

# Operating System (CSC 3150)

## Tutorial 5

*LeLe Li*

*E-mail: 221019053@link.cuhk.edu.cn*

# Content

In this tutorial, we will practice related functions for Assignment 2.

Functions you may required:

- Keyboard Hit
- Terminal Control
- Suspend the executing thread
- Hints for the game “Frog crosses river”
- Pthread conditional wait and signal (Bonus Task)

# Keyboard Hit

- In Assignment 2, we've provided a similar function named `int kbhit(void)`, you could use it directly.
- `int kbhit(void)`, If a key has been pressed then it returns a non zero value, otherwise it returns zero.
- The termios general terminal interface provides an interface to asynchronous communications devices.
  - <http://man7.org/linux/man-pages/man3/termios.3.html>
  - <http://man7.org/linux/man-pages/man2/fcntl.2.html>

# Keyboard Hit

W, A, S,D : up, left, right, down (4 directions), Q (Quit the game)

```
int kbhit(void){
    struct termios oldt, newt;
    int ch;
    int oldf;

    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);

    if(ch != EOF)
    {
        ungetc(ch, stdin);
        return 1;
    }
    return 0;
}

int main( int argc, char *argv[] ){

    int isQuit = 0;
    while (!isQuit){
        if( kbhit() ){
            char dir = getchar() ;
            if( dir == 'w' || dir == 'W' )
                printf ("UP Hit!\n");
            if( dir == 'a' || dir == 'A' )
                printf ("LEFT Hit!\n");
            if( dir == 'd' || dir == 'D' )
                printf ("RIGHT Hit!\n");
            if( dir == 's' || dir == 'S' )
                printf ("DOWN Hit!\n");
            if( dir == 'q' || dir == 'Q' ){
                printf("Quit!\n");
                isQuit= 1;
            }
        }
    }
    return 0;
}
```



```
Terminal
[10/15/18]seed@VM:~/.../Khit$ gcc kbhit.c
[10/15/18]seed@VM:~/.../Khit$ ./a.out
wUP Hit!
DOWN Hit!
aLEFT Hit!
RIGHT Hit!
LEFT Hit!
UP Hit!
dRIGHT Hit!
sDOWN Hit!
wUP Hit!
aLEFT Hit!
aLEFT Hit!
aLEFT Hit!
aLEFT Hit!
LEFT Hit!
aLEFT Hit!
aLEFT Hit!
LEFT Hit!
Quit!
[10/15/18]seed@VM:~/.../Khit$
```

# Keyboard Hit

- `int getchar(void)` function is used to get/read a character from keyboard input.
- `int putchar(int char)` function is a file handling function which is used to write a character on standard output/screen.
- `int puts(const char *str)` writes a string to stdout up to but not including the null character. A newline character is appended to the output.
- In Assignment 2, you may use above functions to complete your keyboard read and map write.

# Terminal Control

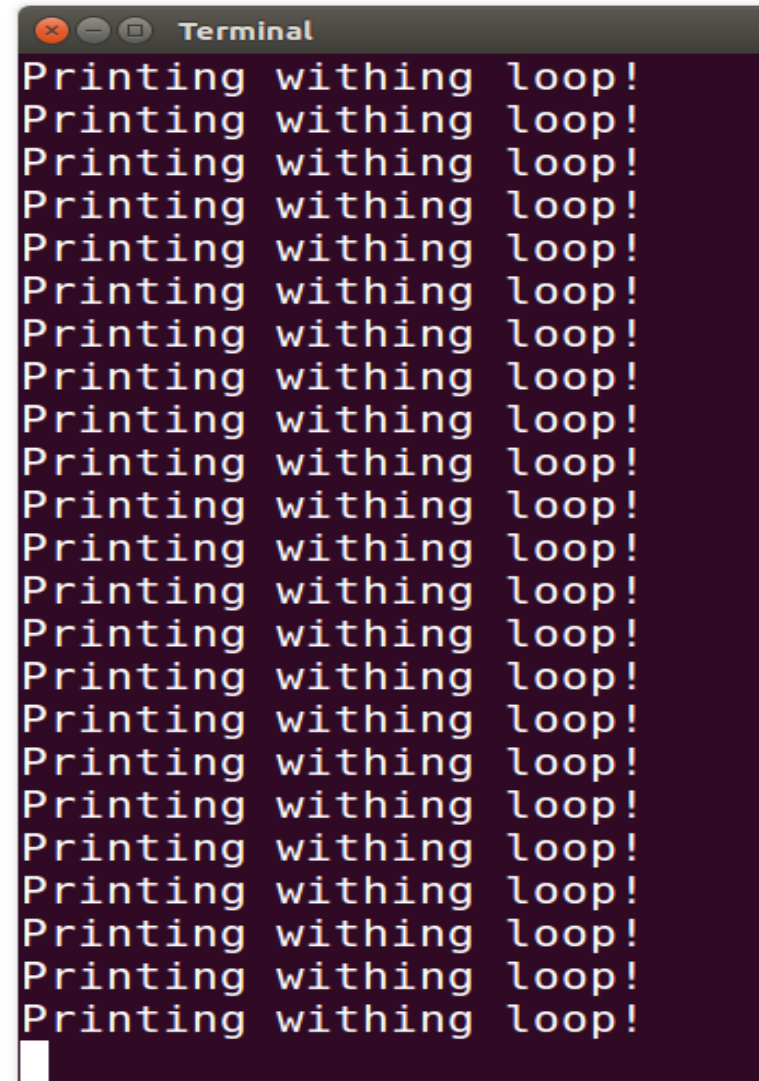
- When printing the message, you could use “\033” to control the **cursor in terminal**.
  - <https://www.student.cs.uwaterloo.ca/~cs452/terminal.html>

