Operating Systems Lab

Assignment 3

Name: Aditya Kangune
Batch: K11
Roll number: 33323
Date of Submission: 05/10/2021

Problem Statement:

Implement C program for CPU scheduling algorithms: Shortest Job First(Preemptive)and Round Robin with different arrival times.

Theory:

Process Scheduling:

- Process Scheduling is an important task of the Operating system which involves the employment of a specific algorithm to allocate the resources of the CPU to a process.
- It involves taking the process into a structure termed as the "Ready Queue" which keeps track of the process to be served (incomplete processes).
- If there is no process remaining in the ready queue, all the processes are flagged as completed.
- There are various types of scheduling algorithms that are broadly categorized into Preemptive and Non Preemptive algorithms.

Preemptive:

The context of the process is switched on the basis of some criteria regardless of the fact that the current process is completed its execution.

Non-Preemptive:

The context of the process is not switched unless the current process is completed.

Examples of Job Scheduling Algorithms:

- First Come First Serve.
- Shortest Job First.
- Round Robin Algorithm.
- Priority Scheduling.

Shortest Job First:

- Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution.
- This scheduling method can be preemptive or non-preemptive.
- It significantly reduces the average waiting time for other processes awaiting execution.

Characteristics of SJF:

- It is associated with each job as a unit of time to complete.
- This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical.
- It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time.
- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Advantages of SJF:

- SJF is frequently used for long-term scheduling.
- It reduces the average waiting time over FIFO (First in First Out) algorithm.
- SJF method gives the lowest average waiting time for a specific set of processes.
- It is appropriate for the jobs running in batch, where run times are known in advance.

Disadvantages of SJF:

- Job completion time must be known earlier, but it is hard to predict.

- May cause starvation.
- Elapsed time should be recorded, which results in more overhead on the processor.
- It is appropriate for the jobs running in batch, where run times are known in advance.
- In Preemptive SJF Scheduling, jobs are put into the ready queue as they come.
- A process with the shortest burst time begins execution. If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.

Round Robin Algorithm :

- A round-robin is an arrangement of choosing all elements in a group equally in some rational order, usually from the top to the bottom of a list and then starting again at the top of the list and so on.
- A simple way to think of round-robin is that it is about "taking turns."
- Used as an adjective, round-robin becomes "round-robin."
- In computer operation, one method of having different program processes take turns using the resources of the computer is to limit each process to a certain short time period, then suspending that process to give another process a turn.

Characteristics of Round Robin:

- The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
- The process that is preempted is added to the end of the queue.
- It is a real-time algorithm that responds to the event within a specific time limit.
- Widely used scheduling method in traditional OS.

Advantages of Round Robin:

- It does not face the issues of starvation.
- Fair allocation of CPU.
- It deals with all processes without any priority.

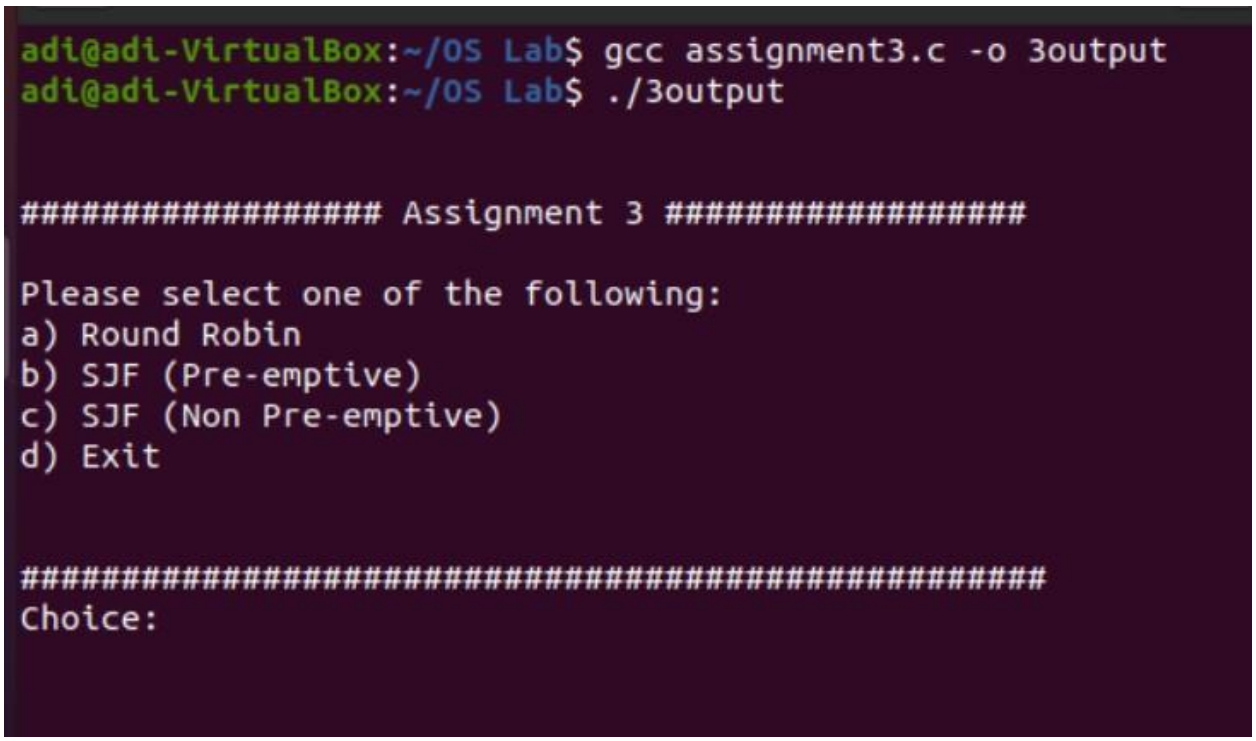
- It gives the best performance in terms of average waiting time.

