

QUESTION 1

AIM AND ALGORITHM

DEV DATTA GOLWALKAR
19BCE0551


Q.1

Ans(a) AIM: To implement a menu driven program to implement FCFS and SSTF disk scheduling algorithms. ~~Take~~ Take total number of tracks (or cylinders) and current head position as run-time input. Assume head is moving towards 0. Compute and display seek length and average seek length.

Algorithm:

FCFS

1. Let the request array represents an array storing indexes of tracks that have been requested in ~~ascending~~ ascending order of their time of arrival. 'head' is the disk head position of disk head.
2. Let us one by one take the tracks in default ~~in~~ order and calculate and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

 FOR EDUCATIONAL USE

SSTF

Algorithm

- 1) Find the positive distance of all tracks in the request array from the head.
- 2) Find a track from the requested track array which has not been accessed / serviced yet and has a minimum distance from the head.
- 3) Increment the total seek count with this distance.
- 4) Currently services track position now becomes new head position
- 5) Go to step 2 until all tracks in request array have not been serviced.

CODE:

```
#include<stdio.h>
```

```
int absolute(int a,int b)
```

```
{int c;
```

```
c=a-b;
```

```
if(c<0)
```

```
    return -c;
```

```
else
```

```
    return c;
```

```

    }
int main()
{
    printf("MENU DRIVEN C CODE FOR FCFS AND SSTF DISK SCHEDULING ALGORITHM");
    int choice,m,n,x,start,i,j,pos,min,a[15],count;
    count=0;
    printf("\nEnter the number of cylinders :");
    scanf("%d",&m);
    printf("\nEnter the number of requests :");
    scanf("%d",&n);
    printf("\nEnter current position :");
    scanf("%d",&start);
    printf("\nEnter the request queue :");
    for(i=0;i<n;i++)
        {scanf("%d",&a[i]);
        if(a[i]>=m)
            {printf("\ninvalid input");
            scanf("%d",&a[i]);
            }
        }
    do
    {printf("\n\nDISK SCHEDULING ALGORITHMS\n1. FCFS\n2. SSTF\n");
    printf("\nEnter choice :");
    scanf("%d",&choice);
    count=0;
    x=start;
    switch(choice)
        {case 1:printf("\nFCFS :\n");
        printf("Scheduling services the request in the order that follows:\n%d\t",start);
        for(i=0;i<n;i++)
            {x-=a[i];

