Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology

(An Autonomous Institute affiliated to Savitribai Phule Pune University)

**Assignment-4**

|  |  |
| --- | --- |
| **Subject** | Operating System |
| **Name** | Ajinkya Walunj |
| **Class** | CS-A |
| **Roll No.** | 05 |

**Title**: Implement CPU scheduling algorithms.

1. **FCFS Scheduling**

FCFS (First Come, First Served) is a scheduling algorithm where the process that requests the CPU first is allocated the CPU first. It is a non-pre-emptive algorithm, meaning that once a process is given the CPU, it continues to run until it completes or is blocked. This can lead to poor performance if a process with a long burst time (the amount of time a process needs to execute) is placed ahead of shorter processes in the queue, as the shorter processes will have to wait longer to be executed.

*Code :-*

#include<stdio.h>

struct PCB{

    int pid, arrival\_t,burst\_t,turnaround\_t,finish\_t;

};

void pline(int x);

void main(){

    int i,num,j;

    float avg=0.0,sum=0.0;

    struct PCB p[10], temp;

    printf("Enter the total number of processes: ");

    scanf("%d",&num);

    for(int i=0;i<num;i++){

        printf("Enter the arriaval time and burst time for process %d: \n",i+1);

        scanf("%d %d",&p[i].arrival\_t,&p[i].burst\_t);

        p[i].pid=i+1;

    }

    //For sorting according to arival time

    for(i=0;i<num;i++){

        for(j=0;j<num-i-1;j++){

            if(p[j].arrival\_t>p[j+1].arrival\_t){

                temp=p[j];

                p[j]=p[j+1];

                p[j+1]=temp;

            }

        }

    }

    //Calculation of Finish time

    for(i=0;i<num;i++){

        sum=sum+p[i].burst\_t;

        p[i].finish\_t=sum;

    }

    //Calculation of Turnaround time

    for(i=0;i<num;i++){

        p[i].turnaround\_t=p[i].finish\_t-p[i].arrival\_t;

    }

    sum=0;

    //Following function used to draw line

    pline(44);

    printf("PID\tArrival\tBurst\tFinish\tTurnaround\n");

    pline(44);

    for(i=0;i<num;i++){

        printf("%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].arrival\_t,p[i].burst\_t,p[i].finish\_t,p[i].turnaround\_t);

        sum+=p[i].turnaround\_t;//To sum all the processes turnaround

    }

    pline(44);

    avg = sum/(float)num;

    printf("\nTotal turnaround time : %f.",sum);

    printf("\nAverage turnaround time : %.3f",avg);

}

void pline(int x){

    int i;

    for(i=0;i<x;i++){

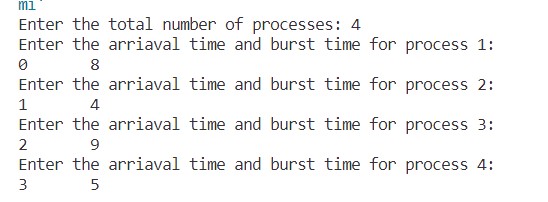
        printf("-");

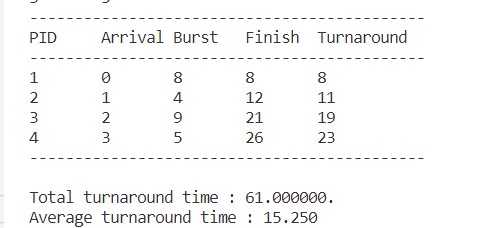
    }

    printf("\n");

}

*Output :-*

**



1. **Shortest Job First Non-Pre-emptive (SJF)**

SJF (Shortest Job First) is a scheduling algorithm where the process with the shortest burst time is allocated the CPU first. It is a non-pre-emptive algorithm, meaning that once a process is given the CPU, it continues to run until it completes or is blocked. This can lead to better performance than the FCFS algorithm, as shorter processes are executed first, reducing the amount of time that other processes have to wait. However, since it is a non-pre-emptive algorithm, it is not suitable for real-time systems, where a process may have a higher priority and needs to be executed immediately.

