Source Code

**HeadMovement.cpp**

//Model data, and methods for determining the seek time of a disk

#pragma once

#ifndef HEAD\_MOVEMENT\_CPP

#define HEAD\_MOVEMENT\_CPP

#include <cmath> //for power function

static const double xMax = 8057; //Cylinders

static const double C = 9.1; //GB, disk capacity

static const double N = 7200; //RPM

static const double xStar = 1686; //Number cyl before max speed

static const double t = 1.5455; //ms, min seek time

static const double c = 0.3197; //ms, second cylinder increment time

static const double r = 0.3868; //

static const double calculateTime(double x); //calculates time to traverse x cylinders

static const double calculateTime(int x);

static const double calculateTime(double x) //using a double instead of int to prevent type conversion

{

if (x == 0) return 0;

else if (x <= xStar) return t + c\*pow(x - 1.0, r);

else return c\*r\*(x - xStar) / pow(xStar - 1.0, 1.0 - r) + t + c\*pow(xStar - 1.0, r);

}

static const double calculateTime(int x) //this converts int to double before operations

{

double xDouble = double(x);

if (xDouble == 0) return 0;

else if (xDouble <= xStar) return t + c\*pow(xDouble - 1.0, r);

else return c\*r\*(xDouble - xStar) / pow(xStar - 1.0, 1.0 - r) + t + c\*pow(xStar - 1.0, r);

}

static const double calculateTimeRootEquation(double x) //The square root model

{

return pow(x / xMax, .5);

}

static const double calculateTimeRootEquation(int x) //The square root model

{

return \*pow( double(x) / xMax, .5);

}

#endif

**QueueGeneration.h**

//Queue Generation And Shortest Seeking Operation

#pragma once

#ifndef QUEUE\_GENERATION\_H

#define QUEUE\_GENERATION\_H

#include <ctime> //for seeding values

#include <cstdlib> //for rand

#include <iostream>

class Queue

{

private:

friend std::ostream &operator<<(std::ostream&, Queue &);

int size;

int \*dq;

int counter;

int xMax;

public:

Queue(int Q, int xMax);

int X; //HEAD POSITION;

int moveToNext();

};

#endif

**QueueGeneration.cpp**

//Queue Generation And Shortest Seeking Operation

//Values are uniformly distributed

#include "QueueGeneration.h"

Queue::Queue(int Q, int xMax)

{

srand(NULL);

size = Q;

dq = new int[ size ];

this -> xMax = xMax;

X = rand() % xMax + 1; //head generation

for (counter = 0; counter < size; counter++)

{

dq[counter] = rand() % xMax + 1; //place random numbers into the array

}

}

std::ostream &operator<<(std::ostream &output, Queue &queue)//output

{

for (int counter = 0; counter < queue.size; counter++)

{

output << queue.dq[counter] << " ";

}

output << "\n";

return output;

}

/\*==================================================

\* moveToNext() : Int

\* Returns: The distance the head moved

\*

\* Moves the head from current position to next position

\* utilizing a shortest seek time first implementation.

\* Once the head is moved, the cylinder location is

\* then removed and another location is added using

\* a random number generator.

==================================================\*/

int Queue :: moveToNext()

{

int smallestDistance = abs(X - dq[0]); //Just so it is initialized

int smallestIndex = 0;

if (smallestDistance == 0) //If there is no distance, no need to check

{ //for the closest location

dq[0] = rand() % xMax + 1;

return smallestDistance;

}

for (counter = 0; counter < size; counter++) //Loop to find closest value to X

{

if ( abs(X - dq[counter]) < smallestDistance )

{

smallestDistance = abs(X - dq[counter]);

smallestIndex = counter;

}

}

X = dq[smallestIndex]; //Assign X

dq[smallestIndex] = rand() % xMax + 1; //Replace old value with new random value

return smallestDistance;

}

**DiskSimulation.cpp**

//The testing environment utilizing the disk model

#include <iostream>

#include "QueueGeneration.h"

#include "HeadMovement.cpp"

using namespace std;

const int XMAX = int(xMax);

static const int NUMBERTRIALS = 100000;

struct SeekStatistics //Holds data for various queue sizes

{

int queueSize;

double averageTime;

double averageDistance;

};

//Executes the simulation for a particular queue size and extracts data

static SeekStatistics simulate( int x)

{

static Queue \*queuePtr;

static int seekCounter;

static int seekDistance, totalSeekDistance;

static double seekTime, totalSeekTime;

static SeekStatistics simulatedSeekStatistics;

queuePtr = new Queue(x, XMAX);

totalSeekDistance = 0;

totalSeekTime = 0;

for (seekCounter = 0; seekCounter < NUMBERTRIALS; seekCounter++)

{

seekDistance = queuePtr->moveToNext();

totalSeekDistance += seekDistance;

seekTime = calculateTime(seekDistance);

totalSeekTime += seekTime;

}

simulatedSeekStatistics.queueSize = x;

simulatedSeekStatistics.averageTime = double(totalSeekTime) / double(NUMBERTRIALS);

simulatedSeekStatistics.averageDistance = double(totalSeekDistance) / double(NUMBERTRIALS);

return simulatedSeekStatistics;

}

//Output method

static void printSeekStatistics(SeekStatistics statisticsToPrint)

{

cout << "\nQueue Size: " << statisticsToPrint.queueSize

<< "\tAvg Seek Distance: " << statisticsToPrint.averageDistance

<< "\tAvg Seek Time: " << statisticsToPrint.averageTime;

}

int main()

{

SeekStatistics seekData[20];

for (int i = 0; i <= 20; i++) seekData[i] = simulate( i + 1 ); //Run the simulations

for (int i = 0; i < 20; i++) printSeekStatistics(seekData[i]); //Print results

cin.get(); //OS independant way of "pausing"

return 0;

}