```

```

        if(x<0)
            x=-x;
        count+=x;
        x=a[i];
        printf("%d\t",x);
    }

    printf("\nTotal Head Movement :%d Cylinders",count);
    break;
case 2:printf("\nSSTF :\n");
    printf("Scheduling services the request in the order that follows:\n%d\t",start);
    for(i=0;i<n;i++)
        {min=absolute(a[i],x);
        pos=i;
        for(j=i;j<n;j++)
            if(min>absolute(x,a[j]))
                {pos=j;
                min=absolute(x,a[j]);
                }
        count+=absolute(x,a[pos]);
        x=a[pos];
        a[pos]=a[i];
        a[i]=x;
        printf("%d\t",x);
        }

    printf("\nTotal Head Movement: %d Cylinders",count);
    break;
}

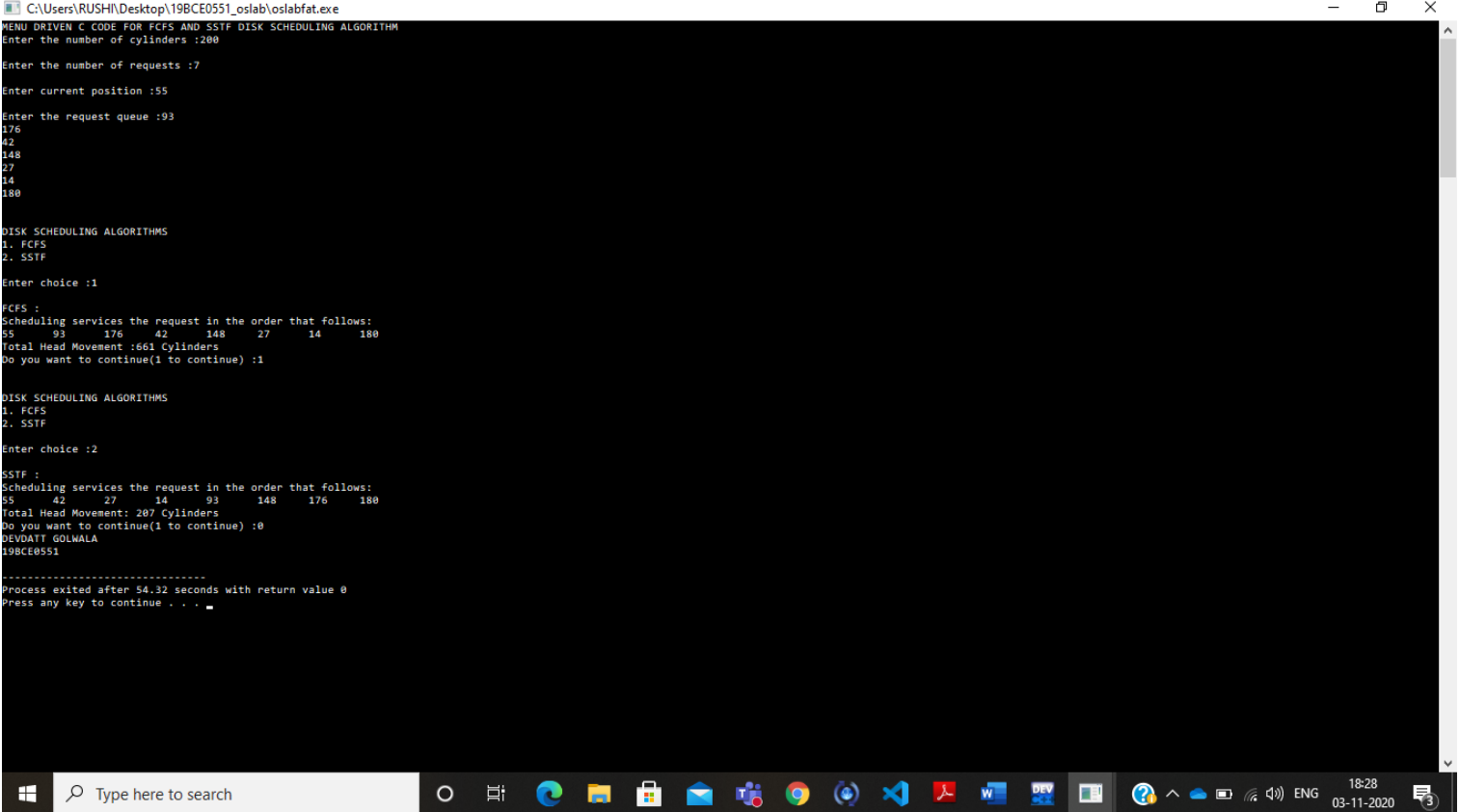
printf("\nDo you want to continue(1 to continue) :");
scanf("%d",&choice);
}while(choice==1);
printf("DEVDAAT GOLWALA\n");

```

```
printf("19BCE0551\n");
```

```
}
```

Screenshot (OUTPUT)



The screenshot shows a Windows command prompt window titled "C:\Users\RUSHI\Desktop\19BCE0551_oslab\oslabfat.exe". The program is a menu-driven C code for FCFS and SSTF disk scheduling algorithms. It prompts the user to enter the number of cylinders (200), the number of requests (7), and the current position (55). The request queue is entered as 93, 176, 42, 148, 27, 14, 180. The user chooses FCFS (1). The program outputs the scheduling sequence: 55, 93, 176, 42, 148, 27, 14, 180, with a total head movement of 661 cylinders. The user chooses to continue (1). The program then prompts for the SSTF algorithm choice (2). It outputs the scheduling sequence: 55, 42, 27, 14, 93, 148, 176, 180, with a total head movement of 207 cylinders. The user chooses not to continue (0). The program displays the user's name "DEVDAAT GOLWALA" and the ID "19BCE0551". It then shows the process exit message: "Process exited after 54.32 seconds with return value 0". The Windows taskbar at the bottom shows the search bar, taskbar icons, and system tray with the date 03-11-2020 and time 18:28.

```
C:\Users\RUSHI\Desktop\19BCE0551_oslab\oslabfat.exe
MENU DRIVEN C CODE FOR FCFS AND SSTF DISK SCHEDULING ALGORITHM
Enter the number of cylinders :200

Enter the number of requests :7

Enter current position :55

Enter the request queue :93
176
42
148
27
14
180

DISK SCHEDULING ALGORITHMS
1. FCFS
2. SSTF

Enter choice :1

FCFS :
Scheduling services the request in the order that follows:
55    93    176    42    148    27    14    180
Total Head Movement :661 Cylinders
Do you want to continue(1 to continue) :1

DISK SCHEDULING ALGORITHMS
1. FCFS
2. SSTF

Enter choice :2

SSTF :
Scheduling services the request in the order that follows:
55    42    27    14    93    148    176    180
Total Head Movement: 207 Cylinders
Do you want to continue(1 to continue) :0
DEVDAAT GOLWALA
19BCE0551

-----
Process exited after 54.32 seconds with return value 0
Press any key to continue . . .
```