*Code :-*

#include<stdio.h>

struct PCB{

    int p\_id,burst\_t,finish\_t,turnaround\_t,wait\_t;

};

void pline(int);

void main(){

    struct PCB p[10],temp;

    int i,n,j=1,sum=0,w\_tot=0,tt\_tot=0;

    float w\_avg,tt\_avg;

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

    scanf("%d",&n);

    for(i=0;i<n;i++){

        printf("Enter the burst\_time of process %d : \n",i+1);

        scanf("%d",&p[i].burst\_t);

        p[i].p\_id=i+1;

    }

    for(i=0;i<n;i++){

        for(j=0;j<n-i-1;j++){

            if(p[j].burst\_t>p[j+1].burst\_t){

                temp=p[j];

                p[j]=p[j+1];

                p[j+1]=temp;

            }

        }

    }

    for(i=0;i<n;i++){

        p[i].wait\_t=sum;

        sum+=p[i].burst\_t;

        p[i].turnaround\_t=sum;

    }

    //calculating finish time

    p[0].finish\_t=p[0].burst\_t;

    for(i=1;i<n;i++){

        p[i].finish\_t=p[i].burst\_t+p[i-1].finish\_t;

    }

    pline(35);

    printf("PID\tBurst\tFinish\tWait\tTurnaround\n");

    pline(35);

    //Below for loop to print values

    for(i=0;i<n;i++){

        printf("%d\t%d\t%d\t%d\t%d\n",p[i].p\_id,p[i].burst\_t,p[i].finish\_t,p[i].wait\_t,p[i].turnaround\_t);

        w\_tot+=p[i].wait\_t;

        tt\_tot+=p[i].turnaround\_t;

    }

    w\_avg=w\_tot/(float)n;

    tt\_avg=tt\_tot/(float)n;

    printf("\nTotal waiting time : %d",w\_tot);

    printf("\nAverage waiting time : %.3f",w\_avg);

    printf("\nTotal turnaround time : %d",tt\_tot);

    printf("\nAverage turnaround time : %.3f",tt\_avg);

}

void pline(int x){

    int i;

    for(i=0;i<x;i++){

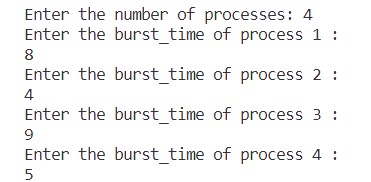
        printf("-");

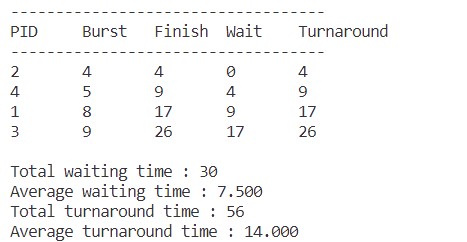
    }

    printf("\n");

}

*Output :-*





1. **Shortest Job First Preemptive (SJF)**

SJF (Shortest Job First) is a scheduling algorithm where the process with the shortest burst time is allocated the CPU first. It is a preemptive algorithm, meaning that once a process is given the CPU, it continues to run until it completes or is blocked, unless a new process with a shorter burst time arrives and the current process is preempted and the new process is executed. This can lead to better performance than the non-preemptive version of SJF and FCFS algorithm, as shorter processes are executed first, reducing the amount of time that other processes have to wait. It is also suitable for real-time systems, where a process may have a higher priority and needs to be executed immediately.

*Code:-*

*Output:-*

1. **Round Robin Scheduling**

Round Robin (RR) is a scheduling algorithm that allocates the CPU to each process for a fixed time called a time slice or quantum. Once the time slice is over, the process is moved to the end of the queue and the next process in the queue is given the CPU. This process continues until all the processes in the queue have been given the CPU. The algorithm is simple to implement, but it requires a fixed time slice, and if the time slice is too short, it can lead to a lot of context switching, which can decrease system performance. On the other hand, if the time slice is too long, it can lead to poor response time for interactive processes. It is suitable for time-sharing systems, where multiple users are connected to the system simultaneously and need a fair share of the resources.

*Code:-*

*Output:-*

1. **Priority Scheduling (Non – pre-emptive)**

Priority Scheduling is a scheduling algorithm that assigns a priority to each process, and the process with the highest priority is allocated the CPU first. If two or more processes have the same priority, then they are scheduled using a tie-breaker algorithm such as FCFS or Round Robin. It is a non-preemptive algorithm, meaning that once a process is given the CPU, it continues to run until it completes or is blocked. This algorithm ensures that high-priority processes are executed quickly, but it can lead to lower-priority processes being starved, i.e., never getting a chance to execute.

*Code :-*

#include<stdio.h>

struct PCB{

    int pid,burst\_t,priority,turnaround\_t,wait\_t;

};

void pline(int x);

void main(){

    int i,num,j=1,sum=0,w\_tot=0,tt\_tot=0;

    float w\_avg,tt\_avg;

    struct PCB p[10], temp;

    printf("Enter the total number of processes: ");

    scanf("%d",&num);

    for(int i=0;i<num;i++){

        printf("Enter the burst time and priority for process %d: \n",i+1);

        scanf("%d %d",&p[i].burst\_t,&p[i].priority);

        p[i].pid=i+1;

    }

    //For sorting according to arival time

    for(i=0;i<num;i++){

        for(j=0;j<num-i-1;j++){

            if(p[j].priority>p[j+1].priority){

                temp=p[j];

                p[j]=p[j+1];

                p[j+1]=temp;

