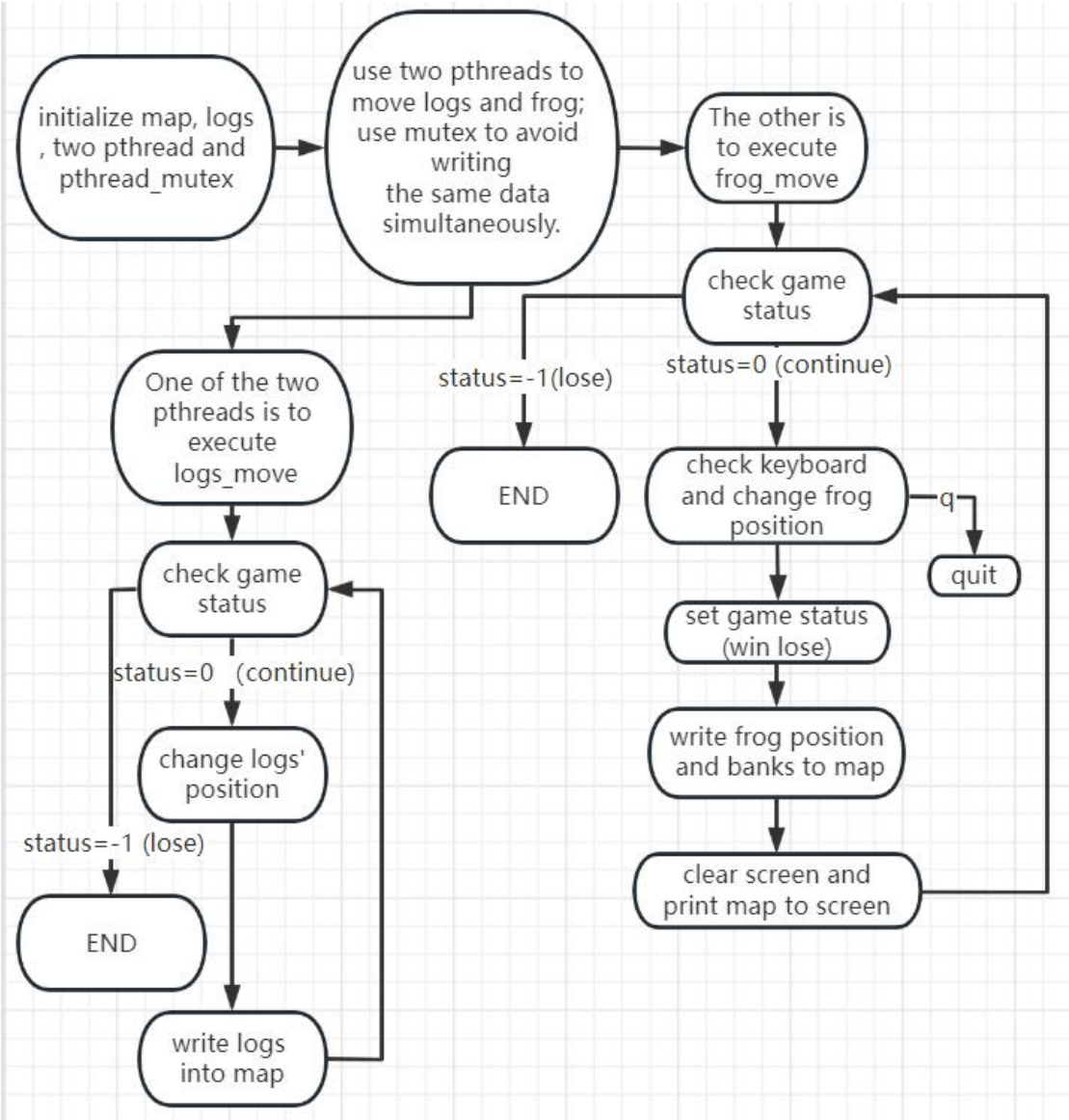
Report_CSC3150_Assignment2

121090406 Tengfei Ma

I have not finished bonus.

