**Homework #3**

**Disk as a Load-dependent Server**

Name: Peitong Shi

SFSU ID: 915604994

CSC641 – 01

Instructor: Dr. Jozo J. Dujmović

October 6th 2017

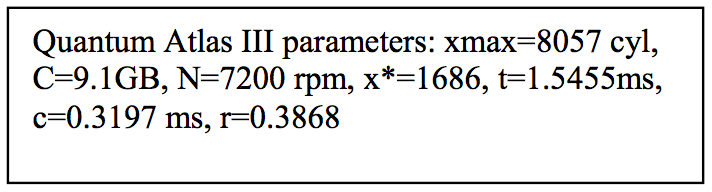
1. **Introduction:**

In all operation systems, the designers and the programmers have different algorithms to minimize the seek time in the I/O mechanism. In this program, we are going to design a disk seeking simulation as a load-dependent server. To achieve that, the idea is to have shortest seek time first (SSTF). We are going to simulate how I/O seek the disk, and it will always seek to the nearest position of the current position in the cylinder. Thus, if there is only one request in the disk queue, then there will be no optimization. However, with multiple requests in the disk queue, we always seek to the position which has shortest seek time, which allows the seek time to be less. The longer queue we have, the shorter seek time it would be.

1. **Workload:**

In order to find the average seek distance and average seek time, we are going to simulate a model of disk seeking. The star function of the program is **seek(int size):**

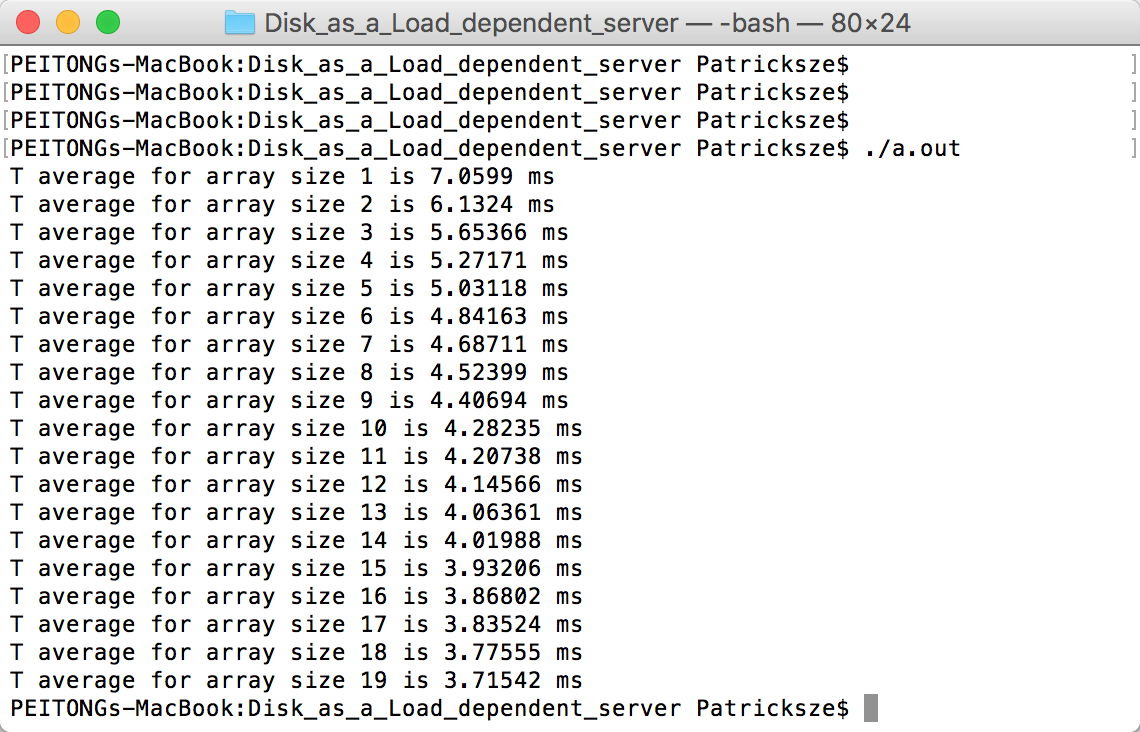
1. Create array **dq** with size **N** that represents the disk queue, and for N elements in the array, the numbers which are uniformly distributed represent the position of the cylinder of disk files. In this case, **N** is from 1-20.
2. First, to initialize a start position of head, and go through the whole queue to find the nearest position.
3. After that, “points” the head to the nearest position we found, then update the number of the same position and go through the array again to find the nearest position.
4. Repeat step 3 for 10000 times. And record the x, which is the position of the cylinder.
5. Compute the average seek time in every loop by following formula:

