**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**OPERATING SYSTEM LAB REPORT #09**

**SUBMITTED BY:**

Pradeep Dahal

017BSCIT029

2nd year/ 4th sem

|  |  |
| --- | --- |
|  | Signature |
| Mr. Rabin Maharjan  (Lecturer) |  |
| Department of Computer Science | |

**SUBMITTED TO:**

**Process Scheduling**

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Algorithms for Process Scheduling:

There are five popular process scheduling algorithms −

1. First-Come, First-Served (FCFS) Scheduling
2. Shortest-Job-Next (SJN) Scheduling
3. Shortest Remaining Time First (SRTF) Scheduling
4. Round Robin(RR) Scheduling
5. Priority Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

**First-Come, First-Served (FCFS) Scheduling**

Simplest scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is a non-preemptive scheduling algorithm.

**Source Code:**

#include<stdio.h>

int main()

{

    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

    printf("Enter total number of processes(maximum 20):");

    scanf("%d",&n);

    printf("\nEnter Process Burst Time\n");

    for(i=0;i<n;i++)

    {

        printf("P[%d]:",i+1);

        scanf("%d",&bt[i]);

    }

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

    //calculating waiting time

    for(i=1;i<n;i++)

    {

        wt[i]=0;

        for(j=0;j<i;j++)

            wt[i]+=bt[j];

    }

    printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

    //calculating turnaround time

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];

        avwt+=wt[i];

        avtat+=tat[i];

        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

    }

    avwt/=i;

    avtat/=i;

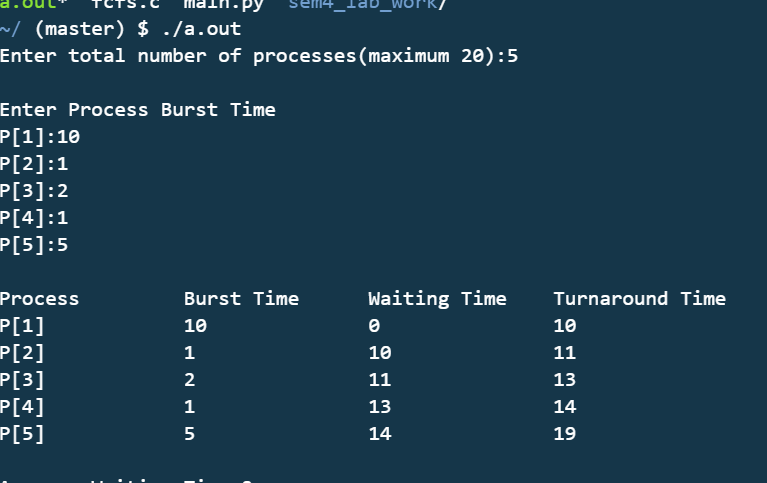
    printf("\n\nAverage Waiting Time:%d",avwt);

    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;

}

**Output:**



**Shortest-Job-Next (SJN) Scheduling**

Process which have the shortest burst time are scheduled first. If two processes have the same burst time, then FCFS is used to break the tie. It is a non-preemptive scheduling algorithm.

**Source Code:**

#include<stdio.h>

void main()

{

    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

    float avg\_wt,avg\_tat;

    printf("Enter number of process:");

    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");

    for(i=0;i<n;i++)

    {

        printf("p%d:",i+1);

        scanf("%d",&bt[i]);

        p[i]=i+1;           //contains process number

    }

    //sorting burst time in ascending order using selection sort

    for(i=0;i<n;i++)

    {

        pos=i;

        for(j=i+1;j<n;j++)

        {

            if(bt[j]<bt[pos])

                pos=j;

        }

        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 will be 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=(float)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]);

