Operating System (CSC 3150)

Tutorial 4 – Part 1

KAI SHEN

OFFICE HOUR: WED 9PM-10PM @ ZHI XIN 101

E-MAIL: 118010254@LINK.CUHK.EDU.CN

Target

In this tutorial, we will practice pthread programming.

- Thread
- Thread and Process
- Pthread creation
- Pthread termination
- Pthread join
- Pthread mutex
- Pthread condition

Thread

- Technically, a thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system.

- To the software developer, the concept of a "procedure" that runs independently from its main program may best describe a thread.

- Pthread: POSIX Thread, a standard-based thread API for C.
- other options: openMP, std::thread

- When compiling Pthread in gcc/g++, should add option “-lpthread”.
  - Compile: gcc test.c –lpthread or g++ test.cpp -lpthread
  - Execution: ./a.out

Thread and Process

- A process is created by the operating system. Processes contain information about program resources and program execution state。

- Threads use and exist within these process resources, yet are able to be scheduled by the operating system and run as independent entities within a process.

- A process can have multiple threads, all of which share the resources within a process and all of which execute within the same address space. Within a multi-threaded program, there are at any time multiple points of execution.

Pthread creation – side thread

- pthread_create:
  - int pthread_create(pthread_t *thread,
    const pthread_attr_t *attr,
    void *(*start_routine) (void *),
    void *arg);

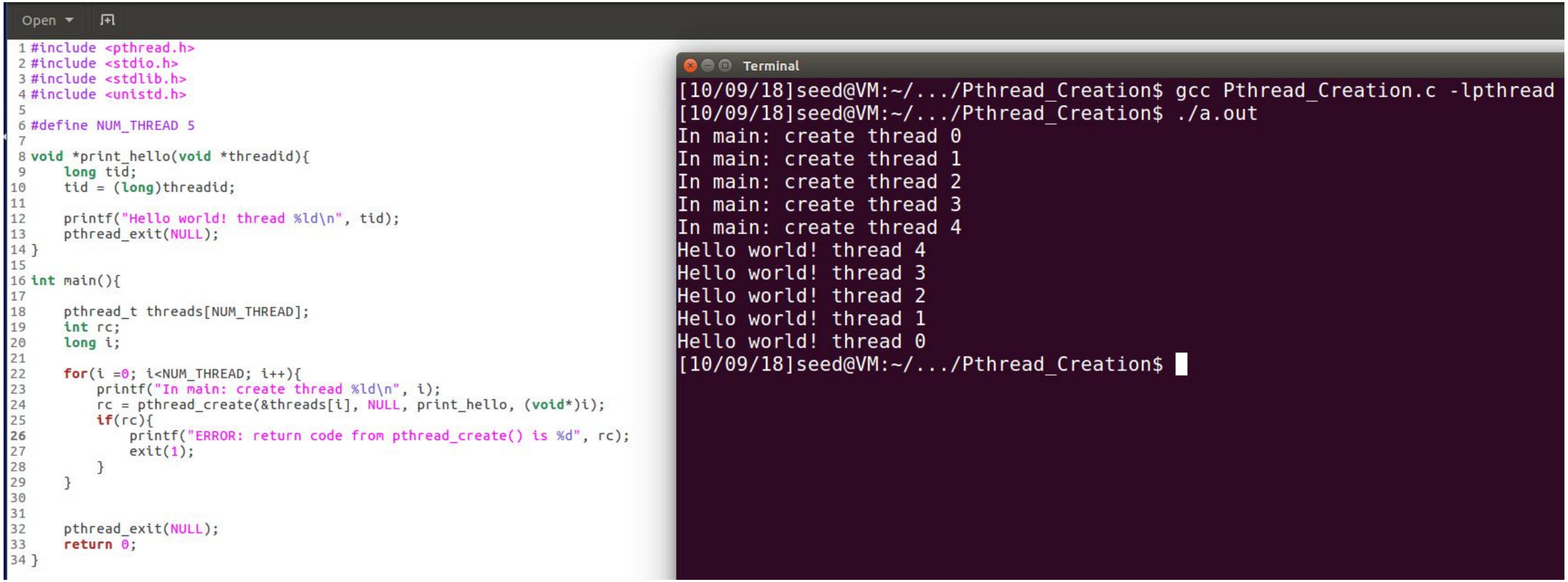
- This routine creates a new thread and makes it executable. Typically, threads are first created from within main() inside a single process. Once created, threads are peers, and may create other threads.
- The attr parameter is used to set thread attributes. You can specify a thread attributes object, or NULL.
- The start_routine is the C routine that the thread will execute once it is created.

Pthread creation - side thread

- Return value
  - On success, pthread_create() returns 0;
  - On error, it returns an error number, and the contents of *thread are undefined.

- Pthread is declared with type:
  - pthread_t (defined in “sys/types.h”)

Pthread creation - side thread



```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NUM_THREAD 5

void *print_hello(void *threadid){
    long tid;
    tid = (long)threadid;

    printf("Hello world! thread %ld\n", tid);
    pthread_exit(NULL);
}

int main(){

    pthread_t threads[NUM_THREAD];
    int rc;
    long i;

    for(i =0; i<NUM_THREAD; i++){
        printf("In main: create thread %ld\n", i);
        rc = pthread_create(&threads[i], NULL, print_hello, (void*)i);
        if(rc){
            printf("ERROR: return code from pthread_create() is %d", rc);
            exit(1);
        }
    }


    pthread_exit(NULL);
    return 0;
}
```

```
[10/09/18]seed@VM:~/.../Pthread_Creation$ gcc Pthread_Creation.c -lpthread
[10/09/18]seed@VM:~/.../Pthread_Creation$ ./a.out
In main: create thread 0
In main: create thread 1
In main: create thread 2
In main: create thread 3
In main: create thread 4
Hello world! thread 4
Hello world! thread 3
Hello world! thread 2
Hello world! thread 1
Hello world! thread 0
[10/09/18]seed@VM:~/.../Pthread_Creation$
```

Pthread termination – main vs side

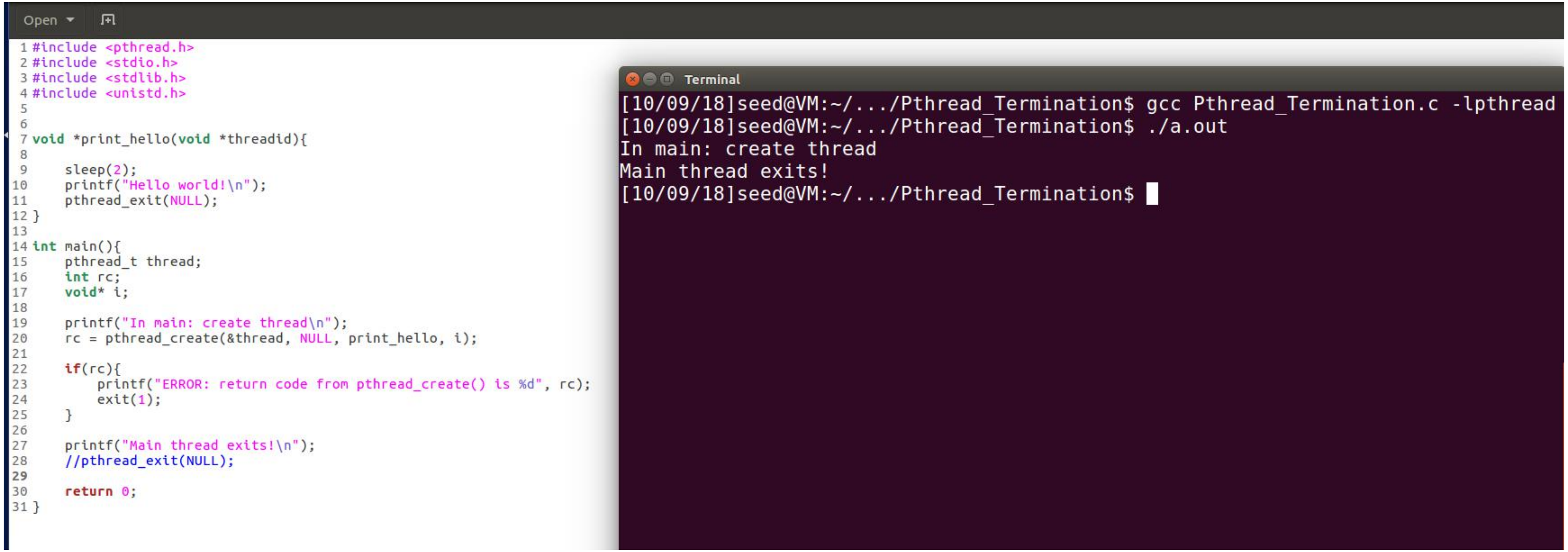
- pthread_exit:
  - void pthread_exit(void *retval);

