OPERATING SYSTEM\_CSE316

**Name:** Lovely Singh

**ID:** 11812005

**Email Address:** lovelys.s129@gmail.com

**GitHub Link:**

**QUESTION 11**

Write a C program to solve the following problem:

Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the SCAN disk-scheduling algorithms?

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#define CYLINDERS 5000

#define REQUESTS 1000

int start = 0;

int ran\_array[REQUESTS];

int test\_array[REQUESTS];

int\* sort\_array() {

int temp = 0, i = 0, j = 0;

for (i = 0; i < REQUESTS; ++i) {

for (j = i + 1; j < REQUESTS; ++j) {

if (ran\_array[i] > ran\_array[j]) {

temp = ran\_array[i];

ran\_array[i] = ran\_array[j];

ran\_array[j] = temp;

}

}

}

return ran\_array;

}

/\* First-Come-First-Serve (fcfs) starts from the index after the starting

index and continually adds the headmovement from the starting index in

order recieved. If at end of array, start from index zero and continually

add until starting index \*/

int fcfs(int \*ran\_array) {

int i = 0, head\_movement = 0, this\_start = ran\_array[start];

for(i = start; i < REQUESTS; i++) {

head\_movement += abs(ran\_array[i] - this\_start);

}

for(i = 0; i < start; i++) {

head\_movement += abs(this\_start - ran\_array[i]);

}

return head\_movement;

}

/\* Shortest-Seek-Time-First (SSTF) takes the current head position, and

adds the position closest to the current head. This new position now becomes

the head, then this system repeats.

First we sort the array. Then We have counters for above and below start

index that we decrement if used. Once these equal to REQUEST-2 (excluding

start index) we exit. \*/

int sstf(int \* ran\_array) {

ran\_array = sort\_array();

int small\_i = start-1, large\_i = start+1;

int small\_diff = 0, large\_diff = 0;

int head\_movement = 0, total = REQUESTS-2, new\_head = start, head\_value = ran\_array[start];

while(total >= 0) {

small\_diff = abs(ran\_array[new\_head] - ran\_array[small\_i]);

large\_diff = abs(ran\_array[large\_i] - ran\_array[new\_head]);

if(small\_diff < large\_diff) {

head\_movement += small\_diff;

new\_head = small\_i;

small\_i--;

} else {

head\_movement += large\_diff;

new\_head = large\_i;

large\_i++;

}

total--;

}

return head\_movement;

}

/\* SCAN - array is already sorted from sstf. SCAN starts from one left of start,

and continually goes down to zero (if included in randome array or not). Then

starts at one higher than start and continually goes up to highest value (not 5000) \*/

int scan(int \* ranArray) {

int i = 0, curr\_val = 0, sav\_val = ran\_array[start], difference = 0;

int head\_movement = 0, curr\_i = 0;

for(i = start-1; i >= 0; --i) {

curr\_val = ran\_array[i];

difference = abs(sav\_val - curr\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

/\* used to subtract value from zero, or just add same value \*/

head\_movement += sav\_val;

sav\_val = 0;

for(i = start+1; i < REQUESTS; i++) {

curr\_val = ran\_array[i];

difference = abs(curr\_val - sav\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

return head\_movement;

}

/\* Circular Scan (C-SCAN) - start at start index, increase to upper boundary

(even if no value at boundary), save boundary value, go to start boundary

(zero value) increase till last value before start value \*/

int cscan(int \* ranArray) {

int i = 0, curr\_val = 0, sav\_val = ran\_array[start], difference = 0;

int head\_movement = 0, curr\_i = 0, upper\_bound = 4999;

for(i = start+1; i < REQUESTS; i++) {

curr\_val = ran\_array[i];

difference = abs(sav\_val - curr\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

/\* add last val - upper bound, go to and add zero bounday (4999)\*/

head\_movement += upper\_bound - sav\_val;

sav\_val = 0;

head\_movement += 4999;

for(i = 0; i < start; i++) {

curr\_val = ran\_array[i];

difference = abs(curr\_val - sav\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

return head\_movement;

}

/\* Look - start from value above start, increase to highest value.