Code	Effect
"\033[2J"	Clear the screen.
"\033[H"	Move the cursor to the upper-left corner of the screen.
"\033[r;cH"	Move the cursor to row <i>r</i> , column <i>c</i> . Note that both the rows and columns are indexed starting at 1.
"\033[?251"	Hide the cursor.
"\033[K"	Delete everything from the cursor to the end of the line.

## Terminal Control

Move the cursor to the upper-left corner of the screen and clear the screen

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6
7 int main( int argc, char *argv[] ){
8     int isStop = 0;
9     while(!isStop)
10     {
11         printf("Printing withing loop!\n");
12         //printf("\033[0;0H\033[2J");
13     }
14 }
```


A terminal window titled "Terminal" with a dark background. It displays 20 lines of the text "Printing withing loop!" stacked vertically. The cursor is visible at the bottom left of the terminal window.

```
Terminal
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
Printing withing loop!
```

# Terminal Control

Move the cursor to the upper-left corner of the screen and clear the screen

```
TerminalControl.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6
7 int main( int argc, char *argv[] ){
8
9     int isStop = 0;
10
11     while(!isStop)
12     {
13         printf("Printing withing loop!\n");
14         printf("\033[H\033[2J");
15
16     }
17
18
19 }
```

A terminal window titled "Terminal" with a dark background. It displays the text "Printing withing loop!" in white. The rest of the terminal is empty, showing the dark background color.



# Suspend executing thread with three functions

- `int usleep(useconds_t usec)` suspends execution for microsecond intervals.
- The `usleep()` function suspends execution of the calling thread for (at least) usec **microseconds**.
- The sleep may be lengthened slightly by any system activity or by the time spent processing the call or by the granularity of system timers.
- The `usleep()` function returns 0 on success. On error, -1 is returned, with `errno` set to indicate the cause of the error.
- The `sleep()` function accepts time in **seconds** while `usleep()` accepts in microseconds.
- The `nanosleep()` allows the user to specify the sleep period with **nanosecond precision**.

```
8  int main(int argc, char* argv[])
9  {
10     printf("Sleep program for 5 seconds\n");
11     sleep(5);
12
13     printf("Sleep program for 1000000 micro seconds (1 second)\n");
14     usleep(1000000);
15
16     printf("Program finished\n");
17
18     return 0;
19 }
```

## Output

/tmp/PljkFvLBgA.o

Sleep program for 5 seconds

Sleep program for 1000000 micro seconds (1 second)

Program finished

# Assignment 2 Hints

- The river is regarded as a **map**. You could create multiple threads to control the map printing (**logs position**).
- Use the **mutex\_lock** when reading and writing the **shared data like the map**.
- You could set suspend for **screen printing(rendering)**, to control the logs moving speed.
- You could use **srand() and rand()** to set the random interval among logs moving or the lengths of the logs.
- You could set flags to check game status.

## rand() and srand() in C

### ◆ What is rand() ?

The function rand() is used to generate the **pseudo random number**. It returns an integer value and its range is from 0 to RAND\_MAX.

#### Parameters

The function accepts no parameter(s) –

#### Return value

This function returns an integer value between 0 to RAND\_MAX.

Like we want to generate a random number between 1-6 then we use this function :