- This routine is used to explicitly exit a thread. Typically, the pthread_exit() routine is called after a thread has completed its work and is no longer required to exist.

- If main() finishes before the threads it has created, and exits with pthread_exit(), the other threads will continue to execute. Otherwise, they will be automatically terminated when main() finishes.

- Recommendation: Use pthread_exit() to exit from all threads...especially main().

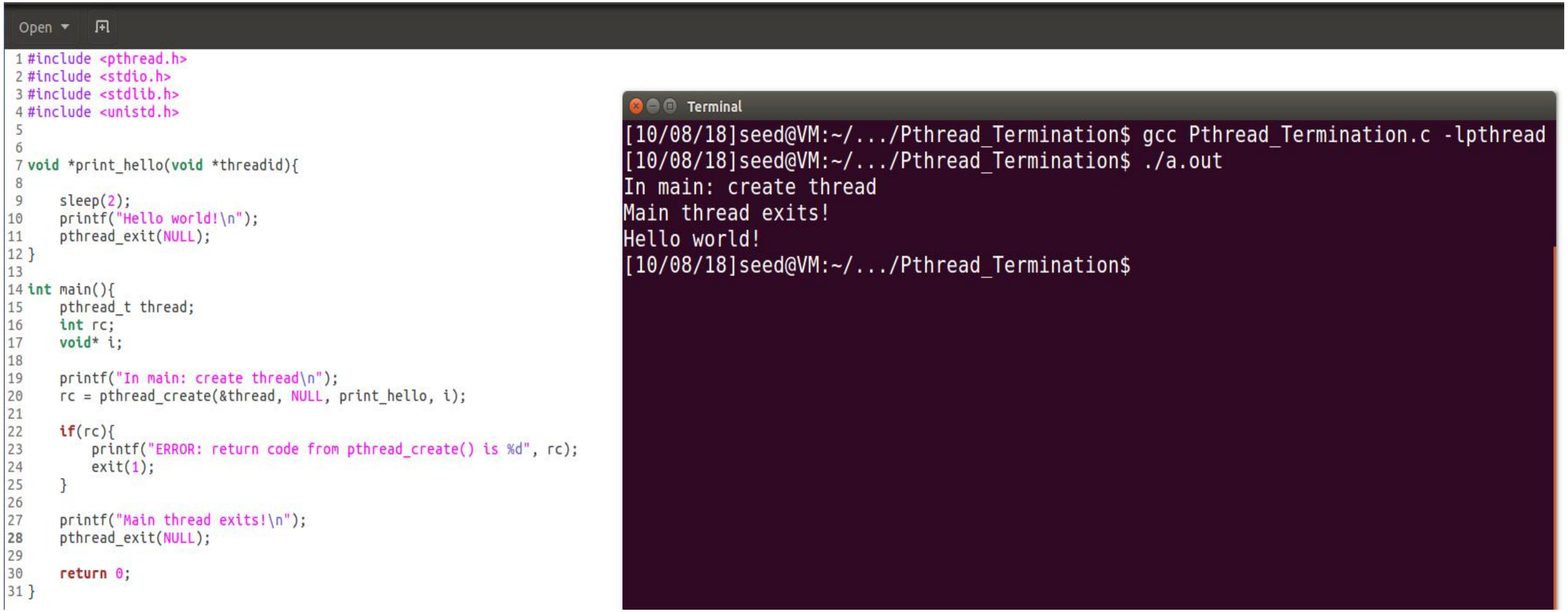
Pthread termination – main vs side



```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


void *print_hello(void *threadid){

    sleep(2);
    printf("Hello world!\n");
    pthread_exit(NULL);
}

int main(){
    pthread_t thread;
    int rc;
    void* i;

    printf("In main: create thread\n");
    rc = pthread_create(&thread, NULL, print_hello, i);

    if(rc){
        printf("ERROR: return code from pthread_create() is %d", rc);
        exit(1);
    }

    printf("Main thread exits!\n");
    //pthread_exit(NULL);

    return 0;
}
```

```
[10/09/18]seed@VM:~/.../Pthread_Termination$ gcc Pthread_Termination.c -lpthread
[10/09/18]seed@VM:~/.../Pthread_Termination$ ./a.out
In main: create thread
Main thread exits!
[10/09/18]seed@VM:~/.../Pthread_Termination$ 
```

Pthread termination – main vs side



```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


void *print_hello(void *threadid){

    sleep(2);
    printf("Hello world!\n");
    pthread_exit(NULL);
}

int main(){
    pthread_t thread;
    int rc;
    void* i;

    printf("In main: create thread\n");
    rc = pthread_create(&thread, NULL, print_hello, i);

    if(rc){
        printf("ERROR: return code from pthread_create() is %d", rc);
        exit(1);
    }

    printf("Main thread exits!\n");
    pthread_exit(NULL);

    return 0;
}
```

```
[10/08/18]seed@VM:~/.../Pthread_Termination$ gcc Pthread_Termination.c -lpthread
[10/08/18]seed@VM:~/.../Pthread_Termination$ ./a.out
In main: create thread
Main thread exits!
Hello world!
[10/08/18]seed@VM:~/.../Pthread_Termination$
```

Pthread join - synchronization

- pthread_join:
  - int pthread_join(pthread_t thread, void **retval);

- "Joining" is one way to accomplish synchronization between threads.

- The pthread_join() subroutine blocks the calling thread until the specified threadid thread terminates.

- The programmer is able to obtain the target thread's termination return status if specified through pthread_exit(), in the status parameter.