QUESTION 2

AIM AND ALGORITHM

Ans (b) AIM: Consider 3 process, all arriving at time 0, with total execution time of 10, 20 and 30 units respectively. Each process spends the first 20% of execution time during I/O, the next 70% of time during computation, & the last 10% of time doing I/O again.

Assume that all I/O operations can be overlapped. Use Shortest Remaining Time First scheduling algorithm. Develop an algorithm and write code to find out the percentage of the CPU remain idle.

Algorithm

SRTF

Basically SRTF algorithm is shortest job first with preemption.

- 1) Traverse until all process gets completed.
- a) Find process with minimum remaining time at every single time lap.
- b) Reduce its time by 1

- c) Check if its remaining time becomes 0.
- d) Increment the counter of process completion.
- e) Completion time = current time + 1
- f) Waiting time = Completion time - arrival time - burst time.

g) Increment time cap by one.

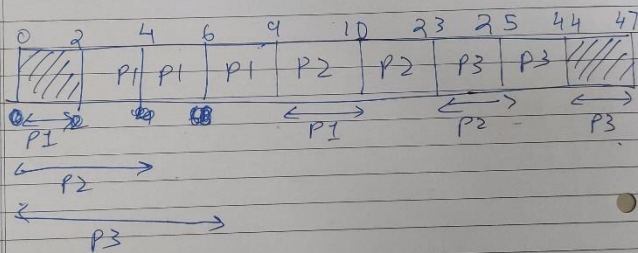
2) Find CPU idle time.

- Calculate total time spent
- Calculate idle time
- Percentage idle time = $\left(\frac{\text{Idle time}}{\text{Total time spent}} \right) \times 100$

② Theoretical solution.

Process	Total Burst time	I/O	CPU	I/O
P1	10	2	7	1
P2	20	4	14	2
P3	30	6	21	3

Grant Chart



Total time spent = 47 units

Idle time = 2 + 3 = 5

Percentage of idle time = $\left(\frac{5}{47}\right) \times 100$
 = 10.6%

CODE:

// C++ program to implement Shortest Remaining Time First

// Shortest Remaining Time First (SRTF)

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
struct Process {
```

```
    int pid; // Process ID
```

```
    int bt; // Burst Time
```



```

        int art; // Arrival Time
    };

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n,

                                                                    int wt[])
{
    int rt[n];

    // Copy the burst time into rt[]
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;

    // Process until all processes gets
    // completed
    while (complete != n) {

        // Find process with minimum
        // remaining time among the
        // processes that arrives till the
        // current time`
        for (int j = 0; j < n; j++) {
            if ((proc[j].art <= t) &&
                (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
            }
        }
    }
}

```

```

        check = true;
    }
}

if (check == false) {
    t++;
    continue;
}

// Reduce remaining time by one
rt[shortest]--;

// Update minimum
minm = rt[shortest];
if (minm == 0)
    minm = INT_MAX;

// If a process gets completely
// executed
if (rt[shortest] == 0) {

    // Increment complete
    complete++;
    check = false;

    // Find finish time of current
    // process
    finish_time = t + 1;

    // Calculate waiting time
    wt[shortest] = finish_time -

```

```
proc[shortest].bt -
```

```
proc[shortest].art;
```

```
    if (wt[shortest] < 0)
```

```
        wt[shortest] = 0;
```

```
    }
```

```
    // Increment time
```

```
    t++;
```

```
}
```

```
}
```

```
// Function to calculate turn around time
```

```
void findTurnAroundTime(Process proc[], int n,
```

```
int wt[], int tat[])
```

```
{
```

```
    // calculating turnaround time by adding
```

```
    // bt[i] + wt[i]
```

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

```
        tat[i] = proc[i].bt + wt[i];
```

```
}
```

```
// Function to calculate average time
```

```
void findavgTime(Process proc[], int n)
```

```
{
```

```
    int wt[n], tat[n], total_wt = 0,
```

```
        total_tat = 0;
```

```
    // Function to find waiting time of all
```

```
    // processes
```

```
    findWaitingTime(proc, n, wt);
```

```

// Function to find turn around time for
// all processes
findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all
// details
cout << "Processes "
        << " Burst time "
        << " Waiting time "
        << " Turn around time\n";

