VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

DIGITAL ASSIGNMENT 2

CPU Scheduling Algorithms

Anushka Dixit
[19BCE0577]

This assignment contains CPU scheduling algorithms. The execution of system calls exec(), kill() and wait() has been submitted in the previous assignment.

CPU SCHEDULING ALGORITHMS

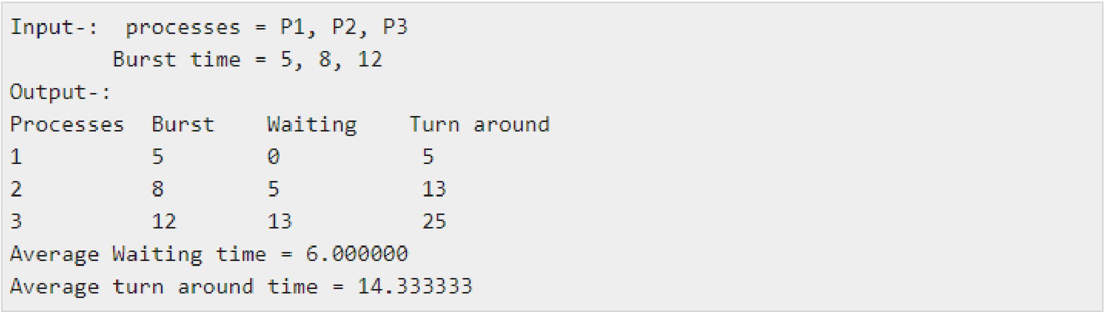
FIRST COME FIRST SERVE [FCFS]

First Come, First Served (FCFS) also known as First In, First Out(FIFO) is the CPU scheduling algorithm in which the CPU is allocated to the processes in the order they are queued in the ready queue. FCFS follows non-pre emptive scheduling which mean once the CPU is allocated to a process it does not leave the CPU until the process will not get terminated or may get halted due to some I/O interrupt.

Algorithm:

```
Start
 Step 1-> In function int waitingtime(int proc[], int n, int burst_time[], int wait_time[])
   Set wait_time[0] = 0
   Loop For i = 1 and i < n and i++
     Set wait_time[i] = burst_time[i-1] + wait_time[i-1]
   End For
Step 2-> In function int turnaroundtime( int proc[], int n, int burst_time[], int wait_time[], int tat[])
   Loop For  i = 0 and i < n and i++
     Set tat[i] = burst_time[i] + wait_time[i]
   End For
Step 3-> In function int avgtime( int proc[], int n, int burst_time[])
   Declare and initialize wait_time[n], tat[n], total_wt = 0, total_tat = 0;
   Call waitingtime(proc, n, burst_time, wait_time)
   Call turnaroundtime(proc, n, burst_time, wait_time, tat)
   Loop For  i=0 and i<n and i++
     Set total_wt = total_wt + wait_time[i]
     Set total_tat = total_tat + tat[i]
     Print process number, burstime wait time and turnaround time
   End For
   Print "Average waiting time =i.e. total_wt / n
   Print "Average turn around time = i.e. total_tat / n
Step 4-> In int main()
   Declare the input int proc[] = { 1, 2, 3}
   Declare and initialize n = sizeof proc / sizeof proc[0]
   Declare and initialize burst_time[] = {10, 5, 8}
   Call avgtime(proc, n, burst_time)
Stop
```

Example

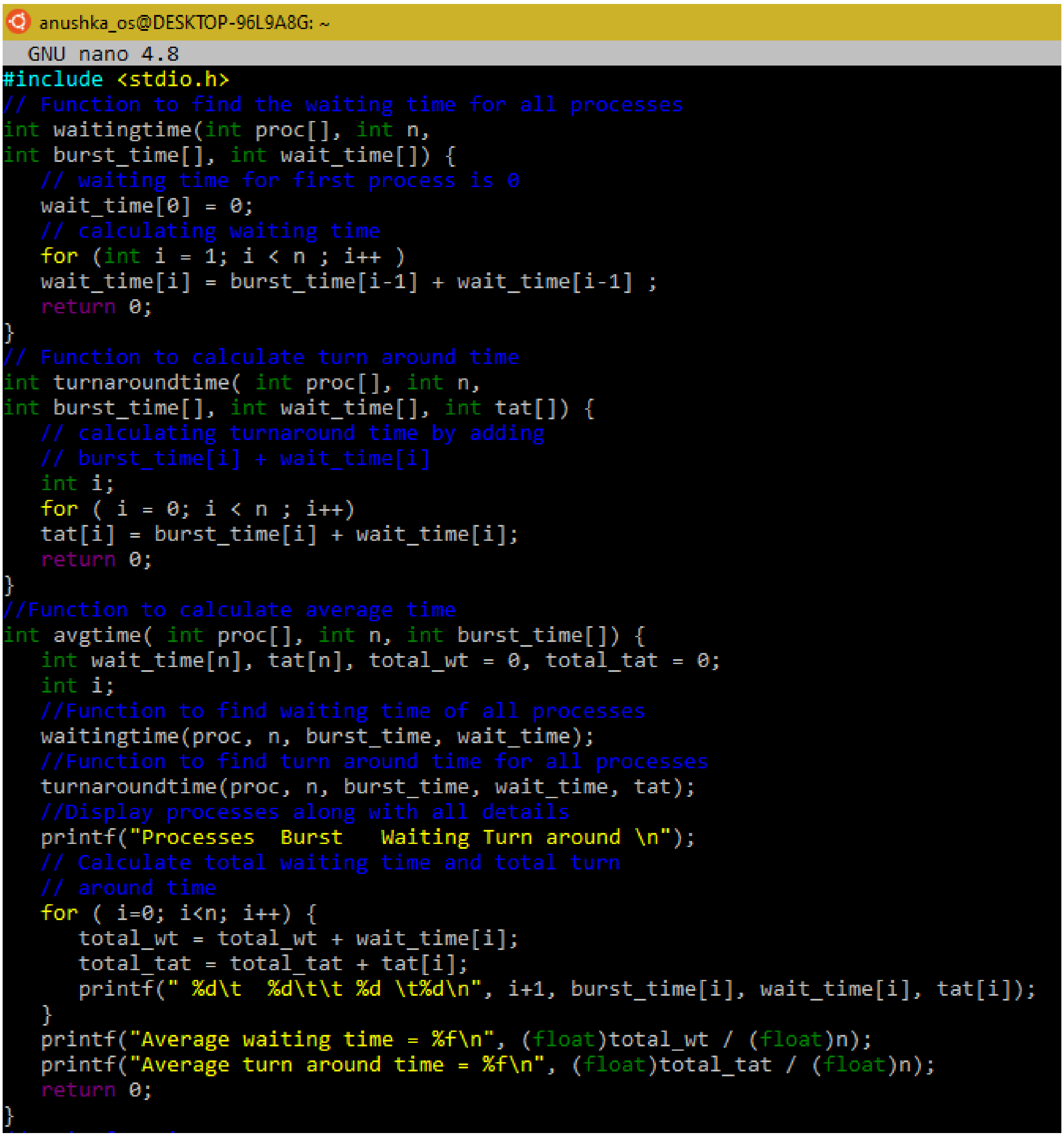


```
Input-:  processes = P1, P2, P3
        Burst time = 5, 8, 12
Output-:
Processes  Burst    Waiting    Turn around
1          5        0           5
2          8        5           13
3          12       13          25
Average Waiting time = 6.000000
Average turn around time = 14.333333
```

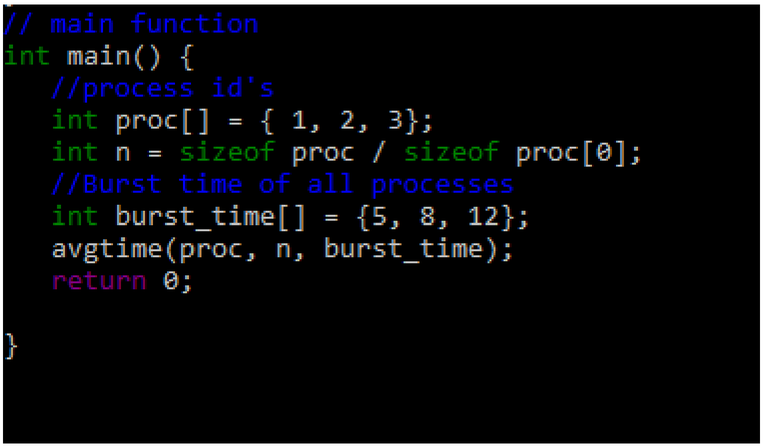
Anushka Dixit [19BCE0577]