Disadvantages of SJF:

- • Spends more time on context switching.
- Performance heavily depends on time quantum.
- Priorities cannot be set for the processes.
- Finding a correct time quantum is a quite difficult task.

Main program:

Menu for choices:



Round Robin:

```
################## Assignment 3 ##################

Please select one of the following:
a) Round Robin
b) SJF (Pre-emptive)
c) SJF (Non Pre-emptive)
d) Exit


################################################
Choice: a

===== Enter process data ====
Enter number of processes: 4

********************************************************************************

PROCESS NUMBER [1]
Process name: P1
Burst time : 5
Arrival time : 0

********************************************************************************

PROCESS NUMBER [2]
Process name: P2
Burst time : 4
```

```
********************************************************************************

PROCESS NUMBER [2]
Process name: P2
Burst time : 4
Arrival time : 1

********************************************************************************

PROCESS NUMBER [3]
Process name: P3
Burst time : 2
Arrival time : 2

********************************************************************************

PROCESS NUMBER [4]
Process name: P4
Burst time : 1
Arrival time : 3

********************************************************************************
```

Process ID	Burst Time	Arrival Time
P1	5	0
P2	4	1
P3	2	2
P4	1	3

Time-quantum = 2

```
Enter time quantum: 2

Gantt chart for Round Robin Algorithm :

*************************************************************************
************
       P1 ||   P2 ||   P3 ||   P1 ||   P4 ||   P2 ||   P1 ||
*************************************************************************
************
0        2||     4||     6||     8||     9||     11||    12||
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^

Analysis of Round Robin Algorithm :

Process Id      Arrival Time    Burst Time      Waiting Time    Turn Around Tim
e
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^
P1              0               5               7               12
P2              1               4               6               10
P3              2               2               2               4
P4              3               1               5               6


Average Waiting Time is : 20.000000 / 4 = 5.000000
Average Turn Around Time is : 32.000000 / 4 = 8.000000
```

Pre-emptive:

```
################################################
Choice: b

===== Enter process data =====
Enter number of processes: 4

********************************************************************************

PROCESS NUMBER [1]
Process name: P1
Burst time : 5
Arrival time : 0

********************************************************************************

PROCESS NUMBER [2]
Process name: P2
```

```
********************************************************************************

PROCESS NUMBER [2]
Process name: P2
Burst time : 4
Arrival time : 1

********************************************************************************

PROCESS NUMBER [3]
Process name: P3
Burst time : 2
Arrival time : 2

********************************************************************************

PROCESS NUMBER [4]
Process name: P4
Burst time : 1
Arrival time : 3
```

