OS EXPERIMENTS

EXPERIMENT NO: 01 **R HARIKRISHNAN**
DATE: 06-08-24 **ROLL NO:62**

CPU SCHEDULING ALGORITHMS

AIM

To simulate following CPU scheduling algorithms to find turnaround time and waiting time :
a) FCFS
b) SJF
c) Round Robin
d) Priority

ALGORITHM

- **FCFS**

1. Declare the variables
2. Declare the variable i,n as integer, totalwtime and total time is equal to zero
3. Get the value of btime[i]
4. Assign wtime[0] as zero and tatime[0] as btime[0] and inside the loop calculate waiting time and turnaround time.
5. Calculate total waiting time and total turnaround time and calculate average waiting time and turnaround time by dividing it by the total number of process
6. Print total waiting time, total turnaround time, average waiting time and average turnaround time
7. Stop the program

- **SJF**

1. Declare the variables
2. Declare the variable i,j as integer,totaltatime and totalwtime is equal to zero
3. Get the value of n and assign burst time for each process
4. Assign wtime[0] as zero and tatime[0] as btime[0] and inside the loop calculate wait time and turnaround time
5. Calculate total waiting time and total turnaround time and calculate average waiting time and average Turnaround time by dividing it by total number of process
6. Print total waiting time, total turnaround time, average waiting time, and average turnaround time
7. Stop the program

- **ROUND ROBIN**

1. Declare the variables
2. Declare the variable i,j as integer, totalwtime and totalttime is equal to zero
3. Get the number of processes n and time quantum
4. Inside the for loop get the value of burst time and arrival time
5. Check burst time of process is greater than time quantum or not
6. Calculate waiting time and turnaround time of processes
7. Calculate the total of waiting time and turnaround time and find average of waiting time and turnaround time by dividing it by number of processes
8. Stop the program

- **PRIORITY**

1. Declare the variables
2. Declare the variable i,j as integer,totaltatime and totalwtime is equal to zero
3. Get the value of n and assign burst time for each process
4. Assign wtime[0] as zero and tatime[0] as btime[0] and inside the loop calculate wait time and turnaround time
5. Calculate total waiting time and total turnaround time and calculate average waiting time and average turnaround time by dividing it by totalnumber of process
6. Print total waiting time, total turnaround time, average waiting time, and average turnaround time
7. Stop the program