1) FIRST COME FIRST SERVE SCHEDULING

CODE:

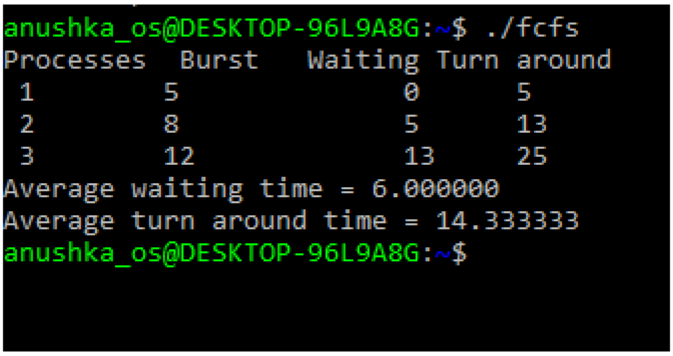


```
#include <stdio.h>
// Function to find the waiting time for all processes
int waitingtime(int proc[], int n,
int burst_time[], int wait_time[]) {
   // waiting time for first process is 0
   wait_time[0] = 0;
   // calculating waiting time
   for (int i = 1; i < n ; i++ )
   wait_time[i] = burst_time[i-1] + wait_time[i-1] ;
   return 0;
}
// Function to calculate turn around time
int turnaroundtime( int proc[], int n,
int burst_time[], int wait_time[], int tat[]) {
   // calculating turnaround time by adding
   // burst_time[i] + wait_time[i]
   int i;
   for ( i = 0; i < n ; i++)
   tat[i] = burst_time[i] + wait_time[i];
   return 0;
}
//Function to calculate average time
int avgtime( int proc[], int n, int burst_time[]) {
   int wait_time[n], tat[n], total_wt = 0, total_tat = 0;
   int i;
   //Function to find waiting time of all processes
   waitingtime(proc, n, burst_time, wait_time);
   //Function to find turn around time for all processes
   turnaroundtime(proc, n, burst_time, wait_time, tat);
   //Display processes along with all details
   printf("Processes  Burst   Waiting Turn around \n");
   // Calculate total waiting time and total turn
   // around time
   for ( i=0; i<n; i++) {
      total_wt = total_wt + wait_time[i];
      total_tat = total_tat + tat[i];
      printf(" %d\t  %d\t\t %d \t%d\n", i+1, burst_time[i], wait_time[i], tat[i]);
   }
   printf("Average waiting time = %f\n", (float)total_wt / (float)n);
   printf("Average turn around time = %f\n", (float)total_tat / (float)n);
   return 0;
}
```



```
// main function
int main() {
   //process id's
   int proc[] = { 1, 2, 3};
   int n = sizeof proc / sizeof proc[0];
   //Burst time of all processes
   int burst_time[] = {5, 8, 12};
   avgtime(proc, n, burst_time);
   return 0;

}
```

OUTPUT:



```
anushka_os@DESKTOP-96L9A8G:~$ ./fcfs
Processes  Burst   Waiting Turn around
 1         5            0       5
 2         8            5       13
 3         12           13      25
Average waiting time = 6.000000
Average turn around time = 14.333333
anushka_os@DESKTOP-96L9A8G:~$
```

SHORTEST JOB FIRST [SJF]

Shortest job first scheduling is the job or process scheduling algorithm that follows the non pre-emptive scheduling discipline. In this, scheduler selects the process from the waiting queue with the least completion time and allocates the CPU to that job or process. Shortest Job First is more desirable than FIFO algorithm because SJF is more optimal as it reduces average wait time which will increase the throughput.

1) NON PRE-EMPTIVE

A CPU scheduling technique where the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state. No process is interrupted until it is completed, and after that processor switches to another process.

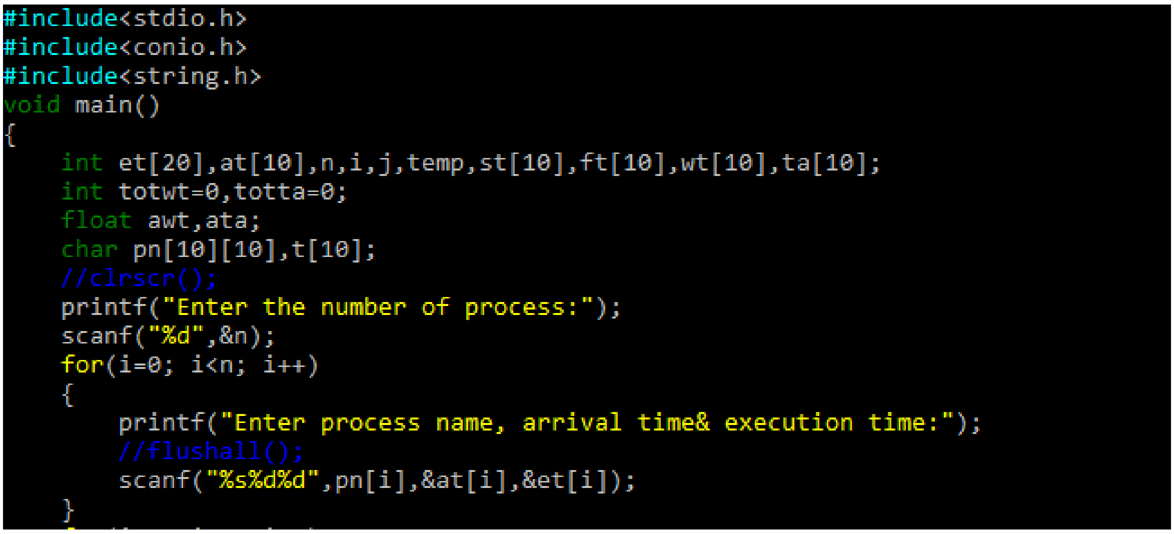
Algorithm:

```
Start
Step 1-> In function swap(int *a, int *b)
  Set temp = *a
  Set *a = *b
  Set *b = temp
Step 2-> In function arrangeArrival(int num, int mat[][3])
  Loop For i=0 and i<num and i++
    Loop For j=0 and j<num-i-1 and j++
      If mat[1][j] > mat[1][j+1] then,
        For k=0 and k<5 and k++
        Call function swap(mat[k][j], mat[k][j+1])
Step 3-> In function completionTime(int num, int mat[][3])
  Declare temp, val
  Set mat[3][0] = mat[1][0] + mat[2][0]
  Set mat[5][0] = mat[3][0] - mat[1][0]
  Set mat[4][0] = mat[5][0] - mat[2][0]
   Loop For i=1 and i<num and i++
    Set temp = mat[3][i-1]
    Set low = mat[2][i]
    Loop For j=i and j<num and j++
      If temp >= mat[1][j] && low >= mat[2][j] then,
        Set low = mat[2][j]
        Set val = j
        Set mat[3][val] = temp + mat[2][val]
        Set mat[5][val] = mat[3][val] - mat[1][val]
        Set mat[4][val] = mat[5][val] - mat[2][val]
        Loop For  k=0; k<6; k++
        Call function swap(mat[k][val], mat[k][i])
Step 4-> In function int main()
  Declare and set num = 3, temp
  Declare and set mat[6][3] = {1, 2, 3, 3, 6, 4, 2, 3, 4}
  Print Process ID, Arrival Time, Burst Time
  Loop For i=0 and i<num and i++
    Print the values of mat[0][i], mat[1][i], mat[2][i]
    Call function arrangeArrival(num, mat)
    Call function completionTime(num, mat)
    Print Process ID, Arrival Time, Burst Time, Waiting Time, Turnaround Time
    Loop For i=0 and i<num and i++
    Print the values of  mat[0][i], mat[1][i], mat[2][i], mat[4][i], mat[5][i]
Stop
```