    }

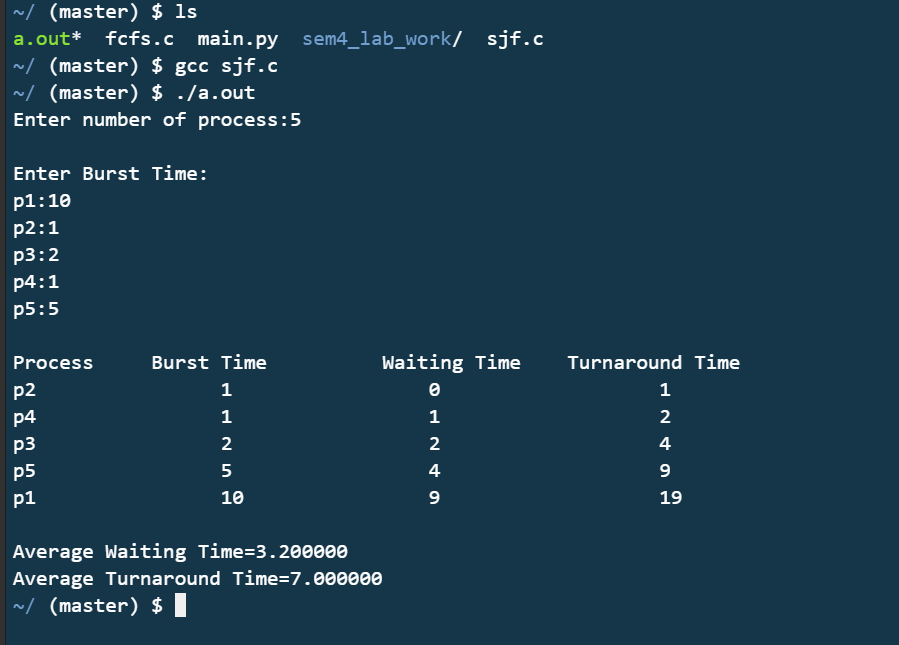
    avg\_tat=(float)total/n;     //average turnaround time

    printf("\n\nAverage Waiting Time=%f",avg\_wt);

    printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

**Output**



**Shortest Remaining Time First (SRTF) Scheduling**

It is preemptive mode of SJF algorithm in which jobs are schedule according to shortest remaining time.

**Source Code:**

#include<stdio.h>

int main()

{

int a[10],b[10],x[10];

int waiting[10],turnaround[10],completion[10];

int i,j,smallest,count=0,time,n;

double avg=0,tt=0,end;

printf("\nEnter the number of Processes: ");

scanf("%d",&n);

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

{

printf("\nEnter arrival time of process %d : ",i+1);

scanf("%d",&a[i]);

}

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

{

printf("\nEnter burst time of process %d : ",i+1);

scanf("%d",&b[i]);

}

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

x[i]=b[i];

b[9]=9999;

for(time=0;count!=n;time++)

{

smallest=9;

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

{

if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )

smallest=i;

}

b[smallest]--;

//printf("\n%d => p%d",time+1,smallest);

if(b[smallest]==0)

{

count++;

end=time+1;

completion[smallest] = end;

waiting[smallest] = end - a[smallest] - x[smallest];

turnaround[smallest] = end - a[smallest];

// printf("\n %d %d %d",smallest,wt[smallest],ttp[smallest]);

}

}

printf("pid \t burst \t arrival \twaiting \tturnaround \tcompletion");

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

{

printf("\n %d \t %d \t %d\t\t%d \t\t%d\t\t%d",i+1,x[i],a[i],waiting[i],turnaround[i],completion[i]);

avg = avg + waiting[i];

tt = tt + turnaround[i];