PROGRAM CODE

```
#include<stdio.h>
#include<stdlib.h>
struct process
{
    int no,bt,at,tat,wt,ct,prior,id;
}p[20];
int ready,n,a,ct,b,t,i,j,q[50],f=-1,r=-1;
float sum_wt=0.0,sum_tat=0.0,avg_wt,avg_tat;
void sort(int n)
{
struct process temp;
for(i=0;i<n-1;i++)
{
 for(j=0;j<n-i-1;j++)
    {
    if(p[j].at>p[j+1].at)
    {
        temp=p[j];
        p[j]=p[j+1];
        p[j+1]=temp;
    }  }  }
}
void FCFS(){
int flag;
```

```
printf("\nEnter the number of processes : ");
scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("\nEnter arrival time and burst time of process P%d : ",i);
 scanf("%d%d",&p[i].at,&p[i].bt);

  p[i].no=i+1;
 }
 sort(n);
  p[0].ct=p[0].at+p[0].bt;
       for(i=1;i<n;i++)
       {
            if(p[i].at>p[i-1].ct)
            {
                 p[i].ct=p[i].at+p[i].bt;
            }
            else
            {
                 p[i].ct=p[i-1].ct+p[i].bt;
            }
}
```

```
for(i=0;i<n;i++)
        {
        p[i].tat=p[i].ct-p[i].at;
                p[i].wt=p[i].tat-p[i].bt;
sum_wt+=p[i].wt;
sum_tat+=p[i].tat;
}
avg_wt=sum_wt/n;
avg_tat=sum_tat/n;
printf("\nPROCESS\t ARRIVAL TIME \t BURST TIME \t TURNARROUND TIME
\t WAITING TIME\n");
for(i=0;i<n;i++)


{
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",p[i].no,p[i].at,p[i].bt,p[i].tat,p[i].wt);
}
printf("\nAverage waiting time is %.2f\n\n",avg_wt);
printf("Average turnarround time %.2f\n\n",avg_tat);
}

void SJF()
{
int count=0,t=0,short_p,temp[10],n,i;

floattotal_wt=0,total_tat=0,awt,atat;
        printf("\nEnter the number of
        proceses:\n"); scanf("%d",&n);
        for(i=0;i<n;i++)
        {
        printf("\nEnter arrival time and burst time of process P%d : ",i);
        scanf("%d%d",&p[i].at,&p[i].bt);
        temp[i]=p[i].bt;

        }
        p[19].bt=10000;
        for(t=0;count!=n;t++)
        {
```

```
        short_p=19;
for(i=0;i<n;i++)
                {
                        if(p[i].bt<p[short_p].bt&& (p[i].at<=t && p[i].bt>0))
                        {
                                short_p=i;
                        }

                }
                p[short_p].bt=p[short_p].bt-1;
                if(p[short_p].bt==0)
                {
                        count++;
                        p[short_p].wt=t+1-p[short_p].at-temp[short_p];
        p[short_p].tat=t+1-p[short_p].at;

        total_wt+=p[short_p].wt;
        total_tat+=p[short_p].tat;
        }
        }
        awt=total_wt/n;
        atat=total_tat/n;
   printf("process , wt, tat\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t%d\t%d\n",i+1,p[i].wt,p[i].tat);
        }

        printf("Average waiting time :%.2f\n",awt);
        printf("\nAverage turnarround time:%.2f\n",atat);


  }
  void RR()
  {
  int queue[100];
```

```
 int F=-1;
 int R=-1;
        void insert(int n)
        {
if(F==-1)
F=0; R+=1;
                queue[R]=n;
        }
        int delete()
        {
                int n;
                n=queue[F];
                F+=1;
                return n;
        }
                int n,TQ,a,time=0;
                int temp[10],exist[10]={0};
                float total_wt=0,total_tat=0,avg_wt,avg_tat;
                printf("\nEnter the number of process:\n");
  scanf("%d",&n);
                for(int i=0;i<n;i++)

                {

                        printf("\nEnter arrival time and burst time of process P%d : ",i);
                       scanf("%d%d",&p[i].at,&p[i].bt); p[i].id=i;


                       temp[i]=p[i].bt;
                }printf("\nEnter the time quantum:\n");
                scanf("%d",&TQ);
                insert(0);
                exist[0]=1;
```

```
        while(F<=R)
        {
            a=delete();
            if(p[a].bt>=TQ)
            {
                p[a].bt=p[a].bt-TQ;
                time+=TQ;
            }
            else
            {
                time+=p[a].bt;
                p[a].bt=0;
            }



             for(int i=0;i<n;i++)
            {
                if(exist[i]==0 && p[i].at<=time)
                {
                insert(i);
                exist[i]=1;
}
            }
            if(p[a].bt==0)
            {
                p[a].tat=time-p[a].at;
                p[a].wt=p[a].tat-temp[a];
                total_tat=total_tat+p[a].tat;
                total_wt=total_wt+p[a].wt;
            }
            else
```

```
{
                        insert(a);
                }
        }
avg_tat=total_tat/n;
avg_wt=total_wt/n;
        // printing of the answer
        printf("ID WT TAT\n");
        for(int i=0;i<n;i++)
        {
                printf("%d %d %d\n",p[i].id,p[i].wt,p[i].tat);
        }
        printf("Average waiting time of the processes is : %.2f\n",avg_wt);
        printf("\nAverage turn around time of the processes is : %.2f\n\n",avg_tat);
    }
  void Priority(){
    int i,n,temp[20],t,count=0,sp;
    float to_wt=0,to_tat=0,avg_wt,avg_tat;
    printf("Enter the no of processes : ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
         printf("\nEnter Arrival time and burst time,priority of process P%d : \n",i);
         scanf("%d%d%d",&p[i].at,&p[i].bt,&p[i].prior);
        p[i].no=i;
        temp[i]=p[i].bt;
    }
    p[9].prior=1000;
    for(t=0;count!=n;t++)
    {
        sp=9;
        for(i=0;i<n;i++)
        {
            if(p[sp].prior>p[i].prior && p[i].at<=t && p[i].bt>0)
```

```
                {
                    sp=i;
                }
            }
            p[sp].bt=p[sp].bt-1;
            if(p[sp].bt==0)
            {
                count++;
                p[sp].tat=t+1-p[sp].at;
                p[sp].wt=p[sp].tat-temp[sp];
                to_wt+=p[sp].wt;
                to_tat+=p[sp].tat;
            }
        }
        avg_tat=to_tat/n;
        avg_wt=to_wt/n;
        printf("P\tARRIVAL TIME\tBURST TIME\tWAITING TIME\tTURNARROUND
TIME\tPRIORITY\n");
        for(i=0;i<n;i++)
        {

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,p[i].at,temp[i],p[i].wt,p[i].tat,p[i].prior);
        }
        printf("Average turnarrounf time : %.2f\n",avg_tat);
        printf("\nAverage waiting time : %.2f\n",avg_wt);
}
 int main()
{
        int opt;
        do{
        printf("Enter the choice :\n 1.FCFS\n2.SJF\n3.RR\n4.Priority\n5.Exit\n");
        scanf("%d",&opt);
            switch(opt)
            {
```

```
            case 1:
            FCFS();
            break;
            case 2:
            SJF();
            break;
            case 3:
            RR();
            break;
            case 4:
            Priority();
            break;
            case 5:
            printf("Exit");
            break;
            default:
            printf("Enter the choice:");
            break;
        }
}
while(opt!=5);
return 0;
}
```

OUTPUT

FCFS

```
PS C:\Users\HP\Desktop\CODES ARE HERE\SS LAB> ./cpu
Enter the choice :
1.FCFS
2.SJF
3.RR
4.Priority
5.Exit
1

Enter the number of processes : 5

Enter arrival time and burst time of process P0 : 0 8

Enter arrival time and burst time of process P1 : 2 6

Enter arrival time and burst time of process P2 : 2 1

Enter arrival time and burst time of process P3 : 1 9

Enter arrival time and burst time of process P4 : 3 3

PROCESS  ARRIVAL TIME    BURST TIME      TURNARROUND TIME       WAITING TIME
1               0               8               8               0
4               1               9               16              7
2               2               6               21              15
3               2               1               22              21
5               3               3               24              21

Average waiting time is 12.80

Average turnarround time is 18.20
```

SJF

```
Enter the choice :
1.FCFS
2.SJF
3.RR
4.Priority
5.Exit
2

Enter the number of proceses:
3

Enter arrival time and burst time of process P0 : 5 7

Enter arrival time and burst time of process P1 : 6 14

Enter arrival time and burst time of process P2 : 7 12
process , wt, tat
1       0       7
2       18      32
3       5       17
Average waiting time :7.67

Average turnarround time:18.67
```

ROUNDROBIN

```
Enter the choice :
1.FCFS
2.SJF
3.RR
4.Priority
5.Exit
3

Enter the number of process:
4

Enter arrival time and burst time of process P0 : 0 5

Enter arrival time and burst time of process P1 : 0 4

Enter arrival time and burst time of process P2 : 0 2

Enter arrival time and burst time of process P3 : 0 1

Enter the time quantum:
2
ID WT TAT
0 7 12
1 7 11
2 4 6
3 6 7
Average waiting time of the processes is : 6.00

Average turn around time of the processes is : 9.00
```