Then goes to value below start value and decreases until smallest value \*/

int look(int\* ranArray) {

int i = 0, curr\_val = 0, sav\_val = ran\_array[start], difference = 0;

int head\_movement = 0, curr\_i = 0;

for(i = start+1; i < REQUESTS; i++) {

curr\_val = ran\_array[i];

difference = abs(sav\_val - curr\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

for(i = start-1; i >= 0; --i) {

curr\_val = ran\_array[i];

difference = abs(curr\_val - sav\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

return head\_movement;

}

/\* C-Look - Starts from value after start value, goes to highest value,

then goes to smallest value and increases until value before start value \*/

int clook(int\* ranArray) {

int i = 0, curr\_val = 0, sav\_val = ran\_array[start], difference = 0;

int head\_movement = 0, curr\_i = 0;

for(i = start+1; i < REQUESTS; i++) {

curr\_val = ran\_array[i];

difference = abs(sav\_val - curr\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

for(i = 0; i < start; i++) {

curr\_val = ran\_array[i];

difference = abs(curr\_val - sav\_val);

head\_movement += difference;

sav\_val = curr\_val;

}

return head\_movement;

}

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

int i = 0;

start = atoi(argv[1]);

if(argc != 2) {

printf("Please compile program with starting index from 0-4999. Ex. ./diskAlgorithms 423\n");

exit(-1);

}

for(i = 0; i < REQUESTS; i++) {

ran\_array[i] = rand() % 5000;

}

printf("\nStart index: %d, start value: %d\n\n", start, ran\_array[start]);

printf("FCFS head movements: %d\n", fcfs(ran\_array));

printf("SSTF head movements: %d\n", sstf(ran\_array));

printf("SCAN head movements: %d\n", scan(ran\_array));

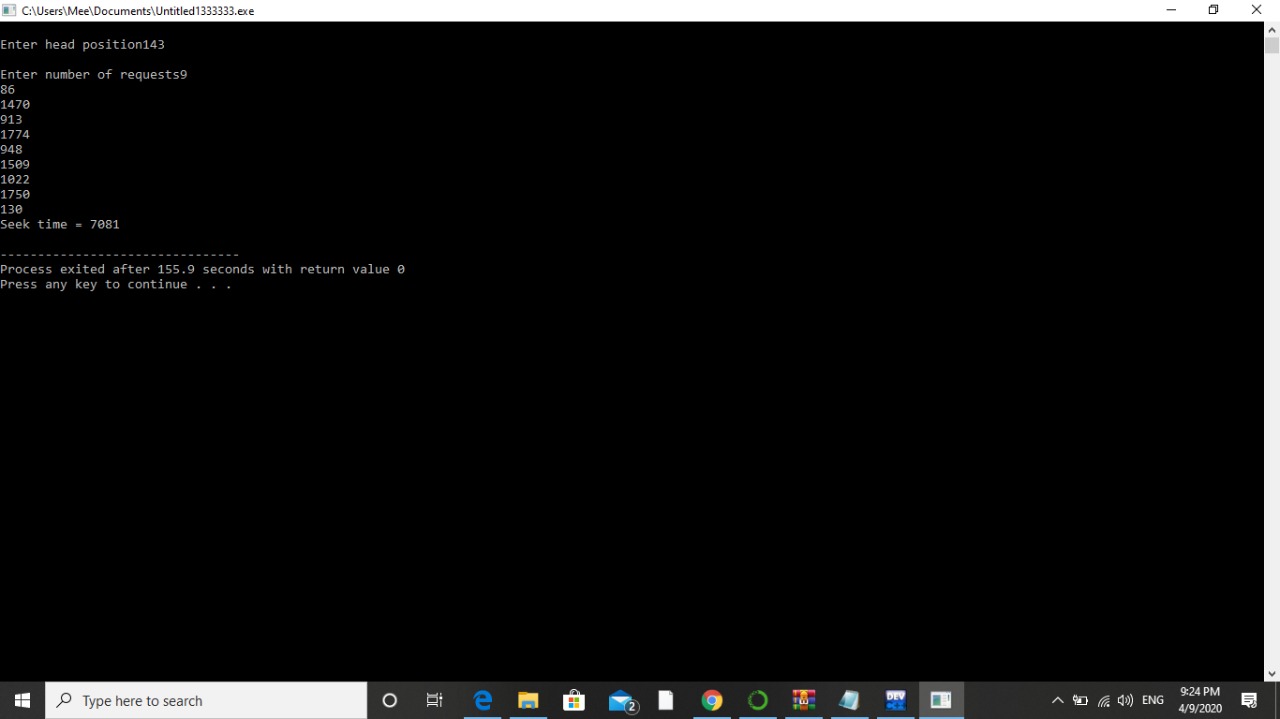
printf("CSCAN head movements: %d\n", cscan(ran\_array));

printf("LOOK head movements: %d\n", look(ran\_array));

printf("C-LOOK head movements: %d\n\n", clook(ran\_array));

return 0;

}

**OUTPUT:**

**ALGORITHM:**

FOR FNC. int scan(int \*ran\_array) :-

STEP 0: START

STEP 1 : DECLARE integer variables 'i', 'curr\_val' and 'difference' equal to 0.

