# ACADEMIC TASK-3(OPERATING SYSTEM)

ASSIGNMENT REPORT

**by**

ABDULLAH(57)

Email id:abdul123aslam@gmail.com
Github link: https://github.com/Abdul123aslam/ahmad

**Department of Intelligent Systems**
**School of Computer Science Engineering**
**Lovely Professional University, Jalandhar**
April – 2019

# DECLARATION STATEMENT

I hereby declare that the Assignment entitled to me for Question number 3 & 30, Submitted at Lovely Professional University, Phagwara, Punjab is an authentic work and has not been submitted elsewhere.

I understand that the work presented herewith is in direct compliance with Lovely Professional University's Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this case study represents authentic and honest effort conducted, in its entirety, by me. I am fully responsible for the contents of my case study report.

ABDULLAH

11717001

.

# CONTENTS

**Q.no- 3** Considering the arrival time and burst time requirement of the process the scheduler schedules the processes by interrupting the processor after every 6 units of time and does consider the completion of the process in this iteration. The scheduler than checks for the number of process waiting for the processor and allots the processor to the process but interrupting the processor every 10 unit of time and considers the completion of the processes in this iteration. The scheduler checks the number of processes waiting in the queue for the processor after the second iteration and gives the processor to the process which needs more time to complete than the other processes to go in the terminated state.

The inputs for the number of requirements, arrival time and burst time should be provided by the user. Consider the following units for reference.

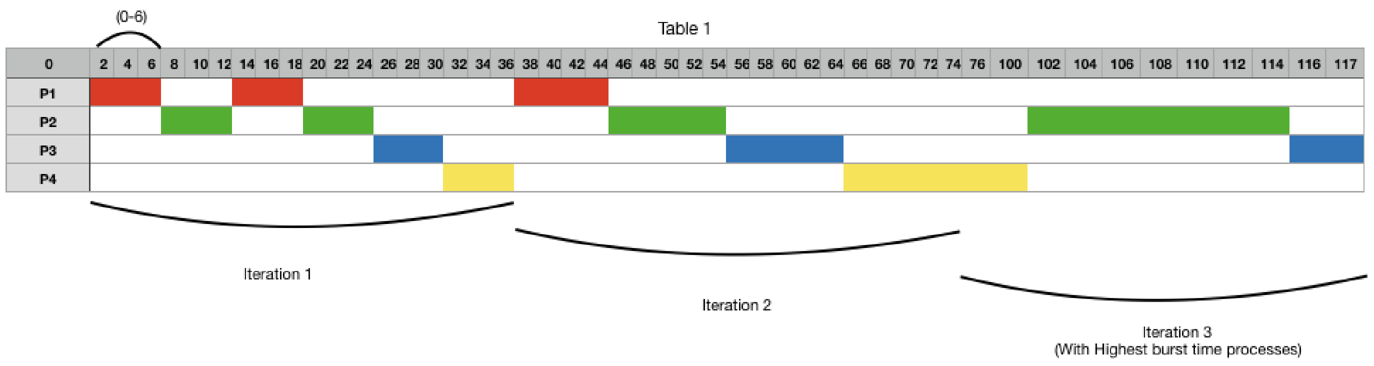
| Process | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| Arrival time | 0 | 5 | 13 | 26 |
| Burst time | 20 | 36 | 19 | 42 |

Develop a scheduler which submits the processes to the processor in the defined scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time.

**Algorithm –**

- **Step-1:** Create an array like AT (Arrival Time), BT (Burst Time), TAT (Turn Around Time), WT (Waiting Time).
- **Step-2:** Apply Round Robin with time quantum of 6ms upto each process traverse once
- **Step-3:** Apply Round Robin with time quantum of 10ms upto each process traverse once after each process execute with time quantum 6.
- **Step-4:** Apply FCFS with highest burst time process execute first.
- **Step-5:** find TAT and WT.

**Gantt chart** will be as following below,



Therefore,

**Output:**

Total Turn Around Time = 314
So, Average Turn Around Time = 78.5

And, Total Waiting Time = 189
So, Average Waiting Time = 47.25

## Methodology

### Common terms for users:

**Arrival Time:** Time at which the process arrives in the ready queue.
**Completion Time:** Time at which process completes its execution.
**Burst Time:** Time required by a process for CPU execution.
**Turn Around Time:** Time Difference between completion time and arrival time.