PRIORITY

```
Enter the choice :
1.FCFS
2.SJF
3.RR
4.Priority
5.Exit
4
Enter the no of processes : 5

Enter Arrival time and burst time,priority of process P0 :
0 3 3

Enter Arrival time and burst time,priority of process P1 :
1 6 9

Enter Arrival time and burst time,priority of process P2 :
3 1 9

Enter Arrival time and burst time,priority of process P3 :
2 2 7

Enter Arrival time and burst time,priority of process P4 :
4 4 8
P       ARRIVAL TIME    BURST TIME      WAITING TIME    TURNARROUND TIME
0               0               3               0               3
1               1               6               8               14
2               3               1               12              13
3               2               2               1               3
4               4               4               1               5
Average turnarrounf time : 7.60

Average waiting time : 4.40
```

RESULT

PROGRAM EXECUTED SUCCESSFULLY.

EXPERIMENT NO: 02 **R HARIKRISHNAN**
DATE: 10-08-24 **ROLL NO:62**

BANKER'S ALGORITHM

AIM

To write a C program to simulate the banker's algorithm for deadlock avoidance

ALGORITHM

Data Strutures Used :-

1. Available :- A vector of length 'm' indicates the number of available process of each type
2. Max :- An n*m matrix defines the maximum demand of each process
3. Allocation :- n*m matrix defines the number of resources of each type currently allocated to each process
4. Need :- n*m matrix indicates the remaining resource need of each processNeed[i][j]=Max[i][j]-Alloaction[i][j]

1 . Work are finish are the vectors of length m and n

Initialize Work=Available

Finish[i]=false for i=0,1,..,n-1

2 . Find index i such that Finish[i]==false and Needi <= Work

If no such I exists , go to step 4

3 .Work = Work + Allocation

Finish[i]=true

Go to step 2

4 . If Finish[i]==true , then the system is in safe state

PROGRAM

```
#include<stdio.h>

int main()


 {

 int i,j,k,y,n,m,p,alloc[10],Alloc[20][20],Max[20][20],Need[20][20],Avail[20],finish[20],safe[20],ind=0,fl
 ag;

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

 scanf("%d",&n);
```

```
printf("\nEnter the number of resource types : ");
scanf("%d",&m);
printf("\nEnter the current allocations of each process\n");
for(i=0;i<n;i++)
{
        for(j=0;j<m;j++)
        {
                scanf("%d",&Alloc[i][j]);
        }
}
printf("\nEnter the maximum allocations for each process : ");
for(i=0;i<n;i++)
{
        for(j=0;j<m;j++)
        {
                scanf("%d",&Max[i][j]);
        }
}
printf("\nEnter the available resources : ");
for(i=0;i<m;i++)
{
        scanf("%d",&Avail[i]);
}
printf("\nNeed matrix is :
\n");for(i=0;i<n;i++)
{
        printf("\n");
        for(j=0;j<m;j++)
        {
                Need[i][j]=Max[i][j]-Alloc[i][j];
                printf("%d ",Need[i][j]);
        }
}
printf("\nEnter the process which needs extra allocation : ");
scanf("%d",&p)
```

```
printf("\nEnter the request : ");
for(i=0;i<m;i++)
{
        scanf("%d",&alloc[i]);
        Alloc[p][i]+alloc[i];
}
printf("\nNeed matrix changed to :
\n");for(i=0;i<n;i++)
    {
        printf("\n");
        for(j=0;j<m;j++)
        {
            Need[i][j]=Max[i][j]-Alloc[i][j];
            printf("%d ",Need[i][j]);
        }
    }
for(i=0;i<n;i++)
{
        finish[i]=0;
}
for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        {
            if(finish[j]==0)
            {
                flag=0;
                for(k=0;k<m;k++)
                {
                    if(Need[j][k]>Avail[k])
                    {
                        flag=1;
                        break;
                    }
                }
```

```
                    if(flag==0)
                    {
                        safe[ind++]=j;
                        for(y=0;y<m;y++)
                            {
                                Avail[y]+=Alloc[j][y];
                            }
                        finish[j]=1;
                    }}}
}
flag=1;
for(i=0;i<n;i++)
{
    if(finish[i]==0)
    {
        flag=0;
        printf("\nThe system is not in safe state\n");
        break;
     }}
 if(flag==1)
{
    printf("\nSafe sequence is :-
    \n");for(i=0;i<n;i++)
    {
        printf("P%d",safe[i]);
        if(i<n-1)
        {
        printf("\n");
        }}]
        return 0;
}
```

OUTPUT

```
Enter the number of processes : 5

Enter the number of resource types : 3

Enter the current allocations of each process
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter the maximum allocations for each process :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter the available resources : 3 3 2

Need matrix is :

7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
Enter the process which needs extra allocation : 2

Enter the request : 1 0 2

Need matrix changed to :

7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
Safe sequence is :-
P1-->P3-->P4-->P0-->P2
```

RESULT

PROGRAM EXECUTED SUCCESSFULLY.

EXPERIMENT NO: 03 **R HARIKRISHNAN**
DATE: 10-08-24 **ROLL NO:62**

DISK SCHEDULING ALGORITHMS

AIM

To Write a C program to simulate the following disk scheduling algorithms
a) FCFS
b) SCAN
c) C-SCAN

ALGORITHM

FCFS:

1. Let us one by one take the tracks in default order and calculate the absolutedistance of the track from the head.
2. Increment the total seek count with this distance.
3. Currently serviced track position now becomes the new head position.
4. Go to step 2 until all tracks in the request array have not been serviced.
5. Let Request array represents an array storing indexes of tracks that have beenrequested in ascending order of their time of arrival. 'head' is the position ofdisk head.

