CSC3150 Assignment2

119010054 Dai Yulong

1. Design Ideas

1.1 Overview

The assignment 2 mainly focus on multi-thread programming. It's required to implement the game "Frog crosses river". The file structure is shown below

```
dai@ubuntu:~/Desktop/Assignment_2_119010054$ tree
.
├── demo_bonus.mp4
├── demo.mp4
├── Report.pdf
├── source
│   ├── hw2.cpp
│   └── makefile
└── source_bonus
    ├── hw2.cpp
    └── makefile

2 directories, 7 files
```

- In the `source` folder, the basic version is implemented (with random log size). It has created totally ten threads, nine for each log and one for frog. It used `kbhit(void)` to monitor the keyboard hit and `printf("\033[H\033[2J");` to repeatedly clear the terminal window.
- In the `source_bonus` folder, the improved version with GUI is implemented, which accomplish all requirements (including all bonus). It uses OpenGL (GLUT) to draw the graphical output. And it also includes a slide bar to adjust the speed of floating logs. The steps of set up environment (very easy) and sample output is listed below.

1.2 Basic version

- **Multi-thread Programming**

  This program has created ten threads, nine for logs and one for log. Thus, high parallel efficiency can improve game performance. Every thread is connected by `pthread_join(pthread_t thread, void **retval)`, which can run terminal print in the main thread after every child thread exits. In every child thread, when reading and writing the shared data (e.g. `map[ROW+10][COLUMN]`), `pthread_mutex_lock(&mutex)` is used to protect access to a shared data resource (one thread can access at one time). And in each log thread, after one loop, the program uses `pthread_cond_signal(&threshold_cv)` to implement synchronization with frog thread by controlling thread access to data. In this way, the program can ensure the synchronization of real-time keyboard hit and modification of `map[ROW+10][COLUMN]`. And no breaking-ups will appear. After each thread is terminated by `pthread_exit(NULL)`, the mutex and condition variable are destroyed by `pthread_mutex_destroy(&mutex)` and `pthread_cond_destroy(&threshold_cv)`.

- **Logs Moving**

  There're totally nine logs in the river. The 1st, 3rd, 5th, 7th and 9th logs move towards the left. The 2nd, 4th, 6th and 8th logs move towards the right. The length and initial positions of logs are randomly created by `rand()`. The length of log is between ten and twenty. Notice that I use `srand((unsigned)time(0))` as the random sand, which make sure the logs status will be different in every game. The speed of moving logs is determined by `usleep(50000)`.

- **Keyboard Hit and Frog Moving**

  The signal `isOver` determines whether the game is over (exit, win or lose). If the game is not over, the function `kbhit()` will monitor the keyborad tap in real-time and control the move the frog. Notice that the frog cannot move down at the lowest level. And the game will lose if the frog cross either any left or right boundary.

1.3 Improved version (with GUI)

- **Multi-thread Programming**

  This version also has created ten threads, but nine for logs and one for GUI window. The moving of frog is integrated into the GUI thread. The usage of multi-thread functions is the same as basic version.

- **GUI Design**

To implement a graphical output and slide bar, I use openGL (mainly GLUT). The method of setting up environment will be shown later. A new thread `void *init_window( void *t )` is created, where uses `glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA)` to initialize the openGL window, `glutKeyboardFunc( onKeyTap )` to monitor the keyboard hit, `glutMouseFunc( onMouseTap )` to monitor the mouse tap, and `glutTimerFunc(10, display, 1)` to registers a timer callback to be triggered in 10 milliseconds (refresh the window in 10 milliseconds). In the `void display(int nTimerID)`, if the game is not over (`isOver = false`), it will draw the GUI window normally. And if the game is over (exit, win or lose), the window will be clear and print the end message for two-seconds, then the GUI thread exit and the main thread print the message also in the terminal.

- **Slide Bar**

  In the lower half of window, I create a slide bar to adjust the speed of the moving logs. Just use mouse to click different position of the bar, the speed will be change. Notice that the leftmost end represents the lowest speed, and the rightmoset end represents the highest speed. The initial position is in the middle, whose speed equals to the basic version's (`usleep(50000)`). The speed range of moving logs is between `usleep(80000)` to `usleep(20000)`. The shorter the thread hang interval, the faster the logs runs.

