David Veronin
Homework 3

1) The aging algorithm with a = ½ 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?

Taking the four previous run times into consideration, the prediction is:

(((40 + 20)/2 + 40) / 2 + 15) / 2

= ((30 + 40) / 2 + 15) / 2

= (35 + 15) / 2

= **25**

Taking the previous two run times into consideration, the prediction is:

(40 + 15) / 2

= **27.5**

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:

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

Number of times switched = (T/Q) -> Time wasted switching = (T/Q) * S
Therefore the efficiency = T / (T + Time wasted switching) = T / (T + ST/Q)

a) S = 0-> CPU efficiency = T /T = 100%
b) S = 0 -> CPU efficiency = T / T = 100%
c) CPU Efficiency = T / ((ST/Q) + T) = (varies from 100% down to 50% depending on how much less Q is than T)
d) S = Q -> CPU efficiency = T / ((QT/Q) + T) = T / 2T = 50%
e) Q ~= 0 -> CPU efficiency = T / ((ST/Q) +T) = T / ~infinity = ~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,

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

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.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <iostream>


#include <SDL/SDL_thread.h>

#include <vector>

//global

int matrixA[3][2] = { {1, 2}, {5, 8}, {7, 12} };
int matrixB[2][3] = { {3, 14, 0}, {6, 10, 15} };
int matrixC[3][3] = { {0, 0, 0}, {0, 0, 0}, {0, 0, 0} };

using namespace std;

class matrix
{
public:
void printA (int m[][2]);
void printB (int m[][3]);
void printC (int m[][3]);

private:
int row;
int col;
int product;
};

/* This function is a thread entry point. */

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


cout << "This is " << threadname << endl;;


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


}
```

```
61  return 0;
62  }
63  //void print ( const vector<double> &v )
64  void matrix::printA (int m[][2])
65  {
66  cout << "Matrix A: " << endl;
67  for (row = 0; row < 3; row++)
68  {
69  for (col = 0; col < 2; col++)
70  cout << matrixA[row][col] << " ";
71  cout << endl;
72  }
73  cout << endl;
74  }
75  void matrix::printB (int m[][3])
76  {
77  cout << "Matrix B: " << endl;
78  for (row = 0; row < 2; row++)
79  {
80  for (col = 0; col < 3; col++)
81  cout << matrixB[row][col] << " ";
82  cout << endl;
83  }
84
85  cout << endl;
86  }
87  void matrix::printC (int m[][3])
88  {
89  cout << "Matrix C: " << endl;
90  for(row = 0; row < 3; row++)
91  {
92  for(col = 0; col < 3; col++)
93  {
94  cout << matrixC[row][col] << " ";
95  }
96  cout << endl;
97  }
98  }
99
100
101 //main
102
103
104 int main()
105 {
106 //initVectors();
107 matrix matrices;
108
109
110 SDL_Thread *sumThread;
111
112
113 sumThread = SDL_CreateThread( dotProduct, ( void *) "Dot Product Thread");
114
115
117   if (sumThread == NULL)
118   {
119   cout << "\nSDL_CreateThread failed: " << SDL_GetError() << end
120   }
121   else
122   {
123   // Wait for the thread to complete.
124   int returnValue;
125   SDL_WaitThread( sumThread, &returnValue);
126   cout << "Dot product of matrices A and B: " << endl;
127   matrices.printA (matrixA);
128   matrices.printB (matrixB);
129   cout << "equal matrix C: " << endl;
130   matrices.printC (matrixC);
131   cout << endl;
132   }
133
134   return 0;
```

```
Script started on Thu Feb  21 10:25:50 2017
[?1034hbash-3.2$ g++ -o mdp dot_product.cpp -lSDL
[?1034hbash-3.2$ ./mdp
This is Dot Product Thread
Dot product of matrices A and B:
Matrix A:
1 2
5 8
7 12

Matrix B:
3 14 0
6 10 15

equal matrix C:
Matrix C:
15 34 30
63 150 120
93 218 180

bash-3.2$
```

4) 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.

