OPERATING SYSTEMS
CS342

Project 4

Ramazan Mert Cinar
21601985
Section 2

Part B)

Outline

In this paper, first, I am providing the output consisting of the average and standard deviation of the algorithms in Figure 1. After that, I am stating my observations on the disk scheduling algorithms. At the end, you can find my code in the Appendices.

Experimental Outputs

	St. Deviation	Average
FCFS	37251.94	1666900
SCAN	1356.35	7535
C-SCAN	1101.55	10682
LOOK	1355.7	7525
C-LOOK	1103.5	10668
SSTF	2322.01	5632

Figure1

FCFS

FCFS algorithm is an algorithm which is not optimized but serves according to the arrival time of the requests. Therefore, its average head movements and standard deviation is quite high when compared to the other algorithms.

SCAN

SCAN algorithm does not care about the priority of the request and basically moves through the end/beginning. Therefore, it takes at most two traversals of the cylinders. Hence, its average and standard deviation is much better than FCFS.

C-SCAN

C-SCAN algorithm is a derivation of SCAN algorithm. We basically run the SCAN algorithm in one direction and instead of going backwards, we move head to the beginning and run in the same direction. It generally provides better standard deviation but not necessarily better average.

LOOK

LOOK algorithm is a derivation of SCAN algorithm. We need to calculate the min and max values of the requests and then run the SCAN algorithm in the interval [min, max]. So, it gives slightly better result and it is because of the head position. In general, it gives almost the same results as SCAN algorithm, however, in the worst-case scenarios, LOOK algorithm is better than SCAN algorithm.

C-LOOK

C-LOOK algorithm is sort of a combination of C-SCAN and LOOK. It runs in an interval just like LOOK algorithm but instead of going backwards, it moves

the head to the beginning and runs towards the same direction. In general, it gives almost the same results as SCAN algorithm, however, in the worst-case scenarios, C-LOOK algorithm is better than C-SCAN algorithm.

SSTF

SSTF algorithm gives the best result among 6 of them. But it runs in longer time when compared to the others. It is because the algorithm calculates the nearest request each time.

APPENDICES

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>

int req[1000];
int headPos;

int mvFcfs = 0;
int mvScan = 0;
int mvCScan = 0;
int mvLook = 0;
int mvCLook = 0;
int mvSstf = 0;

int st[6][100];

void generateRandomly() {
   int i;
   for(i = 0; i < 1000; i++){
      int res = (int) ((5000) * ((double)rand() / RAND_MAX));
      if (res > 4999)
         res--;
      req[i] = res;
   }
}

void readFromFile(char* fileName) {
   FILE* fp;
   fp = fopen(fileName, "r");

   if (fp == NULL) {
      printf("%s", "invalid file");
   }

   int a, b;
```

```
    while ((fscanf(fp, "%d %d", &a, &b) != -1)) {
        assert(b < 5000 && a > 0 && a < 1001);
        req[a-1] = b;
    }

}

int findMax() {
    int max = req[0];
    int i;

    for (i = 1; i < 1000; i++){
        if (req[i] > max)
            max = req[i];
    }
    return max;
}

int findMin() {
    int min = req[0];
    int i;

    for (i = 1; i < 1000; i++){
        if (req[i] < min)
            min = req[i];
    }
    return min;
}

int findMaxLessThanHead() {
    int res = req[0];
    int i;

    for (i = 0; i < 1000; i++){
        if (req[i] > res && req[i] < headPos)
            res = req[i];
    }
    return res;
}
```

```
int findNearest(int base) {
   int res = req[0], i, pos = 0;

   for (i = 1; i < 1000; i++) {
      if(req[i] == -1)
         continue;
      if (abs(res - base) > abs(req[i] - base)) {
         res = req[i];
         pos = i;
      }
   }

   req[pos] = -1;
   return res;
}

void cLook() {
   int min = findMin();
   int max = findMax();
   int mv = 0;
   int lHeadPos = headPos;

   if (max < lHeadPos) {
      mvCLook = mv = lHeadPos - min + (max - min);
//       printf("%s:\t%d\n", "C-LOOK", mv);
      return;
   }

   if (min > lHeadPos) {
      mvCLook = mv = max - lHeadPos;
//       printf("%s:\t%d\n", "C-LOOK", mv);
      return;
   }

   mvCLook = mv = max - lHeadPos + (max - min) + findMaxLessThanHead();
//    printf("%s:\t%d\n", "C-LOOK", mv);
}

void look() {
   int min = findMin();
```

```
    int max = findMax();
    int mv = 0;
    int lHeadPos = headPos;