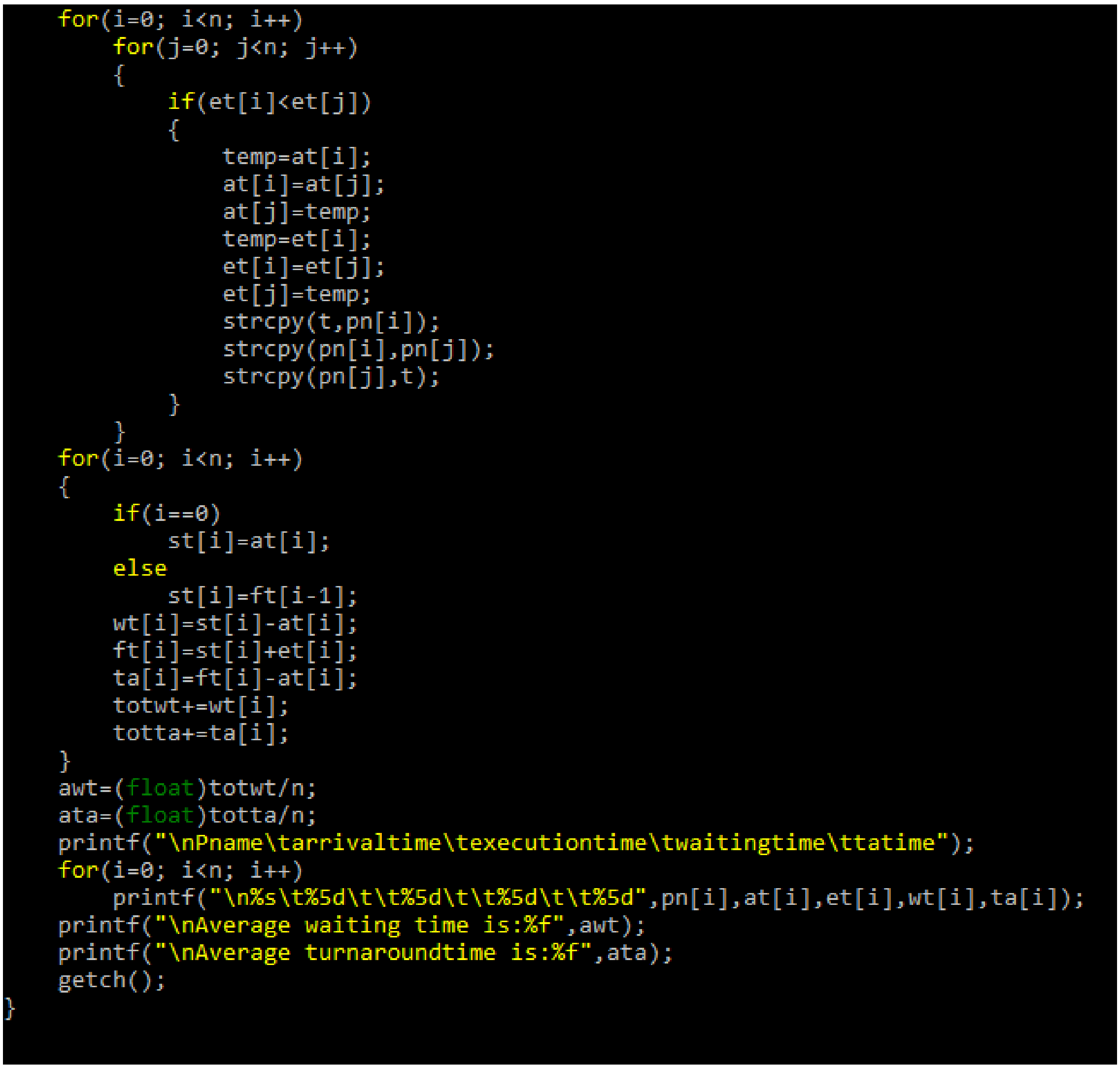
Example

Process ID	Arrival Time	Execution Time
P1	0	4
P2	3	6
P3	5	4
P4	6	8

Code:

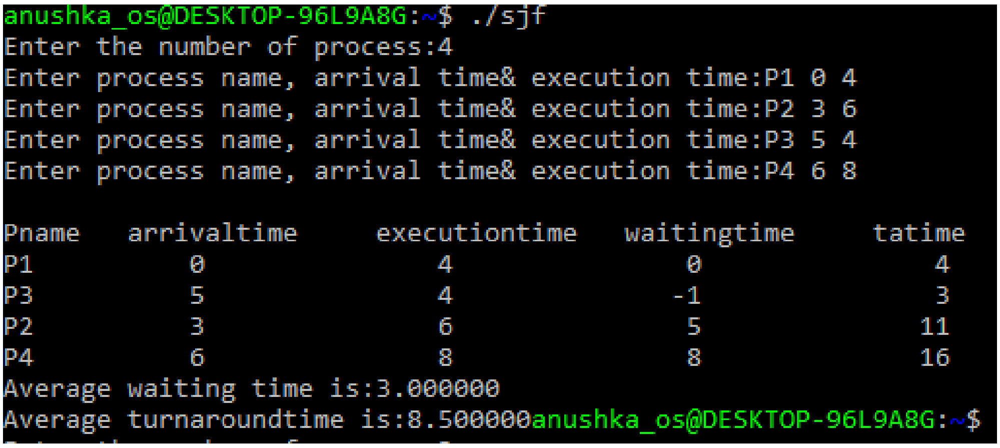


```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
    int totwt=0,totta=0;
    float awt,ata;
    char pn[10][10],t[10];
    //clrscr();
    printf("Enter the number of process:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("Enter process name, arrival time& execution time:");
        //flushall();
        scanf("%s%d%d",pn[i],&at[i],&et[i]);
    }
```



```
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
        {
            if(et[i]<et[j])
            {
                temp=at[i];
                at[i]=at[j];
                at[j]=temp;
                temp=et[i];
                et[i]=et[j];
                et[j]=temp;
                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }
        }
    for(i=0; i<n; i++)
    {
        if(i==0)
            st[i]=at[i];
        else
            st[i]=ft[i-1];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];
        totwt+=wt[i];
        totta+=ta[i];
    }
    awt=(float)totwt/n;
    ata=(float)totta/n;
    printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
    for(i=0; i<n; i++)
        printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
    printf("\nAverage waiting time is:%f",awt);
    printf("\nAverage turnaroundtime is:%f",ata);
    getch();
}
```

Output:



```
anushka_os@DESKTOP-96L9A8G:~$ ./sjf
Enter the number of process:4
Enter process name, arrival time& execution time:P1 0 4
Enter process name, arrival time& execution time:P2 3 6
Enter process name, arrival time& execution time:P3 5 4
Enter process name, arrival time& execution time:P4 6 8

Pname   arrivaltime     executiontime   waitingtime     tatime
P1          0               4               0               4
P3          5               4              -1               3
P2          3               6               5              11
P4          6               8               8              16
Average waiting time is:3.000000
Average turnaroundtime is:8.500000anushka_os@DESKTOP-96L9A8G:~$
```

2) PRE-EMPTIVE [SHORTEST REMAINING TIME FIRST]

In Pre emptive approach, the new process arises when there is already executing process. If the burst of newly arriving process is lesser than the burst time of executing process than scheduler will pre-empt the execution of the process with lesser burst time.

Algorithm:

