**CS 332/532 – 1G- Systems Programming**

**Lab 12**

### ****Objectives****

* Create threads using POSIX threads library
* Thread synchronization using Mutexes

### ****Create threads using POSIX threads library****

In the previous labs we focused on how to create processes, in this lab we will focus on creating threads and mechanisms for establishing synchronization among threads.

First, let us understand the difference between a process and a thread. A process could be considered to have two characteristics: (a) resource ownership and (b) scheduling or execution. The unit of scheduling and dispatching is usually referred to as a **thread** or **lightweight process and**the ability of to support multiple, concurrent paths of execution within a single process is often referred to as multithreading.  Threads offer several benefits compared to a process:

* Threads takes less time to create a new thread than a process
* Threads take less time to terminate a thread than a process
* Switching between two threads (context switching) takes less time than switching between processes
* All of the threads in a process share the state and resources of that process (since threads reside in the same address space and have access to the same data)
* Threads enhance efficiency in communication between programs (since threads share memory and files within the same process and can communicate without invoking the kernel)

As a result of the above advantages, if we have to implement a set of functions that are closely related, implementing this functionality using multiple threads is far more efficient than using multiple processes.

We will use the POSIX threads library, usually referred to as Pthreads library, that provides C APIs to create and manage threads. We have to include the file pthread.h and link with -lpthread to compile and link.

We can create new threads using the pthread\_create() function which has the following function definition:

#include <pthread.h>  
int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,  
 void \*(\*start\_routine) (void \*), void \*arg);

The new thread that will be created by the pthread\_create function will invoke the function start\_routine. Note that the function start\_routine takes one argument of type void \* and has the return type as void \*. In other words, the function start\_routine has the following function definition:

void \*start\_routine(void \*arg)

When the pthread\_create call returns successfully, it returns the thread ID associated with the new thread created in the variable thread. This can be used by the main thread in subsequent pthread function calls such as pthread\_join. The second argument, attr, provides a reference to the pthread\_attr\_t structure that describes the various attributes of the new thread to be created. It can be initialized using pthread\_attr\_init call or set to NULL if default attributes must be used. You can find out more about the different thread attributes that can be specified by looking at the man page for pthread\_attr\_init.

The new thread created will terminate when the function start\_routine returns or when a call to pthread\_exit is made inside the start\_routine. We can use the pthread\_join function to wait for a thread to complete using the thread ID that was returned when pthread\_create call was invoked. If a thread has already completed, pthread\_join will return immediately, otherwise, it will wait for the corresponding thread to complete.

The following example shows how to create threads and wait for the threads to complete. You can download this program here: pthread1.c

/\*  
  Simple Pthread Program to illustrate the create/join threads.  
 To Compile: gcc -O -Wall pthread1.c -lpthread  
  To Run: ./a.out 4  
\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
  
int nthreads; /\* variable shared between threads \*/  
  
void \*compute(void \*arg) {  
    long tid = (long)arg;  
  
    printf("Hello, I am thread %ld of %d\n", tid, nthreads);  
  
    return (NULL);  
}  
  
int main(int argc, char \*\*argv) {  
    long i;  
    pthread\_t \*tid;  
  
    if (argc != 2) {  
      printf("Usage: %s <# of threads>\n",argv[0]);  
       exit(-1);  
    }  
  
    nthreads = atoi(argv[1]); // no. of threads  
  
    // allocate vector and initialize  
    tid = (pthread\_t \*)malloc(sizeof(pthread\_t)\*nthreads);  
  
    // create threads  
    for ( i = 0; i < nthreads; i++)  
      pthread\_create(&tid[i], NULL, compute, (void \*)i);  
  
    // wait for them to complete  
    for ( i = 0; i < nthreads; i++)  
      pthread\_join(tid[i], NULL);  
  
    printf("Exiting main program\n");  
  
    return 0;  
}

Here is an updated version of the program that prints the thread ids: pthread2.c

Here is another version of the program that passes a structure as the argument for pthread\_create instead of using the global variables: pthread3.c