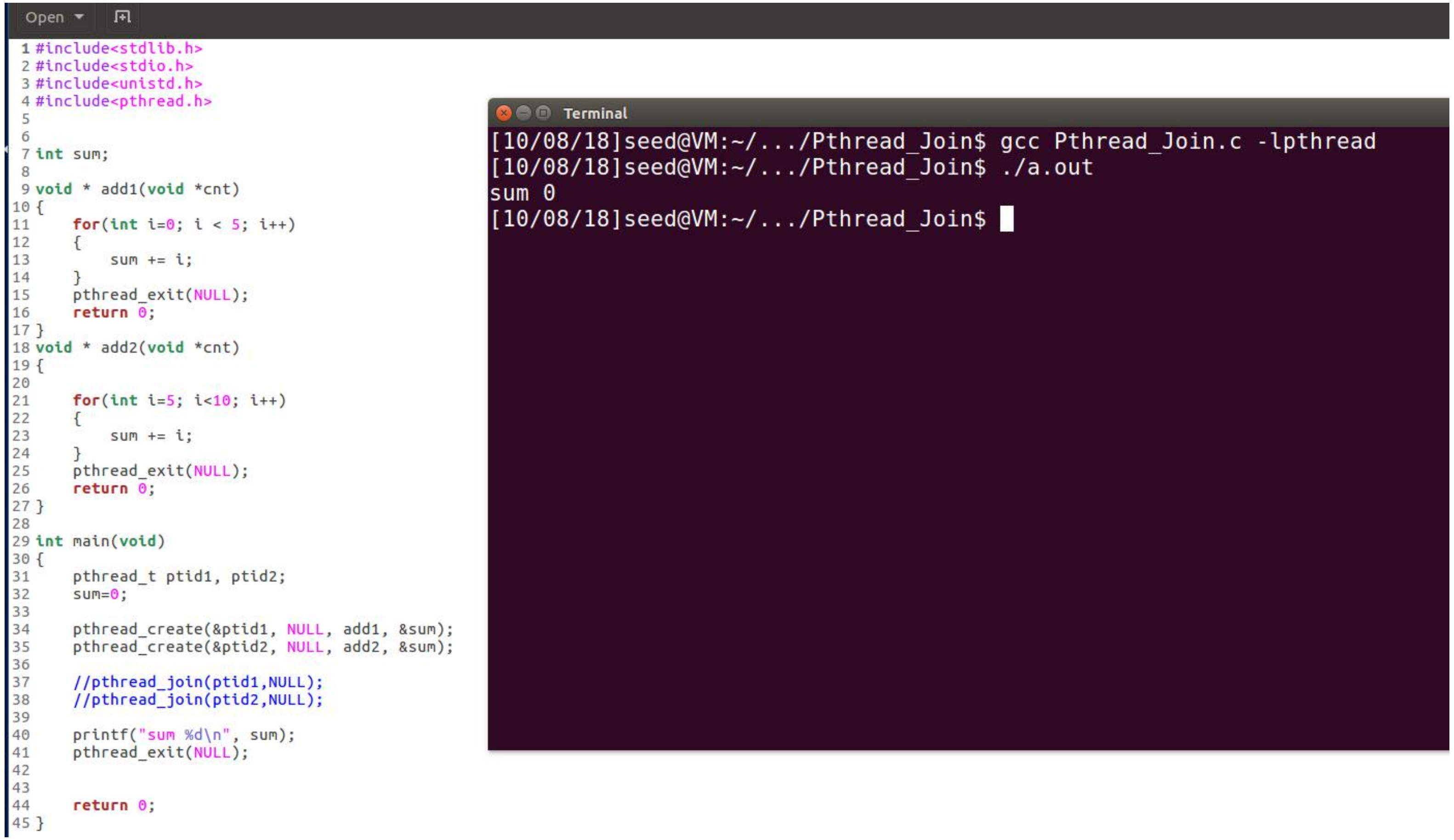
Pthread join - synchronization

- Return value
  - On success, pthread_join() returns 0;
  - On error, it returns an error number.

- It is impossible to join a detached thread.

- When a thread is created, one of its attributes defines whether it is joinable or detached. Detached means it can never be joined. (PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE)

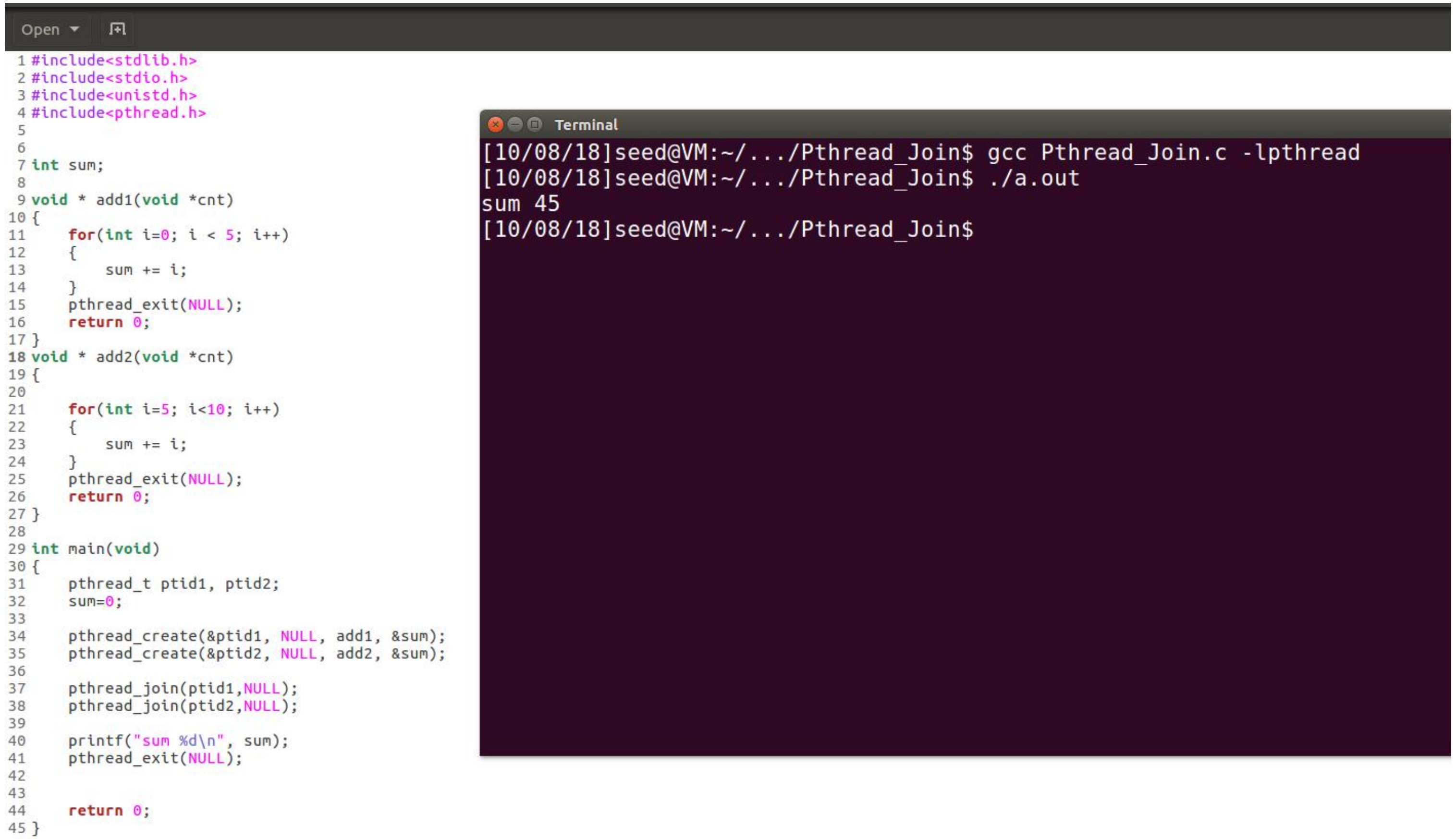
Pthread join - synchronization



```c
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>


int sum;

void * add1(void *cnt)
{
    for(int i=0; i < 5; i++)
    {
        sum += i;
    }
    pthread_exit(NULL);
    return 0;
}
void * add2(void *cnt)
{

    for(int i=5; i<10; i++)
    {
        sum += i;
    }
    pthread_exit(NULL);
    return 0;
}

int main(void)
{
    pthread_t ptid1, ptid2;
    sum=0;

    pthread_create(&ptid1, NULL, add1, &sum);
    pthread_create(&ptid2, NULL, add2, &sum);

    //pthread_join(ptid1,NULL);
    //pthread_join(ptid2,NULL);

    printf("sum %d\n", sum);
    pthread_exit(NULL);


    return 0;
}
```

```
[10/08/18]seed@VM:~/.../Pthread_Join$ gcc Pthread_Join.c -lpthread
[10/08/18]seed@VM:~/.../Pthread_Join$ ./a.out
sum 0
[10/08/18]seed@VM:~/.../Pthread_Join$
```

