CSEN2253
OS Lab

WEEK - III Threads (I)

compile : gcc -pthread p1.c -o p1

1. Recall Basics about Threads

What is a Thread?
A thread is a path of execution within a process. A process can contain multiple threads.

Process vs Thread
The primary difference is that threads within the same process run in a *shared memory space*, while processes run in separate memory spaces.
Threads are *not independent* of one another like processes are, and as a result, threads share with other threads their code section, data section, and OS resources (like open files and signals).

Advantages of Thread over Process

a) **Responsiveness**: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
b) **Faster context switch**: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
c) **Resource sharing:** Resources like code, data, and files can be shared among all threads within a process.
d) **Communication:** Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two processes.
e) **Enhanced throughput:** If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Types of Threads
There are two types of threads.
User Level Thread
Kernel Level Thread

Linux Thread Basics
Thread operations include thread creation, termination, synchronization (joins,blocking), scheduling, data management and process interaction.

A thread does not maintain a list of created threads, nor does it know the thread that created it.
All threads within a process share the same address space.

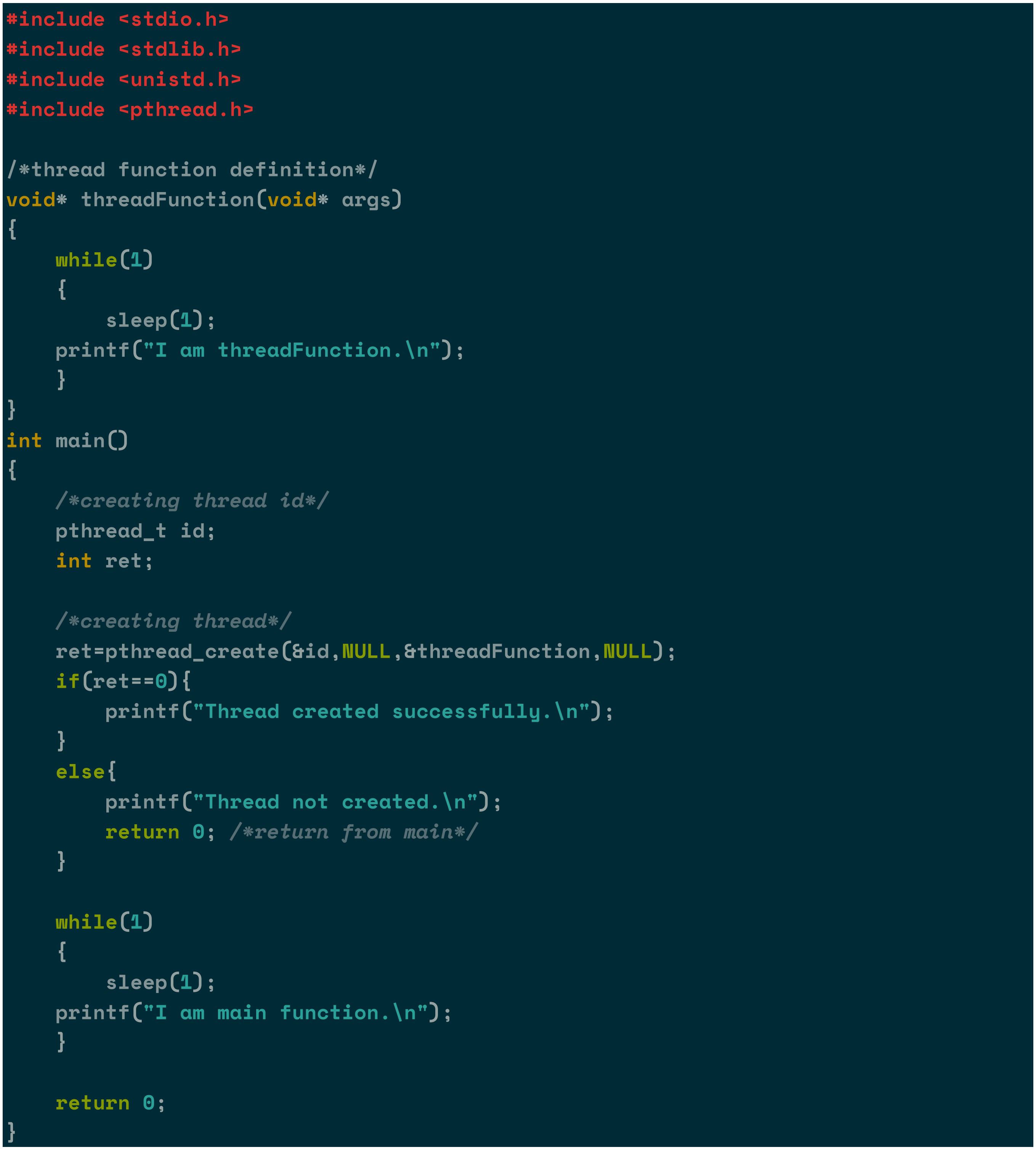
Threads in the same process share:
Process instructions, Most data, open files (descriptors), signals and signal handlers,
current working directory, User and group id