SCAN:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival .'head' is the position of diskhead.
2. Let direction represents whether the head is moving towards left or right.
3. In the direction in which head is moving service all tracks one by one.
4. Calculate the absolute distance of the track from the head.
5. Increment the total seek count with this distance.
6. Currently serviced track position now becomes the new head position.
7. Go to step 3 until we reach at one of the ends of the disk.
8. If we reach at the end of the disk reverse the direction and go tostep 2 until all tracks in request array have not been serviced

C-SCAN

1. Let Request array represents an array storing indexes of tracks that have been requested

in ascending order of their time of arrival. 'head' is the position of disk head.

2. The head services only in the right direction from 0 to size of the disk.
3. While moving in the left direction do not service any of the tracks.
4. When we reach at the beginning (left end) reverse the direction.
5. While moving in right direction it services all tracks one by one.
6. While moving in right direction calculate the absolute distance of the track from the head.
7. Increment the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach at right end of the disk.
10. If we reach at the right end of the disk reverse the direction and go to step 3 until all tracks in request array have not been serviced.

PROGRAM

```
#include<stdio.h>
#include<stdlib.>
int i,j;
void fcfs(void)
{
int n,RQ[20],head,THM=0;
printf("\nEnter the number of requests : ");
scanf("%d",&n);
printf("\nEnter the request sequence : ");
for(i=0;i<n;i++)
{
scanf("%d",&RQ[i]);
}
printf("\nEnter the initial head position : ");
scanf("%d",&head);
for(i=0;i<n;i++)
{
THM+=abs(RQ[i]-
head);head=RQ[i];
}
printf("\nTotal head moment is : %d\n",THM);
```

```
}
void scan(void)
{
int indx,temp,size,n,RQ[20],head,move;
printf("\nEnter the number of requests : ");
      scanf("%d",&n);
      printf("\nEnter the request sequence : ");
      for(i=0;i<n;i++)
      {
            scanf("%d",&RQ[i]);
      }
      printf("\nEnter the initial head position : ");
      scanf("%d",&head);
printf("\nEnter the size of disk : ");
scanf("%d",&size);
printf("\nEnter the head movement direction(1 for high and 0 for low) : ");
scanf("%d",&move);
for(i=0;i<n-1;i++){
for(j=0;j<n-i-1;j++)
{
if(RQ[j]>RQ[j+
1])
{
temp=RQ[j];
RQ[j]=RQ[j+1
];
RQ[j+1]=temp;
}}}
for(i=0;i<n;i++)
{
if(head<RQ[i])
{
```

```
index=i;
break;
}
}
int THM=0;
if(move==1
)
{
for(i=indx;i<n;i++)
{
THM+=abs(RQ[i]-
head);head=RQ[i];
}
THM+=abs(size-RQ[i-1]-
1);head=size-1;
for(i=indx-1;i>=0;i--)
{
THM+=abs(RQ[i]-
head);head=RQ[i];
}
}
else
{
for(i=indx-1;i>=0;i--)
{
        THM+=abs(RQ[i]-head);
        head=RQ[i];
    }
    THM+=abs(RQ[i+1]-0);
    head=0;
    for(i=indx;i<n;i++)
    {
```

```
THM+=abs(RQ[i]-head);
head=RQ[i];
            }
 }
 printf("\nTotal head moment is : %d\n",THM);
 }
 void cscan(void)
 {
 int indx,temp,n,size,RQ[20],head,move;
 printf("\nEnter the number of requests : ");
      scanf("%d",&n);
      printf("\nEnter the request sequence : ");
      for(i=0;i<n;i++)
      {
           scanf("%d",&RQ[i]);
      }
      printf("\nEnter the initial head position : ");
      scanf("%d",&head);
 printf("\nEnter the size of disk : ");
      scanf("%d",&size);
      printf("\nEnter the head movement direction(1 for high and 0 for low) :
      "); scanf("%d",&move);
 for(i=0;i<n;i++)
 {
 for(j=0;j<n-i-1;j++)
 {
 if(RQ[j]>RQ[j+
 1])
 {
 temp=RQ[j];
 RQ[j]=RQ[j+1];
 RQ[j+1]=temp;}}}
```

```
for(i=0;i<n;i++)
        {
              if(head<RQ[i])
              {
                    indx=
                    i;
                    break
                    ;
              }
        }
int THM=0 if(move==1)
        {
              for(i=indx;i<n;i++)
              {
                    THM+=abs(RQ[i]-head);
                    head=RQ[i];
              }
              THM+=abs(size-RQ[i-1]-
  1);THM+=abs(size-1-0);

              head=0;
              for(i=0;i<indx;i++)
              {
   THM+=abs(RQ[i]-head);
                    head=RQ[i];
}}
else {
  for(i=indx-1;i>=0;i--)

{
THM+=abs(RQ[i]-head);

                    head=RQ[i]
```

```
THM+=abs(RQ[i+1]-0);
 THM+=abs(size-1-0);

 head=size-1;
            for(i=n-1;i>=indx;i--)
            {
                THM+=abs(RQ[i]-head);
                head=RQ[i];
            }
       }

       printf("Total head moment is : %d\n",THM);
 }

 int main()
 {
 int ch;
 do{

printf("\nEnter choice\n1 for fcfsc\n2 for scan\n3 for c-scan\t : ");
scanf("%d",&ch);
 switch(ch){
 case 1: fcfs(); break;
 case 2: scan(); break;
 case 3: cscan();break;
  default:
            printf("Exit\n");
            break;
 }
 }

 while(ch!=4);
```

