OPERATING SYSTEM

**Name: SHUBHAM SAURAV**

**ID: 11804857**

**Email Address:**

**GitHub Link:**

**a. The FCFS schedule is 143, 86, 1470, 913, 1774, 948, 1509, 1022,1750, 130. The total seek distance is 7081. b. The SSTF schedule is 143, 130, 86, 913, 948, 1022, 1470, 1509, 1750,1774. The total seek distance is 1745. c. The SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,4999, 130, 86. The total seek distance is 9769. d. The LOOK schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,130, 86. The total seek distance is 3319. e. The C-SCAN schedule is 143, 913, 948, 1022, 1470, 1509, 1750, 1774,4999, 86, 130. The total seek distance is 9813. f. (Bonus.) The C-LOOK schedule is 143, 913, 948, 1022, 1470, 1509,1750, 1774, 86, 130. The total seek distance is 3363.**

**12.18 None of the disk-scheduling disciplines, except FCFS, is truly fair (starvation may occur). a. Explain why this assertion is true. b. Describe a way to modify algorithms such as SCAN to ensure fairness. c. Explain why fairness is an important goal in a time-sharing system. d. Give three or more examples of circumstances in which it is important that the**

**operating system be unfair in serving I/O requests. Ans : a. New requests for the track over which the head currently resides can theoretically arrive as quickly as these requests are being serviced. b. All requests older than some predetermined age could be “forced” to the top of the queue, and an associated bit for each could be set to indicate that no new request could be moved ahead of these requests. For SSTF, the rest of the queue would have to be reorganized with respect to the last of these “old” requests. c. To prevent unusually long response times. d. Paging and swapping should take priority over user requests. It may be desirable for other kernel-initiated I/O, such as the writing of file system metadata, to take precedence over user I/O. If the kernel supports real-time process priorities, the I/O requests of those processes should be favored.**

**12.28 Compare he throughput achieved by a RAID level 5 organization with that achieved b a RAID level 1 organization for the following: a. Read operations on single blocks b. Read operations on multiple contiguous blocks Ans : 1)The amount of throughput depends on the number of disks in the RAID system. A RAID Level 5 comprising of a parity block for every set of four blocks spread over five disks can support four to five operations simultaneously. A RAID Level 1 comprising of two disks can support two simultaneous operations. Of course, there is greater flexibility in RAID Level 1 as to which copy of a block could be accessed and that could provide performance benefits by taking into account position of disk head. 2) RAID Level 5 organization achieves greater bandwidth for accesses to multiple contiguous blocks since the adjacent blocks could be simultaneously accessed. Such bandwidth improvements are not possible in RAID Level 1.**

****

**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:**

#define CYLINDERS 5000

#define REQUESTS 1000

int start =0;

Int ran\_array[REQUESTS];

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;

}

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

return 0;

}

**OUTPUT:**

**ALGORITHM:**

1 : Declare integer variables 'i', 'curr\_val' and 'difference' equal to 0, integer variable 'sav\_val' equal to 'ran\_array[start].

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

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

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

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

7 : Assign 'sav\_val' equal to 'curr\_val'

8: Increase value of 'i' by 1

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

10: Assign 'sav\_val' equal to 0

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

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

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

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

15: Assign 'sav\_val' equal to 'curr\_val'

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

17: Return head\_movement

**USE:**

**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 Turnaround Time is --- %f",tatavg/n);

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

return 0;

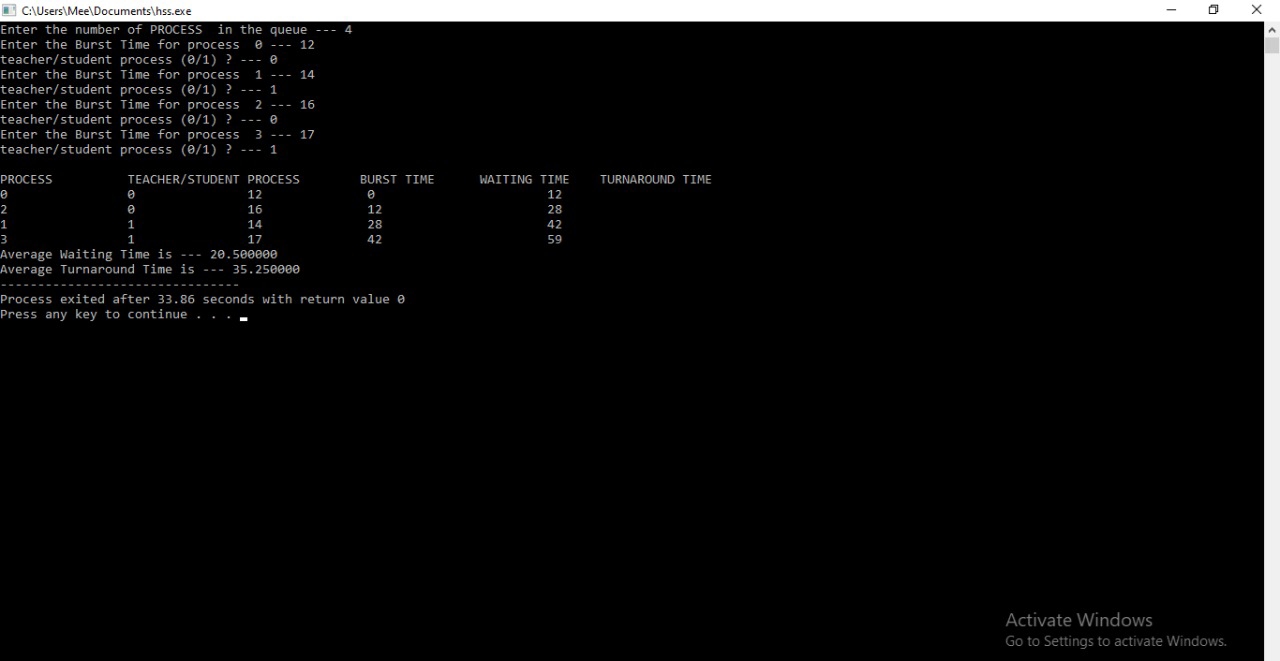
}

**OUTPUT:**

**ALGORITHM:**

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

**USE:**

****