```
Num = rand() % 6 + 1;
```

### ◆ What is srand()?

srand() is used to initialise random number generators. **The argument is passed as a seed for generating a pseudo-random number**. Whenever a different seed value is used in srand the pseudo number generator can be expected to generate different series of results the same as rand().

<https://www.tutorialspoint.com/rand-and-srand-in-c-cplusplus>

[https://www.tutorialspoint.com/compile\\_c\\_online.php](https://www.tutorialspoint.com/compile_c_online.php)

# Pthread conditional wait and signal

- **Condition variables** provide yet another way for threads to synchronize.

Condition variables allow threads to wait until some event or condition has occurred.

A condition variable has attributes that specify the characteristics of the condition.

Typically, a program uses the following objects:

- **A boolean variable, indicating whether the condition is met**
  - **A mutex to serialize the access to the boolean variable**
  - **A condition variable to wait for the condition**
- A condition variable is always used in conjunction with a mutex lock.

# Pthread condition - signals

- Condition variables must be declared with type: `pthread_cond_t`
  - `pthread_cond_t` (defined in “sys/types.h”)
- Condition variables must be initialized before it is used:
  - `int pthread_cond_init(pthread_cond_t *, const pthread_condattr_t *);`
- Condition variables should be freed if it is no longer used:
  - `int pthread_cond_destroy(pthread_cond_t *);`

# Pthread condition - signals

- Pthread condition routines:
  - `int pthread_cond_wait(pthread_cond_t *, pthread_mutex_t *);`
  - `int pthread_cond_signal(pthread_cond_t *);`
  - `int pthread_cond_broadcast(pthread_cond_t *);`

# Pthread conditional wait and signal

- **pthread\_cond\_wait()** blocks the calling thread until **the specified condition** is signalled. This routine should be called **while mutex is locked**, and it will **automatically release the mutex while it waits**.
- **The pthread\_cond\_signal()** routine is used **to signal (or wake up) another thread which is waiting** on the condition variable. It should be called after mutex is locked, and must unlock mutex in order for pthread\_cond\_wait() routine to complete.
- **The pthread\_cond\_broadcast()** routine should be used instead of pthread\_cond\_signal() if more than one thread is in a blocking wait state.



# Example

```
13 pthread_cond_t cond;
14 pthread_mutex_t mutex;
15
16 int footprint = 0;
17
18 void *thread_1_fun(void *arg) {
19     time_t T;
20
21     if (pthread_mutex_lock(&mutex) != 0) {
22         perror(s: "pthread_mutex_lock() error");
23         exit(status: 6);
24     }
25     time(&T);
26     printf(format: "Thread-1: starting wait at %s", ctime(&T));
27     footprint++;
28     printf(format: "Thread-1: incrementing the variable footprint by 1, footprint=%d\n", footprint);
29     if (pthread_cond_wait(&cond, &mutex) != 0) {
30         perror(s: "pthread_cond_timedwait() error");
31         exit(status: 7);
32     }
33     time(&T);
34     printf(format: "Thread-1: wait over at %s", ctime(&T));
35 }
```

```
38 pthread_t thread_1;
39 time_t T;
40 struct timespec t;
41
42 if (pthread_mutex_init(&mutex, mutexattr: NULL) != 0) {
43     perror(s: "pthread_mutex_init() error");
44     exit(status: 1);
45 }
46
47 if (pthread_cond_init(&cond, cond_attr: NULL) != 0) {
48     perror(s: "pthread_cond_init() error");
49     exit(status: 2);
50 }
51
52 if (pthread_create(&thread_1, attr: NULL, thread_1_fun,
53     perror(s: "pthread_create() error");
54     exit(status: 3);
55 }
62 puts(s: "Thread-main: is about ready to release the thread");
63 sleep(seconds: 2);
64
65 if (pthread_cond_signal(&cond) != 0) {
66     perror(s: "pthread_cond_signal() error");
67     exit(status: 4);
68 }
69
70 if (pthread_join(thread_1, thread_return: NULL) != 0) {
71     perror(s: "pthread_join() error");
72     exit(status: 5);
73 }
```

# References

- <https://www.baeldung.com/cs/async-vs-multi-threading>
- <https://www.javatpoint.com/process-vs-thread>
- <https://www.geeksforgeeks.org/multithreading-c-2/>
- <https://www.geeksforgeeks.org/condition-wait-signal-multi-threading/>
- [http://www.cs.unibo.it/~ghini/didattica/sistop/pthreads\\_tutorial/POSIX\\_Threads\\_Programming.htm](http://www.cs.unibo.it/~ghini/didattica/sistop/pthreads_tutorial/POSIX_Threads_Programming.htm)
- <https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/>

## **Next Week's Tutorial (Tutorial 6):**

- **Review of Pthread**
- **Q&A(Question and Answer)**

Thank you