### ****Thread Synchronization using Mutexes****

We can use the mutexes provided by the Pthreads library to control access to critical sections of the program and provide synchronization across the threads. We will use the example of computing the sum of the elements in a vector to illustrate the use of mutexes. We have a vector of N elements, we would like to assign each thread to compute the partial sum of N/P elements (where P is the number of threads), then we will update the shared global variable sum with the partial sums using mutex locks. The API for the mutex lock and unlock functions are shown below:

#include <pthread.h>  
int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);  
int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

The mutex variable is of type pthread\_mutex\_t and can be initially statically by assigning the value PTHREAD\_MUTEX\_INITIALIZER. Note that the mutex variable must be declared in global scope since it will be shared among multiple threads. A mutex can also be initialized dynamically using the function pthread\_mutex\_init. The pthread\_mutex\_destroy function can be used to destroy the mutex that was initialized using pthread\_mutex\_init. The complete program is shown below and can be downloaded here: pthread\_sum.c

/\*  
  Simple Pthread Program to find the sum of a vector.  
  Uses mutex locks to update the global sum.  
  Author: Purushotham Bangalore  
  Date: Jan 25, 2009  
  
 To Compile: gcc -O -Wall pthread\_sum.c -lpthread  
  To Run: ./a.out 1000 4  
\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <unistd.h>  
  
pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;  
  
double \*a=NULL, sum=0.0;  
int    N, size;  
  
void \*compute(void \*arg) {  
    int myStart, myEnd, myN, i;  
    long tid = (long)arg;  
  
    // determine start and end of computation for the current thread  
    myN = N/size;  
    myStart = tid\*myN;  
    myEnd = myStart + myN;  
    if (tid == (size-1)) myEnd = N;  
  
    // compute partial sum  
    double mysum = 0.0;  
    for (i=myStart; i<myEnd; i++)  
      mysum += a[i];  
  
    // grab the lock, update global sum, and release lock  
    pthread\_mutex\_lock(&mutex);  
    sum += mysum;  
    pthread\_mutex\_unlock(&mutex);  
  
    return (NULL);  
}  
  
int main(int argc, char \*\*argv) {  
    long i;  
    pthread\_t \*tid;  
  
    if (argc != 3) {  
       printf("Usage: %s <# of elements> <# of threads>\n",argv[0]);  
       exit(-1);  
    }  
  
    N = atoi(argv[1]); // no. of elements  
    size = atoi(argv[2]); // no. of threads  
  
    // allocate vector and initialize  
    tid = (pthread\_t \*)malloc(sizeof(pthread\_t)\*size);  
    a = (double \*)malloc(sizeof(double)\*N);  
    for (i=0; i<N; i++)  
      a[i] = (double)(i + 1);  
  
    // create threads  
    for ( i = 0; i < size; i++)  
      pthread\_create(&tid[i], NULL, compute, (void \*)i);  
  
    // wait for them to complete  
    for ( i = 0; i < size; i++)  
      pthread\_join(tid[i], NULL);  
  
    printf("The total is %g, it should be equal to %g\n",  
           sum, ((double)N\*(N+1))/2);  
  
    return 0;  
}

Here is another version of the above program that does not use the mutex locks: pthread\_sum2.c

### ****Homework****

Modify the pthread\_sum.c program to create a structure and pass the structure as argument to the thread creation function instead of using global variables a, sum, N, and size. You have to create a structure that contains the variables a and sum with type double, variables N and size with type int, and variable tid with type long or int. You have to create an instance of this structure specific to each thread and pass the structure as an argument to the corresponding thread creation function. Test the program for different values of N and number of threads and make sure that the result is correct.

### ****Submission****

You are required to submit the lab solution to Canvas

**1. To upload solution to Canvas**: Upload the C source code, PDF version of C source code, and README.txt file to Canvas. Do not include any executable or object files.

Use the following command to create PDFs of your source code.(replace the <Source\_code\_File\_name> and <Output\_Source\_code\_File\_name> with your C source code file name):

$enscript <Source\_code\_File\_Name>.c -o - | ps2pdf - <Output\_Source\_code\_File\_Name>.pdf