Pthread join - synchronization



```c
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>


int sum;

void * add1(void *cnt)
{
    for(int i=0; i < 5; i++)
    {
        sum += i;
    }
    pthread_exit(NULL);
    return 0;
}
void * add2(void *cnt)
{

    for(int i=5; i<10; i++)
    {
        sum += i;
    }
    pthread_exit(NULL);
    return 0;
}

int main(void)
{
    pthread_t ptid1, ptid2;
    sum=0;

    pthread_create(&ptid1, NULL, add1, &sum);
    pthread_create(&ptid2, NULL, add2, &sum);

    pthread_join(ptid1,NULL);
    pthread_join(ptid2,NULL);

    printf("sum %d\n", sum);
    pthread_exit(NULL);


    return 0;
}
```

```
[10/08/18]seed@VM:~/.../Pthread_Join$ gcc Pthread_Join.c -lpthread
[10/08/18]seed@VM:~/.../Pthread_Join$ ./a.out
sum 45
[10/08/18]seed@VM:~/.../Pthread_Join$
```

Pthread mutex – flag for privacy/security

- Mutex is an abbreviation for "mutual exclusion". Mutex variables are one of the primary means of implementing thread synchronization and for protecting shared data when multiple writes occur.

- A mutex variable acts like a "lock" protecting access to a shared data resource.

Pthread mutex - flag

- Mutex should be declared with type:
  - pthread_mutex_t (defined in “sys/types.h”)

- Mutex should be initialized before it is used:
  - int pthread_mutex_init(pthread_mutex_t *mutex,
    const pthread_mutexattr_t *attr);
  - It initialises the mutex referenced by mutex with attributes specified by attr.
  - If attr is NULL, the default mutex attributes are used; the effect is the same as passing the address of a default mutex attributes object.
  - Upon successful initialisation, the state of the mutex becomes initialised and unlocked.

- Mutex should be free if it is no longer used:
  - int pthread_mutex_destroy(pthread_mutex_t *);

Pthread mutex - flag

- Pthread mutex lock routines:
  - int pthread_mutex_lock(pthread_mutex_t *mutex);
  - int pthread_mutex_trylock(pthread_mutex_t *mutex);
  - int pthread_mutex_unlock(pthread_mutex_t *mutex);

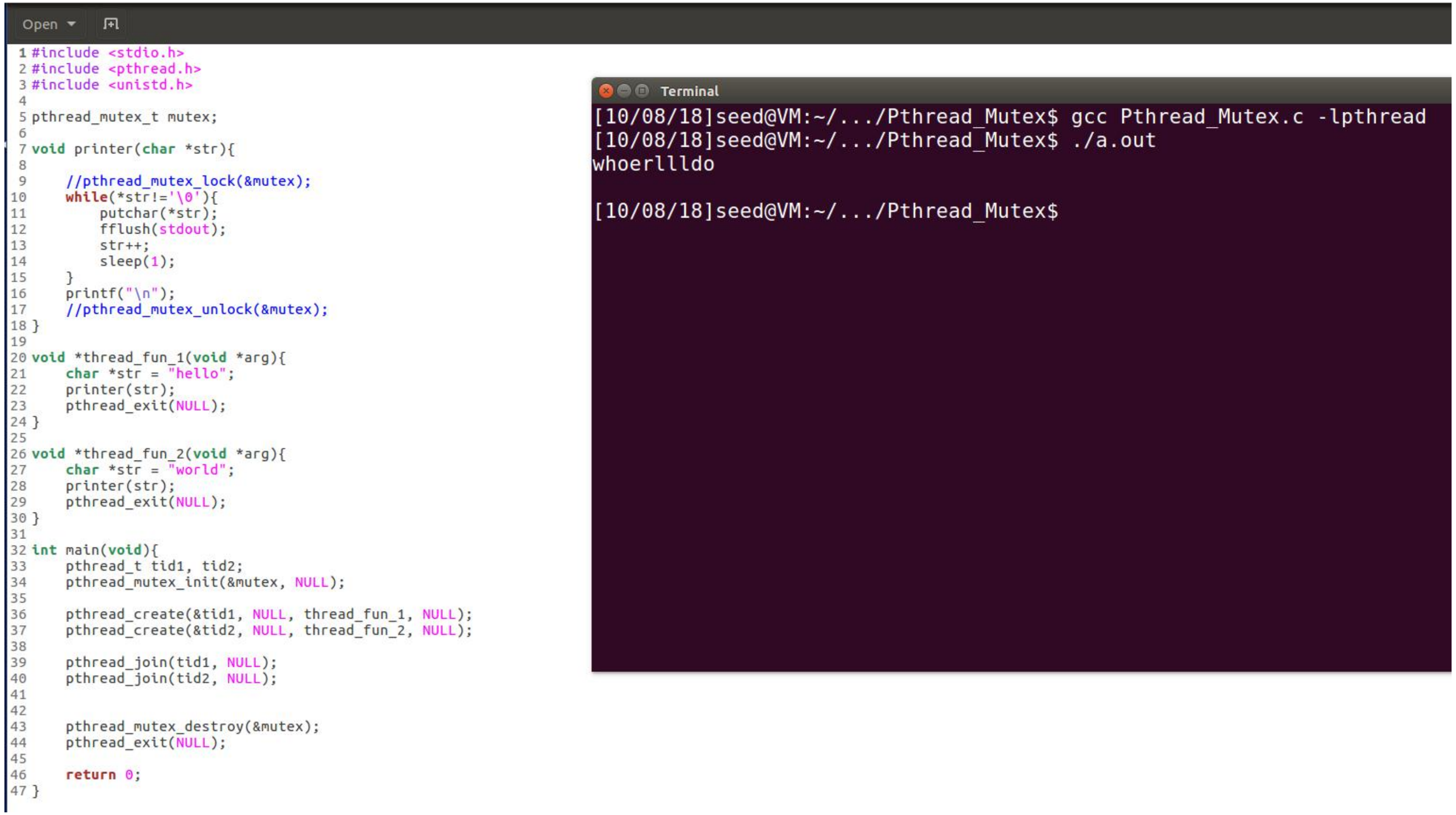
Pthread mutex - flag

- The pthread_mutex_lock() routine is used by a thread to acquire a lock on the specified mutex variable. If the mutex is already locked by another thread, this call will block the calling thread until the mutex is unlocked.

- pthread_mutex_trylock() will attempt to lock a mutex. However, if the mutex is already locked, the routine will return immediately with a "busy" error code. This routine may be useful in preventing deadlock conditions, as in a priority-inversion situation.

- pthread_mutex_unlock() will unlock a mutex if called by the owning thread. Calling this routine is required after a thread has completed its use of protected data if other threads are to acquire the mutex for their work with the protected data. An error will be returned if:
  - If the mutex was already unlocked
  - If the mutex is owned by another thread