OUTPUT

FCFS

```
PS C:\Users\HP\Desktop\CODES ARE HERE\SS LAB> ./disk

Enter choice
1 for fcfsc
2 for scan
3 for c-scan     : 1

Enter the number of requests : 7

Enter the request sequence : 82 170 43 140 24 16 190

Enter the initial head position : 50

Total head moment is : 642
```

SCAN

```
Enter choice
1 for fcfsc
2 for scan
3 for c-scan     : 2

Enter the number of requests : 7

Enter the request sequence : 82 170 43 140 24 16 190

Enter the initial head position : 50

Enter the size of disk : 200

Enter the head movement direction(1 for high and 0 for low) : 1

Total head moment is : 332

Enter choice
1 for fcfsc
2 for scan
3 for c-scan     : 2

Enter the number of requests : 7

Enter the request sequence : 82 170 43 140 24 16 190

Enter the initial head position : 50

Enter the size of disk : 200

Enter the head movement direction(1 for high and 0 for low) : 0

Total head moment is : 240
```

C-SCAN

```
Enter choice
1 for fcfsc
2 for scan
3 for c-scan     : 3

Enter the number of requests : 8

Enter the request sequence : 98 183 41 122 14 124 65 67

Enter the initial head position : 53

Enter the size of disk : 200

Enter the head movement direction(1 for high and 0 for low) : 1
Total head moment is : 386

Enter choice
1 for fcfsc
2 for scan
3 for c-scan     : 3

Enter the number of requests : 8

Enter the request sequence : 98 183 41 122 14 124 65 67

Enter the initial head position : 53

Enter the size of disk : 200

Enter the head movement direction(1 for high and 0 for low) : 0
Total head moment is : 386
```

RESULT

PROGRAM EXECUTED SUCCESSFULLY.

SS EXPERIMENTS

EXPERIMENT NO: 04 **R HARIKRISHNAN**
DATE: 11-09-24 **ROLL NO:62**

PASS 1 OF TWO PASS ASSEMBLER

AIM
To implement pass 1 of a two pass assembler

INPUT
Assembly Language program, Operation code table

OUTPUT
Intermediate file, Symbol table, Program length

ALGORITHM

Begin
Read input line
if OPCODE = ‘START’, then
Set Starting Address as #[Operand]
Initialize LOCCTR to Starting Address
Write line to Intermediate file
Read next line
else
Initialize LOCCTR to 0
Set Starting Address to 0
while OPCODE != ‘END’, do
if line is not a comment, then
if there is a symbol in the LABEL field, then
Search SYMTAB for LABEL
if found, then
Set error flag(duplicate symbol)
else
Add symbol to SYMTAB with it’s address
end if
end if
Search OPTAB for OPCODE
if found, then
Add 3 to LOCCTR // 3 = Instruction length
else if OPCODE = ‘WORD’, then
Add 3 to LOCCTR
else if OPCODE = ‘RESW’, then
Add 3 * #[Operand] to LOCCTR
else if OPCODE = ‘RESB’, then
Add #[Operand] to LOCCTR
else if OPCODE = ‘BYTE’, then

Find length of constant in bytes
Add length to LOCCTR
else
Set error flag(invalid Operation Code)
end if
end if
Write line to Intermediate file
Read next input line
end while
Write last line to Intermediate file
Save (LOCCTR - Starting Address) as Program Length
End Pass 1

PROGRAM

```
 #include<stdio.h>
#include<stdlib.h>
#include<string.h>
char label[10],opcode[10],operand[10],code[10],mnemonic[3];
int length;
void pass1()
{
int locctr=0x0,str;
int start;
FILE *f1,*f2,*f3,*f4,*f5;
f1 = fopen("input.txt","r");
f2 = fopen("optab.txt","r");
f3 = fopen("intermediate.txt","w");
f4 = fopen("symtab.txt","w");
f5 = fopen("length.txt","w");
fscanf(f1,"%s\t%s\t%s",label,opcode,operand);
if(strcmp(opcode,"START")==0) {
start = strtol(operand,NULL,16);
locctr=(0x1)*start;
printf("%x\n",locctr);
fprintf(f3,"\t%s\t%s\t%s\n",label,opcode,operand);
fscanf(f1,"%s\t%s\t%s",label,opcode,operand);
}
else {
locctr=0x0;
}
while(strcmp(opcode,"END")!=0) {
fprintf(f3,"%x\t%s\t%s\t%s\n",locctr,label,opcode,operand);
if(strcmp(label,"**")!=0) {
fprintf(f4,"%s\t%x\n",label,locctr);
}
fscanf(f2,"%s\t%s",code,mnemonic);
while(strcmp(code,"END")!=0) {
if(strcmp(opcode,code)==0) {
locctr+=0x3;
break;
}
```

```
fscanf(f2,"%s\t%s",code,mnemonic);
}
if(strcmp(opcode,"WORD")==0) {
locctr+=0x3;
}
else if(strcmp(opcode,"RESW")==0) {
locctr+=((0x3)*atoi(operand));
}
else if(strcmp(opcode,"RESB")==0) {
locctr+=(0x1)*atoi(operand);
}
else if(strcmp(opcode,"BYTE")==0) {
locctr+=strlen(operand)*(0x1);
}
fscanf(f1,"%s\t%s\t%s",label,opcode,operand);
}
fprintf(f3,"%x\t%s\t%s\t%s",locctr,label,opcode,operand);
length = locctr-start;
fprintf(f5,"%x",length);
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);
}
void display()
{
char ch;
FILE *f1;
printf("---INPUT CODE---\n");
f1 = fopen("input.txt","r");
ch = fgetc(f1);
while(ch!=EOF)
{
printf("%c",ch);
ch = fgetc(f1);
}
fclose(f1);
printf("\n\n");
printf("---INTERMEDIATE FILE---\n");
f1 = fopen("intermediate.txt","r");
ch = fgetc(f1);
while(ch!=EOF)
{
printf("%c",ch);
ch = fgetc(f1);
}
fclose(f1);
printf("\n\n");
printf("---SYMTAB---\n");
f1 = fopen("symtab.txt","r");
ch = fgetc(f1);
while(ch!=EOF)
```

```
{
printf("%c",ch);
ch = fgetc(f1);
}
fclose(f1);
}
int main()
{
pass1();
display();
printf("\n");
return 0;
}
```

