Assignment2

Name: Xinyu Xie
Student ID: 119020059

Frog crosses river

How did I design my program

The task is to implement a frog crosses river game using pthread. I divide my design into the following parts:

Log movement design

For all logs, I randomly generate the length of the log and the starting left position of the log.
Since all logs move at the same speed, I use `LOGSPEED` to control the speed of all logs.
I assign the moving direction of each log using this rule: if the row of log is odd, it moves to the left, otherwise, it moves to the right. Thus, every `LOGSPEED` time, all logs will update their positions by one step.

Frog movement design

The frog can only move on the log. If the frog is on the log, the frog will move with the log. If the frog is not on the log, the frog will fall into the river. The frog will die if it falls into the river. The frog will win if it reaches the other side of the river.
I use `kbhit` function to catch user's keyboard input. If the user presses `w`, the frog will move up. If the user presses `s`, the frog will move down. If the user presses `a`, the frog will move left. If the user presses `d`, the frog will move right. If the user presses `q`, the game will exit.
After catching the user input, I use `check_status` function to check whether the game should be in normal status, lose, win or exit. If the game is in normal status, I update the position of the frog accordingly.

Game status design

I use a global variable `status` to denote the status of the game, with 0 denoting normal status, 1 denoting win status, 2 denoting lose status, and 3 denoting quit status.
I use `check_status` function to check the status of the game. If the frog is on the log, the game is in normal status. If the frog reaches other side of the bank, the game is in win status. If the frog jumps into the river or reaches the left or right boudary of the river, the game is in lose status. If the user presses `q`, the game is in quit status.
If the game is not in normal status, we end the game and use `display_outputs` to print corresponding messages.

Thread design

I create 10 threads in total. One thread denoted by `thread_frog` controls the frog move. For other nine threads stored in `threads_logs`, each thread controls the movement of a log. Each pthread create is paired with pthread exit. Use `pthread_join` to wait for all threads to finish.
In order to protect the shared data between different threads, I use `pthread_mutex_lock` to lock the pthread

while it is modifying the shared data, and I use `pthread_mutex_unlock` to unlock the pthread after it finishes modifying the shared data. In detail, I create two mutex `map_mutex` and `frog_mutex`, the `map_mutex` locks the map resource denoting the whole river with logs, and the `frog_mutex` locks the frog position resource.

The environment of running my program

My Linux version is:

```
vagrant@csc3150:~$ cat /etc/issue
Ubuntu 16.04.7 LTS \n \l
```

My Linux kernel version is:

```
vagrant@csc3150:~$ uname -r
5.10.146
```

My gcc version is:

```
vagrant@csc3150:~$ 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.
```