}

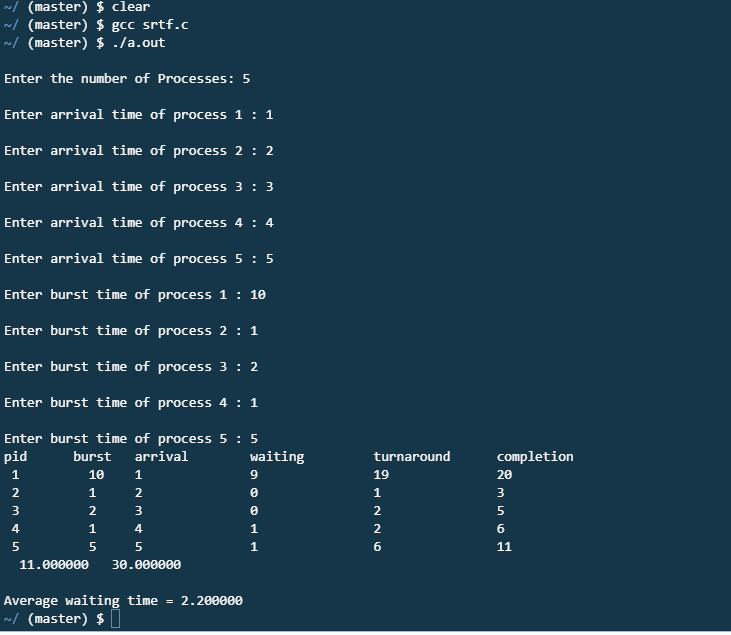
printf("\n %If %If",avg,tt);

printf("\n\nAverage waiting time = %lf\n",avg/n);

printf("Average Turnaround time = %lf",tt/n);

}

**Output:**



**Round Robin (RR) Scheduling**

To schedule processes fairly, a round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum (its allowance of CPU time), and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. If the process terminates or changes its state to waiting during its attributed time quantum, the scheduler selects the first process in the ready queue to execute. In the absence of time-sharing, or if the quanta were large relative to the sizes of the jobs, a process that produced large jobs would be favored over other processes.

Round-robin algorithm is a pre-emptive algorithm as the scheduler forces the process out of the CPU once the time quota expires.

**Source Code:**

#include<stdio.h>

int main()

{

      int i, limit, total = 0, x, counter = 0, time\_quantum;

      int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

      float average\_wait\_time, average\_turnaround\_time;

      printf("\nEnter Total Number of Processes:\t");

      scanf("%d", &limit);

      x = limit;

      for(i = 0; i < limit; i++)

      {

            printf("\nEnter Details of Process[%d]\n", i + 1);

            printf("Arrival Time:\t");

            scanf("%d", &arrival\_time[i]);

            printf("Burst Time:\t");

            scanf("%d", &burst\_time[i]);

            temp[i] = burst\_time[i];

      }

      printf("\nEnter Time Quantum:\t");

      scanf("%d", &time\_quantum);

      printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

      for(total = 0, i = 0; x != 0;)

      {

            if(temp[i] <= time\_quantum && temp[i] > 0)

            {

                  total = total + temp[i];

                  temp[i] = 0;

                  counter = 1;

            }

            else if(temp[i] > 0)

            {

                  temp[i] = temp[i] - time\_quantum;

                  total = total + time\_quantum;

            }

            if(temp[i] == 0 && counter == 1)

            {

                  x--;

                  printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

                  wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

                  turnaround\_time = turnaround\_time + total - arrival\_time[i];

                  counter = 0;

            }

            if(i == limit - 1)

            {

                  i = 0;

            }

            else if(arrival\_time[i + 1] <= total)

            {

                  i++;

            }

            else

            {

                  i = 0;

            }

      }

      average\_wait\_time = wait\_time \* 1.0 / limit;

