Brian Duenas

CSE 460

Professor Dr. Tong Lai Yu

60/60 Total points

HW 3

Q1:

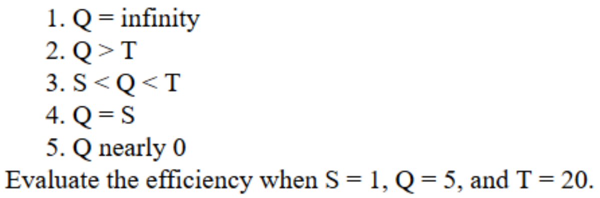
The aging algorithm with a = 1/2 is being used to predict run times. The previous four runs, from oldest to most recent are 40, 20, 40, and 15 msec. What is the next run time?

A:

($15/2 + 40/4 + 20/8 + 40/16 + T_0/40$) = 25msec

Q2:

Measurement of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). For round robin scheduling with quantum Q, give a formula for the CPU efficiency for each of the following.



1. Q = infinity
2. Q > T
3. S < Q < T
4. Q = S
5. Q nearly 0

Evaluate the efficiency when S = 1, Q = 5, and T = 20.

A:

1) T/(T+S)

2) T/(T+S)

3) T/(T+(ST/Q)) → Q/(Q+S)

4) Q/(Q+Q)

5) Efficiency goes to 0 as Q goes to 0

Q3:

Write a multithreaded program using SDL threads or POSIX threads. The program uses a number of threads to multiply two matrices. The multiplication of an M X L matrix A and an L X N matrix B gives an M X N matrix C, and is given by the formula,

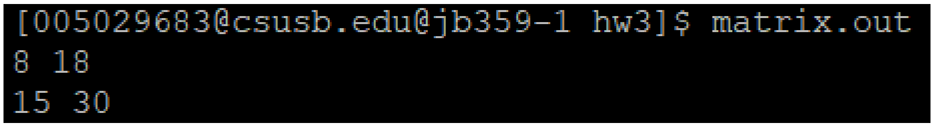
$$C_{ij} = \sum_{k=0}^{L-1} A_{ik} B_{kj} \quad 0 \le i < M,\ 0 \le j < N$$

Basically, each element C_{ij} is the dot product of the i-th row vector of A with the j-th column vector of B. The program uses one thread to calculate a dot product. Therefore, it totally needs M x N threads to calculate all the elements of matrix C.

A:

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

#define M 2
#define K 2
#define N 2
#define NUM_THREADS 5

int A[M][K] = { { 2,2 },{ 3,4 } };
int B[K][N] = { { 1,6 },{ 3,3 } };
int C[M][N];

struct v {
	int i; /* row */
	int j; /* column */
};
void *runner(void *param); /* the thread */
int main(int argc, char *argv[]) {
	int i, j, count = 0;
	for (i = 0; i < M; i++) {
		for (j = 0; j < N; j++) {
			//Assign a row and column for each thread
			struct v *data = (struct v *) malloc(sizeof(struct v));
			data->i = i;
			data->j = j;
			/* Now create the thread passing it data as a parameter */
			pthread_t tid;       //Thread ID
			pthread_attr_t attr; //Set of thread attributes
								//Get the default attributes
			pthread_attr_init(&attr);
			//Create the thread
			pthread_create(&tid, &attr, runner, data);
			//Make sure the parent waits for all thread to complete
			pthread_join(tid, NULL);
			count++;
		}
	}
	//Print out the resulting matrix
	for (i = 0; i < M; i++) {
		for (j = 0; j < N; j++) {
			printf("%d ", C[i][j]);
		}
		printf("\n");
	}
}
//The thread will begin control in this function
void *runner(void *param) {
	struct v *data = param; // the structure that holds our data
	int n, sum = 0; //the counter and sum

					//Row multiplied by column
	for (n = 0; n< K; n++) {
		sum += A[data->i][n] * B[n][data->j];
	}
	//assign the sum to its coordinate
	C[data->i][data->j] = sum;

	//Exit the thread
	pthread_exit(0);
}
```

Output:



