George Suarez

CSE 460

Homework 3

1. 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?

   (((40 + 20) / 2 + 40) / 2 + 15) / 2
   = ((30 + 40) / 2 + 15) /2
   = (35 + 15) / 2 = **25**

2. 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.
   a. Q = infinity
   b. Q > T
   c. S < Q < T
   d. Q = S
   e. Q nearly 0

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

   a. T / (T + S) = 20 / (20 + 1) = 20 / 21 = **.95 => 95%**
   b. T / (T + S)) = 20 / (20 + 1) = 20 / 21 = **.95 => 95%**
   c. T / (T + (ST / Q)) = 20 / (20 + (1 * 20 / 25)) = 20 / (20 + .8) = **.96 => 96%**
   d. Q / (Q + Q) = 5 / (5 + 5) = 5 / 10 = **.50 => 50%**
   e. Efficiency goes to zero as Q goes to 0.

3. 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,

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

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.

sdl_matrix.cpp:

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_thread.h>
#include <vector>
#include <ctime>
#include <iostream>

using namespace std;

int matrixA[3][2] = {{5, 4}, {2,6}, {9, 2}};
int matrixB[2][3] = {{5,6,2}, {4, 2, 8}};
int matrixC[3][3] = {{0, 0, 0},{0, 0, 0},{0, 0, 0}};

int dotProduct(void *data)
{
    char *threadname = (char *)data;


    for (int row = 0; row < 3; row++)
    {
        for (int col = 0; col < 3; col++)
        {
            for (int product = 0; product < 2; product++)
            {
                matrixC[row][col] += matrixA[row][product] * matrixB[product][col];
            }
        }
    }
        return 0;
}

void printMatrixA(int matrix[][2])
{
    cout << "Matrix A: " << endl;
    for (int row = 0; row < 3; row++)
    {
        for (int col = 0; col < 2; col++)
        {
            cout << matrixA[row][col] << " ";
        }
        cout << endl;
    }

    cout << endl;
}

void printMatrixB(int m[][3])
{
    cout << "Matrix B: " << endl;
    for (int row = 0; row < 2; row++)
    {
        for (int col = 0; col < 3; col++)
        {
            cout << matrixB[row][col] << " ";
        }
        cout << endl;
    }

    cout << endl;
}

void printMatrixC(int m[][3])
{
    cout << "Matrix C: " << endl;
    for (int row = 0; row < 3; row++)
    {
        for (int col = 0; col < 3; col++)
        {
            cout << matrixC[row][col] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

int main()
{
    SDL_Thread *sumThread = SDL_CreateThread(dotProduct, "Sum Thread", (void *)"Dot Product Thread");

    if (sumThread == NULL)
    {
        cout << "SDL_CreateThread failed: \n" << SDL_GetError() << endl;
    }
    else
    {
        int returnValue;
        SDL_WaitThread(sumThread, &returnValue);
        printMatrixA(matrixA);
        printMatrixB(matrixB);

        cout << "Equals to ";
        printMatrixC(matrixC);
        cout << endl;
    }
    return 0;
}
```

Output:

```
georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 3 on master*
$ ./sdl_matrix
Matrix A:
5 4
2 6
9 2

Matrix B:
5 6 2
4 2 8

Equals to Matrix C:
41 38 42
34 24 52
53 58 34
```

4. *Sdl_reader_writer.cpp:*

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_thread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <signal.h>
#include <unistd.h>
#include <iostream>
#include <fstream>

using namespace std;

SDL_bool condition = SDL_FALSE;
SDL_mutex *mutex1;
SDL_cond *readerQueue; //condition variable
SDL_cond *writerQueue; //condition variable

int readerCount = 0;
int writerCount = 0;
bool quit = false;

string fileName = "counter.txt";

int reader(void *data)
{

    while (!quit)
    {
        SDL_Delay(rand() % 3000);
        SDL_LockMutex(mutex1);
        while (!(writerCount == 0))
            SDL_CondWait(readerQueue, mutex1);

        readerCount++;

        SDL_UnlockMutex(mutex1);
        //read
        int count = -1;
        ifstream inFile;
        inFile.open(fileName.c_str());
        if (inFile.good())
        {
            inFile >> count;
            inFile.close();
        }
        SDL_LockMutex(mutex1);
        printf("\nThis is %s thread: %d\n", (char *)data, count);
        printf("Counter value: %d\n", count);
        if (--readerCount == 0)
            SDL_CondSignal(writerQueue);
        SDL_UnlockMutex(mutex1);
    }
```

```
}

int writer(void *data)
{
    while (!quit)
    {
        SDL_Delay(rand() % 3000);
        SDL_LockMutex(mutex1);
        while (!((readerCount == 0) && (writerCount == 0)))
            SDL_CondWait(writerQueue, mutex1);

        writerCount++;

        SDL_UnlockMutex(mutex1);
        int count = -1;
        ifstream inFile;
        inFile.open(fileName.c_str());
        if (inFile.good())
        {
            inFile >> count;
            inFile.close();
        }

        ofstream outFile;
        outFile.open(fileName.c_str());
        if (outFile.good())
        {
            outFile << count;
            outFile.close();
        }

        SDL_LockMutex(mutex1);
        writerCount--; //only one writer at one time
        count++;
        printf("\nThis is %s thread: %d\n", (char *)data, count);
        printf("Counter value: %d\n", count);
        SDL_CondSignal(writerQueue);
        SDL_CondBroadcast(readerQueue);
        SDL_UnlockMutex(mutex1);
    }
}

int main()
{
    SDL_Thread *idr[20], *idw[3]; //thread identifiers
    char readerNames[20][10];
    char writerNames[3][10];

    for (int i = 0; i < 20; i++)
    {
        cout << readerNames[i] << "Reader: " << i + 1 << endl;
        idr[i] = SDL_CreateThread(reader, "Reader Thread", readerNames[i]);
    }

    for (int i = 0; i < 3; i++)
    {
        cout << writerNames[i] << "Writer: " << i + 1 << endl;
        idw[i] = SDL_CreateThread(writer, "Writer Thread", writerNames[i]);
    }

    readerQueue = SDL_CreateCond();
    writerQueue = SDL_CreateCond();

    for (int i = 0; i < 20; i++)
    {
        SDL_WaitThread(idr[i], NULL);
    }

    for (int i = 0; i < 3; i++)
    {
        SDL_WaitThread(idw[i], NULL);
    }

    SDL_DestroyCond(readerQueue);
    SDL_DestroyCond(writerQueue);
    SDL_DestroyMutex(mutex1);
    return 0;
}
```

Output:

```
georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 3 on master*
$ ./sdl_readers_writers
Reader: 1
Reader: 2
Reader: 3
Reader: 4
Reader: 5
Reader: 6
Reader: 7
Reader: 8
Reader: 9
Reader: 10
Reader: 11
Reader: 12
Reader: 13
Reader: 14
Reader: 15
Reader: 16
Reader: 17
Reader: 18
Reader: 19
Reader: 20
Writer: 1
Writer: 2
Writer: 3

This is  thread: 0
Counter value: 0

This is  thread: -1
```

```
Counter value: -1

This is  thread: 0
Counter value: 0

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: -1
Counter value: -1

This is  thread: 0
Counter value: 0

This is  thread: -1
Counter value: -1
```

```
        This is  thread: -1
        Counter value: -1

        This is  thread: 0
Counter value: 0
```