where 

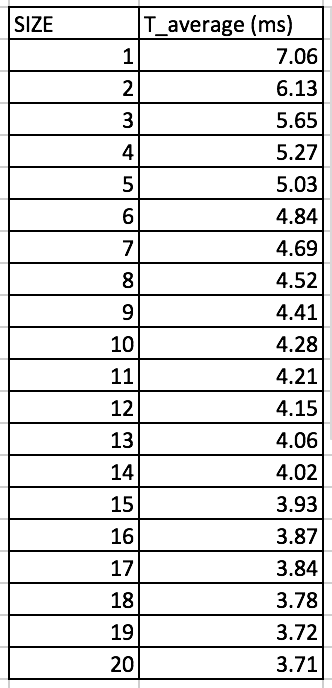
Then we sum all T(x) up, and divide it by 10000, and we get our result.

1. Notice that sometimes the values in the array can be the same since we may read a same file again and again.
2. Return the total seek time and we can compute the average time **T**, by divide it by 10000, which is in main() function.
3. **Measurement and output**

The output of my program including the result (Average seek time) is shown below:



1. **Plot and Graph**

****

According to the Monte Carlo Significance Test, having N = 10000 of repeats allow me to reserve 2 significant decimal digit. With the data on the right, the graph is drawn and shown below.

We can see that we the increase of the array size, the average seek time **T** is decreasing, which means that the program successfully simulate a disk process as a load-dependent server.

1. **Conclusion**

In conclusion, we successfully simulated a disk as a load-dependent server program, and we can see the conclusion that with bigger array’s size, the average seek time become less, which meets our expectation of this project.

In short, by using the idea of shortest seek time first (SSTF), the program shows significantly improvement of the seeking time.

1. **Appendix**
   1. **Source Code**

//

**// main.cpp**

**// Disk\_as\_a\_Load\_dependent\_server**

**//**

**// Created by PEITONG SHI on 04/10/2017.**

**// Copyright © 2017 PEITONG SHI. All rights reserved.**

**//**

**#include <iostream>**

**#include <stdlib.h>**

**#include <time.h>**

**#include <random>**

**static double C = 9.1;**

**static int N = 7200;**

**static int x\_star = 1686;**

**static double t = 1.5455; //ms**

**static double c = 0.3197; //ms**

**static double r = 0.3868;**

**static double x\_max = 8057;**

**using namespace std;**

**double urn()**

**{ //random # generator**

**return int(x\_max \* rand()/RAND\_MAX);**

**}**

**double seek(int size)**

**{**

**double T\_ave = 0; //Average seek time**

**double x = 0; //The distance from current to the next**

**double x\_record = 0; //For recording**

**double dq[size]; //disk queue**

**double now = urn(); //current position, initially the head position**

**int nowIndex = 0; //current position by index**

**for (int i = 0; i< 10000; i++)**

**{ //simulate 10000 times**

**for(int j = 0; j < size; j++)**

**{ //randomly fill in number**

**dq[j] = urn();**

**}**

**x = fabs(dq[0] - x); //Absolute value of distance**

**for(int k = 0; k < size; k++)**

**{ //go through the whole queue**

**x\_record = fabs(dq[k] - now); //record the absolute value of distance**

**if(x > x\_record){ //if previous distance is longer,**

**x = x\_record; //x points to the shorter one**

**nowIndex = k; //index changing to shorter one**

**}**

**}**

**now = dq[nowIndex]; //current head position is now dq[nowIndex]**

**dq[nowIndex] = urn(); //update the value of the served position**

**x = fabs(ceil(x)); //Make sure x is absolute value and round it.**

**if(x < 1) { x = 1; } //For preventing error (when distance =0)**

**if (x <= x\_star && x >= 1)**

**{ //based on pdf’s formula to get the T(x) every time and sum up**

**T\_ave += (t + c \* pow((x - 1), r));**

**}**

**else**

**{**

**T\_ave += ((c\*r\*(x-x\_star)/(pow((x\_star-1.0), (1.0-r))))+t+c\* pow((x\_star-1.0),r));**

**}**

**}**

**return T\_ave;**

**}**

**int main(int argc, const char \* argv[])**

**{**

**for (int i = 1; i <= 20; i++)//call function to pass 1-20 size of array**

**{**

**cout<<"T average for array size " <<i<<" is "<< seek(i)/10000<< " ms"<<endl;**

**}**

**}**