Xi'an Jiaotong-Liverpool University
西交利物浦大学

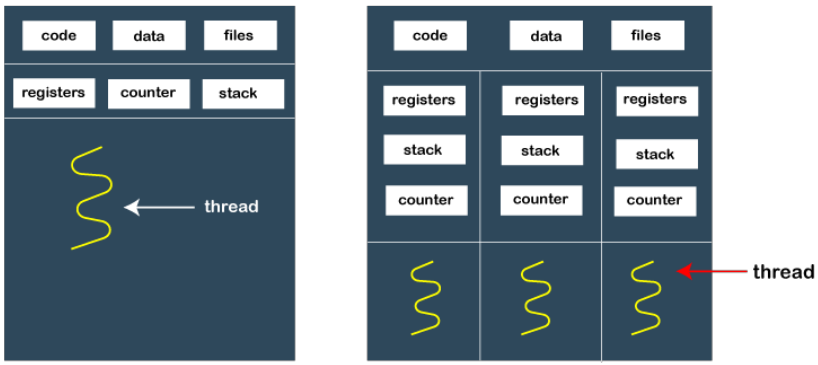
CPT104 - Operating Systems Concepts

Lab 9

Thread management using pthread library in C

A thread is a single sequence stream within in a process.

Threads are not independent of one other like process as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.



Single-threaded process Multi-threaded process

Creating Threads

To create a thread we use the **pthread_create( )** routine. This routine creates a thread based on the settings of the thread attributes object if specified, which your program must have previously initialized. If called without a specified thread attributes object, **pthread_create** creates a new thread that has the default attributes.

Each thread in a process is identified by a thread ID, type **pthread_t**.
Upon creation, each thread executes a **thread function**. This is just an ordinary function and contains the code that the thread should run. When the function returns, the thread exits.

int pthread_create(pthread_t *tidp, const pthread_attr_t *attr, void *(*start_rtn)(void *), void *restrict arg);

pthread_create () function creates a new thread. You provide it with the following:

- A pointer to a **pthread_t** variable, in which the thread ID of the new thread is stored.
- A pointer to a thread attribute object. This object controls details of how the thread interacts with the rest of the program. If you pass **NULL** as the thread attribute, a thread will be created with the default thread attributes.
- A pointer to the thread function and a pointer to the arguments to be passed to the function.

pthread_exit()

*void pthread_exit(void *retval);*

This function terminates a calling thread. It takes one argument as a parameter and returns nothing.

pthread_cancel()

int pthread_cancel(pthread_t thread);

Sometimes an application may wish to stop a thread that is currently executing. The function *pthread_cancel* can help us accomplish this.

pthread_join()

*int pthread_join(pthread_t thread, void **thread_return);*

This function waits for the termination of another thread. It takes two parameters as arguments: the first parameter is the thread for which to wait, and the second argument is a pointer to a pointer that itself points to the return value from the thread. The function returns the integer type with 0 on successful termination and –1 if any failure occurs.

Creating Threads

1. Import the required libraries.

```
#include<stdio.h>          // Standard I/O Routines Library
#include<unistd.h>         // Unix Standard Library
#include<pthread.h>        // POSIX Thread Creation Library
```

2. Develop the thread function to make it multithreaded. **The thread function must have a return type as a pointer.**

```
void *ThreadFunction(){
printf("Thread created by programmer.\n");
return NULL;
}
```

3. In this step, we write the main function. We need to create a thread descriptor variable and method.

pthread_create function returns the status codes 0 and 1 for **success** or **failure**.

If the thread is successful, the thread function executes; otherwise, you exit out of the program.

In the **pthread_create** function:

- the first argument is the address of the thread descriptor variable.
- the second argument takes a NULL value (since this deal with the default threads).
- the third value is a thread function that is executed in the thread.
- the last argument is also **NULL** because, in this thread function, there aren't any arguments to pass.

```
int main(){
pthread_t thread;            // Thread Descriptor
pthread_create(&thread, NULL, ThreadFunction, NULL);
pthread_exit(NULL);
return 0;
}
```