2. Environments and Run Methods

2.1 Environments

My project adopts the recommended environment:

- Linux Version

```
dai@ubuntu:~$ cat /etc/issue
Ubuntu 16.04.5 LTS \n \l
```

- Linux Kernel Version

```
dai@ubuntu:~$ uname -r
4.10.14
```

- GCC Version

```
dai@ubuntu:~$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

!!!How to install OpenGL (GLUT)

Type following instructions in the Ubuntu terminal:

```
sudo apt-get install build-essential libgl1-mesa-dev
sudo apt-get install freeglut3-dev
sudo apt-get install libglew-dev libsdl2-dev libsdl2-image-dev libglm-dev
libfreetype6-dev
```

2.1 Run Methods

For both two programs, I write makefile to complie it. So only two steps are needed to run the program.

1. Using `cd` to enter the `source` or `source_bonus` directory, then type `make` command and enter.
2. Enter `./a.out` to execute the program:

```
   dai@ubuntu:~/Desktop/Assignment_2_119010054/source$ make
   g++ hw2.cpp -lpthread
   dai@ubuntu:~/Desktop/Assignment_2_119010054/source$ ./a.out
```

```
   dai@ubuntu:~/Desktop/Assignment_2_119010054/source_bonus$ make
   g++ hw2.cpp -o a.out -lpthread -lGL -lglut
   dai@ubuntu:~/Desktop/Assignment_2_119010054/source_bonus$ ./a.out
```

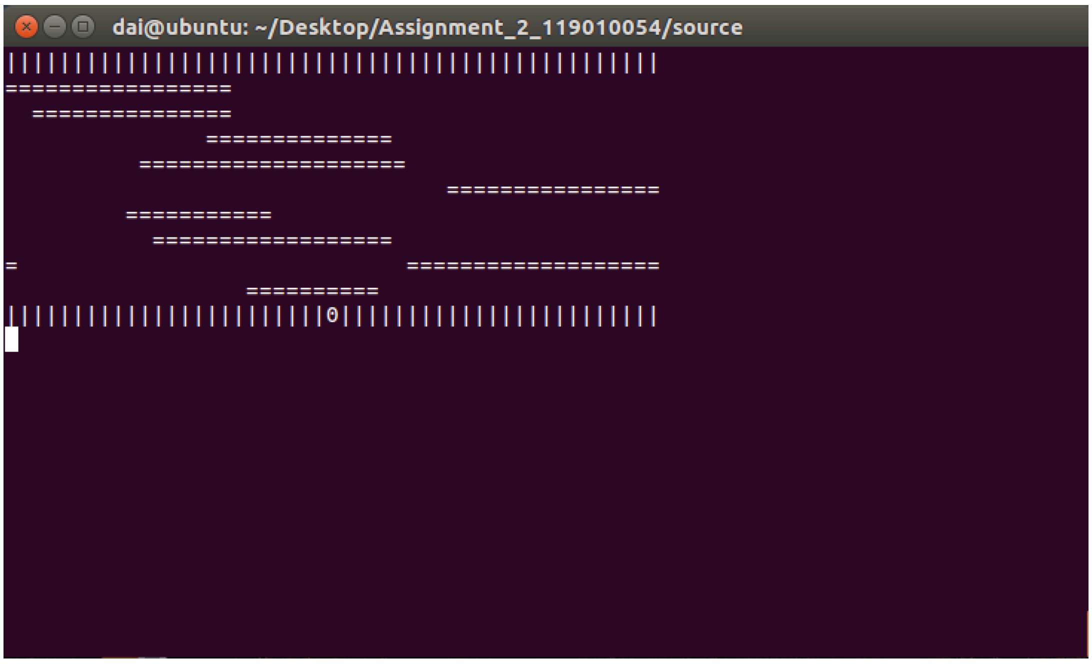
3. In the corresponding directory, type `make clean` command to clear the `a.out`.

3. Sample Output

3.1 Sample Outputs of basic version `source`

- Sample output

  The real-time playing video can be watched in the attached file `demo.mp4`.

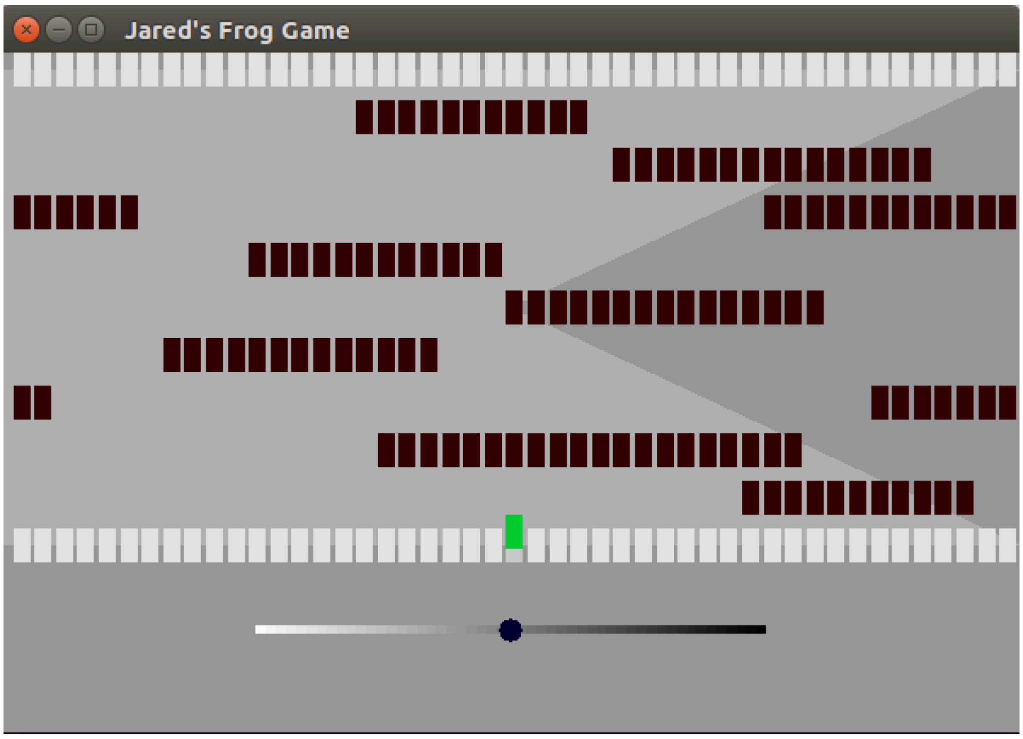