```
[005029683@csusb.edu@jb359-1 hw3]$ matrix.out
8 18
15 30
```

Q4:

In the class the implementation of the readers-writers problem using SDL threads has been presented. However, the read and write tasks of the reader thread and the writer thread are not given. Implement these tasks as reading and writing of a file named counter.txt, which contains an integer counter.

A **reader** thread

reads the counter from the file, and
prints out its thread name and the value of the counter.

A **writer** thread

increments the value of the counter in the file,
prints out its thread name and the new value of the counter.

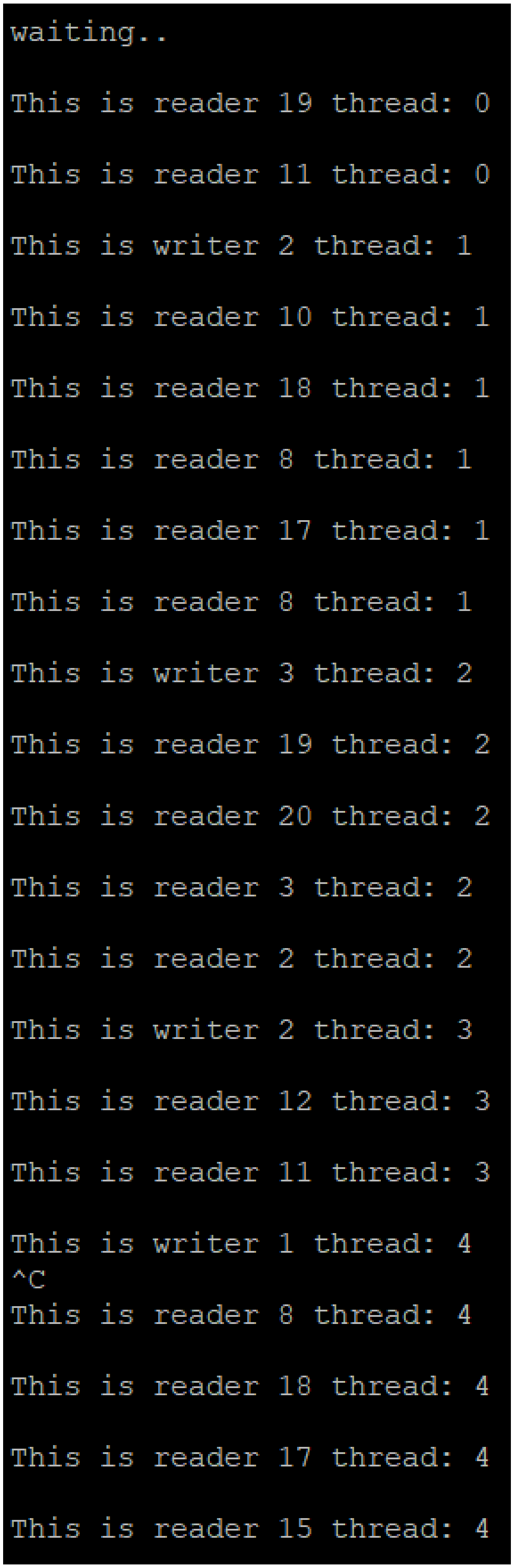
Each thread repeats its task indefinitely in a random amount of time between 0 and 3000 ms. Your main program should create 20 reader threads and 3 writer threads.

A:

```
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <signal.h>
#include <unistd.h>

using namespace std;

SDL_bool condition = SDL_FALSE;
SDL_mutex *mutex;
SDL_cond *readerQueue;   //condition variable
SDL_cond *writerQueue;   //condition variable

int readers = 0;
int writers = 0;
int count = 0;
bool quit = false;
int reader(void *data)
{
       while (!quit) {
              SDL_Delay(rand() % 3000);
              SDL_LockMutex(mutex);
              while (!(writers == 0))
                     SDL_CondWait(readerQueue, mutex);

              readers++;

              SDL_UnlockMutex(mutex);
              //read

              SDL_LockMutex(mutex);
              printf("\nThis is %s thread: %d\n", (char *)data, count);
              if (--readers == 0)
                     SDL_CondSignal(writerQueue);
              SDL_UnlockMutex(mutex);
       }
}
```

```
int writer(void *data)
{
        while (!quit) {
                SDL_Delay(rand() % 3000);
                SDL_LockMutex(mutex);
                while (!((readers == 0) && (writers == 0)))
                        SDL_CondWait(writerQueue, mutex);

                writers++;

                SDL_UnlockMutex(mutex);
                //write

                SDL_LockMutex(mutex);
                writers--;        //only one writer at one time
                count++;
                printf("\nThis is %s thread: %d\n", (char *)data, count);
                SDL_CondSignal(writerQueue);
                SDL_CondBroadcast(readerQueue);
                SDL_UnlockMutex(mutex);
        }
}

void func(int sig)
{
        quit = true;
}

// note: error checking codes omitted
int main()
{
        SDL_Thread *idr[20], *idw[3];                       //thread identifiers
        char *rnames[20] = { (char *) "reader 1", (char *)"reader 2", (char *)"reader 3",(char
*) "reader 4",(char *) "reader 5",(char *) "reader 6",(char *) "reader 7",
                (char *) "reader 8",(char *) "reader 9",(char *) "reader 10",(char *) "reader
11",(char *) "reader 12",(char *) "reader 13",(char *) "reader 14",(char *) "reader 15",
                (char *) "reader 16",(char *) "reader 17",(char *) "reader 18",(char *) "reader
19",(char *) "reader 20" }; //names of threads

        char *wnames[] = { (char *) "writer 1", (char *) "writer 2", (char *) "writer 3" };
//names of threads

        mutex = SDL_CreateMutex();
        readerQueue = SDL_CreateCond();
        writerQueue = SDL_CreateCond();
        for (int i = 0; i < 20; i++) {
                idr[i] = SDL_CreateThread(reader, rnames[i]);
        }
        for (int j = 0; j < 3; j++) {
                idw[j] = SDL_CreateThread(writer, wnames[j]);
        }
        printf("\nwaiting..\n");
        (void)signal(SIGINT, func);       //catch terminal interrupts
                                                                    //wait for the threads to
exit
        for (int i = 0; i < 20; i++) {
                SDL_WaitThread(idr[i], NULL);
        }
        for (int j = 0; j < 3; j++) {
                SDL_WaitThread(idw[j], NULL);
        }

        SDL_DestroyCond(readerQueue);
        SDL_DestroyCond(writerQueue);
        SDL_DestroyMutex(mutex);
        return 0;
}
```

Output:



```
waiting..

This is reader 19 thread: 0

This is reader 11 thread: 0

This is writer 2 thread: 1

This is reader 10 thread: 1

This is reader 18 thread: 1

This is reader 8 thread: 1

This is reader 17 thread: 1

This is reader 8 thread: 1

This is writer 3 thread: 2

This is reader 19 thread: 2

This is reader 20 thread: 2

This is reader 3 thread: 2

This is reader 2 thread: 2

This is writer 2 thread: 3

This is reader 12 thread: 3

This is reader 11 thread: 3

This is writer 1 thread: 4
^C
This is reader 8 thread: 4

This is reader 18 thread: 4

This is reader 17 thread: 4

This is reader 15 thread: 4
```