      average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

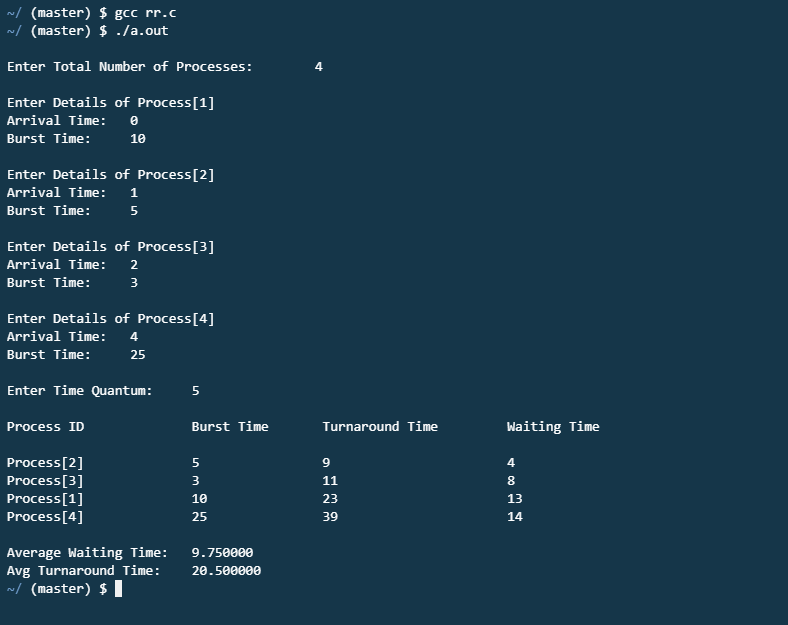
      printf("\n\nAverage Waiting Time:\t%f", average\_wait\_time);

      printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);

      return 0;

}

**Output:**



**Priority Scheduling**

 In this scheduling, processes are scheduled according to their priorities, i.e., highest priority process is scheduled first. If priorities of two processes match, then schedule according to arrival time. Here starvation of process is possible.

**Source Code:**

#include<stdio.h>

void main()

{

int x,n,p[10],pp[10],pt[10],w[10],t[10],awt,atat,i;

printf("Enter the number of process : ");

scanf("%d",&n);

printf("\n Enter process : time priorities \n");

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

{

printf("\nProcess no %d : ",i+1);

scanf("%d %d",&pt[i],&pp[i]);

p[i]=i+1;

}

for(i=0;i<n-1;i++)

{

for(int j=i+1;j<n;j++)

{

if(pp[i]<pp[j])

{

x=pp[i];

pp[i]=pp[j];

pp[j]=x;

x=pt[i];

pt[i]=pt[j];

pt[j]=x;

x=p[i];

p[i]=p[j];

p[j]=x;

}

}

}

w[0]=0;

awt=0;

t[0]=pt[0];

atat=t[0];

for(i=1;i<n;i++)

{

w[i]=t[i-1];

awt+=w[i];

t[i]=w[i]+pt[i];

atat+=t[i];

}

printf("\n\n Job \t Burst Time \t Wait Time \t Turn Around Time Priority \n");

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

printf("\n %d \t\t %d \t\t %d \t\t %d \t\t %d \n",p[i],pt[i],w[i],t[i],pp[i]);

awt/=n;

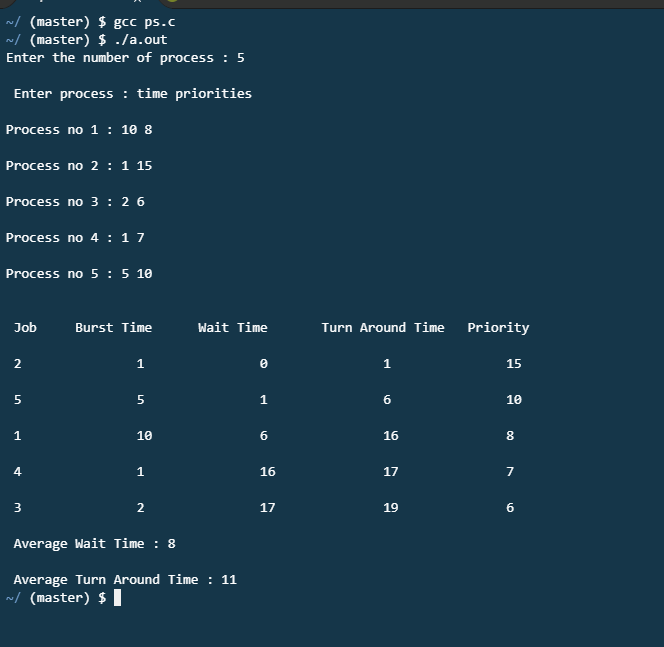
atat/=n;

printf("\n Average Wait Time : %d \n",awt);

printf("\n Average Turn Around Time : %d \n",atat);

}

**Output:**

****