Aissgnment1 Report

Name: Zhang Yixuan

Student No: 120010058

How did I design my program?

There are four functions in my program: **moveleft**, **moveright**, **kbhit**, **logs_move**, and **main** function.

Before the functions, there are several definition of variables: **ROW**, **COLUMN**, **length_log**, **Node** as frog, **status** showing whether the game is running, **game_status** showing the result of the game, **map** as the river, **pthread_mutex_t mutex** and **log_mutex**, **pthread_cond_t gameover** (showing the game is over).

Game_status: 1 as win; -1 as lose; other value as quit.

Status: true as game is running, false as game is over.

In moveleft function, it implements the logs moving with left direction (when row = 1, 3, 5, 7, 9).

Moveleft: 2 conditions, 3 variables;

Front: first element of log

Rear: last element of log;

I: column coordinate;

Front > **rear**: front on the right of rear (divided log), within the position of log, **putchar** '=', other position, **putchar** ' ';

```
} else {
    //divided log
    if (i <= front && i > rear) {
        putchar('=');
    } else {
        putchar(' ');
    }
}
```

Front < **rear**: front on the left of rear (continuing log), within the position of log, **putchar** '=', other position, **putchar** ' ';

```
if (front < rear) {
    //continuing log
    if (i <= front || i > rear) {
        putchar('=');
    } else {
        putchar(' ');
    }
```

Moveright: 2 conditions, 3 variables;

Front: first element of log

Rear: last element of log;

I: column coordinate;

Front > **rear**: front on the right of rear (continuing log), within the position of log, **putchar** '=', other position, **putchar** ' ';

```
if (front > rear) {
    //divided log
    if (i < rear || i >= front) {
        putchar('=');
    } else {
        putchar(' ');
    }
```

Front < **rear**: front on the left of rear (divided log), within the position of log, **putchar** '=', other position, **putchar** ' ';

```
} else {
    //continuing log
    if (i >= front && i < rear) {
        putchar('=');
    } else {
        putchar(' ');
    }
```

In **kbhit** function, it is used to determine a keyboard is hit or not. If yes return 1. if not, return 0. It is used in **logs_move** function.

In **logs_move** function, some variables are defined. **Log_index** as which log, **front**, **rear**, **front_left**, **rear_left** as the first/last element of left/right moving logs.

While moving logs part, I use **while** to check the status. Within **while** loop, first use

pthread_mutex_lock(&log_mutex);

to lock each log in order to moving independently. Then try to print each row using for loop and

printf("\033[%iH\033[K", log_index + 1);

at each column position, check which row is the log in, 2, 4, 6, 8 using **moveright** function; 1, 3, 5, 7, 9 using **moveleft** function. Then **unlock** the **log_mutex**.

Then let frog move with logs. Checking condition is whether **frog.x == log_index**. In the row 2, 4, 6, 8, frog.y++ (moving right); in the row 1, 3, 5, 7, 9, frog.y-- (moving left).

```
if (frog.x == log_index) {
    if (log_index == 1 || log_index == 3 || log_index == 5 || log_index == 7 || log_index == 9) {
        frog.y--;
    } else if (log_index == 2 || log_index == 4 || log_index == 6 || log_index == 8) {
        frog.y++;
    }
}
```

Using **usleep** to wait for some time before next move.

Next check if logs are out of rage and let it rotate in one row. Let **front**++ **front_left**--, and when each becomes **COLUMN** and 0, correct its value to 0/**COLUMN**. The same thing is done on **rear**, **rear_left**.

The next part is operating part, checking keyboard hits, change frog's position or quit the game. First use **pthread_mutex_lock(&mutex)** to make the operating part independent (so that operating and auto moving of logs/frog will not influence each other). **Mutex** will unlock at the end. Check if **kbhit** is 1 and use **char ch** to store the input signal: **'w' as frog.x--**, **'s' as frog.x++**, **'a' as frog.y--**, **'d' as frog.y++**, 'q' as the end of game (**status = false** and **pthread_cond_signal(&gameover);**). In the first four conditions, frog cannot move out the map using the signal (ex. Cannot use 's' in the beginning).

```
pthread_mutex_lock(&mutex);
char ch;
if (kbhit()) {
    ch = getchar();
    if (ch == 'w' && frog.x > 0) {
        frog.x--;
    } else if (ch == 's' && frog.x < ROW) {
        frog.x++;
    } else if (ch == 'a' && frog.y >= 0) {
        frog.y--;
    } else if (ch == 'd' && frog.y <= COLUMN) {
        frog.y++;
    } else if (ch == 'q') {
        status = false;
        pthread_cond_signal(&gameover);
    }
}
```

Then lock **log_mutex** again to print river band and frog. Print river band is to fill in the position where frog leave on the band. Frog is printed according to **frog.x+1** (correct because if without 1, frog will start above original place), and frog.y.

The following part is to check if frog is out of range and the player fails. First is out of range on left/right side.

```
//check if frog is out of the left/right side
if (frog.y < 0 || frog.y > COLUMN) {
    status = false;
    game_status = -1;
    pthread_cond_signal(&gameover);
}
```

Next is in four conditions: **moving left/right, divided/continuing log**. Check when frog is on one of the logs instead of on river band. If the frog is out of the range of logs, player fails.

Moving left, divided log: frog.y < front_left + 2 && frog.y >= rear_left + 2

Moving left, continuing log: frog.y >= front_left + 3 || frog.y < rear_left + 3

Moving right, continuing log: frog.y <= rear && frog.y > front

Moving right, divided log: frog.y < front || frog.y >= rear

The constant after **front_left** and **rear_left** is to correct the error when checking frog place.

If one of the four conditions is checked, send the **status** signal as **false** to end the game and **game_status** as -1 to output fail content.

Then check the success result, that is, when **frog.x == 0**. Send **status** as **false**, **game_status** as 1 to output success content.

Unlock the log_mutex;

Unlock the mutex;

Here is the end of logs_move function.

In the main function, first set ROW amount thread and initialize mutex (**mutex** and **log_mutex**) and condition (**gameover**). The next part is the initialization of map. After printing the map, create pthreads for wood move and frog control. Create threads from i = 1 to i = ROW-1 as the index of logs.

```
for (int i = 1; i < ROW; i++) {
    rc = pthread_create(&thread[i], NULL, logs_move, (void*) i);
    if (rc) {
        printf("ERROR: return code from pthread_create() is %d", rc);
        exit(1);
    }
}
```

After the running of **logs_move** function, use **pthread_cond_wait(&gameover, &mutex);** to wait for the gameover signal.

Finally, check the value of game_status to print out correct output.

```
if (game_status == -1) {
    printf("You lose the game!!\n");
} else if (game_status == 1) {
    printf("You win the game!!\n");
} else {
    printf("You exit the game.\n");
}
```

The environment of running your program

Version: `Ubuntu 16.04.7 LTS \n \l`

Linux kernel version: 5.10.145

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.
```

The steps to execute your program

Use **g++ hw2.cpp -lpthread** to compile cpp file.

Use **./a.out** to run the program.

Screenshot of your program output:

What did you learn from the tasks?

How to create multi thread to implement process. How to use multi thread to run different parts in a process, especially in games. How to divide a process into small segments to simplify the implements of it. The usage of lock and unlock of mutex in using multi threads to implement a process. How to send signals of condition to the process (as the end of multi threads). How to use \033 orders in printf to clean and print the content I want. How to ubuntu version and linux kernel version. How to compile a file with multi thread using pthread function. How to implement pthread function in pthread.h and the conditions to implement them.