Process ID	Burst Time	Arrival Time
P1	5	0
P2	4	1
P3	2	2
P4	1	3

Gantt chart:

```
+------------+------------+-------------+
| Process ID | Burst Time | Arrival Time|
+------------+------------+-------------+
|    P1      |      5     |     0       |
|    P2      |      4     |     1       |
|    P3      |      2     |     2       |
|    P4      |      1     |     3       |
+------------+------------+-------------+
        P1 ||   P4 ||   P3 ||   P2 ||
*********************************************
0       5||     6||     8||     12||
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Analysis of Shortest Job First(Non-Preemptive) Algorithm :

Process Id      Arrival Time    Burst Time      Waiting Time    Turn Around Tim
e
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
-----
P4              3               1               2               3
P3              2               2               4               6
P2              1               4               7               11
P1              0               5               0               5
```

```
Average Waiting Time is : 13.000000 / 4 = 3.250000
Average Turn Around Time is : 25.000000 / 4 = 6.250000
```

Non-preemptive:

```
################## Assignment 3 ##################

Please select one of the following:
a) Round Robin
b) SJF (Pre-emptive)
c) SJF (Non Pre-emptive)
d) Exit


##########################################
Choice: c

===== Enter process data =====
Enter number of processes: 4

*****************************************************************

PROCESS NUMBER [1]
Process name: P1
Burst time : 5
Arrival time : 0

*****************************************************************

PROCESS NUMBER [2]
Process name: P2
Burst time : 4
Arrival time : 1
```

```
**********************************************************************

PROCESS NUMBER [3]
Process name: P3
Burst time : 2
Arrival time : 2

**********************************************************************

PROCESS NUMBER [4]
Process name: P4
Burst time : 1
Arrival time : 3

**********************************************************************
```

Process ID	Burst Time	Arrival Time
P1	5	0
P2	4	1
P3	2	2
P4	1	3

Gantt Chart:

```
 Gantt chart for SJF Preemptive :
****************************************
        P1||    P4||    P3||    P2||    P1||
**************************************************
 0||     3||     4||     6||     10||    12||

Analysis of Shortest Job First(Preemptive) Algorithm :


-------------------------------------------------------------
Process Id      Arrival Time    Burst Time      Waiting Time    Turn Around Tim
e
-------------------------------------------------------------
P4              3               1               0               1

-------------------------------------------------------------------------------
P3              2               2               2               4

-------------------------------------------------------------------------------
P2              1               4               5               9

-------------------------------------------------------------------------------
P1              0               5               7               12

-------------------------------------------------------------------------------
```

```
Average Waiting Time: 14.000000 / 4 = 3.500000
Average Turn Around Time: 26.000000 / 4 = 6.500000
```

Conclusion:

- The theory for job scheduling algorithms was studied and explored.
- Shortest Job First(SJF) and Round Robin algorithms with different arrival times and burst times were implemented.
- Round Robin algorithm performs better than SJF in terms of Average waiting time.

Code:

Link to code file:
https://drive.google.com/file/d/1zuIgF7QMdqDXoSyf77HLtmYZw2CarC60/view?usp=sharing