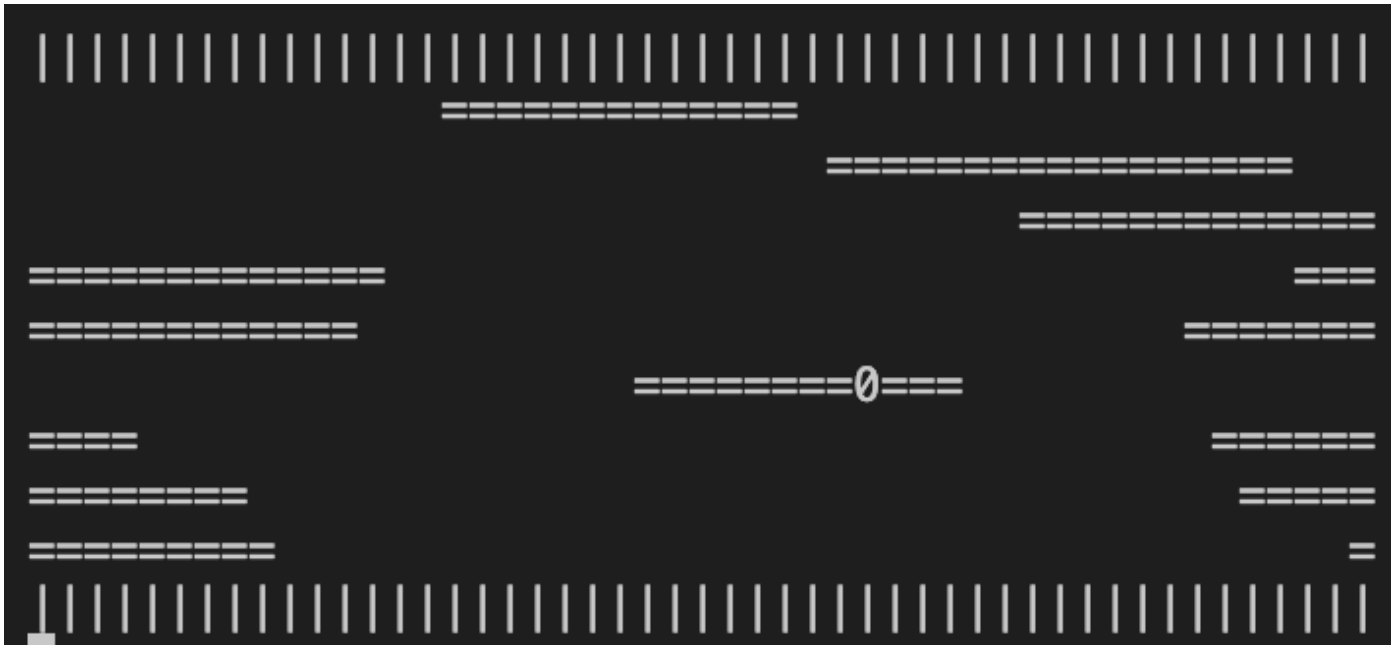
Steps to excecute my program

The following are the steps to execute the program:

1. Go to folder `source`.
2. In terminal, type `make all` to compile all programs.
3. In terminal, type `make run` to run the game.

Screenshot of my program output

Normal Status



Win Status

```
You win the game!!
vagrant@csc3150:~/csc3150/CSC3150-Assignment2/source$
```

Lose Status

```
You lose the game!!
vagrant@csc3150:~/csc3150/CSC3150-Assignment2/source$
```

Exit Status

```
You exit the game.
vagrant@csc3150:~/csc3150/CSC3150-Assignment2/source$
```

What did I learn from the tasks

In this task, I learned the following knowledge:

1. Through the `kbhit` function, I learned how to catch the use keyboard input.
2. I have a deeper understanding about pthreads, especially how to design multiple threads in a game.
3. I also have a better understanding about mutex and how to use mutex to protect shared data between different threads. Using mutex ensures that
   code being controlled will only be hit by a single thread each time, thus also need to remember to release the mutex when the thread finishes job.

Bonus

How did I design my program

In Bonus part, I need to implement two essential functions of a thread pool.

Thread pool design

The thread pool is a struct containing number of threads, a pointer of array to store all threads, a linked list to store all
tasks that need to be executed. The task queue is a linked list, with each task being one node.

async_init

In this function, I firstly do some initialization work to allocate memory of the thread pool. Then use `pthread_create()` to create
specified number of threads in the thread pool, and all threads will run the `thread_run()` function.

thread_run

While there is no task in the task queue, I use `pthread_cond_wait()` to make
threads sleep until it is signaled. When a thread is signaled, it will repeatedly get the first task in the task queue and execute it.
After a task is finished, it is deleted from the task queue.

async_run

Upon receiving a task, it is added to the end of task queue. And since a new task is arrived, I use `pthread_cond_signal()`
to wake up all threads that are waiting on the condition variable.

The environment of running my program

My Linux version is:

```
vagrant@csc3150:~$ cat /etc/issue
Ubuntu 16.04.7 LTS \n \l
```

My Linux kernel version is:

```
vagrant@csc3150:~$ uname -r
5.10.146
```

My gcc version is:

```
vagrant@csc3150:~$ 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.
```

Steps to execute my program

The following are the steps to execute the program:

1. Go to folder `thread_poll`.
2. In terminal, type `make all` to compile all programs.
3. Run httpserver with appropriate command line arguments.

Screenshot of my program output

```
Listening on port 8000...
Accepted connection from 127.0.0.1 on port 42166
Thread 139759599044160 will handle proxy request 0.
response thread 0 start to work
request thread 0 start to work
request thread 0 read failed, status 0
request thread 0 write failed, status 0
request thread 0 exited
Socket closed, proxy request 0 finished.
```

What did I learn from the tasks

In this task, I learned the following knowledge:

1. I have a deeper understanding about the thread pool and its detailed design.
2. I have a better understanding about mutex and how to use mutex to protect shared data between different threads.
3. I learned how to use pthread condition variables to provide another way for threads to synchronize. In detail, I learned
   how to use pthread_cond_wait() to block the calling thread until the specified condition is signalled, and how to use
   pthread_cond_signal() to wake up another thread which is waiting on the condition variable.