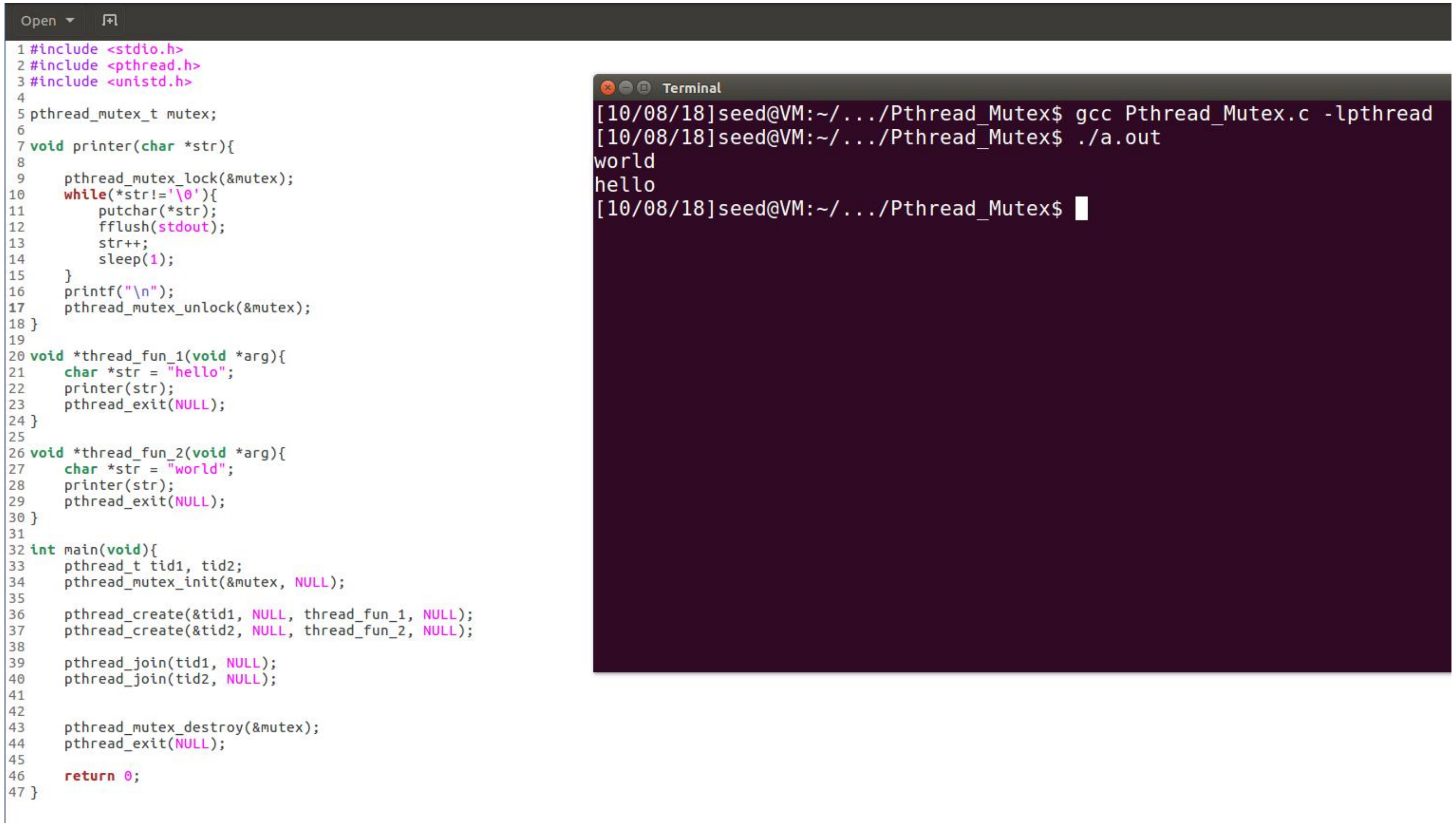
Pthread mutex - Flag



```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t mutex;

void printer(char *str){

    //pthread_mutex_lock(&mutex);
    while(*str!='\0'){
        putchar(*str);
        fflush(stdout);
        str++;
        sleep(1);
    }
    printf("\n");
    //pthread_mutex_unlock(&mutex);
}

void *thread_fun_1(void *arg){
    char *str = "hello";
    printer(str);
    pthread_exit(NULL);
}

void *thread_fun_2(void *arg){
    char *str = "world";
    printer(str);
    pthread_exit(NULL);
}

int main(void){
    pthread_t tid1, tid2;
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&tid1, NULL, thread_fun_1, NULL);
    pthread_create(&tid2, NULL, thread_fun_2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);


    pthread_mutex_destroy(&mutex);
    pthread_exit(NULL);

    return 0;
}
```

```
[10/08/18]seed@VM:~/.../Pthread_Mutex$ gcc Pthread_Mutex.c -lpthread
[10/08/18]seed@VM:~/.../Pthread_Mutex$ ./a.out
whoerllldo

[10/08/18]seed@VM:~/.../Pthread_Mutex$
```

Pthread mutex - flag



```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t mutex;

void printer(char *str){

    pthread_mutex_lock(&mutex);
    while(*str!='\0'){
        putchar(*str);
        fflush(stdout);
        str++;
        sleep(1);
    }
    printf("\n");
    pthread_mutex_unlock(&mutex);
}

void *thread_fun_1(void *arg){
    char *str = "hello";
    printer(str);
    pthread_exit(NULL);
}

void *thread_fun_2(void *arg){
    char *str = "world";
    printer(str);
    pthread_exit(NULL);
}

int main(void){
    pthread_t tid1, tid2;
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&tid1, NULL, thread_fun_1, NULL);
    pthread_create(&tid2, NULL, thread_fun_2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);


    pthread_mutex_destroy(&mutex);
    pthread_exit(NULL);

    return 0;
}
```

```
[10/08/18]seed@VM:~/.../Pthread_Mutex$ gcc Pthread_Mutex.c -lpthread
[10/08/18]seed@VM:~/.../Pthread_Mutex$ ./a.out
world
hello
[10/08/18]seed@VM:~/.../Pthread_Mutex$ 
```

Pthread condition - signals

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

- While mutexes implement synchronization by controlling thread access to data, condition variables allow threads to synchronize based upon the actual value of data.

- 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 *);

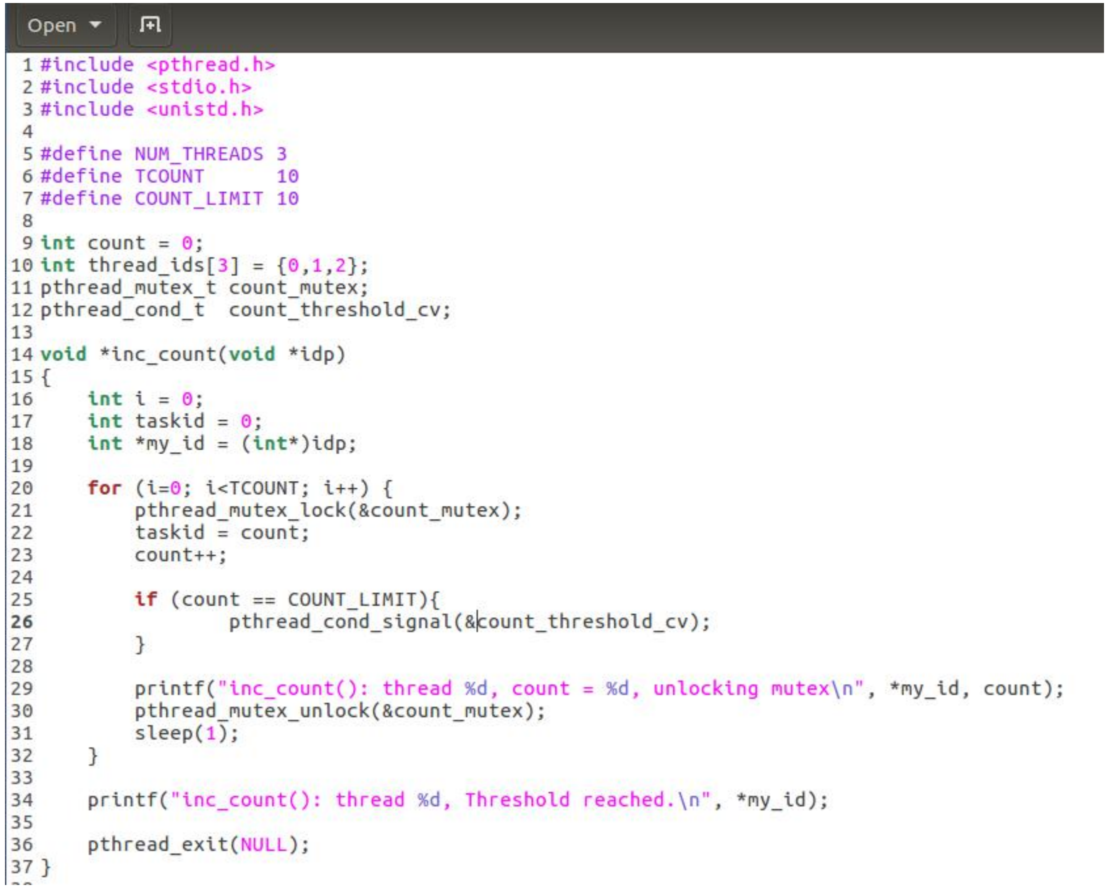
Pthread condition - signals

- 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. Should also unlock mutex after signal has been received.