1. How I design my program

Main idea:



(1) Initialize map

① Clear map, set all elements as ‘ ‘

1)

```
memset( map , 0, sizeof( map ) ) ;
int i , j ;
for( i = 1; i < ROW; ++i ){
    for( j = 0; j < COLUMN - 1; ++j )
        map[i][j] = ' ' ;
}
```

② Write banks to map

1)
```
for( j = 0; j < COLUMN - 1; ++j ) map[ROW][j] = map[0][j] = '|' ;
for( j = 0; j < COLUMN - 1; ++j ) map[0][j] = map[0][j] = '|' ;
```

③ Write frog to map

1)
```
frog = Node( ROW, (COLUMN-1) / 2 ) ;
map[frog.x][frog.y] = '0' ;
```

④ Initialize logs

1) Create a 9*3 matrix to record 9 logs' status, like head position, tail position and moving direction

a.
```
int logs[9][3];// 9 logs with 3 parameters.
// logs[i][0] means head position
// logs[i][1] means tail position
// logs[i][2] means direction, -1 for left, 1 for right
```

a) logs is a global array.

2) Set every log's head position randomly, and set tail position such that the length of log is 15, and set moving direction as left and right alternately

a.
```
for (int log_id = 0; log_id < 9; log_id++)
{
    int length = 15;
    int head = rand() % 48;
    int tail = head + length - 1;
    tail = (tail + COLUMN-1) % (COLUMN-1);
    if (log_id % 2 == 1)
    {
        logs[log_id][0] = head;
        logs[log_id][1] = tail;
        logs[log_id][2] = 1;
    }
    else
    {
        logs[log_id][0] = head;
        logs[log_id][1] = tail;
        logs[log_id][2] = -1;
    }
}
```

3) Write logs to map

a.
```
writeLogsToMap();
```

b.
```
void writeLogsToMap()
{
    pthread_mutex_lock(&map_mutex);
    for (int i = 0; i < 9; i++)// 9 logs in total
    {
        if (logs[i][0] < logs[i][1])//head position < tail position
        {
            for (int col = 0; col < COLUMN-1; col++)
            {
                if (col >= logs[i][0] && col <= logs[i][1]) map[i+1][col] = '=';
                else map[i+1][col] = ' ';
            }
        }
        else
        {
            for (int col = 0; col < COLUMN-1; col++)
            {
                if (col > logs[i][1] && col < logs[i][0]) map[i+1][col] = ' ' ;
                else map[i+1][col] = '=';
            }
        }
    }
    pthread_mutex_unlock(&map_mutex);
}
```

a) Use pthread_mutex_lock to prevent many threads writing map simultaneously.

(2) Initialize pthreads for log move and frog move.

① Initialize mutual exclusion

1)
```
pthread_mutex_t frog_mutex;
pthread_mutex_t map_mutex;
pthread_mutex_t log_mutex;
```
This is global variable.

2)
```
pthread_mutex_init(&frog_mutex, NULL);
pthread_mutex_init(&log_mutex, NULL);
pthread_mutex_init(&map_mutex, NULL);
```

② Create pthreads to execute frog_move and logs_move

1)
```
void *unused;
pthread_t pthread_log;
pthread_t pthread_frog;
pthread_create(&pthread_frog, NULL, frog_move, unused);
pthread_create(&pthread_log, NULL, logs_move, unused);
```

③ Use pthread_join to wait for pthreads to finish.

