Wayne State University

CSC 4421 - Fall 2020
Computer Operating Systems Labs
Lab 06 - Semaphore

Instructors

Rui Chen - section 001

Tasks

1. Write a multithreaded program to add two 256x256 matrices. The program should use 8 threads, and 1-D data partitioning. Thread i is responsible for adding 1-D blocks composed of rows 32 i to 32 i + 31. Each thread should print the statement Thread i: Done when it completes the local summation (i is the thread id). The input matrices should have on each row integers from 1 to 256 in increasing order. The main thread is responsible for generating the initial matrices and printing the final result in a file called output.txt.

2. You must include semaphore to ensure only one thread will modify the result matrix at a time.

```
//HOMEWORK03 by RUI CHEN fb4138

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
//declare globe var, such as the three matrices and mutex lock
int matrix1[256][256], matrix2[256][256], result[256][256];

//to declare semaphore
sem_t mutex;

//this func is to write a matrix into a file called "output.txt"
int matrixOutput(int matrix[256][256]){
  FILE *fp=fopen("output.txt", "w+");
  if(fp==NULL){
    printf("could not open file!\n");
    exit(-1);
  }

  for(int i=0; i<256; i++){
    for(int j=0; j<256; j++){
      fprintf(fp,"%d ", matrix[i][j]);
```

```
    }
    fprintf(fp,"\n");
  }
  fclose(fp);
}


void* matrixAdd(void *i){
  //first need to convert i datatype
  int num=(int)i;

  //put a semaphore so only one thread can modify the result matrix at once
  --------

  //this thread will do addition for rows from num*32 to num*32+31
  ???

  printf("Thread %d Done!\n", num);

  //to release the semaphore
  --------

  pthread_exit(NULL);
}


int main(){
  //to initialize semaphore
  --------

  //firstly initialize the two matrices
  //both matrix 1 and 2 will be like:
  // 1,2,...,256
  // 1,2,...,256
  // ...
  // 1,2,...,256
  for(int i=0; i<256; i++){
    for(int j=0; j<256; j++){
      matrix1[i][j]=j+1;
      matrix2[i][j]=j+1;
      result[i][j]=0;
    }
  }

  //secondly to create the threads and assign jobs for them
  pthread_t thread[8];
  for(int i=7; i>=0; i--) {
    --------
  }

  //thirdly use pthread join to make sure all threads are done before the main thread
  for(int i=7; i>=0; i--) {
    ---------
  }

  //lastly to write the result into output.txt
  matrixOutput(result);

  return 0;
}
```

lab06.c