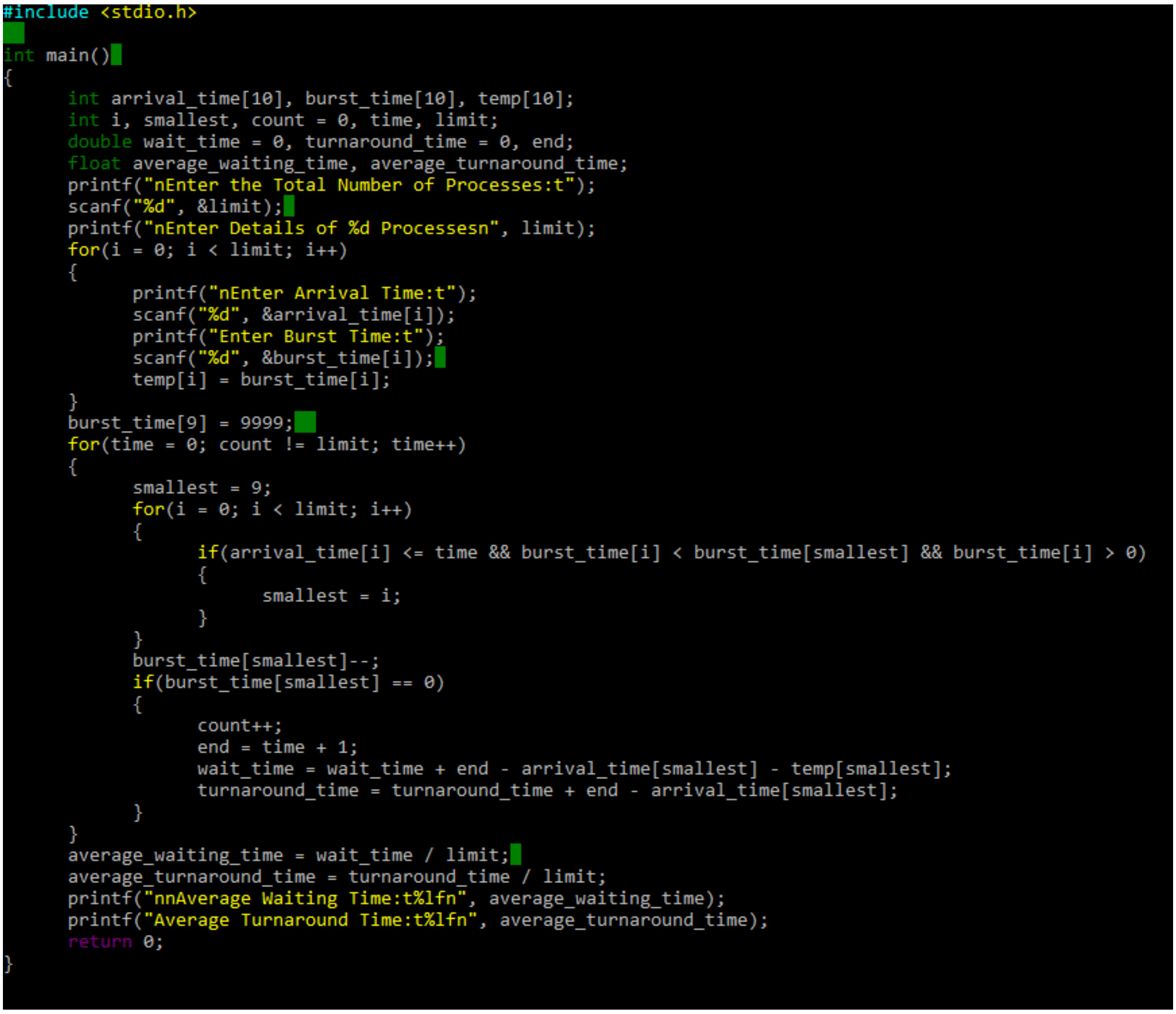
```
Start
Step 1-> Declare a struct Process
  Declare pid, bt, art
Step 2-> In function findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
  Loop For i = 0 and i < n and i++
    Set tat[i] = proc[i].bt + wt[i]
Step 3-> In function findWaitingTime(Process proc[], int n, int wt[])
  Declare rt[n]
  Loop For i = 0 and i < n and i++
    Set rt[i] = proc[i].bt
    Set complete = 0, t = 0, minm = INT_MAX
    Set shortest = 0, finish_time
    Set bool check = false
    Loop While (complete != n)
      Loop For j = 0 and j < n and j++
        If (proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0 then,
          Set minm = rt[j]
          Set shortest = j
          Set check = true
        If check == false then,
          Increment t by 1
          Continue
          Decrement the value of rt[shortest] by 1
          Set minm = rt[shortest]
        If minm == 0 then,
          Set minm = INT_MAX
          If rt[shortest] == 0 then,
          Increment complete by 1
          Set check = false
          Set finish_time = t + 1
          Set wt[shortest] = finish_time - proc[shortest].bt -proc[shortest].art
        If wt[shortest] < 0
          Set wt[shortest] = 0
          Increment t by 1
Step 4-> In function findavgTime(Process proc[], int n)
  Declare and set wt[n], tat[n], total_wt = 0, total_tat = 0
  Call findWaitingTime(proc, n, wt)
  Call findTurnAroundTime(proc, n, wt, tat)
  Loop For i = 0 and i < n and i++
    Set total_wt = total_wt + wt[i]
    Set total_tat = total_tat + tat[i]
    Print proc[i].pid, proc[i].bt, wt[i], tat[i]
    Print Average waiting time i.e., total_wt / n
    Print Average turn around time i.e., total_tat / n
```

```
Step 5-> In function int main()
  Declare and set Process proc[] = { { 1, 5, 1 }, { 2, 3, 1 }, { 3, 6, 2 }, { 4, 5, 3 } }
  Set n = sizeof(proc) / sizeof(proc[0])
  Call findavgTime(proc, n)
Stop
```

Example:

Process ID	Arrival Time	Burst Time
P1	0	3
P2	3	4
P3	8	4
P4	6	3

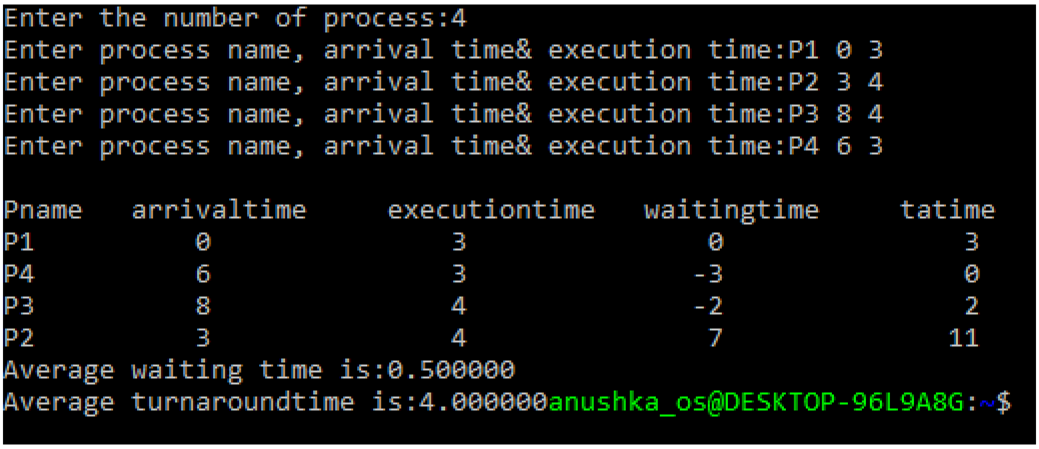
CODE:



```c
#include <stdio.h>

int main()
{
        int arrival_time[10], burst_time[10], temp[10];
        int i, smallest, count = 0, time, limit;
        double wait_time = 0, turnaround_time = 0, end;
        float average_waiting_time, average_turnaround_time;
        printf("nEnter the Total Number of Processes:t");
        scanf("%d", &limit);
        printf("nEnter Details of %d Processesn", limit);
        for(i = 0; i < limit; i++)
        {
                printf("nEnter Arrival Time:t");
                scanf("%d", &arrival_time[i]);
                printf("Enter Burst Time:t");
                scanf("%d", &burst_time[i]);
                temp[i] = burst_time[i];
        }
        burst_time[9] = 9999;
        for(time = 0; count != limit; time++)
        {
                smallest = 9;
                for(i = 0; i < limit; i++)
                {
                        if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
                        {
                                smallest = i;
                        }
                }
                burst_time[smallest]--;
                if(burst_time[smallest] == 0)
                {
                        count++;
                        end = time + 1;
                        wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
                        turnaround_time = turnaround_time + end - arrival_time[smallest];
                }
        }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
        printf("nnAverage Waiting Time:t%lfn", average_waiting_time);
        printf("Average Turnaround Time:t%lfn", average_turnaround_time);
        return 0;
}
```

Anushka Dixit [19BCE0577]

Output:



```
Enter the number of process:4
Enter process name, arrival time& execution time:P1 0 3
Enter process name, arrival time& execution time:P2 3 4
Enter process name, arrival time& execution time:P3 8 4
Enter process name, arrival time& execution time:P4 6 3

Pname   arrivaltime     executiontime   waitingtime     tatime
P1          0               3               0              3
P4          6               3              -3              0
P3          8               4              -2              2
P2          3               4               7             11
Average waiting time is:0.500000
Average turnaroundtime is:4.000000anushka_os@DESKTOP-96L9A8G:~$
```

PRIORITY SCHEDULING

In priority scheduling, every process is associated with a priority ranging from 0-10 where, integer 0 represents the lowest priority and 10 represents the highest priority. Priorities can be defined in two ways i.e. internally and externally. Also, priority scheduling can be either preemptive or nonpreemptive..