```
|||||||||||||||||||||||||||||||||||||||||||||||||
=================
  ===============
               ==============
          ====================
                                 ================
         ===========
           ==================
=                             ===================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

3.2 Sample Outputs of improved version with GUI `source_bonus`

- Sample output

  The real-time playing video can be watched in the attached file `demo_bonus.mp4`.



4. Reflection

After finishing the assignment1, I learned several things about multiprocess programming, some basic knowledge about GUI library.

Comprehensive understanding of multiprocess programming:

- Be familar with many basic multithread programming functions under Linux, including pthread creation, pthread termination, pthread join, pthread mutex and pthread condition.
- Get to know how threads communicate with others. Each thread has its own stack to store its own variable. And all threads share one common heap area with each other. In this way, we can create some global variables and allocate some memory in the heap area to store the information from each thread.
- Learn how to use terminal control, for example, using `printf("\033[H\033[2J")` to move the cursor to the upper-left corner of the screen and clear the screen.
- Learn how to monitor keyboard hit and output operation. `int getchar(void)` function is used to get/read a character from keyboard input. And `int puts(const char *str)` can writes a string to stdout up to but not including the null character.

Learn to use a new GUI library - OpenGL

- To implement the graphical output and slide bar, I learn a new GUI library - OpenGL by reading official documents.
- Get to know many functions about drawing window, keyboard and mouse monitoring, refresh operations.
- Apply multithread programming in drawing window. Separate the draw operation thread and logs moving thread, which greatly improves parallel efficiency and increases performance.

Special Learnings

In this assignment, actually I have implemented three versions of multi programming code. The first version is to use two threads, one created thread handle all nine logs and one frog, which is easiest and seemingly working well. But it has very bad parallel performance. The second version is to create nine thread for logs. Then I thought if I could judge the state of the frog in each log thread, why not open a new thread and synchronize it? So the current version comes out. I create ten new threads, nine for logs and one for frog. At the beginning, the screen sometimes flashes when the frog is moving. I guess this may be caused by the untimely update of frog position when each log moves in its thread (`usleep(50000)` is implemented in the log thread). Then the problem is perfectly solved by adding `pthread_cond_signal(&threshold_cv)` to implement synchronization with frog thread by controlling thread access to data. The current version without doubt has the best parallel performance and also runs smoothly. In the process of continuous exploration, I have a deep understanding of multithreaded programming。