STEP 2:DECLARE integer variable 'sav\_val' equal to 'ran\_array[start].

STEP 3: DECLARE integer variables ' head\_movement' and 'curr\_i' equal to 0

STEP 4: FROM STEP 4 TO STEP 9 till 'i' equal to 'start' is greater than equal to 0

STEP 5 :Assign 'curr\_val' equal to ran\_array[i]

STEP 6 :Assign 'difference' equal to absolute of 'sav\_val' subtract 'curr\_val'

STEP 7 : 'head\_movement' equal to the sum of 'head\_movement' and 'difference'

STEP 8 : Assign 'sav\_val' equal to 'curr\_val'

STEP 9: Increase value of 'i' by 1 and Go To Step 4

STEP 10: head\_movement' equal to the sum of 'head\_movement' and 'sav\_val'

STEP 11: Assign 'sav\_val' equal to 0

STEP 12: FROM STEP 12 TO STEP 17 till 'i' equal to 'start' added to 1 is less than REQUESTS

STEP 13: Assign 'curr\_val' equal to 'ran\_array[i]'

STEP 14: 'difference' equal to absolute of the 'curr\_val' subtract 'sav\_val'

STEP 15:head\_movement' equal to the sum of 'head\_movement' and 'difference'

STEP 16: Assign 'sav\_val' equal to 'curr\_val'

STEP 17:Increase value of 'i' by 1 and Go To Step 12

STEP 18: Return head\_movement

STEP 19: END

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

STEP 0: START

STEP 1: DECLARE integer variable 'i' equal to 0

STEP 2: Assign 'start' equal ro atoi(argv[1])

STEP 3: IF 'argc' is not equal to 2 then print "Please compile program with starting index from 0-4999.Ex ./diskAlgorithm 423" And exit

STEP 4: FROM STEP 4 TO STEP 6 till 'i' equal to 0 is less than REQUESTS

STEP 5: Assign 'ran\_array[i] equal to rand()%5000

STEP 6: Increase value of 'i' by 1 and Go to Step 4

STEP 7: Print " Start index , Start value " With values 'start' , 'ran\_array[start]'

STEP 8: Print msg "SCAN head movement" And call function scan(ran\_array)

STEP 9: Return 0

STEP 10: END

**DESCRIPTION:**

First Come First Serve(FCFS) is an operating system process scheduling algorithm and a network routing management mechanism that automatically executes queued requests and processes by the order of their arrival.

**QUETSION 14**

If a teacher is being served at the food mess and during the period when he is being served, another teacher comes, then that teacher would get the service (food) next. This process might continue leading to increase in waiting time of students to get food. Ensure in your program that the waiting time of students is minimized.

**CODE:**

#include<stdio.h>

int main()

{

int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;

float wtavg, tatavg;

printf("Enter the number of PROCESS in the queue --- ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

p[i] = i;

printf("Enter the Burst Time for process %d --- ", i);

scanf("%d",&bt[i]);

printf("teacher/student process (0/1) ? --- ");

scanf("%d", &su[i]);

}

for(i=0;i<n;i++)

{

for(k=i+1;k<n;k++)

{

if(su[i] > su[k])

{

temp=p[i];

p[i]=p[k];

p[k]=temp;

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=su[i];

su[i]=su[k];

su[k]=temp;

}

}

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

for(i=1;i<n;i++)

{

wt[i] = wt[i-1] + bt[i-1];

tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\t TEACHER/STUDENT PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

for(i=0;i<n;i++)

{

printf("\n%d \t\t %d \t\t %d \t\t %d \t\t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]);