1) **Non pre-emptive scheduling**

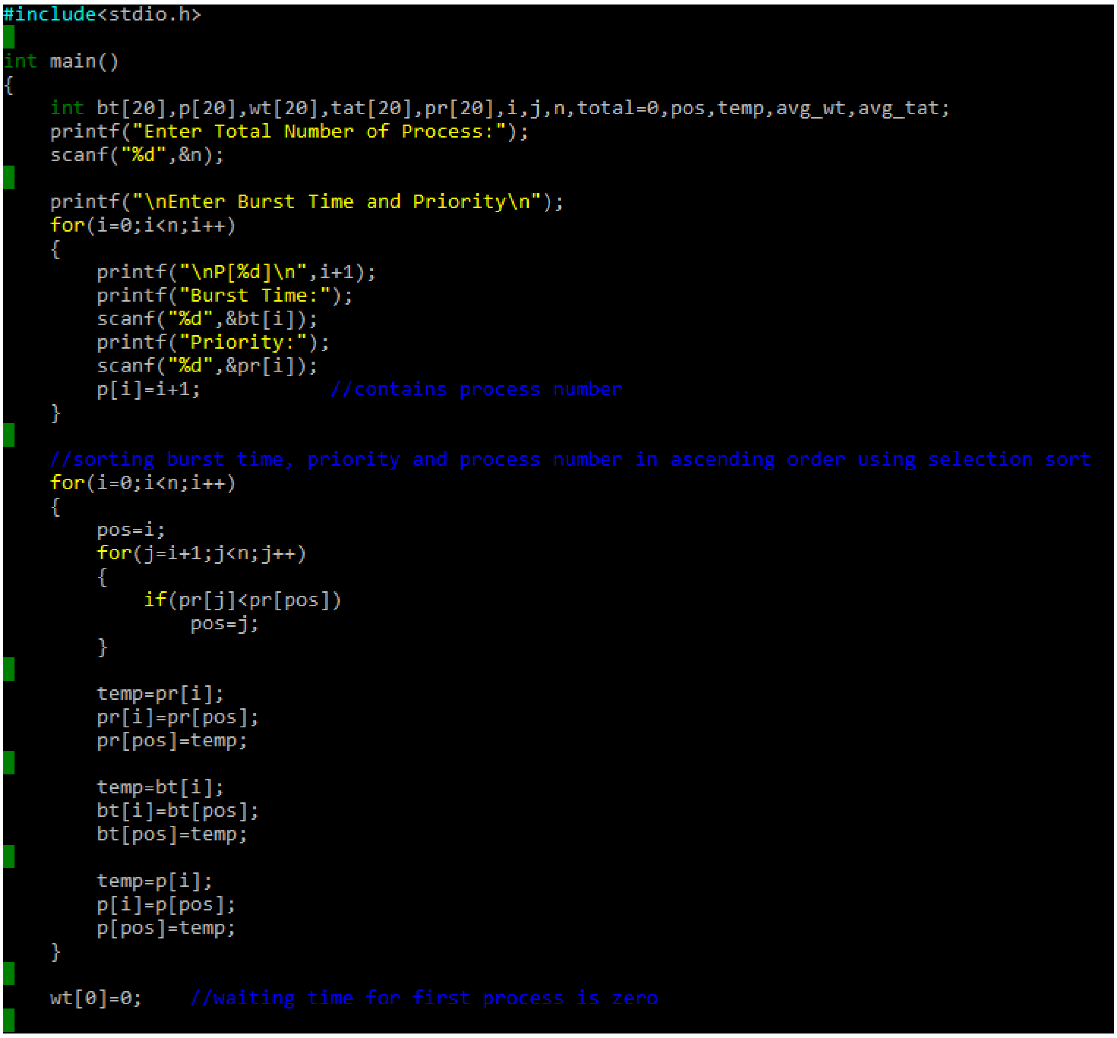
Scheduler will queue the new process at the head of the ready queue.

Algorithm

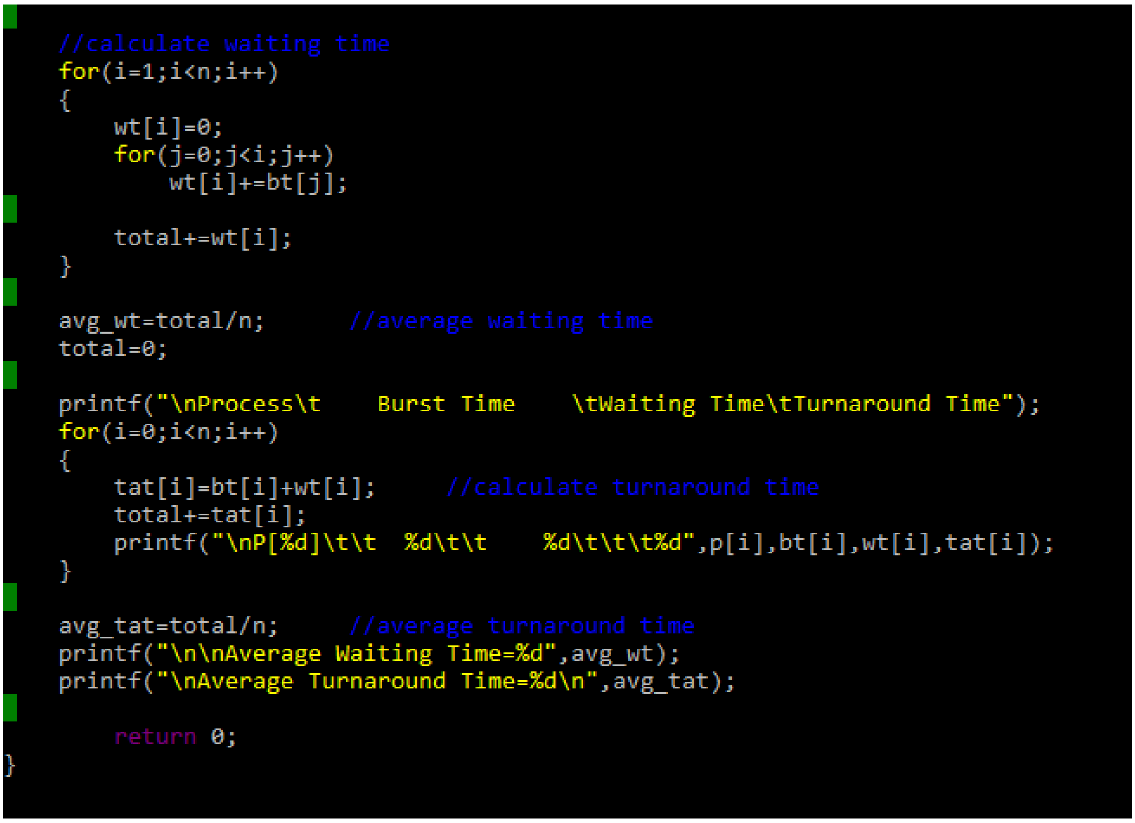
```
Start
Step 1-> Make a structure Process with variables pid, bt, priority
Step 2-> In function bool compare(Process a, Process b)
  Return (a.priority > b.priority)
Step 3-> In function waitingtime(Process pro[], int n, int wt[])
  Set wt[0] = 0
  Loop For i = 1 and i < n and i++
    Set wt[i] = pro[i-1].bt + wt[i-1]
  End
Step 4-> In function turnarround( Process pro[], int n, int wt[], int tat[])
  Loop For i = 0 and i < n and i++
    Set tat[i] = pro[i].bt + wt[i]
  End Loop
Step 5-> In function avgtime(Process pro[], int n)
  Declare and initialize wt[n], tat[n], total_wt = 0, total_tat = 0
  Call function waitingtime(pro, n, wt)
  Call function turnarround(pro, n, wt, tat)
  Print “Processes, Burst time, Waiting time, Turn around time"
  Loop For i=0 and i<n and i++
    Set total_wt = total_wt + wt[i]
    total_tat = total_tat + tat[i]
  End Loop
  Print values of “Processes, Burst time, Waiting time, Turn around time"
  Print Average waiting time, Average turn around time
Step 6-> In function scheduling(Process pro[], int n)
  Call function sort(pro, pro + n, compare)
  Loop For i = 0 and i < n and i++
    Print the order.
  End Loop
  Call function avgtime(pro, n)
Step 7-> In function int main()
  Declare and initialize Process pro[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}}
  Declare and initialize n = sizeof pro / sizeof pro[0]
  Call function scheduling(pro, n)
Stop
```

Anushka Dixit [19BCE0577]

Code:



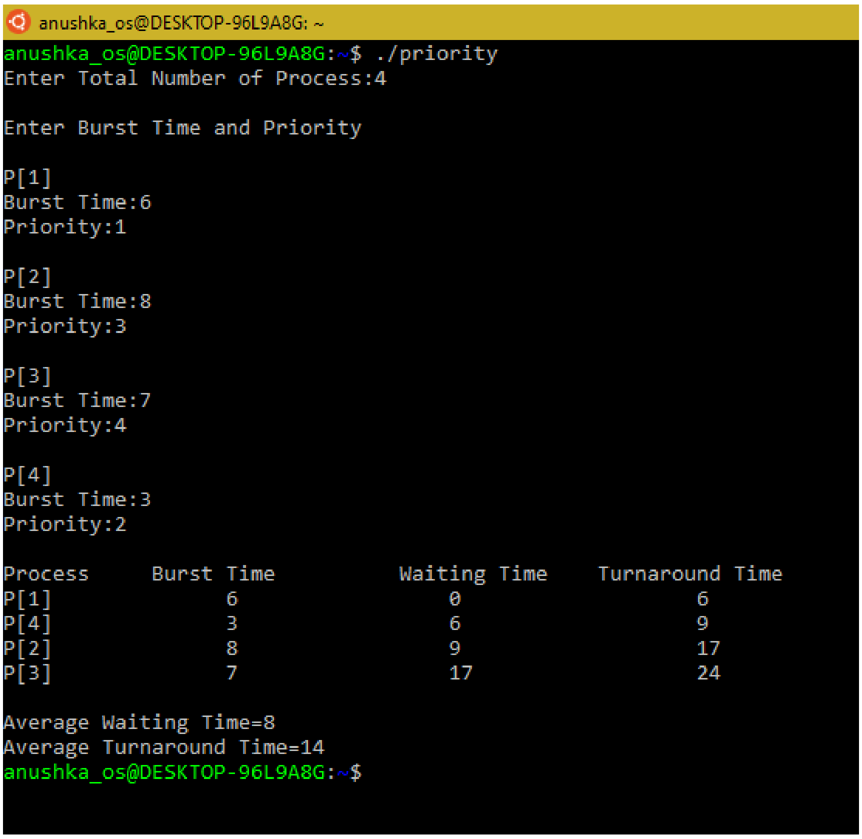
```
#include<stdio.h>

int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;           //contains process number
    }

    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }

        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;    //waiting time for first process is zero

```



```

    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=total/n;      //average waiting time
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

        return 0;
}
```

Anushka Dixit [19BCE0577]

Output:



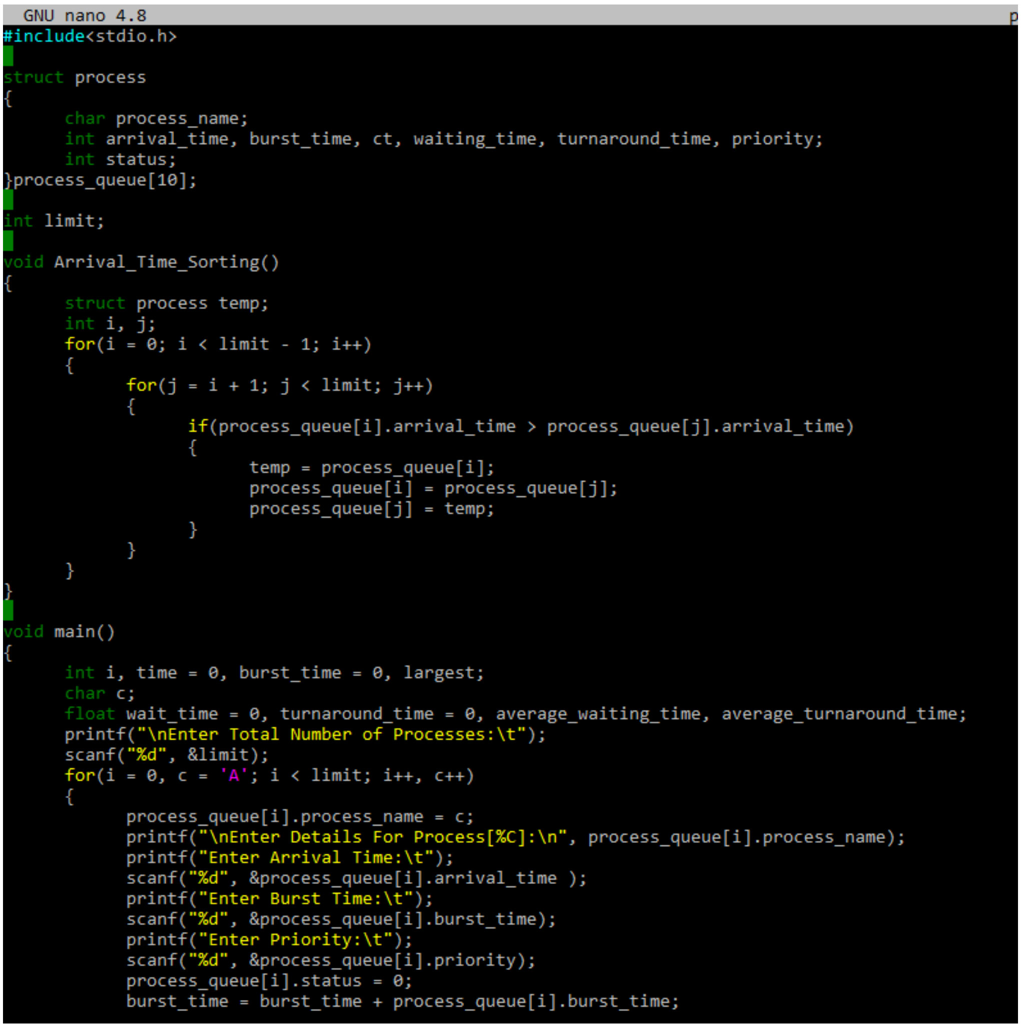
```
anushka_os@DESKTOP-96L9A8G:~$ ./priority
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:1

P[2]
Burst Time:8
Priority:3

P[3]
Burst Time:7
Priority:4

P[4]
Burst Time:3
Priority:2

Process     Burst Time          Waiting Time    Turnaround Time
P[1]              6                 0                   6
P[4]              3                 6                   9
P[2]              8                 9                   17
P[3]              7                 17                  24

Average Waiting Time=8
Average Turnaround Time=14
anushka_os@DESKTOP-96L9A8G:~$
```

Anushka Dixit [19BCE0577]

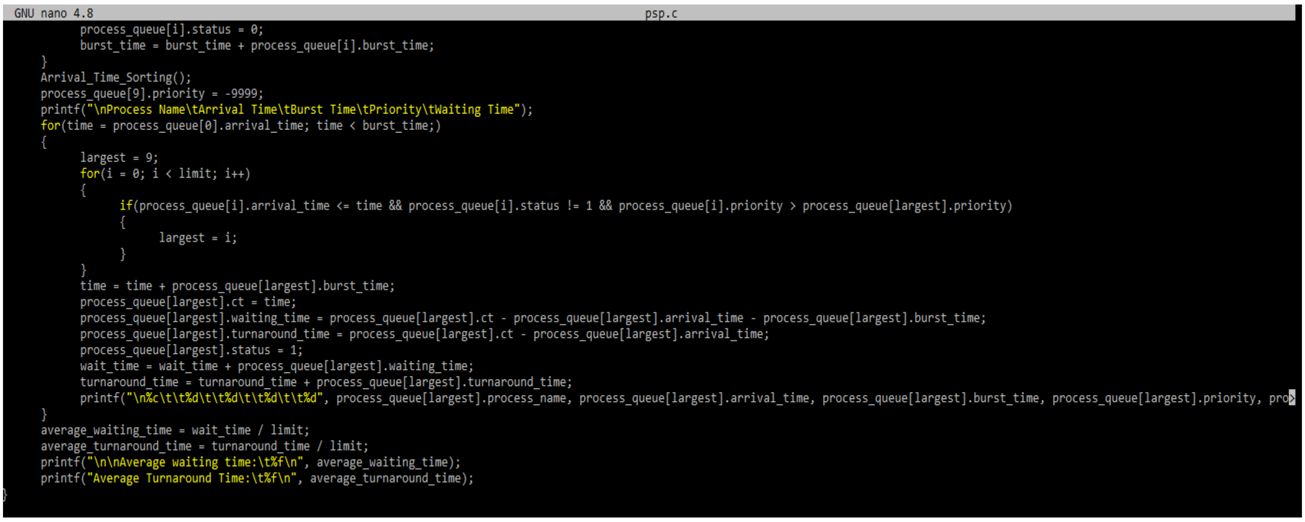
2) Pre-emptive scheduling

Scheduler will pre-empt the CPU if the priority of newly arrived process is higher than the priority of a process under execution