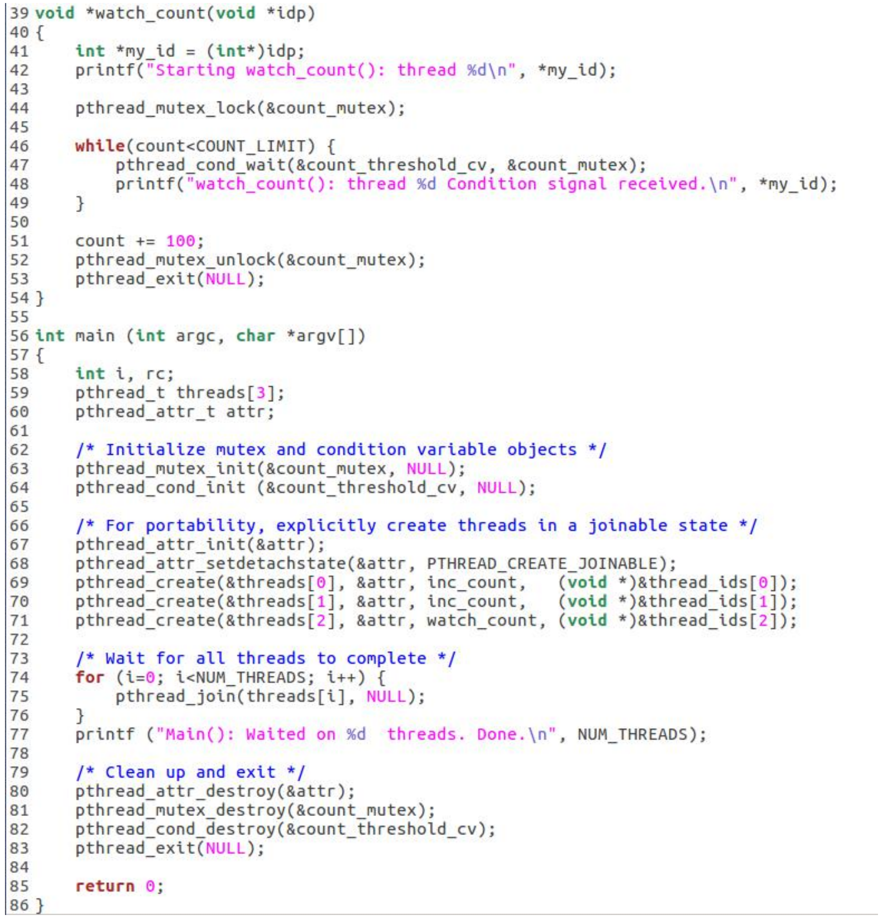
- The pthread_cond_signal() routine is used to signal (or wake up) another thread which is waiting on the condition variable. It should be called after mutex is locked, and must unlock mutex in order for pthread_cond_wait() routine to complete.

- The pthread_cond_broadcast() routine should be used instead of pthread_cond_signal() if more than one thread is in a blocking wait state.

Pthread condition - signals

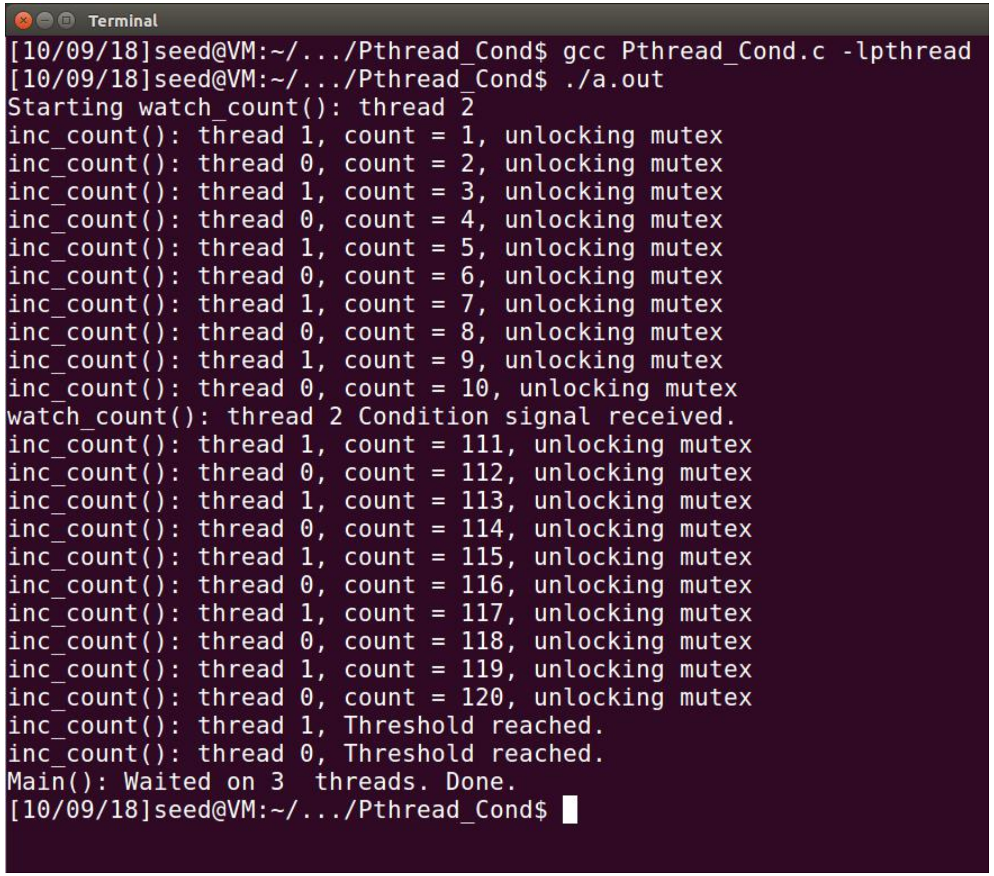


```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

#define NUM_THREADS 3
#define TCOUNT      10
#define COUNT_LIMIT 10

int count = 0;
int thread_ids[3] = {0,1,2};
pthread_mutex_t count_mutex;
pthread_cond_t  count_threshold_cv;

void *inc_count(void *idp)
{
    int i = 0;
    int taskid = 0;
    int *my_id = (int*)idp;

    for (i=0; i<TCOUNT; i++) {
        pthread_mutex_lock(&count_mutex);
        taskid = count;
        count++;

        if (count == COUNT_LIMIT){
                pthread_cond_signal(&count_threshold_cv);
        }

        printf("inc_count(): thread %d, count = %d, unlocking mutex\n", *my_id, count);
        pthread_mutex_unlock(&count_mutex);
        sleep(1);
    }

    printf("inc_count(): thread %d, Threshold reached.\n", *my_id);

    pthread_exit(NULL);
}
```



```
void *watch_count(void *idp)
{
    int *my_id = (int*)idp;
    printf("Starting watch_count(): thread %d\n", *my_id);

    pthread_mutex_lock(&count_mutex);

    while(count<COUNT_LIMIT) {
        pthread_cond_wait(&count_threshold_cv, &count_mutex);
        printf("watch_count(): thread %d Condition signal received.\n", *my_id);
    }

    count += 100;
    pthread_mutex_unlock(&count_mutex);
    pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
    int i, rc;
    pthread_t threads[3];
    pthread_attr_t attr;

    /* Initialize mutex and condition variable objects */
    pthread_mutex_init(&count_mutex, NULL);
    pthread_cond_init (&count_threshold_cv, NULL);

    /* For portability, explicitly create threads in a joinable state */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    pthread_create(&threads[0], &attr, inc_count,   (void *)&thread_ids[0]);
    pthread_create(&threads[1], &attr, inc_count,   (void *)&thread_ids[1]);
    pthread_create(&threads[2], &attr, watch_count, (void *)&thread_ids[2]);

    /* Wait for all threads to complete */
    for (i=0; i<NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
    printf ("Main(): Waited on %d  threads. Done.\n", NUM_THREADS);

    /* Clean up and exit */
    pthread_attr_destroy(&attr);
    pthread_mutex_destroy(&count_mutex);
    pthread_cond_destroy(&count_threshold_cv);
    pthread_exit(NULL);

    return 0;
}
```