Each thread has a unique:
Thread ID, set of registers, stack pointer, stack for local variables, return addresses
signal mask, priority, Return value

pthread functions return "0" if OK.

2. Creating a thread

Creating a process using **`pthread_create()`**function



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

/*thread function definition*/
void* threadFunction(void* args)
{
    while(1)
    {
        sleep(1);
    printf("I am threadFunction.\n");
    }
}
int main()
{
    /*creating thread id*/
    pthread_t id;
    int ret;

    /*creating thread*/
    ret=pthread_create(&id,NULL,&threadFunction,NULL);
    if(ret==0){
        printf("Thread created successfully.\n");
    }
    else{
        printf("Thread not created.\n");
        return 0; /*return from main*/
    }

    while(1)
    {
        sleep(1);
    printf("I am main function.\n");
    }

    return 0;
}
```

SYNOPSIS

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

Compile and link with **-pthread**.

DESCRIPTION

The **pthread_create()** function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().

ARGUMENTS

thread - returns the thread id. (unsigned long int defined in bits/pthreadtypes.h)

attr - Set to NULL if default thread attributes are used. (else define members of the struct pthread_attr_t defined in bits/pthreadtypes.h)

Attributes include:

- *detached state* (joinable? Default: PTHREAD_CREATE_JOINABLE.
  Other option: PTHREAD_CREATE_DETACHED)
- *scheduling policy* (real-time? PTHREAD_INHERIT_SCHED,PTHREAD_EXPLICIT_SCHED,SCHED_OTHER)
- *scheduling parameter*
- *inheritsched attribute* (Default: PTHREAD_EXPLICIT_SCHED Inherit from parent thread: PTHREAD_INHERIT_SCHED)
- *scope* (Kernel threads: PTHREAD_SCOPE_SYSTEM User threads: PTHREAD_SCOPE_PROCESS Pick one or the other not both.)
- *guard size*
- *stack address* (See unistd.h and bits/posix_opt.h _POSIX_THREAD_ATTR_STACKADDR)
- *stack size* (default minimum PTHREAD_STACK_SIZE set in pthread.h)

void * (*start_routine) - pointer to the function to be threaded. Function has a single argument: pointer to void.

***arg** - pointer to argument of function. To pass multiple arguments, send a pointer to a structure.

3. Terminating a Threads