1)
```
pthread_join(pthread_frog, NULL);
pthread_join(pthread_log, NULL);
```

(3) logs_move

①
```
while(status == 0)// status is continue
{
    usleep(100000);
    //Move the logs
    for (int log_id = 0; log_id < 9; log_id++)
    {
        if(logs[log_id][2] == -1)//-1 means left, log moves to left
        {
            logs[log_id][0] = ((logs[log_id][0] - 1) + COLUMN-1) % (COLUMN-1);
            logs[log_id][1] = ((logs[log_id][1] - 1) + COLUMN-1) % (COLUMN-1);
        }
        else// log moves to right
        {
            logs[log_id][0] = ((logs[log_id][0] + 1)) % (COLUMN-1);
            logs[log_id][1] = ((logs[log_id][1] + 1)) % (COLUMN-1);
        }
        if (frog.x == log_id+1)//frog on the log
        {
            if (logs[log_id][2] == -1) frogLeft();//frog moves with the log
            else frogRight();
        }
    }
    writeLogsToMap();// write logs to map
}
pthread_exit(NULL);
```

1) First check if "status" is 0 (continue)

a.
```
while(status == 0)// status is continue
```

b. "status" is a global variable.

c.
```
int status = 0;//0 for continue, 1 for win, -1 for Lose, 2 for qui
```

2) Move logs and write the logs' position to map.

3) If there is a frog on certain log, the frog is moved with the log.

(4) frog_move

① Check if "status" is 0 "continue"

1)
```
while(status == 0)
```

② Check keyboard and change frog's position

1)

2)

```c
if(kbhit())// check keyboard hits, to change frog's position or quit game.
{
    char moveDirection = getchar();
    switch(moveDirection)
    {
        case 'W': case 'w':
        {
            frogUp();
            break;
        }
        case 'A': case 'a' :
        {
            frogLeft();
            break;
        }
        case 'S': case 's':
        {
            if (frog.x == ROW) continue;
            frogDown();
            break;
        }
        case 'D': case 'd' :
        {
            frogRight();
            break;
        }
        case 'Q': case 'q':
        {
            status = 2;//quit
            puts("\033[H\033[2J");//clear screen
            pthread_exit(NULL);
        }
    }
}
```

a. Notice: typing “q” makes game over directly

③ Set status and check if status is -1 (lose)

1)

```c
setStatus();// set game status according to frog's position
if (status == -1) pthread_exit(NULL);//lose
```

2)

```
void setStatus()// check game's status
{
    int frogX = frog.x;
    int frogY = frog.y;
    pthread_mutex_lock(&map_mutex);
    if (frogX == 0) // reach the oppposite river bank
    {
        status = 1;// win
        pthread_mutex_unlock(&map_mutex);
        return;
    }

    if (map[frogX][frogY] == ' ') // Fall in to river
    {
        status = -1;//lose
        pthread_mutex_unlock(&map_mutex);
        return;
    }

    if (frogY < 0 || frogY > COLUMN-2) //cross left or right boundary
    {
        status = -1;//lose
        pthread_mutex_unlock(&map_mutex);
        return;
    }
    pthread_mutex_unlock(&map_mutex);
}
```

a. If frog reaches the opposite bank, status is 1(win).
b. If frog is at ‘ ‘, frog fall into water and status is -1(lose).
c. If frog cross left or right boundary, status = -1(lose).

