1. (10 points)
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?

If take all four previous run times into consideration, the prediction is:

$$= \frac{\frac{(40 + 20)}{2} + 40}{2} + \frac{15}{2}$$

$$= \frac{\frac{30 + 40}{2} + 15}{2}$$

$$= \frac{35 + 15}{2}$$

$$= 25$$

Or if take only two previous run times into consideration, the prediction is:

$$= \frac{40 + 15}{2}$$

$$= \frac{27.5}{2}$$

2. (10 points)
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. $Q = \text{infinity}$
2. $Q > T$
3. $S < Q < T$
4. $Q = S$
5. $Q \text{ nearly 0}$
# of times switched = (T/Q) -> Time wasted switching = (T/Q) * S
so, 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 than T Q is)
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. (20 points)
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.

Code: For my code I adopted class in order to make this code with a better readability:

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <iostream>
/* We must include SDL_thread.h separately. */
#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++)
            {
            }
        }
    }
    return 0;
}

//void print ( const vector<double> &v )
void matrix::printA ( int m[][2])
{
    cout << "Matrix A: " << endl;
    for (row = 0; row < 3; row++)
    {
        for (col = 0; col < 2; col++)
            cout << matrixA[row][col] << " ";
        cout << endl;
    }
}

void matrix::printB ( int m[][3])
{
    cout << "Matrix B: " << endl;
    for (row = 0; row < 2; row++)
    {
        for (col = 0; col < 3; col++)
            cout << matrixB[row][col] << " ";
        cout << endl;
    }
}
```cpp
void matrix::printC (int m[][3])
{
    cout << "Matrix C: " << endl;
    for(row = 0; row < 3; row++)
    {
        for(col = 0; col < 3; col++)
        {
            cout << matrixC[row][col] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

/****************************
main
******************************/

int main()
{
    //initVectors();
    matrix matrices;

    SDL_Thread *sumThread;

    sumThread = SDL_CreateThread( dotProduct, ( void *) "Dot Product Thread");

    if (sumThread == NULL)
    {
        cout << "
SDL_CreateThread failed: " << SDL_GetError() << endl;
    }
    else
    {
        // Wait for the thread to complete.
        int returnValue;
        SDL_WaitThread( sumThread, &returnValue);
        cout << "Dot product of matrices A and B: " << endl;
        matrices.printA (matrixA);
        matrices.printB (matrixB);
        cout << "equal matrix C: " << endl;
        matrices.printC (matrixC);
        cout << endl;
    }
    return 0;
}
```
Screenshots continue:

```
flavio@flavio-Dell-System-XPS-L502X ~/460 $ ls
460hwk3.odt dot_product.cpp mdp SDL-1.2.15 SDL-1.2.15.tar.gz SDL2-2.0.3
flavio@flavio-Dell-System-XPS-L502X ~/460 $ g++ -o mdp dot_product.cpp -lSDL
flavio@flavio-Dell-System-XPS-L502X ~/460 $ ./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
93 210 180
```

Being working on this assignemnt for at least 3 days.
4. (20 points)

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.

---

**Code:**

```c++
/**************************************************************************
*    Flavio dos Santos - Ross
*           Compile:  g++ -o  mrw my_readers_writers.cpp -lSDL -lpthread
**************************************************************************/ 

#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 "
        file.close();
    }
else
    {
        cout "Unable to read counter.txt "
    }
    SDL_LockMutex (mutex);
    //printf("\nThis is %s thread\n", (char *) data);
    if(--readers == 0)
    {
        SDL_CondSignal (writerQ);
    }
    SDL_UnlockMutex (mutex);
}

int RW::writer (void* data)
{
    while(1)
    {
        SDL_Delay (rand() % 3000);
        SDL_LockMutex(mutex);
        while (!readers == 0) && !writers == 0))
        {
            SDL_CondWait ( writerQ, mutex );
        }
        ++writers;
        SDL_UnlockMutex (mutex);
        int count = -1;
        ifstream read("counter.txt");
        if(read.good())
        {
            read >> count;
            read.close();
        }
else
{
    cout << "write file failed" << endl;
}
++count;

ofstream write("counter.txt", ios::trunc);
write << count;
write.close();

cout << *((string*)data) << " with value: " << count << endl;
SDL_LockMutex (mutex);
    // only one writer at one time
SDL_CondSignal (writerQ);
SDL_CondSignal (readerQ);
SDL_UnlockMutex (mutex);
}

int main ()
{
    RW result;
    // thread identifiers
    SDL_Thread* idr[20];
    SDL_Thread* idw[3];

    mutex = SDL_CreateMutex();
    readerQ = SDL_CreateCond();
    writerQ = SDL_CreateCond();

    for (int i = 0; i < 3; ++i)
    {
        stringstream flavio;
        flavio << "writer: " << i;
        string* name = new string(flavio.str());
        idw[i] = SDL_CreateThread(result.writer(void*)name);
    }

    for (int i = 0; i < 20; ++i)
    {
        stringstream flavio;
        flavio << "reader: " << i;
        string* name = new string(flavio.str());
        idr[i] = SDL_CreateThread(result.reader(void*)name);
    }

    SDL_WaitThread(idw[0], NULL);
    SDL_DestroyCond(readerQ);
    SDL_DestroyCond(writerQ);
    SDL_DestroyMutex(mutex);
    return 0;
Program runs and compiles: - counter value 300