OR

Hard code:

```
// COMPILE : gcc assignment3.c -o 3output
// EXECUTE : ./3output

#include<stdio.h>
#include<stdlib.h>
#include <string.h>

// typedef keyword is used to redefine the name
// of an already existing variable.
typedef struct process
{
    // Declararing array for process name
    char name[5];
    // Declarartion of burst time
    int burst_time;
    // Declarartion of arrival time
    int arrival_time;

    // Declaration of waiting time and turn around time
    int waiting_time, turnaround_time;
    int remaining_Time;
}
processes,ready_queue;

// Function for sorting the processes
// according to arrival time.
void sort(processes proc[], int n)
{
```

```
    processes t;
    int i, j;
    for (i = 1; i < n; i++)
        for (j = 0; j < n - i; j++)
        {
            if (proc[j].arrival_time > proc[j + 1].arrival_time)
            {
                t = proc[j];
                proc[j] = proc[j + 1];
                proc[j + 1] = t;
            }
        }
}

// Function for sorting the processes
// according to arrival time.
void sort_by_burst_time(processes process[],int N)
{
    // Declaring a temporary process for sorting
   processes temp;

    for(int i=0; i<N; i++)
    {
        for(int j=i; j<N; j++ )
        {
            if(process[i].burst_time > process[j].burst_time)
            {
                temp = process[i];
                process[i] = process[j];
                process[j] = temp;
            }
        }
    }

}
```

```
// Function to get the process data from input of the user
int get_Processes(processes P[])
{
    int i, n;
    printf("\nEnter number of processes: ");
    scanf("%d", & n);
    printf("\n**********************************************************************\n");
    for (i = 0; i < n; i++)
    {
        printf("\nPROCESS NUMBER [%d]", i + 1);
        printf("\nProcess name: ");
        scanf("%s",  P[i].name);
        printf("Burst time : ");
        scanf("%d", & P[i].burst_time);
        P[i].remaining_Time = P[i].burst_time;
        printf("Arrival time : ");
        scanf("%d", & P[i].arrival_time);

printf("\n**********************************************************************\n");
    }
    puts("+------------+------------+-------------+");
    // Displaying process data in tabular form
    puts("| Process ID | Burst Time | Arrival Time|");
    puts("+------------+------------+-------------+");
    for (i = 0; i < n; i++)
    printf("|    %s      |      %d     |     %d       |\n", P[i].name, P[i].burst_time,
P[i].arrival_time);
    puts("+------------+------------+-------------+");
    return n;
}

int find_process(char current_name[], processes P[], int n)
{
    for(int i=0;i<n;i++)
```

```
    {
        int value=strcmp(P[i].name,current_name);
        if(value == 0)
            return i;
    }
}

// Function to display the Gantt chart
void print_gantt_chart(processes P[], int n)
{
    int i, j;
    // Displaying top bar
    printf(" ");
    for(i=0; i<n; i++)
    {
        for(j=0; j<P[i].burst_time; j++) printf("--");
        printf(" ");
    }
    printf("\n|");

    // Displaying process id
    for(i=0; i<n; i++)
    {
        for(j=0; j<P[i].burst_time - 1; j++) printf(" ");
        printf("P%s", P[i].name);
        for(j=0; j<P[i].burst_time - 1; j++) printf(" ");
        printf("|");
    }
    printf("\n ");
    // Displaying bottom bar
    for(i=0; i<n; i++)
    {
        for(j=0; j<P[i].burst_time; j++) printf("--");
        printf(" ");
    }
```

```
    printf("\n");

    // Displaying the time line
    printf("0");
    for(i=0; i<n; i++)
    {
        for(j=0; j<P[i].burst_time; j++) printf("  ");
        // backspace : remove 1 space
        if(P[i].turnaround_time > 9) printf("\b");
        printf("%d", P[i].turnaround_time);

    }
    printf("\n");

}

// Round Robin algorithm
void round_robin(processes ready_queue[],processes P[], int n)
{
   int current_time=0, i=0, j=0, rq_pointer=1;
   // Assigning remain process to number of processes initially
   int remaining_process = n;
   // time_elapse is array used to store values
   // of current_time when process gets execute.
   int time_elapse[50];
   float total_waiting=0,total_tat=0;
   int counter=0;

   int time_quantum;
    //T aking time_quantum input
   printf("Enter time quantum: ");
   scanf("%d",&time_quantum);

    sort(P,n);
    printf("\nGantt chart for Round Robin Algorithm :");
```

```
    printf("\n\n**********************************************************\n");

    ready_queue[0] = P[0];
    counter++;

    // Loop will execute until the remaining processes become 0.
    while(remaining_process != 0)
    {

        // If arrival time of process is less than or equal to current time then
only it will get execute
        if(ready_queue[i].arrival_time <= current_time)
        {
            // If remaining time equals to zero then it will iterate over next
process
            if(ready_queue[i].remaining_Time == 0)
            {
                i++;
                continue;
            }
            // Choiceecking if process's remaining time is less than given time
quantum if yes then after executing it's remaining will become 0.
            else if(ready_queue[i].remaining_Time <= time_quantum)
            {
                // Incrementing current_time with remaining time as it is less than
or equals to time quantum.