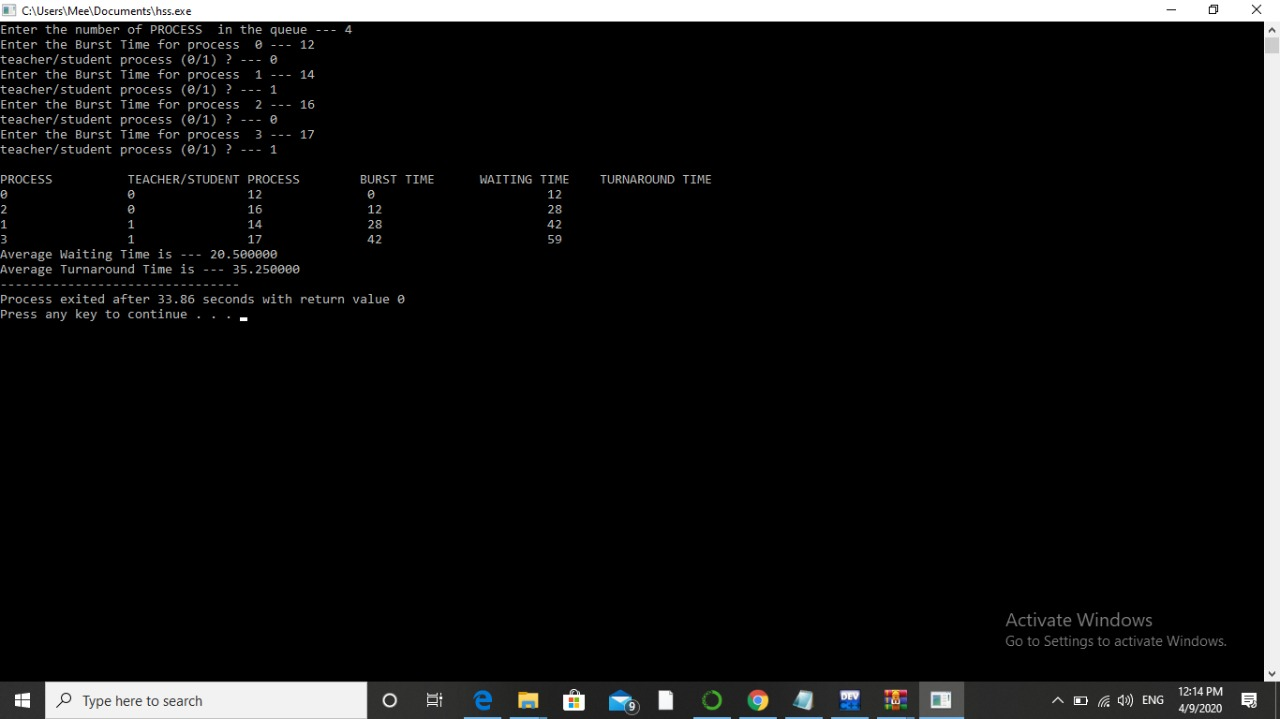
}

printf("\nAverage Waiting Time is --- %f",wtavg/n);

printf("\nAverage Turnaround Time is --- %f",tatavg/n);

return 0;

}

**OUTPUT:**

**ALGORITHM:**

1. Start
2. Declare integer type arrays p, bt, su, wt, tat with capacity of string 20 values.
3. Declare integer variables i, n and temp
4. Declare float type variables
5. Take input from the user in variable n
6. From step 6 to step 12 until i is greater than n
7. Store the value of i in p[i]
8. Print the message “Enter the burst time for the process” i
9. Take input from the user in bt[i]
10. Print message “teacher/student process(0/1)?”
11. Input value from the user in su[i]
12. Increment value of i by 1 and go to step 6
13. From step 13 to step 14 until i is greater than n where i is initially 0
    1. From 13.1 to 13.3 until is equivalent to i is greater than n
    2. If su[i] is greater than su[k] then interchange the value of p[i] with p[k]
    3. Increment value of i by 1 and go to 13.1
14. Increment value i by 1 and go to step 13
15. Assign wtarg equivalent wt[0] equivalent to 0
16. From step 16 to step 21 until i is equal to 1 is greater than n
17. wt[i] equal to the sum of wt[t-1] and bt[i-1]
18. tat[i] equal to sum of tat[i-1] and bt[i]
19. Wtarg equal to the sum of wt arg and wt[i]
20. Tatavg equal to the sum of tatavg and tat[i]
21. Increment value to i by 1 and go to step 16
22. Print message
23. From step 23 to step 25 until i equal to 0 is greater than n
24. Print values p[i], su[i], bt[i], wt[i], tat[i] below columns
25. Increment value of i by 1and go to step 23
26. Print message
27. Return 0
28. End .

**DESCRIPTION:**

The code can be implemented in designing a system software in real world efficiently. It can find its application in various restaurants and cafes where we can use this software in reading time to serve various customers with different statuses like normal or premium or VIP or etcetera where we can reduce the waiting time for VIP or premium package customers.

It can also be used in software like mess as mentioned in the question reducing the waiting time for students. It can also come handy in service center of various things where waiting time can be reduced for certain category of clients depending on individual needs.