INPUT : Optab.txt

```
LDA 00
MUL 20
STA  0C
END *
```

OUTPUT

```
PS C:\Users\91812\Desktop\afs\S5\LABS> gcc pass1.c
PS C:\Users\91812\Desktop\afs\S5\LABS> ./a.exe
1000
---INPUT CODE---
PGM1    START   1000
**      LDA     ALPHA
**      MUL     BETA
**      STA     GAMMA
ALPHA   WORD    2
BETA    WORD    4
GAMMA   RESW    1
**      END     **

---INTERMEDIATE FILE---
        PGM1    START   1000
1000    **      LDA     ALPHA
1003    **      MUL     BETA
1006    **      STA     GAMMA
1009    ALPHA   WORD    2
100c    BETA    WORD    4
100f    GAMMA   RESW    1
1012    **      END     **

---SYMTAB---
ALPHA   1009
BETA    100c
GAMMA   100f
```

RESULT

PROGRAM EXECUTED SUCCESSFULLY.

EXPERIMENT NO: 05 **R HARIKRISHNAN**
DATE: 05-10-24 **ROLL NO:62**

PASS 2 OF TWO PASS ASSEMLER

AIM
To implement pass 2 of a two pass assembler

INPUT
Intermediate file (obtained from pass1), Symbol table, Program length

OUTPUT
Object program, Assembly listing

ALGORITHM
Begin
Read first input line(from intermediate

file)if OPCODE = ‘START’, then

Write Listing line

Read next input

lineend if

Write Header record to object

programInitialise first Text record

While OPCODE != ‘END’, do

if line is comment line,

thenRead next input line

continue

end if

Search OPTAB for

OPCODEif found then

if there is a symbol in OPERAND field

thenSearch SYMTAB for OPERAND

if found, then

Search symbol value as operand address

elseStore 0 as operand address

Set error flag (undefined symbol)

end if

else
Store 0 as operand addressend

if

Assemble object code instruction

else if OPCODE = ‘BYTE’ or ‘WORD’,

thenConvert constant to object code

else if OPCODE = ‘RESB’ or ‘RESW’,

then if current Text record is not empty,

thenWrite Text record to object program

end if
Write Listing line

Read next input line
Initialise new Text record

end if

if object code will not fit into the current Text record, then

Write Text record to object program

Initialise new Text record

end if

Add object code to Text record

Write Listing line

Read next input line

end while

Write last Text record to object program

Write End record to object program

Write last Listing line

End Pass 2

PROGRAM

```
#include<stdio.h>

#include<string.h.>

#include<ctype.h>
```

```
 void main()
{
FILE *f1,*f2,*f3,*f4,*f5;
int i,len,j=0;
int ln = 0x0;
char ch;
charl[10],address[10],label[10],opcode[10],op[10],operand[10],code[10],mne[10],sym[10],add[1
0],start[10];
f1 =fopen("intermediate.txt","r");f2 = fopen("length.txt","r");
f3 = fopen("symtab.txt","r");
f4 = fopen("optab.txt","r");
f5 = fopen("assenbly.txt","w");
fscanf(f1,"%s%s%s",label,opcode,operand);
fprintf(f5,"\t%s\t%s\t%s\n",label,opcode,operand);
if(strcmp(opcode,"START")==0)
{
strcpy(start,operand);
fscanf(f2,"%s",l);
}
printf("H%s^%s%s\nT00%s09^",label,start,l,start);fscanf(f1,"%s%s%s%s",address,label,op,operand);

while(strcmp(op,"END")!=0)
{
if(j==3){
printf("\nT00%s09^",address);i=0;
}
fscanf(f4,"%s%s",code,mne);
while(strcmp(code,"END")!=0)
{
if(strcmp(code,op)==0)
{
fclose(f4);
fscanf(f3,"%s%s",sym,add);
while(strcmp(sym,"END")!=0)
{
if(strcmp(sym,operand)==0)
```

```
{
printf("%s%s^",mne,add);
fprintf(f5,"%s\t%s\t%s\t%s\t%s%s\n",address,label,op,operand,mne,add);
break;
}
else
{
fscanf(f3,"%s%s",sym,add);
}
}
break;
}
else
{
fscanf(f4,"%s%s",code,mne);
}
}
if(strcmp(op,"BYTE")==0||strcmp(op,"WORD")==0)
{
if(strcmp(op,"WORD")==0)
{
printf("0000%s^",operand);
fprintf(f5,"%s\t%s\t%s\t%s\t0000%s\n",address,label,op,operand,operand);
}
else
{
len = strlen(operand);
for(i=2;i<len;i++)
{
printf("%d",operand[i]);
}
printf("^");
fprintf(f5,"%s\t%s\t%s\t%s\t0000%s\n",address,label,op,operand,operand);
}
}
if(strcmp(op,"RESW")==0||strcmp(op,"RESB")==0)  {
fprintf(f5,"%s\t%s\t%s\t%s\n",address,label,op,operand);
}
fscanf(f1,"%s%s%s%s",address,label,op,operand);
```

```
f4 = fopen("optab.txt","r");

fseek(f4,SEEK_SET,0);

j++;

}

printf("\n");
printf("E00%s\n",start);

fclose(f1);

fclose(f2);

fclose(f3);

fclose(f4);

fclose(f5);

printf("---ASSEMBLY LISTING---\n");
f1 = fopen("intermediate.txt","r");ch

= fgetc(f1);

while(ch!=EOF) {

printf("%c",ch); ch

= fgetc(f1);

}
fclose(f1);

printf("\n");

}
```

