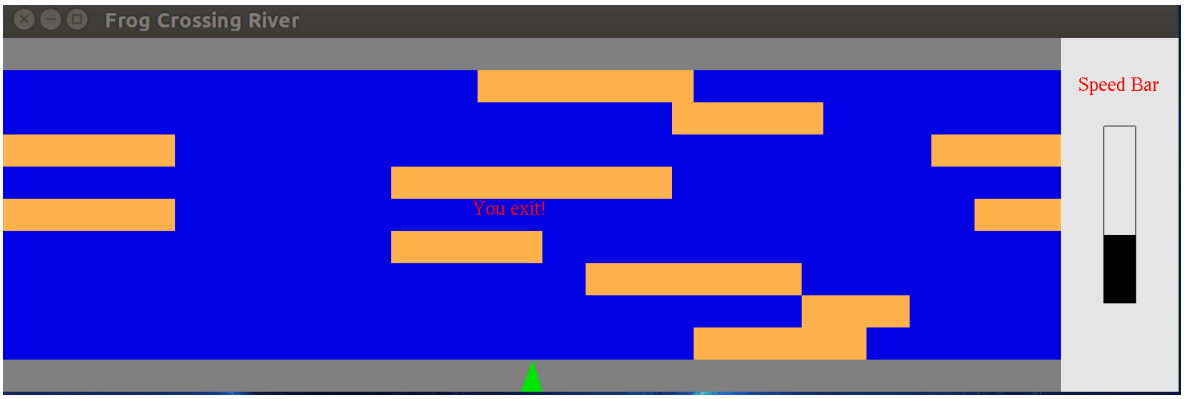

Figure 4.8: Exiting state

5. Things I've learnt from the project:

I learnt a lot of useful techniques for dealing with multi-threaded programming. In the normal part I learnt managing threads (creating, joining and terminating threads) using pthraed

library and isolating the query of those threads on certain data using the mutex locks. In the bonus part I learnt using OpenGL/GLUT for developing GUIs or games. I learnt using the powerful window and clicking event management tools (like glutMouseFunc, glutMotionFunc) to interact the GUI with the user elegantly. Also I learnt that the selection of library for completing a task should be partly based on the documents available for the task (for example, when dealing with the GUI I didn't choose Cairo because there are very few documents about that library and the community for that library, like that on Stackoverflow, is undeveloped).