DHA Suffa University
CS 206 – Operating Systems – Lab
Fall 2017

Lab 12 – Thread Creation and Management

Objective(s):

- Understanding Threads
- Creating Threads
- Managing Threads

Threads

A thread is the smallest unit of processing that can be performed in an OS. In most modern operating systems, a thread exists within a process - that is, a single process may contain multiple threads.

A thread of execution is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources. In particular, the threads of a process share the latter are instructions (its code) and its context (the values that its variables reference at any given moment).

How Threads differ from Processes

Threads differ from traditional multitasking operating system processes in that:

- processes are typically independent, while threads exist as subsets of a process
- processes carry considerably more state information than threads, whereas multiple threads within a process share process state as well as memory and other resources
- processes have separate address spaces, whereas threads share their address space
- processes interact only through system-provided inter-process communication mechanisms
- Context switching between threads in the same process is typically faster than context switching between processes.

Thread Creation:

Each thread in a process is identified by a thread ID. When referring to thread IDs in C or C++ programs, use the 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.

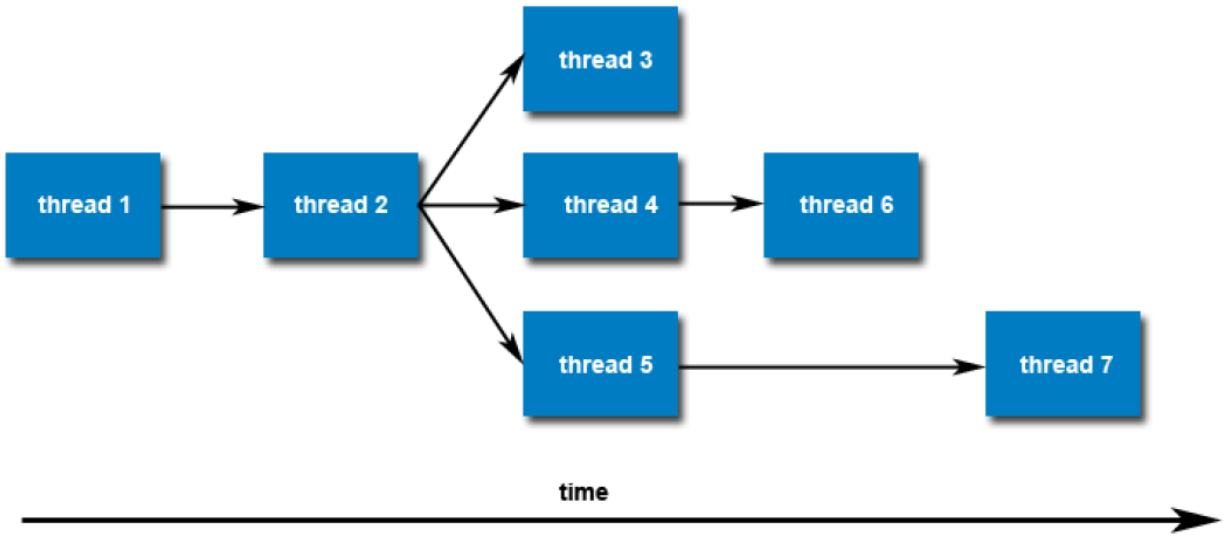
pthread_create

The pthread_create() function is used to create a new thread.

 # include<pthread.h>

*int pthread_create(pthread_t *threadid, const pthread _attr_t *attr, void *(*start_routine)(void*), void *arg);*

Parameter	Description
thread	An opaque, unique identifier for the new thread returned by the subroutine.
attr	An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
start_routine	The C++ routine that the thread will execute once it is created.
arg	A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.



Once created, threads are peers, and may create other threads. There is no implied hierarchy or dependency between threads.

Thread Attributes:

By default, a thread is created with certain attributes. Some of these attributes can be changed by the programmer via the thread attribute object.
pthread_attr_init and pthread_attr_destroy are used to initialize/destroy the thread attribute object.

Other routines are then used to query/set specific attributes in the thread attribute object.
Attributes include:

- Detached or joinable state
- Scheduling inheritance
- Scheduling policy
- Scheduling parameters
- Scheduling contention scope
- Stack size
- Stack address
- Stack guard (overflow) size

Return Values:

If successful it returns 0 otherwise it generates a nonzero number.

Joining Threads:

That function is pthread_join, which takes two arguments: the thread ID of the thread to wait for, and a pointer to a void*variable that will receive the finished thread's return value. If you don't care about the thread return value, pass NULL as the second argument.

pthread_join

The pthread_join() function waits for the thread specified by thread to terminate.

*int pthread_join(pthread_t threadid, void **retval);*

Return Values:

If successful it returns 0 otherwise it generates a nonzero number.

Example 01 (Threads.cpp): Creating Threads in C++

```
#include<iostream>
#include<unistd.h>
#include<pthread.h>
using namespace std;

void *thread(void *str)
{
    cout << (char *)str;
    usleep(2000000);
}

int main()
{
    pthread_t tid1, tid2, tid3;

    pthread_create(&tid1, NULL, thread, (void *) "T1\n");
    pthread_create(&tid2, NULL, thread, (void *) "T2\n");
    pthread_create(&tid3, NULL, thread, (void *) "T3\n");

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    pthread_join(tid3, NULL);

    pthread_exit(0);

    return 0;
}
```

Passing Arguments to Threads

The pthread_create() routine permits to pass one argument to the thread start routine. For cases where multiple arguments must be passed, this limitation is easily overcome by creating a structure which contains all of the arguments, and then passing a pointer to that structure in the pthread_create() routine.