// Calculate total waiting time and
// total turnaround time
for (int i = 0; i < n; i++) {
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    cout << " " << proc[i].pid << "\t\t"
           << proc[i].bt << "\t\t" << wt[i]
           << "\t\t" << tat[i] << endl;
}

cout << "\nAverage waiting time = "
        << (float)total_wt / (float)n;
cout << "\nAverage turn around time = "
        << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    Process proc[] = { { 1, 7, 0 }, { 2, 14, 0 },

```

```
{ 3, 21, 0 } };
```

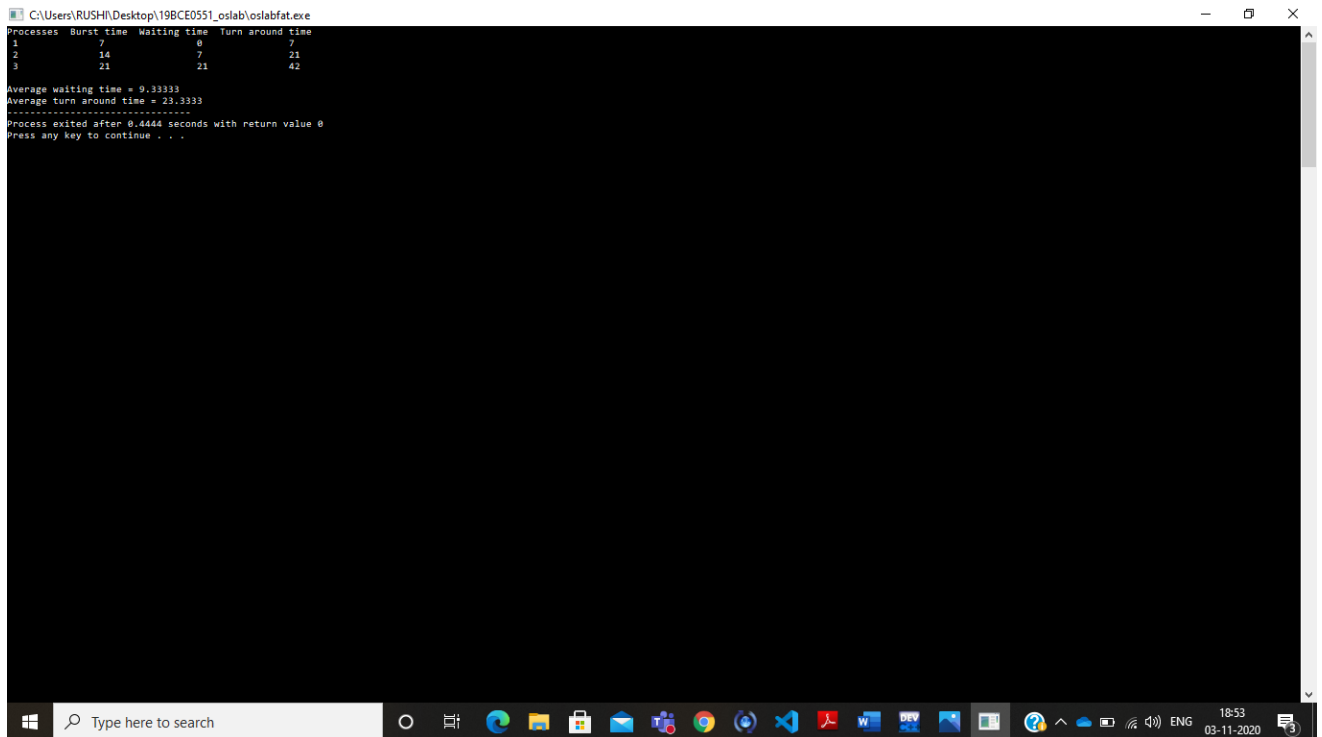
```
int n = sizeof(proc) / sizeof(proc[0]);
```

```
findavgTime(proc, n);
```

```
return 0;
```

```
}
```

Screenshot (OUTPUT)



```
C:\Users\RUSHI\Desktop\19BCE0551_oslab\oslabfat.exe
Processes Burst time Waiting time Turn around time
1          7          0          7
2         14          7         21
3         21         21         42

Average waiting time = 9.33333
Average turn around time = 23.3333
=====
Process exited after 0.4444 seconds with return value 0
Press any key to continue . . .
```