Terminating a process using **pthread_exit()**function

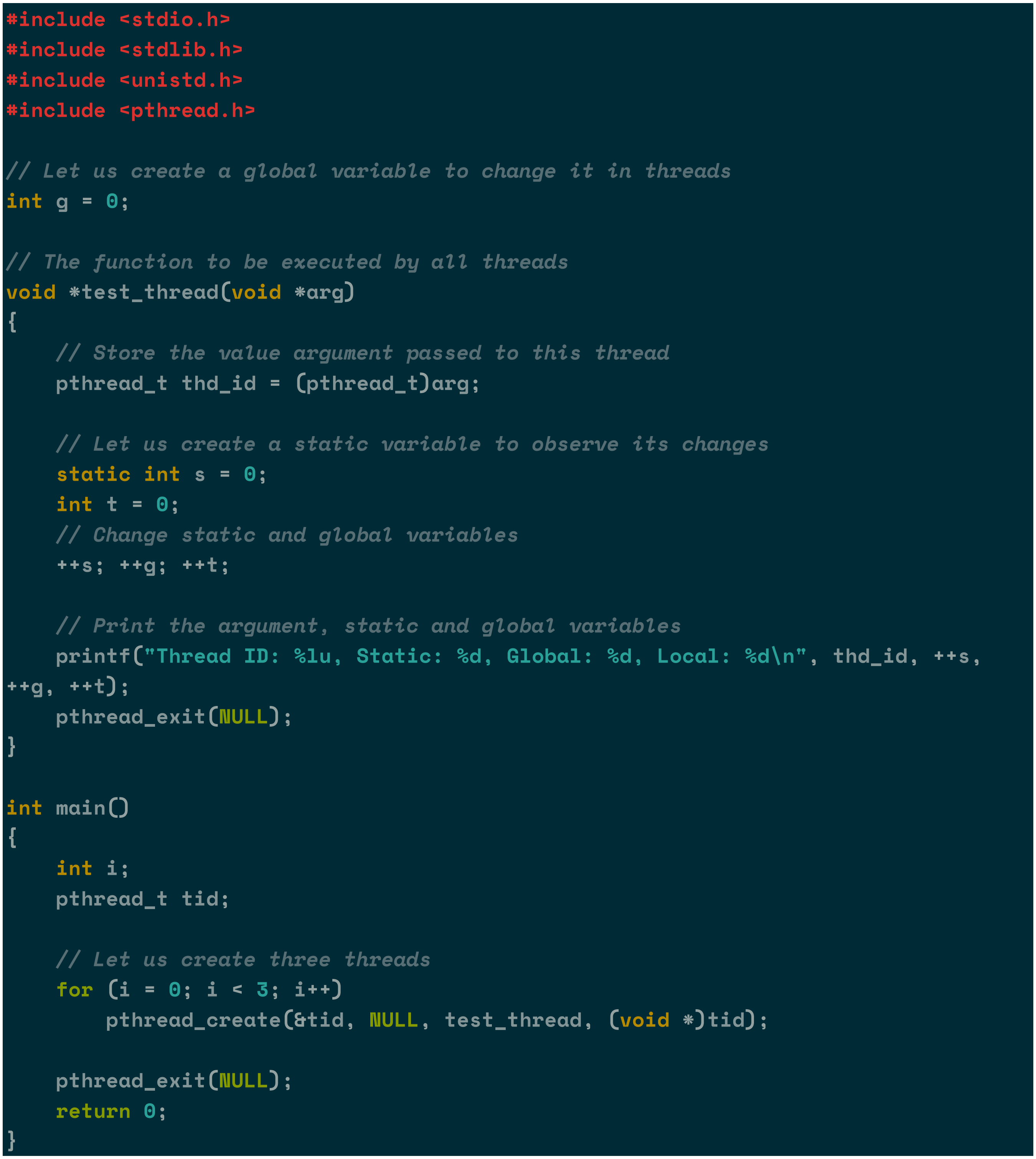
SYNOPSIS

```
void pthread_exit(void *retval);
```

Compile and link with **-pthread**.

DESCRIPTION

The pthread_exit() function terminates the calling thread and returns a value via retval that (if the thread is joinable) is available to another thread in the same process that calls pthread_join(3).



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

// Let us create a global variable to change it in threads
int g = 0;

// The function to be executed by all threads
void *test_thread(void *arg)
{
    // Store the value argument passed to this thread
    pthread_t thd_id = (pthread_t)arg;

    // Let us create a static variable to observe its changes
    static int s = 0;
    int t = 0;
    // Change static and global variables
    ++s; ++g; ++t;

    // Print the argument, static and global variables
    printf("Thread ID: %lu, Static: %d, Global: %d, Local: %d\n", thd_id, ++s,
++g, ++t);
    pthread_exit(NULL);
}

int main()
{
    int i;
    pthread_t tid;

    // Let us create three threads
    for (i = 0; i < 3; i++)
        pthread_create(&tid, NULL, test_thread, (void *)tid);

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

4. Identifying a Thread

Identifying a thread using **pthread_self()** function and **pthread_t *thread**.

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

// The function to be executed by all threads
void *test_thread(void *arg)
{
    // Store the value argument passed to this thread
    pthread_t *thd_id = (pthread_t*)arg;
    pthread_t sid;
    sid = pthread_self();
    printf("Hello from new thread %lu got %lu!\n",sid,*thd_id);
    pthread_exit(NULL);
}

int main()
{
    int i;
    pthread_t tid, sid;

    // Let us create three threads
    for (i = 0; i < 3; i++) {
        if (pthread_create(&tid, NULL, test_thread, (void *)&tid)){
            perror ("\npthread_create() error");
            exit(1);
        }
        sid = pthread_self();
        printf("\n This is thread (%lu): Created new thread (%lu)... \n", sid,
tid);
    }
    pthread_exit(NULL);
    return 0;
}
```

SYNOPSIS

pthread_t pthread_self(void);

Compile and link with -pthread.