All arguments must be passed by reference and cast to (void *).

Example 02 (DetachedThreads.cpp): Creating Detached Threads in C++

```
#include<iostream>
#include<unistd.h>
#include<pthread.h>
using namespace std;
bool thread_finished = 0;
void *thread(void *args)
{
    cout << "Entered the thread" << endl;
    thread_finished = 1;
}

int main()
{
    pthread_attr_t attr;
    pthread_t tid;

    //Create a default thread attributes object
    pthread_attr_init(&attr);

    //Set the detach state thread attribute;
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    //Create a thread using new attributes
    pthread_create(&tid, &attr, thread, NULL);

    pthread_attr_destroy(&attr);
    cout << "Trying Join returns " << (int)pthread_join(tid, NULL) << endl;

    usleep(2000000);

    return 0;
}
```

pthread_attr_init():

The function pthread_attr_init() initializes a thread attributes object attr with the default value

*int pthread_attr_init(pthread_attr_t *attr);*

Return Values:

Upon successful completion, pthread_attr_init() returns a value of 0. Otherwise, an error number is returned to indicate the error.

pthread_attr_setdetachstate():

The detach state attribute controls whether the thread is created in a detached state.

*int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);*

- PTHREAD_CREATE_DETACHED
  Thread state is detached means it cannot be joined with other threads.
- PTHREAD_CREATE_JOINABLE
  Thread state is joinable means it can be joined with other threads.

pthread_attr_destroy():

When a thread attributes object is no longer required, it should be destroyed using the pthread_attr_destroy()

*int pthread_attr_destroy(pthread_attr_t *attr);*

Return Values:

Upon successful completion, pthread_attr_destroy() returns a value of 0. Otherwise, an error number is returned to indicate the error.

Example 03 (StructAndThreads.cpp): Using Structures to store Threads' data

```
#include<iostream>
#include<unistd.h>
#include<stdlib.h>
#include<pthread.h>
#define NUM_THREADS 5
using namespace std;

struct threadData
{
    int threadID;
    char *message;
};

void *printHello(void *arg)
{
    struct threadData *myData;
    myData = (struct threadData*) arg;
    cout << "Thread ID: " << myData->threadID << "\t";
    cout << "Message: " << myData->message << endl;
    pthread_exit(NULL);
}

int main()
{
    pthread_t threads[NUM_THREADS];
    struct threadData td[NUM_THREADS];
    int thread, i;
    for (int i = 0; i < NUM_THREADS; i++)
    {
        cout << "Main creating thread " << i << endl;
        td[i].threadID = i;
        td[i].message = (char *)"This is message";
        thread = pthread_create(&threads[i], NULL, printHello, (void *) &td[i]);
```

Continued…..

```
        if (thread)
        {
            cout << "unable to create thread" << endl;
            exit(-1);
        }
    }
    pthread_exit(NULL);
    return 0;
}
```

Example 04 (DetachedvsJoinable.cpp): Detached vs Joinable Threads in C++

```
#include<iostream>
#include<unistd.h>
#include<stdlib.h>
#include<pthread.h>
#define NUM_THREADS 5
using namespace std;

void *wait(void *arg)
{
    long tid = (long) arg;
    int i;
    cout << "Sleeping in thread" << endl;
    usleep(1000000);
    cout << "Thread with ID: " << tid << " exiting" << endl;
}

int main()
{
    int thread;
    pthread_t threads[NUM_THREADS];
    pthread_attr_t attr;
    void *status;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    for (int i = 0; i < NUM_THREADS; i++)
    {
        cout << "Main creating thread " << i << endl;
        thread = pthread_create(&threads[i], NULL, wait, (void *)(&i));
        if (thread)
        {
            cout << "unable to create thread" << thread  << endl;
            exit(-1);
        }
    }
    pthread_attr_destroy(&attr);
    for (int i = 0; i < NUM_THREADS; i++)
```

Continued…..

```
    {
        thread = pthread_join(threads[i], NULL);
        if (thread)
        {
            cout << "unable to join " << thread << endl;
            exit(-1);
        }
        cout << "Main Completed Thread ID: " << i << endl;
        cout << "Exiting with status " << status << endl;
    }
    cout << "Main Program Exiting" << endl;
    pthread_exit(NULL);
    return 0;
}
```

Lab Task:

1. Calculate the sum of first n odd numbers and first n prime numbers using multithreading in C++.

E.g.

When n = 3

First 5 Odd Numbers: 1, 3, 5; **1 + 3 + 5 = 9**
First 5 Prime Numbers: 1, 2, 3; **1 + 2 + 3 = 6**

9 + 6 = 15

When n = 5

First 5 Odd Numbers: 1, 3, 5, 7, 9; **1 + 3 + 5 + 7 + 9 = 25**
First 5 Prime Numbers: 1, 2, 3, 5, 7; **1 + 2 + 3 + 5 + 7 = 18**

25 + 18 = 43

Lab Assignment 12:

1. **Mergesort.cpp] Create a multithreaded C++ program to sort the elements of a given integer array.**

Submission Instructions:

1. **Compress your .cpp file with your roll number.**
2. **Submit the file on LMS.**
3. **Due Date for assignment submission is Nov 19, 2017.**