Pthread condition - signals



```
[10/09/18]seed@VM:~/.../Pthread_Cond$ gcc Pthread_Cond.c -lpthread
[10/09/18]seed@VM:~/.../Pthread_Cond$ ./a.out
Starting watch_count(): thread 2
inc_count(): thread 1, count = 1, unlocking mutex
inc_count(): thread 0, count = 2, unlocking mutex
inc_count(): thread 1, count = 3, unlocking mutex
inc_count(): thread 0, count = 4, unlocking mutex
inc_count(): thread 1, count = 5, unlocking mutex
inc_count(): thread 0, count = 6, unlocking mutex
inc_count(): thread 1, count = 7, unlocking mutex
inc_count(): thread 0, count = 8, unlocking mutex
inc_count(): thread 1, count = 9, unlocking mutex
inc_count(): thread 0, count = 10, unlocking mutex
watch_count(): thread 2 Condition signal received.
inc_count(): thread 1, count = 111, unlocking mutex
inc_count(): thread 0, count = 112, unlocking mutex
inc_count(): thread 1, count = 113, unlocking mutex
inc_count(): thread 0, count = 114, unlocking mutex
inc_count(): thread 1, count = 115, unlocking mutex
inc_count(): thread 0, count = 116, unlocking mutex
inc_count(): thread 1, count = 117, unlocking mutex
inc_count(): thread 0, count = 118, unlocking mutex
inc_count(): thread 1, count = 119, unlocking mutex
inc_count(): thread 0, count = 120, unlocking mutex
inc_count(): thread 1, Threshold reached.
inc_count(): thread 0, Threshold reached.
Main(): Waited on 3  threads. Done.
[10/09/18]seed@VM:~/.../Pthread_Cond$ 
```

References

- Pthread:
  - http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html
  - Slide “pthread.pdf”
  - Slide “ch04.ppt”

  - Write your own Makefile
  - You can only utilize the libraries listed in the template. Any other libraries are not allowed to use.
  - And your output information must comply with the HW2 description

Operating System (CSC 3150)

Tutorial 4 – Part 2

KAI SHEN

OFFICE HOUR: WED 9PM-10PM @ ZHI XIN 101

E-MAIL: 118010254@LINK.CUHK.EDU.CN

Target

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

Functions you may required:

- Keyboard Hit
- Terminal Control
- Suspend the executing thread

Keyboard Hit

- In Assignment 2, we've provided a similar function named int kbhit(void), you could use it directly.

- 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

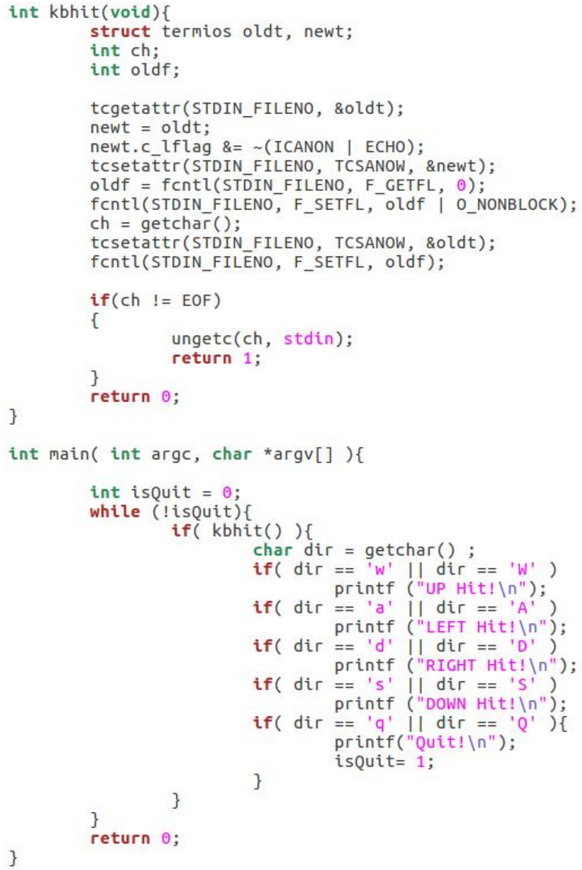
- int getchar(void) function is used to get/read a character from keyboard input.

- int putchar(int char) function is a file handling function which is used to write a character on standard output/screen.

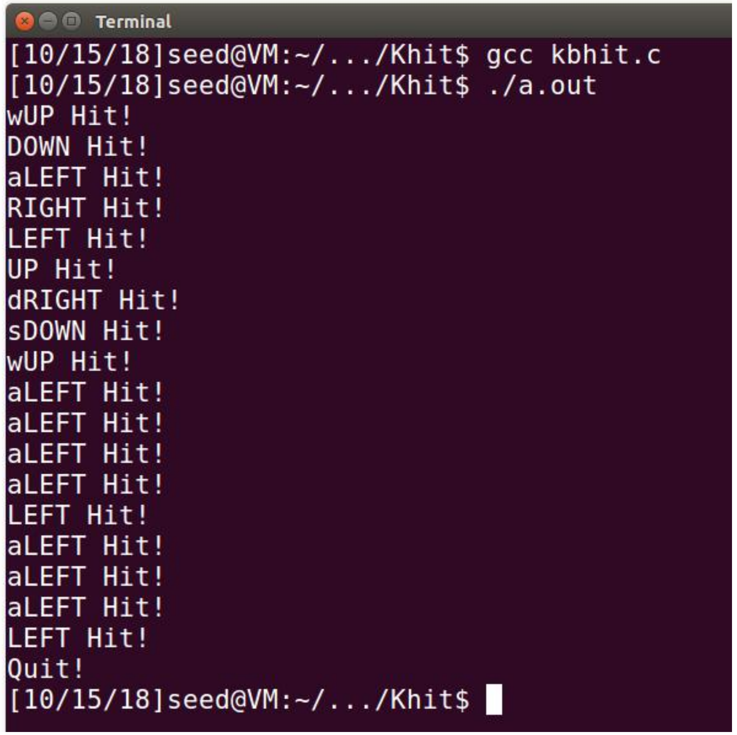
- int puts(const char *str) writes a string to stdout up to but not including the null character. A newline character is appended to the output.

- In Assignment 2, you may use above functions to complete your keyboard read and map write.