INPUT

```
---INTERMEDIATE FILE---
        PGM1    START   1000
1000    **      LDA     ALPHA
1003    **      MUL     BETA
1006    **      STA     GAMMA
1009    ALPHA   WORD    2
100c    BETA    WORD    4
100f    GAMMA   RESW    1
1012    GAMMA   RESW    3
101b    **      END     **

---SYMTAB---
ALPHA   1009
BETA    100c
GAMMA   100f
GAMMA   1012
```

OUTPUT

```
HP@HP ~/ss
$ gcc pass_two.c -o p2

HP@HP ~/ss
$ ./p2
H^PGM1^10001b
T^001000^09^001009^20100c^0C100f^
T^001009^09^00002^00004^
E^001000
---ASSEMBLY LISTING---
        PGM1    START   1000
1000    **      LDA     ALPHA
1003    **      MUL     BETA
1006    **      STA     GAMMA
1009    ALPHA   WORD    2
100c    BETA    WORD    4
100f    GAMMA   RESW    1
1012    GAMMA   RESW    3
101b    **      END     **
```

RESULT

PROGRAM EXECUTED SUCCESSFULL

EXPERIMENT NO: 06 **R HARIKRISHNAN**
DATE: 05-10-24 **ROLL NO:62**

SINGLE PASS ASSEMBLER

AIM

To write a C program to implement the single pass assembler

INPUT

Source file in Assembly Language

OUTPUT

Assembly Listing file, Object Program file

ALGORITHM

```
Begin
  Read input line
  if OPCODE = 'START', then
        Set Starting Address as #[Operand]
        Initialize LOCCTR to Starting Address
        Write line to Intermediate file
        Read next line
  else
        Initialize LOCCTR to 0 Set
        Starting Address to 0
  end if



  Write Header record to Object program file
  Initialise first Text record

  while OPCODE != 'END', do
        if line is a comment, then
              Read next input line
              continue
        end if

        if there is a symbol in the LABEL field, then
              Search SYMTAB for LABEL
              if found, then
                    Set error flag (duplicate symbol)
              else
                    Add symbol to SYMTAB with it's address
```

```
        end if
end if
Search OPTAB for OPCODEif
found, then
        Set LOCCTRincr to 3
else if OPCODE = 'WORD', thenSet
        LOCCTRincr to 3
else if OPCODE = 'RESW', then
        Set LOCCTRincr to 3*#[OPERAND]else
if OPCODE = 'REWB', then
        Set LOCCTRincr to #[OPERAND]else if
OPCODE = 'BYTE', then
        Find length of constant in bytesSet
        LOCCTRincr to length
else
        Set error flag(invalid operation code)
end if



if OPCODE = 'RESB' or 'RESW', then
          if current Text record is not empty, then
                  Write Text record to Object program file
        end if
        Write Listing line
        Read next input line
        Add LOCCTRincr to LOCCTR
        Initialise new Text record

else if OPCODE = 'RESB' or 'RESW', then
        Convert constant to object code
 else
        if there is a symbol in OPERAND field, thenSearch
                SYMTAB for OPERAND
                 if found, then
                          Store symbol value as operand address
                else
                        Store 0 as operand address
                        Set error flag(undefined symbol)
                end if
         else



                Store 0 as operand address
        end if


        Assemble object code instruction
end if
    if object code will not into the current Text record, thenWrite
    Text record to object
```

Initialise new Text record
end if

Add object code to Text recordWrite
line to Intermediate file Read next
input line
Add LOCCTRincr ro LOCCTRend
while

Write last Text record to Object program fileWrite
last Listing line
Write End record to Object program file
Write (LOCCTR] – Starting address) to Program length field inHeader
record in Object program file

End Assemble

PROGRAM

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

void main()
{
char opcode[10], operand[10], label[10], a[10], ad[10], symbol[10], ch;char
code[10][10], code1[10][10] = {"33", "44", "53", "57"};
char mnemonic[10][10] = {"START", "LDA", "STA", "LDCH", "STCH", "END"};
char mnemonic1[10][10] = {"LDA", "STA", "LDCH", "STCH"};
int locctr, start, length, i = 0, j = 0, k, l = 0;
int st, diff, address, add, len, actual_len, finaddr, prevaddr;
FILE *fp1, *fp2, *fp3, *fp4, *fp5, *fp6, *fp7;

fp1 = fopen("input1.txt", "r");
fp2 = fopen("symtab.txt", "w");
fp3 = fopen("intermediate1.txt", "w"); fscanf(fp1,
"%s%s%s", label, opcode, operand);if
(strcmp(opcode, "START") == 0)
{
        start = atoi(operand);
        locctr = start;
        fprintf(fp3, "%s\t%s\t%s\n", label, opcode, operand);
        fscanf(fp1, "%s%s%s", label, opcode, operand);
}
```