    if (lHeadPos < min) {
        mvCLook = mv = max - lHeadPos;
//        printf("%s:\t%d\n", "LOOK", mv);
        return;
    }

    if (max < headPos) {
        mvCLook = mv = headPos - min;
//        printf("%s:\t%d\n", "LOOK", mv);
        return;
    }

    mvLook = mv = max - lHeadPos + (max - min);
//    printf("%s:\t%d\n", "LOOK", mv);
}

void cScan() {
    int min = findMin();
    int max = findMax();
    int mv = 0;
    int lHeadPos = headPos;

    if (lHeadPos < min) {
        mv = max - lHeadPos;
    } else {
        int end = findMaxLessThanHead();
        mv = 4999 - headPos + 4999 + end;
    }

    mvCScan = mv;

//    printf("%s:\t%d\n", "C-SCAN", mv);
}

void scan() {
    int mv = 0;
```

```
    int min = findMin();
    int max = findMax();
    int lHeadPos = headPos;

    if (lHeadPos <= min){
        mv = max - lHeadPos;
    } else {
        mv = 4999 - headPos + (4999 - min);
    }

    mvScan = mv;

//    printf("%s:\t%d\n", "SCAN", mv);
}

void sstf() {
    int i, mv, lHeadPos;
    mv  = 0;
    lHeadPos = headPos;

    for (i = 0; i < 1000; i++){
        int tmp = findNearest(lHeadPos);
        mv = mv + abs(lHeadPos - tmp);
        lHeadPos = tmp;
    }

    mvSstf = mv;

//    printf("%s:\t%d\n", "SSTF", mv);
}

void fcfs() {
    int i;
    int mv = 0;
    int lHeadPos = headPos;
    for (i = 0; i < 1000; i++){
        int add = abs(lHeadPos - req[i]);
        lHeadPos = req[i];
        mv = mv + add;
    }
```

```
    mvFcfs = mv;

//    printf("%s:\t%d\n", "FCFS", mv);
}

double sdv(int i, double avg) {
    int j;
    double sum = 0;
    for (j = 0; j < 100; j++) {
        sum = sum + pow(st[i][j] - avg, 2);
    }

    return sqrt((sum/99));
}

void printRes(int i, double sd, double avg) {
    switch(i) {
        case 0: printf("%s:\tSTD: %lf  AVG: %lf\n", "FCFS", sd, avg); break;
        case 1: printf("%s:\tSTD: %lf  AVG: %lf\n", "SCAN", sd, avg); break;
        case 2: printf("%s:\tSTD: %lf  AVG: %lf\n", "C-SCAN", sd, avg); break;
        case 3: printf("%s:\tSTD: %lf  AVG: %lf\n", "LOOK", sd, avg); break;
        case 4: printf("%s:\tSTD: %lf  AVG: %lf\n", "C-LOOK", sd, avg); break;
        case 5: printf("%s:\tSTD: %lf  AVG: %lf\n", "SSTF", sd, avg); break;
        default:break;
    }
}

void deviationAndAvg(){
    int i, j, sum = 0;
    double sd, avg;
    for (i = 0; i < 6; i++) {
        for (j = 0; j < 100; j++){
            sum = sum + st[i][j];
        }

        avg = sum/100;
        sd = sdv(i, avg);

        printRes(i, sd, avg);
```

```
      sum = 0;
   }
}

int main(int argc, char* argv[]) {
   srand(time(NULL));

   if (argc == 1){
      int i;

      for (i = 0; i < 100; i++){
         headPos = (int) ((5000) * ((double)rand() / RAND_MAX));
         generateRandomly();

         fcfs();
         scan();
         cScan();
         look();
         cLook();
         sstf();

         st[0][i] = mvFcfs;
         st[1][i] = mvScan;
         st[2][i] = mvCScan;
         st[3][i] = mvLook;
         st[4][i] = mvCLook;
         st[5][i] = mvSstf;
      }

      deviationAndAvg();
   } else if (argc < 2 || argc > 3) {
      printf("%s", "Usage: diskscheduling <headpos> <inputfile>\t(inputfile is
optional)\n");
      return -1;
   } else if (argc == 2) {
      headPos = atoi(argv[1]);
      generateRandomly();
   } else if (argc == 3) {
      headPos = atoi(argv[1]);
```

```
        readFromFile(argv[2]);
    }

//    fcfs();
//    scan();
//    cScan();
//    look();
//    cLook();
//    sstf();
    return 0;
}
```