                current_time += ready_queue[i].remaining_Time;

                // Making remaining time of process = 0 as it completes
execution.
                ready_queue[i].remaining_Time = 0;

                // Reducing number of remaining process
                remaining_process--;
```

```
            //Displaying process executed with their id
            printf("\t%s ||",ready_queue[i].name);

            // Inserting current time in time_elapse array
            time_elapse[j] = current_time;
            // Incrementing pointer of time_elapse array
            j++;


            int index = find_process(ready_queue[i].name,P,n);

            P[index].turnaround_time = current_time - P[index].arrival_time;
            P[index].waiting_time = P[index].turnaround_time -
P[index].burst_time;
        }

        // When the remaining time of process is greater than given time
quantum.
        else
        {
            // Incrementing current_time with given time quantum
            current_time += time_quantum;

            printf("\t%s ||",ready_queue[i].name);
            // Inserting current time in time_elapse array
            time_elapse[j] = current_time;

            // Incrementing pointer of time_elapse array
            j++;

            // Substracting time quantum from remaining time so that process
remaining time will be deducted.
            ready_queue[i].remaining_Time =
ready_queue[i].remaining_Time - time_quantum;
```

```
            for(int start=counter;start<n;start++)
            {
               if(P[start].arrival_time <= current_time)
               {
                  ready_queue[rq_pointer] = P[start];
                  rq_pointer++;

                  counter++;
               }
            }

            ready_queue[rq_pointer] = ready_queue[i];
            rq_pointer++;


         }



      }

      // If the iterator reachoicees to end of an array it will again start
iterating from start index of array i.e. 0
      if(i == rq_pointer-1)
         i = 0;
      // Otherwise incrementing iterator by 1
      else
        i++;


   }
   printf("\n**********************************************************\n");


   // Displaying time_elapse array
```

```
    printf("0");
    for(int i=0;i<j;i++)
    {
       printf("\t %d|| ",time_elapse[i]);
    }


    // Displaying entered data

printf("\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^\n\n");
     printf("Analysis of Round Robin Algorithm : \n\n");
     printf("Process Id\tArrival Time\tBurst Time\tWaiting Time\tTurn Around
Time");

printf("\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^\n");
    for(int i=0;i<n;i++)
      {
       //Calculating the total waiting time
       total_waiting += P[i].waiting_time;
       // Calculating total turn around time
       total_tat += P[i].turnaround_time;

       printf("%s\t\t %d\t\t %d\t\t %d\t\t
%d",P[i].name,P[i].arrival_time,P[i].burst_time,P[i].waiting_time,P[i].turnarou
nd_time);
       printf("\n");
      }

    float avg_waiting = total_waiting / n;     //calculating average waiting time
    float avg_tat = total_tat / n;        //calculating average turn around time

    printf("\n\nAverage Waiting Time is : %f / %d = %f
",total_waiting,n,avg_waiting);
```

```
    printf("\nAverage Turn Around Time is : %f / %d = %f
",total_tat,n,avg_tat);
    printf("\n\n");

}


// Function for shortest job first non-preemptive
void non_preemptive_SJF(processes P[], int n)
{

   float avg_waiting,avg_tat,total_waiting=0,total_tat=0;
   int current_time=0, i=0;
   int remaining_process = n;
   int time_elapse[50],j=0;

   sort_by_burst_time(P,n);

    while(remaining_process != 0)
    {
        if(P[i].remaining_Time == 0 || P[i].arrival_time > current_time)
        {
            i++;
            continue;
        }
        else if(P[i].arrival_time <= current_time)
        {
                current_time += P[i].burst_time;
                P[i].remaining_Time = 0;
                remaining_process--;
                printf("\t%s ||",P[i].name);
                time_elapse[j]=current_time;
                j++;

                P[i].turnaround_time = current_time - P[i].arrival_time;
```

```
            P[i].waiting_time = P[i].turnaround_time - P[i].burst_time;
        }

      i = 0;
   }
   printf("\n***********************************************\n");

   printf("0");
   for(int i=0;i<j;i++)
   {
      printf("\t%d|| ",time_elapse[i]);
   }


printf("\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^\n\n");
  printf("Analysis of Shortest Job First(Non-Preemptive) Algorithm : \n\n");
  printf("Process Id\tArrival Time\tBurst Time\tWaiting Time\tTurn Around
Time");

printf("\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^-----\n");
  for(int i=0;i<n;i++)
  {
    total_waiting += P[i].waiting_time;
    total_tat += P[i].turnaround_time;

    printf("%s\t\t %d\t\t %d\t\t %d\t
\t%d",P[i].name,P[i].arrival_time,P[i].burst_time,P[i].waiting_time,P[i].turnaro
und_time);
      printf("\n");
  }
     avg_waiting = total_waiting / n;
    avg_tat = total_tat / n;
```

```
    printf("\n\nAverage Waiting Time is : %f / %d = %f
",total_waiting,n,avg_waiting);
    printf("\nAverage Turn Around Time is : %f / %d = %f
",total_tat,n,avg_tat);
    printf("\n\n");

}


int choiceeck(processes P[], int current_time, int current_remaining, int
current_i, int N)
{
    for(int i=0; i<N; i++)