3) Write frog’s position and banks to map

a.

```
writeFrogAndBanksToMap();// write frog and banks to map
```

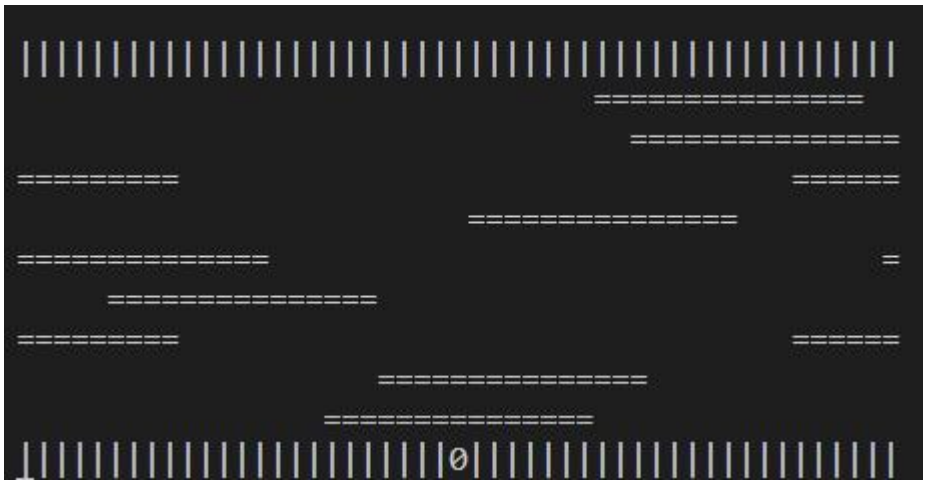
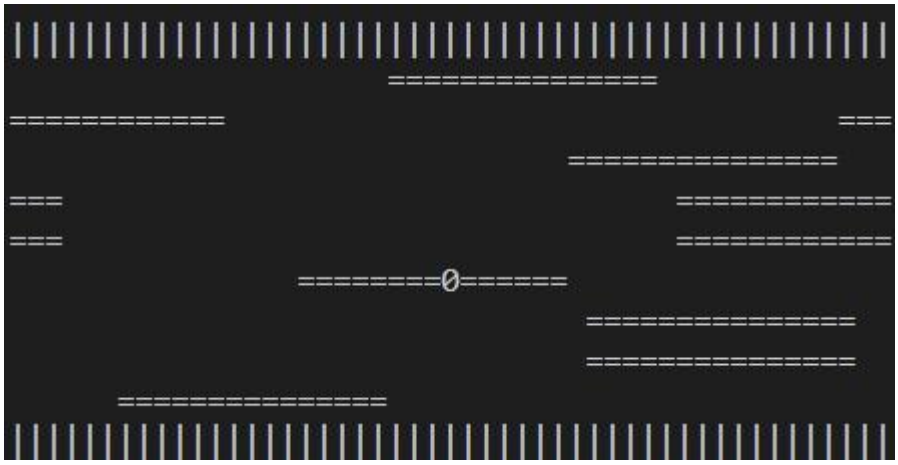
b.

```
void writeFrogAndBanksToMap()
{
    pthread_mutex_lock(&map_mutex);
    for(int j = 0; j < COLUMN - 1; ++j) map[ROW][j] = '|' ;//write lower ban
    for(int j = 0; j < COLUMN - 1; ++j) map[0][j] = '|' ;//write higher bank
    map[frog.x][frog.y] = '0';//write position of frog to map
    pthread_mutex_unlock(&map_mutex);
}
```

4) Clear screen and print map into screen

a.

```
puts("\033[H\033[2J");//clear screen
for(int i = 0; i <= ROW; ++i) puts(map[i]);// Print the map into scree
```

2. Environment
   (1) Linux kernel version is 5.10.195, Ubuntu 16.04
   (2) GCC 5.4.0
3. The steps to execute my program
   (1) Compile: make frog
      ①
```
$ make frog
```
   (2) Execute: ./a.out
      ①
```
$ ./a.out
```
   (3) You can use “make clean” to remove “a.out”
      ①
```
$ make clean
```
4. Output
   (1) Start
      ①

   (2) In the process
      ①

   (3) If you win
      ①
```
You win the game!!
vagrant@csc3150:~/csc3150/Assignment_2_121090406/CSC3150-A2-template/source$
```

(4) If you lose

①
```
You lose the game!!
vagrant@csc3150:~/csc3150/Assignment_2_121090406/CSC3150-A2-template/source$
```

(5) If you quit

①
```
You exit the game.
vagrant@csc3150:~/csc3150/Assignment_2_121090406/CSC3150-A2-template/source$
```

5. What I learn
   (1) I learn how to use multi-thread to implement a game and understand how multi-thread works.
   (2) I can use mutual exclusion to write data, which can avoid writing the same data simultaneously.
   (3) I learn how to monitor and receive keyboard input.
   (4) I learn use pthread_create to create many threads.
   (5) I learn use pthread_join to wait for threads to finish.