            }

        }

    }

    for(i=0;i<num;i++){

        p[i].wait\_t=sum;

        sum+=p[i].burst\_t;

        p[i].turnaround\_t=sum;

    }

    //Following function used to draw line

    pline(44);

    printf("PID\tArrival\tBurst\tPriority\tTurnaround\tWaittime\n");

    pline(44);

    for(i=0;i<num;i++){

        printf("%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].burst\_t,p[i].priority,p[i].turnaround\_t,p[i].wait\_t);

        w\_tot+=p[i].wait\_t;

        tt\_tot+=p[i].turnaround\_t;

    }

    pline(44);

    tt\_avg = tt\_tot/(float)num;

    w\_avg=w\_tot/(float)num;

    printf("\nTotal turnaround time : %d",tt\_tot);

    printf("\nAverage turnaround time : %.3f",tt\_avg);

    printf("\nTotal waiting time : %d.",w\_tot);

    printf("\nAverage waiting time : %.3f",w\_avg);

}

void pline(int x){

    int i;

    for(i=0;i<x;i++){

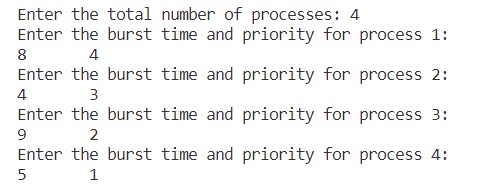
        printf("-");

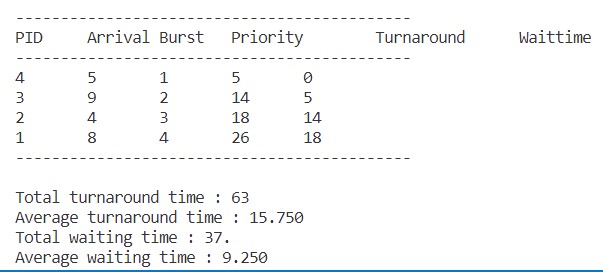
    }

    printf("\n");

}

*Output :-*

**

**

1. **Priority Scheduling (Pre-emptive)**

*Code :-*

#include<stdio.h>

struct process

{

    int WT,AT,BT,TAT,PT;

};

struct process a[10];

int main()

{

    int n,temp[10],t,count=0,short\_p;

    float total\_WT=0,total\_TAT=0,Avg\_WT,Avg\_TAT;

    printf("Enter the number of the process\n");

    scanf("%d",&n);

    printf("Enter the arrival time , burst time and priority of the process\n");

    printf("AT BT PT\n");

    for(int i=0;i<n;i++)

    {

        scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);

        // copying the burst time in

        // a temp array fot futher use

        temp[i]=a[i].BT;

    }

    // we initialize the burst time

    // of a process with maximum

    a[9].PT=10000;

    for(t=0;count!=n;t++)

    {

        short\_p=9;

        for(int i=0;i<n;i++)

        {

            if(a[short\_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)

            {

                short\_p=i;

            }

        }

        a[short\_p].BT=a[short\_p].BT-1;

        // if any process is completed

        if(a[short\_p].BT==0)

        {

            // one process is completed

            // so count increases by 1

            count++;

            a[short\_p].WT=t+1-a[short\_p].AT-temp[short\_p];

            a[short\_p].TAT=t+1-a[short\_p].AT;

            // total calculation

            total\_WT=total\_WT+a[short\_p].WT;

            total\_TAT=total\_TAT+a[short\_p].TAT;

        }

    }

    Avg\_WT=total\_WT/n;

    Avg\_TAT=total\_TAT/n;

    // printing of the answer

    printf("ID WT TAT\n");

    for(int i=0;i<n;i++)

    {

        printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);

    }

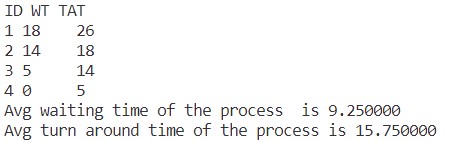
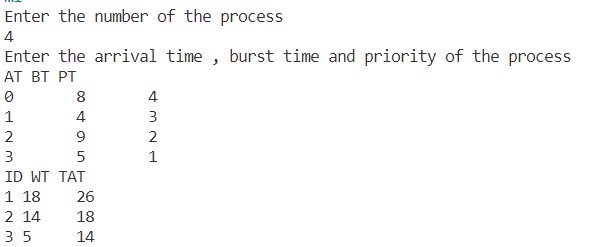
    printf("Avg waiting time of the process  is %f\n",Avg\_WT);

    printf("Avg turn around time of the process is %f\n",Avg\_TAT);

    return 0;

}

*Output :-*

**