Code:



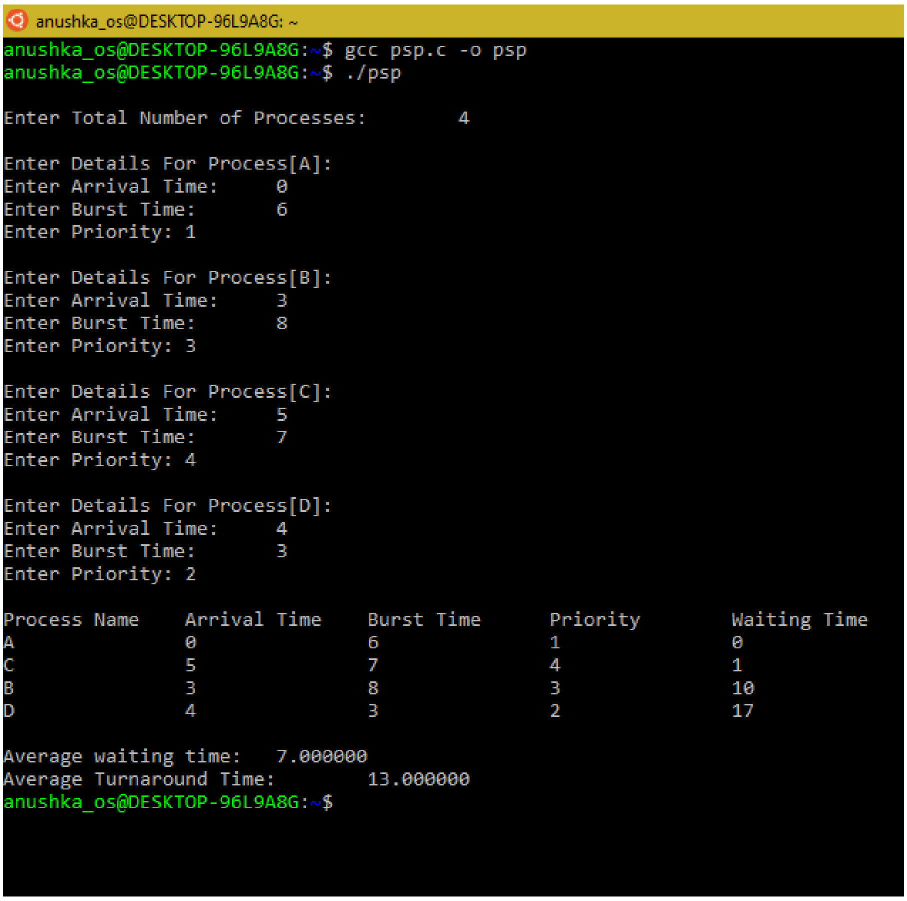
```
 GNU nano 4.8                                                                p
#include<stdio.h>

struct process
{
        char process_name;
        int arrival_time, burst_time, ct, waiting_time, turnaround_time, priority;
        int status;
}process_queue[10];

int limit;

void Arrival_Time_Sorting()
{
        struct process temp;
        int i, j;
        for(i = 0; i < limit - 1; i++)
        {
                for(j = i + 1; j < limit; j++)
                {
                        if(process_queue[i].arrival_time > process_queue[j].arrival_time)
                        {
                                temp = process_queue[i];
                                process_queue[i] = process_queue[j];
                                process_queue[j] = temp;
                        }
                }
        }
}

void main()
{
        int i, time = 0, burst_time = 0, largest;
        char c;
        float wait_time = 0, turnaround_time = 0, average_waiting_time, average_turnaround_time;
        printf("\nEnter Total Number of Processes:\t");
        scanf("%d", &limit);
        for(i = 0, c = 'A'; i < limit; i++, c++)
        {
                process_queue[i].process_name = c;
                printf("\nEnter Details For Process[%C]:\n", process_queue[i].process_name);
                printf("Enter Arrival Time:\t");
                scanf("%d", &process_queue[i].arrival_time );
                printf("Enter Burst Time:\t");
                scanf("%d", &process_queue[i].burst_time);
                printf("Enter Priority:\t");
                scanf("%d", &process_queue[i].priority);
                process_queue[i].status = 0;
                burst_time = burst_time + process_queue[i].burst_time;
```



```
 GNU nano 4.8                                    psp.c
                process_queue[i].status = 0;
                burst_time = burst_time + process_queue[i].burst_time;
        }
        Arrival_Time_Sorting();
        process_queue[9].priority = -9999;
        printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");
        for(time = process_queue[0].arrival_time; time < burst_time;)
        {
                largest = 9;
                for(i = 0; i < limit; i++)
                {
                        if(process_queue[i].arrival_time <= time && process_queue[i].status != 1 && process_queue[i].priority > process_queue[largest].priority)
                        {
                                largest = i;
                        }
                }
                time = time + process_queue[largest].burst_time;
                process_queue[largest].ct = time;
                process_queue[largest].waiting_time = process_queue[largest].ct - process_queue[largest].arrival_time - process_queue[largest].burst_time;
                process_queue[largest].turnaround_time = process_queue[largest].ct - process_queue[largest].arrival_time;
                process_queue[largest].status = 1;
                wait_time = wait_time + process_queue[largest].waiting_time;
                turnaround_time = turnaround_time + process_queue[largest].turnaround_time;
                printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d", process_queue[largest].process_name, process_queue[largest].arrival_time, process_queue[largest].burst_time, process_queue[largest].priority, pro>
        }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
        printf("\n\nAverage waiting time:\t%f\n", average_waiting_time);
        printf("Average Turnaround Time:\t%f\n", average_turnaround_time);
}
```

Anushka Dixit [19BCE0577]

Output:



```
anushka_os@DESKTOP-96L9A8G:~$ gcc psp.c -o psp
anushka_os@DESKTOP-96L9A8G:~$ ./psp

Enter Total Number of Processes:        4

Enter Details For Process[A]:
Enter Arrival Time:     0
Enter Burst Time:       6
Enter Priority: 1

Enter Details For Process[B]:
Enter Arrival Time:     3
Enter Burst Time:       8
Enter Priority: 3

Enter Details For Process[C]:
Enter Arrival Time:     5
Enter Burst Time:       7
Enter Priority: 4

Enter Details For Process[D]:
Enter Arrival Time:     4
Enter Burst Time:       3
Enter Priority: 2

Process Name    Arrival Time    Burst Time      Priority        Waiting Time
A               0               6               1               0
C               5               7               4               1
B               3               8               3               10
D               4               3               2               17

Average waiting time:   7.000000
Average Turnaround Time:        13.000000
anushka_os@DESKTOP-96L9A8G:~$
```

Anushka Dixit [19BCE0577]