```
else
        locctr = 0;
while (strcmp(opcode, "END") != 0)
{
        fprintf(fp3, "%d",  locctr);
        if (strcmp(label, "**") != 0)
           fprintf(fp2, "%s\t%d\n", label, locctr);
        strcpy(code[i], mnemonic[j]);
        while (strcmp(mnemonic[j], "END") != 0)
        {
           if (strcmp(opcode, mnemonic[j]) == 0)
           {
              locctr += 3;
              break;
           }
           strcpy(code[i], mnemonic[j]);
           j++;
        }
        if (strcmp(opcode, "WORD") == 0)
           locctr += 3;
        else if (strcmp(opcode, "RESW") == 0)
           locctr += (3 * (atoi(operand)));
        else if (strcmp(opcode, "RESB") == 0)
           locctr += (atoi(operand));
        else if (strcmp(opcode, "BYTE") == 0)
           ++locctr;
        fprintf(fp3, "\t%s\t%s\t%s\n", label, opcode, operand);
        fscanf(fp1, "%s%s%s", label, opcode, operand);
}
fprintf(fp3, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);
length = locctr - start;

fclose(fp3);
fclose(fp2);
fclose(fp1);

printf("\n\nThe contents of Input file:\n\n");fp1
= fopen("input1.txt", "r");
ch = fgetc(fp1); while
(ch != EOF)
{
        printf("%c", ch);
        ch = fgetc(fp1);
}
```

```
printf("\n\nLength of the input program is %d.", length);
printf("\n\nThe contents of Symbol Table:\n\n");
fp2 = fopen("symtab.txt", "r");ch
= fgetc(fp2);
while (ch != EOF)
{
        printf("%c", ch);
        ch = fgetc(fp2);
}

fclose(fp2);
fclose(fp1);

fp4 = fopen("output.txt", "w");
fp5 = fopen("symtab.txt", "r");
fp6 = fopen("intermediate1.txt", "r");fp7
= fopen("objcode.txt", "w");
fscanf(fp6, "%s%s%s", label, opcode, operand);
while (strcmp(opcode, "END") != 0)
{
        prevaddr = address;
        fscanf(fp6, "%d%s%s%s", &address, label, opcode, operand);
}
finaddr = address;
fclose(fp6);
fp6 = fopen("intermediate1.txt", "r"); fscanf(fp6,
"%s%s%s", label, opcode, operand);if
(strcmp(opcode, "START") == 0)
{
        fprintf(fp4, "\t%s\t%s\t%s\n", label, opcode, operand);
        fprintf(fp7, "H^%s^00%s^00%d\n", label, operand, finaddr);
        fscanf(fp6, "%d%s%s%s", &address, label, opcode, operand);
        st = address;
        diff = prevaddr - st;
        fprintf(fp7, "T^00%d^%d", address, diff);
}
while (strcmp(opcode, "END") != 0)
{
        if (strcmp(opcode, "BYTE") == 0)
        {
           fprintf(fp4, "%d\t%s\t%s\t%s\t", address, label, opcode, operand);
           len = strlen(operand);
           actual_len = len - 3;
           fprintf(fp7, "^");
```

```
            for (k = 2; k < (actual_len + 2); k++)
            {
               itoa(operand[k], ad, 16);
               fprintf(fp4, "%s", ad);
               fprintf(fp7, "%s", ad);
            }
            fprintf(fp4, "\n");
         }
         else if (strcmp(opcode, "WORD") == 0)
         {
            len = strlen(operand);
            itoa(atoi(operand), a, 10);
            fprintf(fp4, "%d\t%s\t%s\t%s\t00000%s\n", address, label, opcode, operand, a);
            fprintf(fp7, "^00000%s", a);
         }
         else if ((strcmp(opcode, "RESB") == 0) || (strcmp(opcode, "RESW") == 0))
            fprintf(fp4, "%d\t%s\t%s\t%s\n", address, label, opcode, operand);
         else
         {
            while (strcmp(opcode, mnemonic1[l]) != 0)
               l++;
            if (strcmp(operand, "COPY") == 0)
               fprintf(fp4, "%d\t%s\t%s\t%s\t%s0000\n", address, label, opcode, operand,
   code1[l]);
            else
            {
               rewind(fp5);
               fscanf(fp5, "%s%d", symbol, &add);
               while (strcmp(operand, symbol) != 0)
                  fscanf(fp5, "%s%d", symbol, &add);
               fprintf(fp4, "%d\t%s\t%s\t%s\t%s%d\n", address, label, opcode, operand, code1[l],
   add);
               fprintf(fp7, "^%s%d", code1[l], add);
         }  }


         fscanf(fp6, "%d%s%s%s", &address, label, opcode, operand);
}
fprintf(fp4, "%d\t%s\t%s\t%s\n", address, label, opcode, operand);
fprintf(fp7, "\nE^00%d", st);
printf("\nObject Program has been generated.");

fclose(fp7);
fclose(fp6);
fclose(fp5);
```

```
fclose(fp4);

printf("\n\nObject Program:\n\n");fp7
= fopen("objcode.txt",  "r"); ch =
fgetc(fp7);
while (ch != EOF)
{
        printf("%c", ch);
        ch = fgetc(fp7);
}
fclose(fp7);
  }
```

INPUT

```
**          START     2000
**          LDA       FIVE
**          STA       ALPHA
**          LDCH      CHARZ
**          STCH      C1
ALPHA     RESW      2
FIVE      WORD      5
CHARZ     BYTE      C'Z'
C1          RESB      1
**          END       **
```

OUTPUT

```
The contents of Input file:

**      START   2000
**      LDA     FIVE
**      STA     ALPHA
**      LDCH    CHARZ
**      STCH    C1
ALPHA   RESW    2
FIVE    WORD    5
CHARZ   BYTE    C'Z'
C1      RESB    1
**      END     **
```

```
Length of the input program is 23.

The contents of Symbol Table:

ALPHA   2012
FIVE    2018
CHARZ   2021
C1      2022

Object Program has been generated.

Object Program:

H^**^002000^002023
T^002000^22^332018^442012^532021^572022^000005^5a
E^002000
```

RESULT

PROGRAM EXECUTED SUCCESSFULLY