Turn Around Time = Completion Time - Arrival Time

**Waiting Time (W.T):** Time Difference between turnaround time and burst time.
Waiting Time = Turn Around Time - Burst Time

### Formulas Used:

Turn Around Time (TAT)
= (Completion Time) - (Arrival Time)

Also, Waiting Time (WT)
= (Turn Around Time) - (Burst Time)

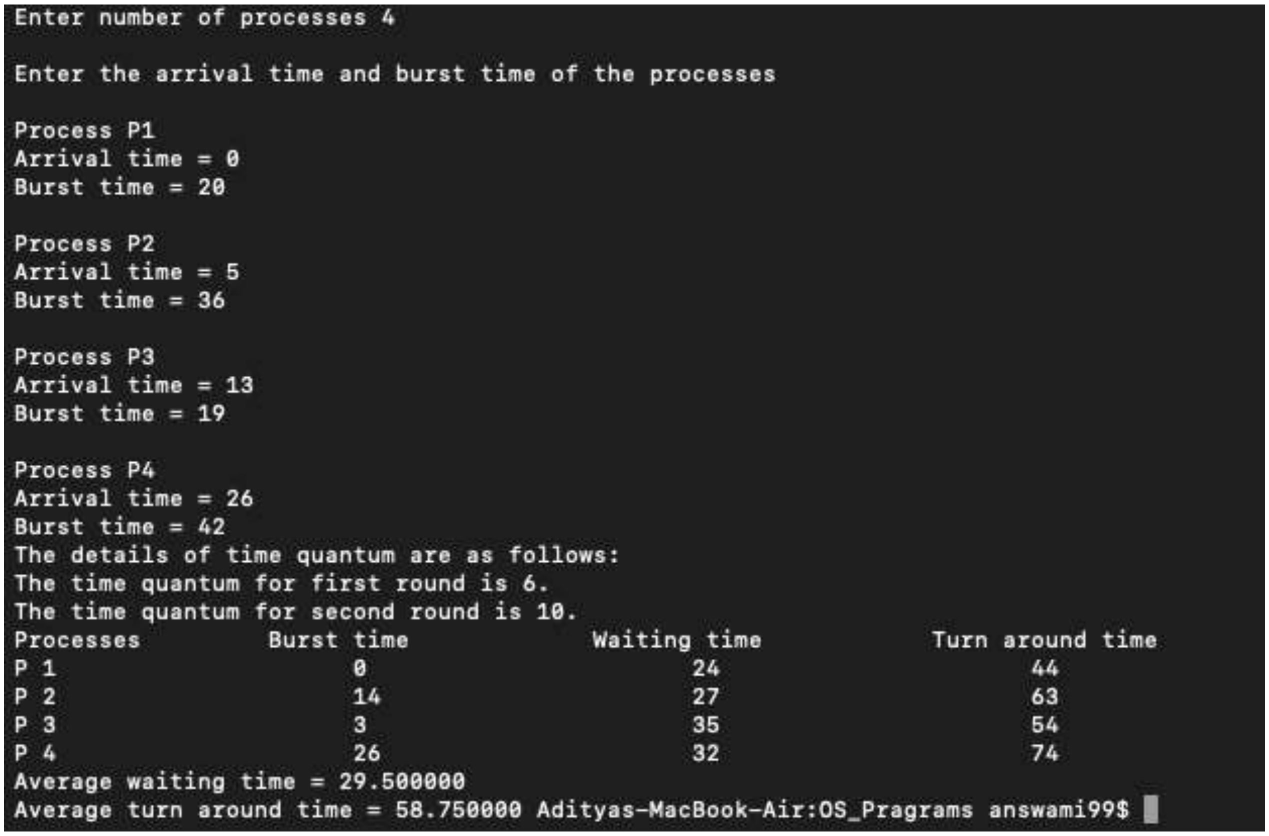
**CODE SNIPPET FOR Q3(With Round Robin and FCFS):**

```
#include<stdio.h>

void findavgTime(int n, int bt[],int curt,int wt[n],int tat[],int bsT[]);
void findWaitingTime(int n,int bt[], int wt[]);
void findTurnAroundTime(int n,int bt[], int wt[], int tat[]);
int main()
{
  int
Proc_no,j,no,CurT,RemProc,indicator,time_quan,wait[10],tut[10],arT[10],bsT[10],remt[10],x=1;
  indicator = 0;
  printf("Enter number of processes ");
  scanf("%d",&no);
  RemProc = no;

  printf("\nEnter the arrival time and burst time of the processes\n");
  for(Proc_no = 0;Proc_no < no;Proc_no++)
  {
    printf("\nProcess P%d\n",Proc_no+1);
    printf("Arrival time = ");
    scanf("%d",&arT[Proc_no]);
    printf("Burst time = ");
    scanf("%d",&bsT[Proc_no]);
    remt[Proc_no]=bsT[Proc_no];
  }
  printf("The details of time quantum are as follows:\n");
  printf("The time quantum for first round is 6.\n");
  time_quan=6;
  CurT=0;
  for(Proc_no=0;RemProc!=0;)
  {
    if(remt[Proc_no]<=time_quan && remt[Proc_no]>0)
    {
        CurT+=remt[Proc_no];
        remt[Proc_no]=0;
        indicator=1;
    }
    else if(remt[Proc_no]>0)
    {
        remt[Proc_no]-=time_quan;
        CurT+=time_quan;
        wait[Proc_no]=CurT-arT[Proc_no]-bsT[Proc_no];
        tut[Proc_no]=CurT-arT[Proc_no];
    }
    if(remt[Proc_no]==0 && indicator==1)
    {
        RemProc--;
        wait[Proc_no]=CurT-arT[Proc_no]-bsT[Proc_no];
        tut[Proc_no]=CurT-arT[Proc_no];