Keyboard Hit



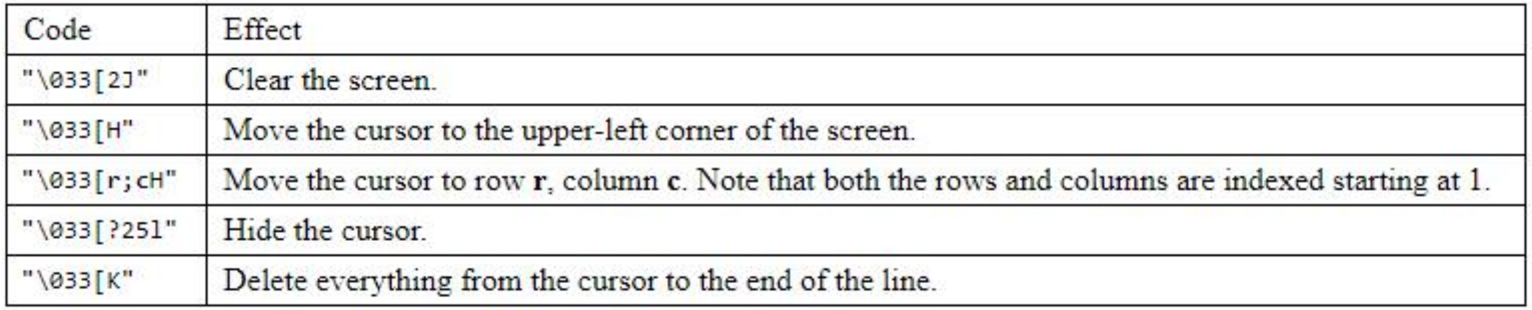
```
int kbhit(void){
        struct termios oldt, newt;
        int ch;
        int oldf;

        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;
}
```



```
[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!
aLEFT Hit!
LEFT Hit!
Quit!
[10/15/18]seed@VM:~/.../Khit$
```

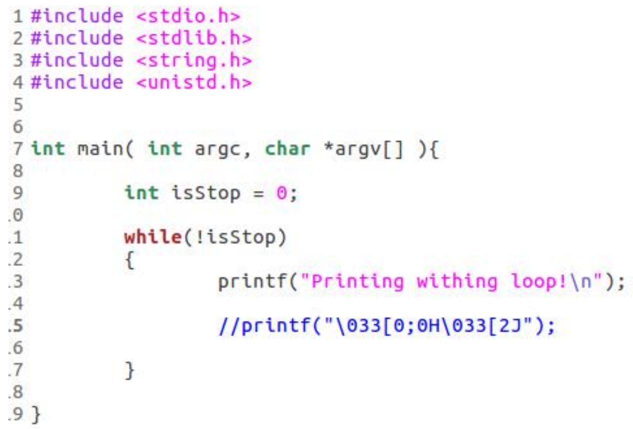
Terminal Control

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

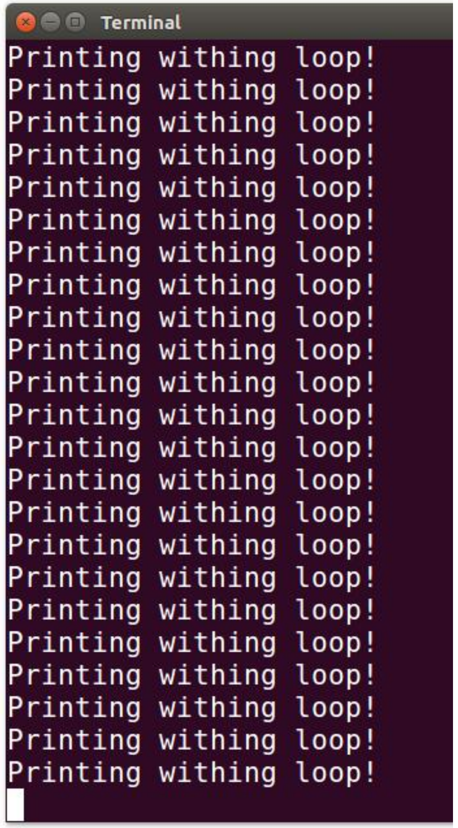


Code	Effect
`"\033[2J"`	Clear the screen.
`"\033[H"`	Move the cursor to the upper-left corner of the screen.
`"\033[r;cH"`	Move the cursor to row **r**, column **c**. Note that both the rows and columns are indexed starting at 1.
`"\033[?25l"`	Hide the cursor.
`"\033[K"`	Delete everything from the cursor to the end of the line.

Terminal Control

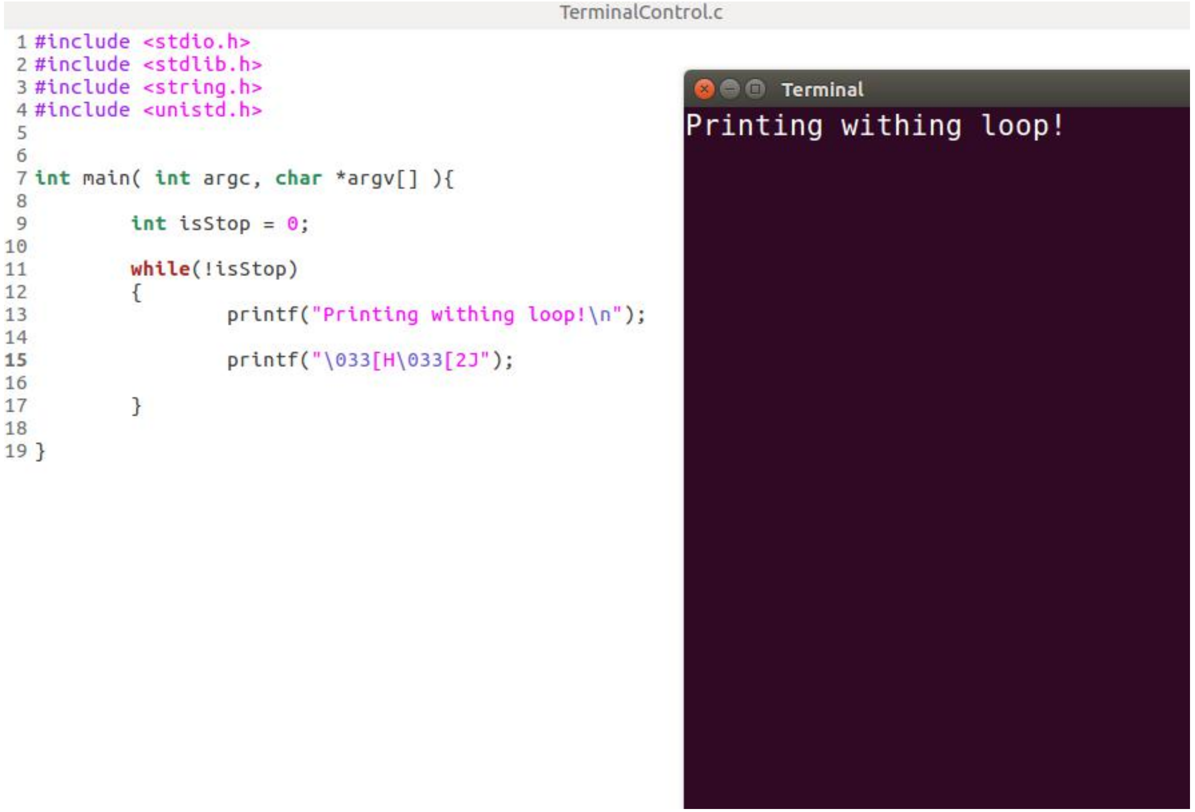


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


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

        int isStop = 0;

        while(!isStop)
        {
                printf("Printing withing loop!\n");

                //printf("\033[0;0H\033[2J");

        }

}
```



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

Terminal Control



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


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

        int isStop = 0;

        while(!isStop)
        {
                printf("Printing withing loop!\n");

                printf("\033[H\033[2J");

        }

}
```

```
Printing withing loop!
```

Suspend executing thread

- 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.

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.

- You could set suspend for screen printing, to control the logs moving speed.

- You could use srand() and rand() to set the random interval among logs moving.

- You could set flags to check game status.

Thank you