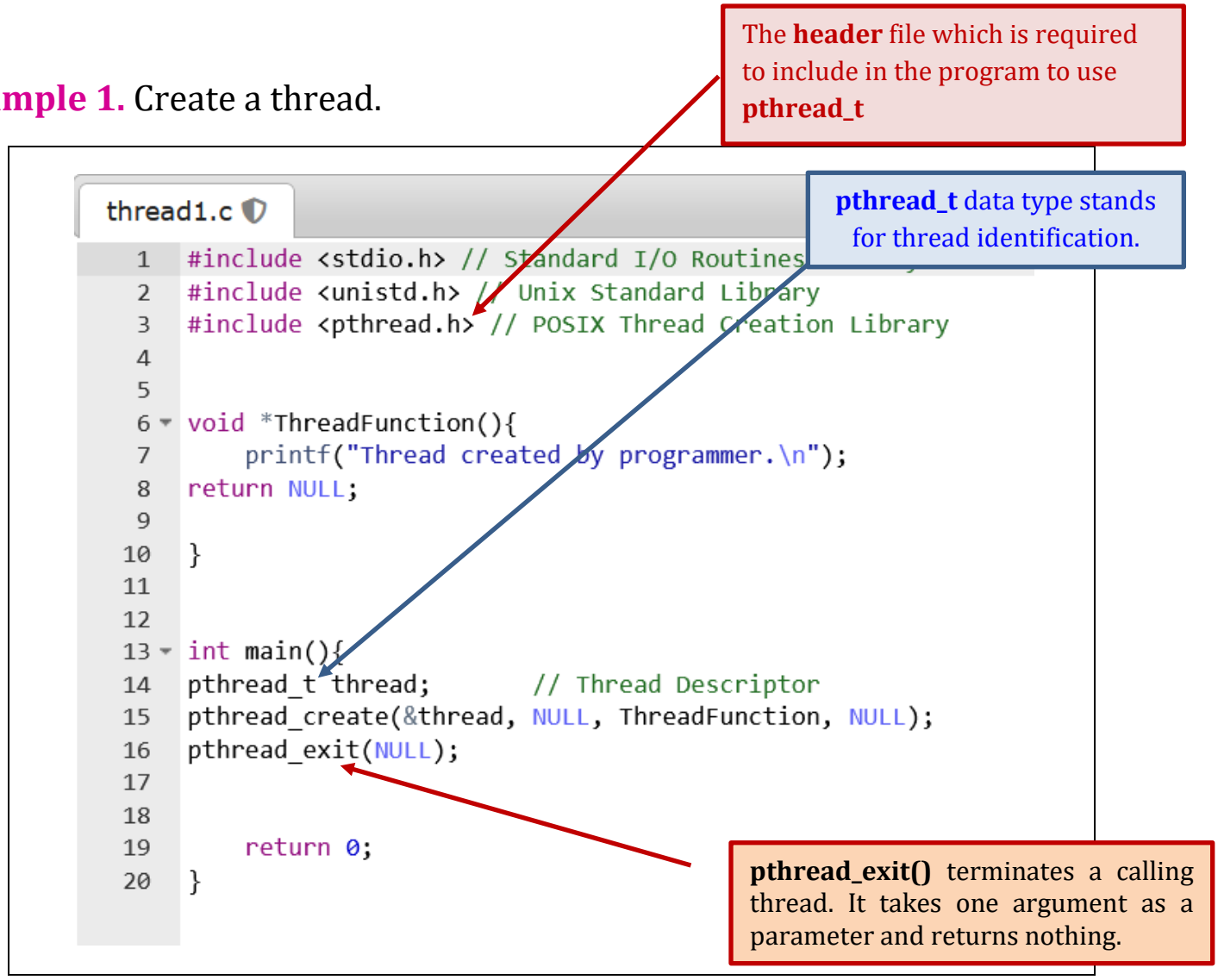
- You will use the

https://remisharrock.fr/sysbuild/#/VM

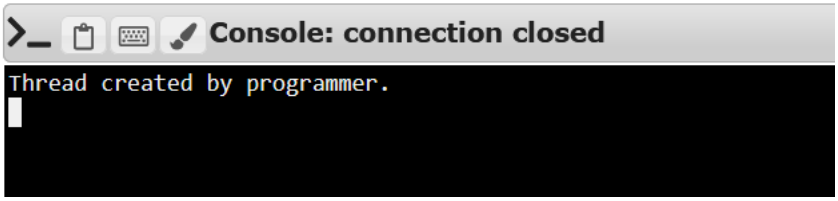
to write and test your code;

- You will also need to submit your code in LMO.

Example 1. Create a thread.