        indicator=0;

    }
    if(Proc_no==no-1){
        x++;
        if(x==2){
            Proc_no=0;
            time_quan=10;

            printf("The time quantum for second round is 10. \n");
        }
        else{
            break;
        }
    }
    else if(CurT >= arT[Proc_no+1]){

        Proc_no++;
    }
    else{
        Proc_no=0;
    }
  }
  findavgTime(no,remt,CurT,wait,tut,bsT);

  return 0;
}
```

```
void findWaitingTime(int n,int bt[], int wt[])
{

   // calculating waiting time
   for (int i = 1; i < n ; i++ ){
      wt[i] = bt[i] + wt[i] ;}
      //printf("\n%d\n",wt[i]);}
}

// Function to calculate turn around time
void findTurnAroundTime(int n,int bt[], int wt[], int tat[])
{
   // calculating turnaround time by adding
   // bt[i] + wt[i]
   for (int i = 0; i < n ; i++)
      tat[i] = bt[i] + wt[i];
}

//Function to calculate average time
void findavgTime(int n, int bt[],int curt,int wt[n],int tat[],int bsT[])
{
   int total_wt = 0, total_tat = 0;

   //Function to find waiting time of all processes
   findWaitingTime(n, bt, wt);

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

   //Display processes along with all details
   printf("Processes \t Burst time \t\t Waiting time \t\t Turn around time\n");

   //Calculate total waiting time and total turn
   // around time
   for (int i=0; i<n; i++)
   {
      total_wt = total_wt + wt[i];
      total_tat = total_tat + tat[i];
      printf("P %d ",(i+1));
      printf("\t\t\t%d ", bt[i] );
      printf("\t\t\t%d",wt[i] );
```

```
        printf("\t\t\t%d\n",tat[i] );
    }
    float s=(float)total_wt / (float)n;
    float t=(float)total_tat / (float)n;
    printf("Average waiting time = %f",s);
    printf("\n");
    printf("Average turn around time = %f ",t);
}
```

