MIDWESTERN STATE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

CMPS 4103- Introduction to Operating Systems

Fall semester 2022

Mini project #2 – Unix Threads - due date 10/25

Problem.

You will be working with a threads program. You will need to be creative and may try different approaches to test the parallelism between threads. Your mission is to have a global array in your code with 10000 integers, initialized with values i%257 for i = 1 to 10000 in the main program (array[i]=i%257). You will also create 5 threads (identified by tid values 1 to 5). Each one of them will add the values of the array with subscripts in the range (tid -1)*2000 to (tid*2000-1), to a global variable TOTAL, one value at a time (do not add them up before adding to the TOTAL, i.e., it should be coded as TOTAL = TOTAL + array[i]). For example, a thread with tid 2, should work the i range from 2000 to 3999. You will run the code initially without using semaphores and in a second run, with semaphores and report any difference in the results. Your program should print the TOTAL value after all threads have completed their work. You will hand in the printout of your source code containing the semaphores solution and the output values on the due date (no late work will be accepted). Use Cygwin or any other Unix/Linux system you may have access to. You can work in a group of at most 3 students. Make sure that your semaphore solution does not make the programs to run sequentially.

Hints: you may need to use the function pthread_join(), but you may not need conditional variables. gcc compiler should work and create a file a.exe, Executing a.exe>out.txt will redirect the output to the file out.txt. The next **two examples** should help with the coding.

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

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
      /* This function prints the thread's identifier and then exits. */
      printf("Hello World. Greetings from thread %d0, tid);
      pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
      /* The main program creates 10 threads and then exits. */
      pthread_t threads[NUMBER_OF_THREADS];
      int status, i;

      for(i=0; i < NUMBER_OF_THREADS; i++) {
            printf("Main here. Creating thread %d0, i);
            status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

            if (status != 0) {
                  printf("Oops. pthread_create returned error code %d0, status);
                  exit(-1);
            }
      }
      exit(NULL);
}
```

```
#include <stdio.h>
#include <pthread.h>
#define MAX 1000000000                          /* how many numbers to produce */
pthread_mutex_t the_mutex;
pthread_cond_t condc, condp;
int buffer = 0;                                 /* buffer used between producer and consumer */

void *producer(void *ptr)                       /* produce data */
{      int i;

       for (i= 1; i <= MAX; i++) {
            pthread_mutex_lock(&the_mutex);    /* get exclusive access to buffer */
            while (buffer != 0) pthread_cond_wait(&condp, &the_mutex);
            buffer = i;                         /* put item in buffer */
            pthread_cond_signal(&condc);        /* wake up consumer */
            pthread_mutex_unlock(&the_mutex);/* release access to buffer */
       }
       pthread_exit(0);
}

void *consumer(void *ptr)                       /* consume data */
{      int i;

       for (i = 1; i <= MAX; i++) {
            pthread_mutex_lock(&the_mutex);    /* get exclusive access to buffer */
            while (buffer ==0 ) pthread_cond_wait(&condc, &the_mutex);
            buffer = 0;                         /* take item out of buffer */
            pthread_cond_signal(&condp);        /* wake up producer */
            pthread_mutex_unlock(&the_mutex);/* release access to buffer */
       }
       pthread_exit(0);
}

int main(int argc, char **argv)
{
       pthread_t pro, con;
       pthread_mutex_init(&the_mutex, 0);
       pthread_cond_init(&condc, 0);
       pthread_cond_init(&condp, 0);
       pthread_create(&con, 0, consumer, 0);
       pthread_create(&pro, 0, producer, 0);
       pthread_join(pro, 0);
       pthread_join(con, 0);
       pthread_cond_destroy(&condc);
       pthread_cond_destroy(&condp);
       pthread_mutex_destroy(&the_mutex);
}
```