Output



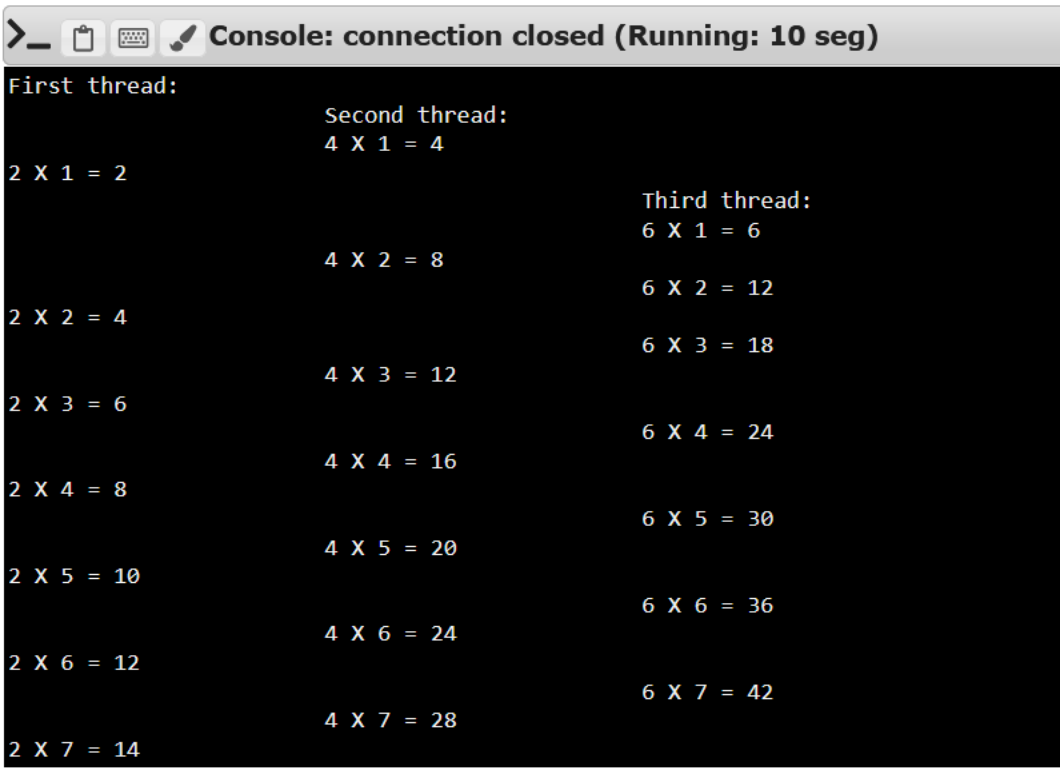
Example 2. Write C code with 3 threads instances execute the multiplication by 2, 4 and 6 in parallel.

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

void* first_thread(void* arg)
{
        int i;

        printf("First thread:\n");
        for (i = 1; i <= 10; i++) {
                printf("2 X %d = %d\n", i, i*2);
                sleep(1);
        }
        pthread_exit(NULL);
}

void* second_thread(void* arg)
{
        int i;

        printf("\t\t\tSecond thread:\n");
        for (i = 1; i <= 10; i++) {
                printf("\t\t\t4 X %d = %d\n", i, i*4);
                sleep(1);
        }

        pthread_exit(NULL);
}

void* third_thread(void* arg)
{
        int i;

        printf("\t\t\t\t\t\tThird thread:\n");
        for (i = 1; i <= 10; i++) {
                printf("\t\t\t\t\t\t6 X %d = %d\n", i, i*6);
                sleep(1);
        }

        pthread_exit(NULL);
}


int main()
{
        pthread_t tid[3];
```

\t - refers to one tab horizontal space

```
        // create 3 thread

        pthread_create(&tid[0], NULL, first_thread, NULL);
        pthread_create(&tid[1], NULL, second_thread, NULL);
        pthread_create(&tid[2], NULL, third_thread, NULL);


        //join 3 thread - waits for the termination of another thread.

        pthread_join(tid[0], NULL);
        pthread_join(tid[1], NULL);
        pthread_join(tid[2], NULL);

        printf("Finished threads execution!\n");
        return 0;

}
```

Output



We now continue with the **Lab Exercise** (see LMO)

- Just like the Lab Example, you can use the

  https://remisharrock.github.io/sysbuild/#/VM

OR

- You can also use to write and test your code **LMO - VPL**.

Reference:
For more information: refer to book chapter 11.1-11.5