ROUND ROBIN

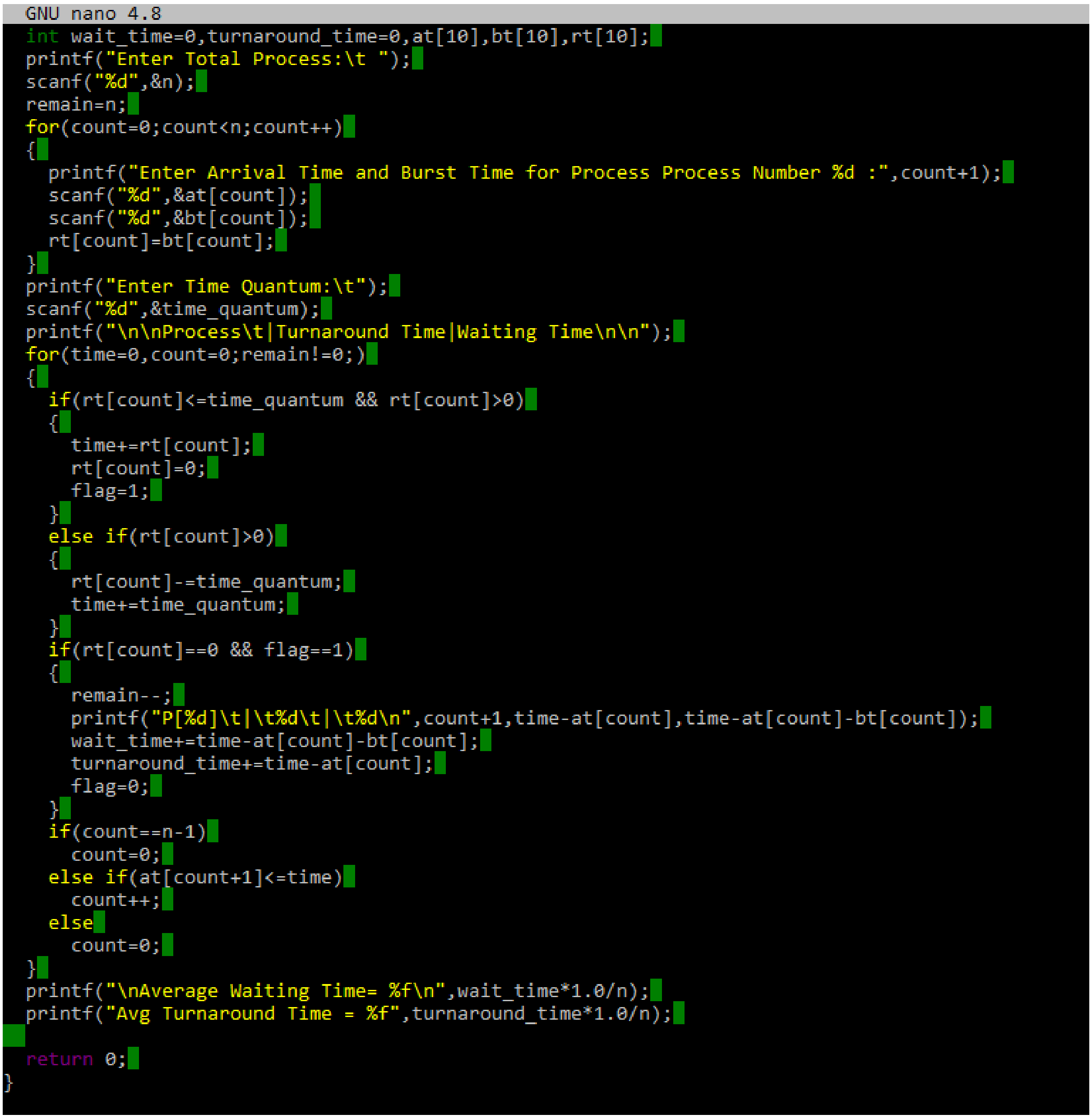
Round robin is a CPU scheduling algorithm that is designed especially for time sharing systems. It is more like a FCFS scheduling algorithm with one change that in Round Robin processes are bounded with a quantum time size. A small unit of time is known as Time Quantum or Time Slice. Time quantum can range from 10 to 100 milliseconds. CPU treats ready queue as a circular queue for executing the processes with given time slice. It follows pre-emptive approach because fixed times are allocated to processes. The only disadvantage of it is overhead of context switching.

Algorithm:

```
Start
Step 1-> In function int turnarroundtime(int processes[], int n, int bt[], int wt[], int tat[])
  Loop For i = 0 and i < n and i++
    Set tat[i] = bt[i] + wt[i]
  return 1
Step 2-> In function int waitingtime(int processes[], int n, int bt[], int wt[], int quantum)
Declare rem_bt[n]
  Loop For i = 0 and i < n and i++
    Set rem_bt[i] = bt[i]
    Set t = 0
  Loop While (1)
    Set done = true
  Loop For i = 0 and i < n and i++
    If rem_bt[i] > 0 then,
      Set done = false
    If rem_bt[i] > quantum then,
      Set t = t + quantum
      Set rem_bt[i] = rem_bt[i] - quantum
    Else
      Set t = t + rem_bt[i]
      Set wt[i] = t - bt[i]
      Set rem_bt[i] = 0
    If done == true then,
  Break
Step 3->In function int findavgTime(int processes[], int n, int bt[], int quantum)
  Declare and initialize wt[n], tat[n], total_wt = 0, total_tat = 0
  Call function waitingtime(processes, n, bt, wt, quantum)
  Call function turnarroundtime(processes, n, bt, wt, tat)
  Print "Processes Burst Time Waiting Time turnaround time "
  Loop For i=0 and i<n and i++
  Set total_wt = total_wt + wt[i]
  Set total_tat = total_tat + tat[i]
  Print the value i+1, bt[i], wt[i], tat[i]
  Print "Average waiting time = total_wt / n
  Print "Average turnaround time =total_tat / n
Step 4-> In function int main()
  Delcare and initialize processes[] = { 1, 2, 3}
  Declare and initialize n = sizeof processes / sizeof processes[0]
  Declare and initialize burst_time[] = {8, 6, 12}
  Set quantum = 2
  Call function findavgTime(processes, n, burst_time, quantum)
```

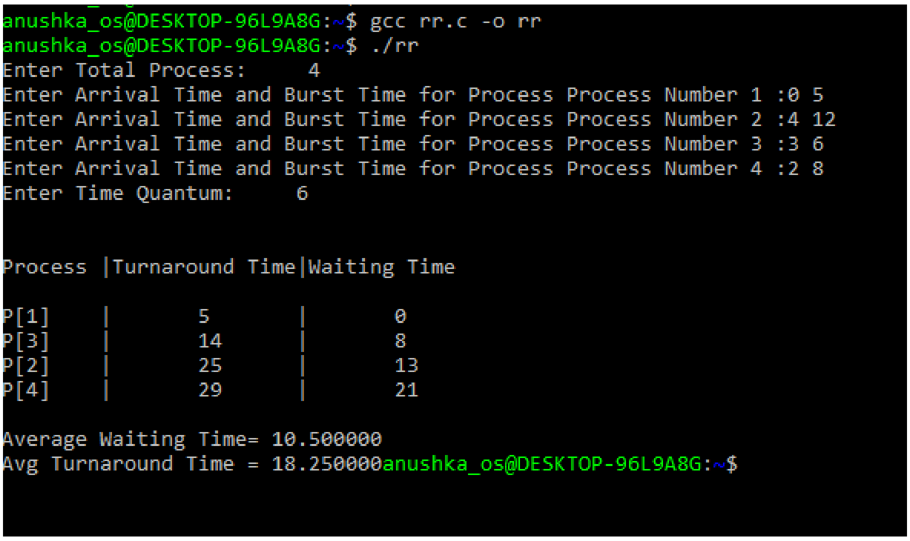
Anushka Dixit [19BCE0577]

Code:



```
 int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
 printf("Enter Total Process:\t ");
 scanf("%d",&n);
 remain=n;
 for(count=0;count<n;count++)
 {
  printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
  scanf("%d",&at[count]);
  scanf("%d",&bt[count]);
  rt[count]=bt[count];
 }
 printf("Enter Time Quantum:\t");
 scanf("%d",&time_quantum);
 printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
 for(time=0,count=0;remain!=0;)
 {
  if(rt[count]<=time_quantum && rt[count]>0)
  {
   time+=rt[count];
   rt[count]=0;
   flag=1;
  }
  else if(rt[count]>0)
  {
   rt[count]-=time_quantum;
   time+=time_quantum;
  }
  if(rt[count]==0 && flag==1)
  {
   remain--;
   printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
   wait_time+=time-at[count]-bt[count];
   turnaround_time+=time-at[count];
   flag=0;
  }
  if(count==n-1)
   count=0;
  else if(at[count+1]<=time)
   count++;
  else
   count=0;
 }
 printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
 printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

 return 0;
}
```

Output:



```
anushka_os@DESKTOP-96L9A8G:~$ gcc rr.c -o rr
anushka_os@DESKTOP-96L9A8G:~$ ./rr
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :0 5
Enter Arrival Time and Burst Time for Process Process Number 2 :4 12
Enter Arrival Time and Burst Time for Process Process Number 3 :3 6
Enter Arrival Time and Burst Time for Process Process Number 4 :2 8
Enter Time Quantum:     6


Process |Turnaround Time|Waiting Time

P[1]    |       5       |       0
P[3]    |       14      |       8
P[2]    |       25      |       13
P[4]    |       29      |       21

Average Waiting Time= 10.500000
Avg Turnaround Time = 18.250000anushka_os@DESKTOP-96L9A8G:~$
```