DESCRIPTION

The pthread_self() function returns the ID of the calling thread. This is the same value that is returned in *thread in the pthread_create(3) call that created this thread.

5. Synchronizing Threads

Using **pthread_join()**, **pthread_equal()**

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

// The function to be executed by all threads
void *test_thread(void *arg)
{
    // Store the value argument passed to this thread
    pthread_t *thd_id = (pthread_t*)arg;
    pthread_t sid;
    sid = pthread_self();
    printf("Hello from new thread %lu got %lu!\n",sid,*thd_id);
    //pthread_exit(NULL);
}

int main()
{
    int i;
    pthread_t tid, sid;

    // Let us create three threads
    for (i = 0; i < 3; i++) {
        if (pthread_create(&tid, NULL, test_thread, (void *)&tid)){
            perror ("\npthread_create() error");
            exit(1);
        }
        sid = pthread_self();
        printf("\n This is thread (%lu): Created new thread (%lu)... \n", sid,
tid);
    }
    pthread_exit(NULL);
    return 0;
}
```

SYNOPSIS

`int pthread_join(pthread_t thread, void **retval);`

Compile and link with -pthread.

DESCRIPTION

The pthread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately. The thread specified by thread must be joinable.

If retval is not NULL, then pthread_join() copies the exit status of the target thread (i.e., the value that the target thread supplied to pthread_exit(3)) into the location pointed to by retval. If the target thread was canceled, then PTHREAD_CANCELED is placed in the location pointed to by retval.

If multiple threads simultaneously try to join with the same thread, the results are undefined. If the thread calling pthread_join() is canceled, then the target thread will remain joinable (i.e., it will not be detached).

6. Classwork

a) Write a program that accepts an integer **num** as command line argument. In your program create two threads. One thread should report the **num!** (Factorial of **num**). The other thread should report the summation of all factorials till **num**. [eg: if the argument is 5, one thread calculates 5!=120, another thread calculates 1!+2!+3!+4!+5!=153]. *Each process should report its own id and caller's id. When a thread ends, it should display meaningful message.*

b) Write a program that accepts two integers (**low, high**) as command line argument. Create two threads in your program. The first thread should calculate the summation of all integers between (**low, high**) as **sum_res**. The second thread should evaluate whether **sum_res** is prime or not. *Each process should report its own id and caller's id. When a thread ends, it should display meaningful message.*

7. Home Assignment

a) Write a multi-threaded program such that it creates 10 more threads. Each thread should print ***Hello from nth thread*** along with the argument received from the main thread. Each thread should return its own id and a unique value to main using **pthread_exit()**. Main thread should be able to print this returned message identifying which thread ended within its own thread.

i) Synchronize it such that all 10 threads are created first and then all of them are terminated.

Eg:
```
0 MAIN
    1 created
    2 created
    3 created
    .
    .
    .
    .
    10 created
    1 ended
    2 ended
    3 ended
    .
    .
    .
    .
    10 ended
  MAIN ended
```

Handed Out: 14th September, 2021
To be turned within: 19th September, 2021

ii) Synchronize it such that one is crea
created and terminated)

Eg:
```
0 MAIN
    1 created
    1 ended
    2 created
    2 ended
    3 created
    3 ended
    .
    .
    .
    .
    10 created
    10 ended
MAIN ended
```

iii) Synchronize it such that a pair is created at a time and terminated in the reverse order of how they were created.

Eg:
```
0 MAIN
    1 created
    2 created
    2 ended
    1 ended
    3 created
    4 created
    4 ended
    3 ended
    .
    .
    .
    .
    9 created
    10 created
    10 ended
    9 ended
MAIN ended
```