// simple program to demonstrate multiple POSIX threads and race condition

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define MAX 1000000 // smaller values means no race condition

// very large value results in occurrence of race condition

int counter = 0 ; // global counter which will be incremented by two threads

// define a function to increment counter; function will be called by both the threads

void \* count(void \*a) {

int i, tmp ;

for ( i=0; I < MAX; i++) {

tmp = counter ;

tmp = tmp + 1 ;

counter = tmp ; // just incrementing counter in a circuitous way

}

printf(“ Counter = %d\n”, counter);

}

// main function creates two threads and each thread increments counter

// through count function; in effect the final value of counter should be 2 \* MAX

Int main () {

pthread\_t thread1,thread2 ; // thread identifiers

pthread\_create(&thread1,NULL,count,NULL); // create thread1 with default args

// thread1 associated with count function

//No args passed

pthread\_create(&thread1,NULL,count,NULL); // create thread2

// thread1 and thread2 automatically call

//counter()

pthread\_join(thread1,NULL); // wait for thread1 to complete

pthread\_join(thread2,NULL); // wait for thread2 to complete

printf(“Final count = %d\n”,counter);

return 0;

}

**Note:**

- compile using –lpthread option

- experiment with different MAX values, say 100, 1000, 1000000 and observe the results; if you get

erroneous values, provide reasons

- remove pthread\_join() and observe the results

// implementation of semaphore to eliminate race condition

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t mutex;

#define MAX 1000000 // smaller values no race condition

// very large value occurrence of race condition

int counter = 0 ; // global counter which will be incremented by two threads

// define a function to increment counter; function will be called by both the threads

void \* count(void \*) {

int i, tmp ;

sem\_wait(&mutex);

for ( i=0; I < MAX; i++) {

tmp = counter ;

tmp = tmp + 1 ;

counter = tmp ; // just incrementing counter in a circuitous way

}

sem\_post(&mutex);

printf(“ Counter = %d\n”, counter);

}

// main function creates two threads and each thread increments counter

// through count function; in effect the final value of counter should be 2 \* MAX

Int main () {

sem\_init(&mutex,0,1); // 0 indicates semaphore is for threads;

// 1 is the mutex value

pthread\_t thread1,thread2 ; // thread identifiers

pthread\_create(&thread1,NULL,count,NULL); // create thread1 will default args

// thread1 associated with count function, No args passed

pthread\_create(&thread1,NULL,count,NULL); // create thread2

// thread1 and thread2 automatically call counter()

pthread\_join(thread1,NULL); // wait for thread1 to complete

pthread\_join(thread2,NULL); // wait for thread2 to complete

printf(“Final count = %d\n”,counter);

sem\_destroy(&mutex);

return 0;

}

NOTE:

-experiment with the position of sem\_wait() and sem\_post()

Reference: Synchronizing Threads with POSIX Semaphores: http://www.csc.villanova.edu/~mdamian/threads/posixsem.html