```
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include <sstream>
#include <iostream>

using namespace std;

//SDL_bool condition = SDL_FALSE;
SDL_mutex *mutex;
SDL_cond *readerQ; //condition variable
SDL_cond *writerQ; //condition variable
int readers = 0;
int writers = 0;
class RW
{
public:
int reader (void* data);
int writer (void* data);


};
int RW::reader (void* data)
{

while(1)
{
SDL_Delay(rand() % 3000);
SDL_LockMutex(mutex);
while(!(writers == 0))
{
SDL_CondWait(readerQ, mutex);
}
++readers;
SDL_UnlockMutex(mutex);
ifstream file("counter.txt");
if(file.good())
{
int count;
file >> count;
cout << *((string*)data) << " with value: " << count <<
endl;
file.close();
}
else
{
cout << "Unable to read counter.txt" << endl;
}
SDL_LockMutex (mutex);
//printf("\nThis is %s thread\n", (char *) data);
if(--readers == 0)
{
SDL_CondSignal (writerQ);
}
```

```
58  SDL_UnlockMutex (mutex);
59  }
60  }
61  int RW::writer (void* data)
62  {
63  while(1)
64  {
65  SDL_Delay (rand() % 3000);
66  SDL_LockMutex(mutex);
67  while (!(readers == 0) && !(writers == 0))
68  {
69  SDL_CondWait ( writerQ, mutex );
70  }
71  ++writers;
72  SDL_UnlockMutex (mutex);
73
74
75  int count = -1;
76  ifstream read("counter.txt");
77  if(read.good())
78  {
79  read >> count;
80  read.close();
81  }
82  else
83
84  {
85  cout << "write file failed" << endl;
86  }
87  ++count;
88  ofstream write("counter.txt", ios::trunc);
89  write << count;
90  write.close();
91
92
93  cout << *((string*)data) << " with value: " << count << endl;
94
95
96  SDL_LockMutex (mutex);
97  --writers; //only one writer at one time
98  SDL_CondSignal (writerQ);
99  SDL_CondSignal (readerQ);
100 SDL_UnlockMutex (mutex);
101 }
102 }
103 int main ()
104 {
105 RW result;
106 //thread identifiers
107 SDL_Thread* idr[20];
108 SDL_Thread* idw[3];
109
110
111 mutex = SDL_CreateMutex();
112 readerQ = SDL_CreateCond();
113 writerQ = SDL_CreateCond();
```

```
115   for (int i = 0; i < 3; ++i)
116   {
117   stringstream flavio;
118   flavio << "writer: " << i;
119   string* name = new string(flavio.str());
120   idw[i] = SDL_CreateThread(result.writer(void*)name);
121
122
123   }
124
125
126   for(int i = 0; i < 20; ++i)
127   {
128   stringstream flavio;
129   flavio << "reader: " << i;
130   string* name = new string(flavio.str());
131   idr[i] = SDL_CreateThread(result.reader(void*)name);
132   }
133   SDL_WaitThread(idw[0], NULL);
134   SDL_DestroyCond(readerQ);
135   SDL_DestroyCond(writerQ);
136   SDL_DestroyMutex(mutex);
137   return 0;
```

//g++ -o readwrite readwrite.cpp -lSDL –lpthread

```
reader: 15 with value: 343
reader: 7 with value: 343
reader: 18 with value: 343
reader: 6 with value: 343
reader: 14 with value: 343
reader: 4 with value: 343
reader: 13 with value: 343
reader: 4 with value: 343
reader: 19 with value: 343
reader: 15 with value: 343
reader: 16 with value: 343
writer: 2 with value: 344
writer: 1 with value: 345
reader: 18 with value: 345
reader: 8 with value: 345
reader: 7 with value: 345
reader: 17 with value: 345
reader: 4 with value: 345
reader: 14 with value: 345
reader: 13 with value: 345
reader: 11 with value: 345
reader: 1 with value: 345
reader: 0 with value: 345
writer: 1 with value: 346
reader: 10 with value: 346
reader: 17 with value: 346
reader: 6 with value: 346
reader: 18 with value: 346
reader: 2 with value: 346
writer: 0 with value: 347
reader: 3 with value: 347
writer: 0 with value: 348
reader: 8 with value: 348
reader: 15 with value: 348
reader: 5 with value: 348
reader: 9 with value: 348
reader: 18 with value: 348
reader: 16 with value: 348
```

Conclusion: After completing all parts, I believe I deserve full credit.