    {
        if(P[i].remaining_Time == 0 || P[i].arrival_time > current_time || i ==
current_i)
        {
            i++;
            continue;
        }
        else if(P[i].remaining_Time < current_remaining)
            return i;
    }
    return -1;
}

// Pre-emptive (Shortest Job First)
void preemptive_SJF(processes P[], int n)
{
   int time_elapse[50];
   int j=0;
   int current_time=0, i=0, previous_i=-1;
   // Assigning remaining process to number of processes initially
   int remaining_process = n;
   float avg_waiting,avg_tat,total_waiting=0,total_tat=0;
```

```
    sort_by_burst_time(P,n);
    printf("\n Gantt chart for SJF Preemptive : ");
    printf("\n*****************************************\n");

    // Loop executes until remaining processes become 0.
    while(remaining_process != 0)
    {
        // If process's remaining time=0 or process's arriaval time is greater
than current_time then it will increment i pointer
        if(P[i].remaining_Time == 0 || P[i].arrival_time > current_time)
        {
            i++;
            continue;
        }
        // If process's arraival time is less or equal to current time
        else if(P[i].arrival_time <= current_time)
        {
            // If previous_i value is not equal to current i then only it will print
process id.
                if(previous_i != i)
                {
                    // Appending current time in time_elapse array
                   time_elapse[j] = current_time;
                   // Incrementing pointer of time_elapse array
                   j++;
                   printf("\t %s||",P[i].name);
                }

                // Incrementing current time
                current_time++;
                // Process's remaining time is decremented
                P[i].remaining_Time--;
```

```
        // After decrementing if process's remaining time becomes zero
        //then it's execution is finished so calculated it's
        // waiting time, turn around time and remaining process counter
reduced,
        // i is pointed to zero to choiceeck from start of array
            if(P[i].remaining_Time == 0)
            {
                P[i].turnaround_time = current_time - P[i].arrival_time;
                P[i].waiting_time = P[i].turnaround_time - P[i].burst_time;

                remaining_process--;
                i=0;
                continue;
            }
            // Previous i value is saved in variable
            previous_i = i;

            // Calling choiceeck function to choiceeck if any process having
less remaining time than current process is present in array or not.
            int cont = choiceeck(P, current_time, P[i].remaining_Time, i, n);

        // If it returns -1 that means no suchoice process is present but if it
returns value other than -1 then pointer of array is located to that index to
execute that process
            if(cont != -1)
            {
                i = cont;
            }
            continue;
        }
    }

    time_elapse[j] = current_time;
    printf("\n**********************************************************\n");
```

```
    // Displaying time_elapse array
    for(int i=0;i<=j; i++)
    {
        printf(" %d|| \t",time_elapse[i]);
    }



  printf("\n\nAnalysis of Shortest Job First(Preemptive) Algorithm : \n\n");
  printf("\n-----------------------------------------------------------\n");
  printf("Process Id\tArrival Time\tBurst Time\tWaiting Time\tTurn Around
Time");
  printf("\n-----------------------------------------------------------------\n");
  for(int i=0;i<n;i++)
  {
    total_waiting += P[i].waiting_time;
    total_tat += P[i].turnaround_time;

    printf("%s\t\t %d\t\t %d\t\t %d\t
\t%d",P[i].name,P[i].arrival_time,P[i].burst_time,P[i].waiting_time,P[i].turnaro
und_time);
        printf("\n");

printf("\n-----------------------------------------------------------------------\n");
  }
      avg_waiting = total_waiting / n;
    avg_tat = total_tat / n;

    printf("\n\nAverage Waiting Time: %f / %d = %f
",total_waiting,n,avg_waiting);
    printf("\nAverage Turn Around Time: %f / %d = %f ",total_tat,n,avg_tat);
    printf("\n\n");
}
```

```
int main()
{
    int n;
    char choice;
    processes P[10];
    processes ready_queue[10];


    do
    {
        printf("\n\n################### Assignment 3
####################\n");
        printf("\nPlease select one of the following:");
        printf("\na) Round Robin");
        printf("\nb) SJF (Pre-emptive)");
        printf("\nc) SJF (Non Pre-emptive)");
        printf("\nd) Exit\n  ");
        printf("\n\n################################################ ");
        printf("\nChoice: ");
        scanf("%c", & choice);
        switch (choice)
        {

            case 'a':
                printf("\n===== Enter process data ====");
                n = get_Processes(P);
                round_robin(ready_queue, P, n);
                break;

            case 'b':
                printf("\n===== Enter process data =====");
                n = get_Processes(P);
                non_preemptive_SJF(P, n);
                break;
```

```
        case 'c':
            printf("\n===== Enter process data =====");
            n = get_Processes(P);
            preemptive_SJF(P, n);
            break;



        case 'd':
            exit(0);
    }
}
while (choice != 'd');
return 0;       }
```