**OUTPUT:-**



```
Enter number of processes 4

Enter the arrival time and burst time of the processes

Process P1
Arrival time = 0
Burst time = 20

Process P2
Arrival time = 5
Burst time = 36

Process P3
Arrival time = 13
Burst time = 19

Process P4
Arrival time = 26
Burst time = 42
The details of time quantum are as follows:
The time quantum for first round is 6.
The time quantum for second round is 10.
Processes         Burst time             Waiting time             Turn around time
P 1                     0                      24                       44
P 2                     14                     27                       63
P 3                     3                      35                       54
P 4                     26                     32                       74
Average waiting time = 29.500000
Average turn around time = 58.750000 Adityas-MacBook-Air:OS_Pragrams answami99$ 
```

**Q.no.30-** 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

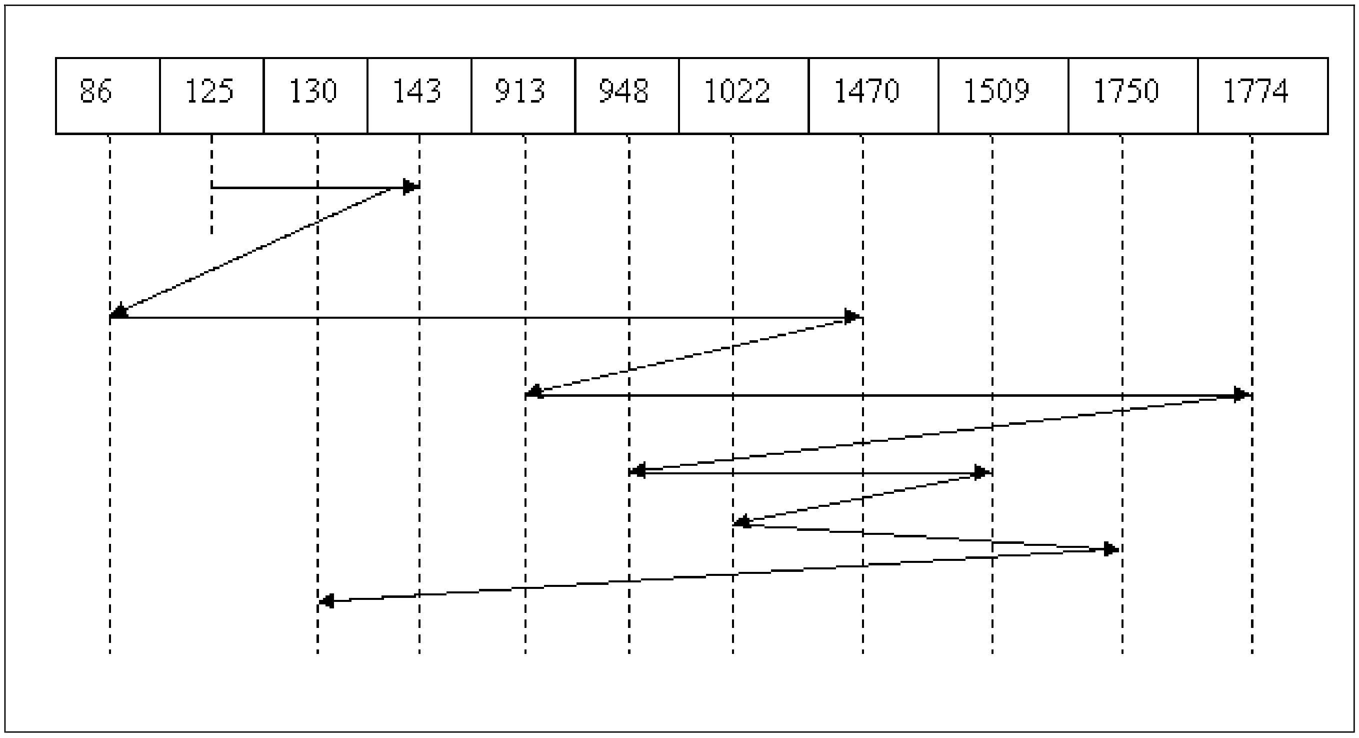
Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the FCFS disk-scheduling algorithms?

**Problem**

From the given position which is 143 move to next positions according to FCFS(First come first serve) manner.

**Explanation**

To do so we have to add all time taken by head to move from one to another position in order of allotting next position



Distance=125 to 143=18; 143 to 86=57; 86 to 1470=1384; 1470 to 913 =557; 913 to 1774=861; 1774 to 948=826; 948 to 1509=561; 1509 to 1022=487; 1022 to 1750=728; 1750 to 130=1620;

**Algorithm –**

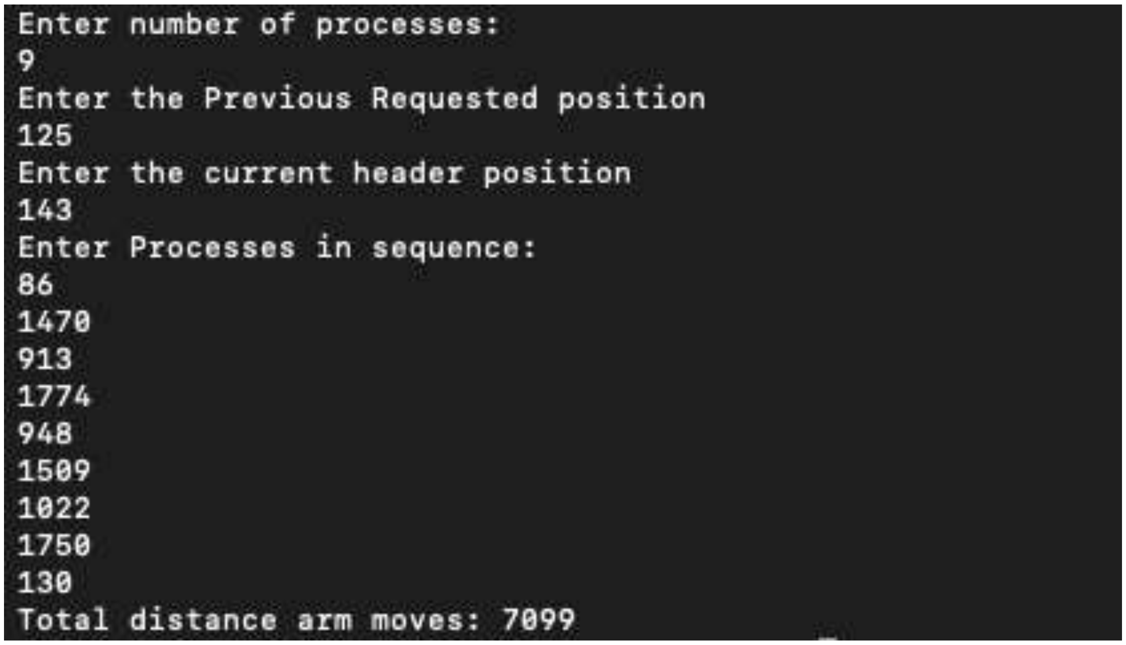
- **Step-1:** Create an Array named Queue to store processes inorder.
- **Step-2:** Take input of Current header and Previous Requested.
- **Step-3:** For all values compare two values and subtract bigger one from smaller one to get distance between them.
- **Step-4:** Add all distances after traversing all positions.

**Code Snippet Q30**

```
#include<stdio.h>
int main(){
   int n;
   int previous,cur;
   printf("Enter number of processes: \n");
   scanf("%d",&n);
   int queue[n];
   printf("Enter the Previous Requested position\n");
   scanf("%d",&previous);
   printf("Enter the current header position\n");
   scanf("%d",&cur);
   printf("Enter Processes in sequence: \n");
   for(int j = 0;j<n;j++){
      scanf("%d",&queue[j]);
   }
   int total =0 ;
   if (previous>cur){
      total = previous+cur;

   }
   else{
      total = cur-previous;
   }
  for (int i = 0;i<n;i++){
      if (queue[i]>cur){
         total += (queue[i]-cur);
         cur = queue[i];

      }
      else{
         total += (cur-queue[i]);
```

```
            cur = queue[i];
      }

      }
      printf("Total distance arm moves: %d\n", total);
}
```

**OUTPUT:-**



```
Enter number of processes:
9
Enter the Previous Requested position
125
Enter the current header position
143
Enter Processes in sequence:
86
1470
913
1774
948
1509
1022
1750
130
Total distance arm moves: 7099
```