Date:

EXPERIMENT NO-6

IMPLEMENTATION OF CPU SCHEDULING ALGORITHM

AIM

Programs to implement various CPU scheduling algorithms.

DESCRIPTION

CPU scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

TYPES OF CPU SCHEDULING

Two kinds of scheduling methods:

Preemptive Scheduling:

The lower priority task holds for sometime & resumes when the higher priority task finishes its execution.

Non-Preemptive Scheduling:

In this, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating.

CPU SHEDULING ALGORITHMS

1. FIRST COME FIRST SERVE (FCFS)

It is the easiest & most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be manged with FIFO queue.

2. SHORTEST JOB FIRST (SJF)

It is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

3. ROUND-ROBIN SCHEDULING

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.

4. PRIORITY BASED SCHEDULING

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority. Priority can be decided based on memory requirements, time requirements etc.

ALGORITHM

1. Algorithm of FCFS

Start

Step 1: Input the process along with their arrival time (at) & burst time (bt).

Step2: Find completion time (ct) & waiting time (wt) for all processes.

Step3: As first process that comes need not to wait so waiting time for process 1 will be 0. ie, wt[0]=0

Step4: Find waiting time for all other processes ie, for process i → wt[i] = tat[i]-bt[i];

Step5: Find turnaround time = ct[i]- at[i] for all processes

Step6: Find average waiting time = totWT/n

Step7: Find average turnaround time = totTAT/n

Stop

2. Algorithm of SJF

```
Algorithm SJF(){
    completed = 0          // Enter the processes along with their burst time & arrival time
    current_time = 0
    while (completed < n){
        find process with minimum burst time among process that are in ready queue at current_time
        if (process found){
            start_time = current_time
            ct[completed] = start_time + bt[completed]
            tat[completed] = ct[completed] - at[completed]
            wt[completed] = tat[completed] - bt[completed]
            mark process as completed
            completed++
```

```
            current_time = ct[completed]
        }
        else{
            current_time++
        }
    }
}
```

3. Algorithm of Round Robin

Start

Step1: Input the process with their arrival time (at) & burst time (bt).

Step2: Input the time quantum (time_quantum).

Step3: if (burst time < time_quantum)

Step 3.1: completion time[i] = completion time[i] + burst time[i]

Step 3.2: burst time[i] = 0

else

Step 3.3: burst time[i] = burst time[i] - time quantum

Step4: Find Turn around time for all processes

Step5: Find waiting time for all processes

Step6: Find average Turn around time = totTAT/n

Step7: Find average waiting time = totWT/n

Stop

4. Algorithm of priority

Start

Step 1: Input the process along with their arrival time (at) & burst time (bt)

Step 2: Sort the process according to the arrival time.

Step 2.1: If two process have same arrival time, then sort according to process priority.

Step 2.1.1: If two process have same priority, then sort according to process number.

Step 3: Assign the processes to CPU according to the priority, process with higher priority gets CPU first & then only lower priority process.

Step 4: As first process that comes need not to wait, so waiting time for process 1 will be 0 i.e, wt[0]=0

Step 5: Find waiting time for all processes.

Step 6: Find turn around time for all processes.

Step 7: Find average waiting time = avg_wt/n;

Step 8: Find average turn around time = avg_tat/n;

Stop

PROGRAMS

1. PROGRAM OF FCFS

```
# include <stdio.h>
void main()
{
    int i, j, temp, n, at[25], bt[25], ct[25], tat[25],
                                        cot[25], sum=0;
    float totWT=0, totTAT=0;
    printf("Enter the no: of processes:");
    scanf("%d", &n);

    for(i=0; i<n; i++)
    {
        printf("\n Enter arrival time of process %d:", i);
        scanf("%d", &at[i]);
        printf("\n Enter burst time of process %d:" i);
        scanf("%d", &bt[i]);
    }
     // CALCULATING COMPLETION TIME
    for(i=0; i<n; i++)
     {
         sum+=bt[i];
         ct[i]+=sum=;
     }
     // CALCULATING TURN AROUND TIME
     for(i=0; i<n; i++)
      {
          tat[i]=ct[i]-at[i];
          totTAT+=tat[i];
      }
```

```
// CALCULATING WAITING TIME
for (i=0; i<n; i++)
 {
    wt[i] = tat[i] - bt[i];
    totWT += wt[i];
 }
printf("\n TABLE");
printf("\n--------");
printf("\n\n P.NO\t AT\t BT\t CT\t TA\t WT");
for (i=0; i<n; i++)
 {
    printf("\n P%d\t %d\t %d\t %d\t %d\t %d",
                    i, at[i], bt[i], ct[i], tat[i],
                                    wt[i]);
 }
printf("\n Average Turnaround Time: %f", totTAT/n);
printf("\n Average waiting Time: %f\n", totWT/n);
```

OUTPUT

```
Enter the no: of processes: 4
Enter arrival time of process 0: 0
Enter burst time of process 0: 6
Enter arrival time of process 1: 1
Enter burst time of process 1: 8
Enter arrival time of process 2: 2
Enter burst time of process 2: 1
Enter arrival time of process 3: 3
Enter burst time of process 3: 3
```

COMPILE: gcc FCFS.c
RUN : ./a.out

TABLE

P.NO	AT	BT	CT	TA	WT
P_0	0	6	6	6	0
P_1	1	8	14	13	5
P_2	2	1	15	13	12
P_3	3	3	18	15	12

Average Turnaround Time = 11.750000
Average waiting time = 7.250000

2. PROGRAM OF SJF

```
#include<stdio.h>
int main()
{
   int at[100], bt[100], process[100], ct[100], tat[100], wt[100];
   int n, i, j, temp, current_time=0, start_time, completed=0,
                                                    count;
   float avg_tat=0.0, avg_wt=0.0;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("\n Enter the arrival time of process %d:", i);
      scanf("%d", &at[i]);
      printf("\n Enter the burst time of process %d:", i);
      scanf("%d", &bt[i]);
      process[i]= i;
      printf("\n");
   }
   for(i=0; i<n-1; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if (at[i]>at[j])
         {
            temp=at[i];
            at[i]=at[j];
            at[j] = temp;
            temp= bt[i];
            bt[i]= bt[j];
            bt[j]= temp;
            temp = process[i];
            process[i]= process[j];
```

```
            process[j]=temp;
        }
    }
}
printf("\n\nProcess\tArrival time\tBurst Time\tCompletion
                time\tTurn Around time\tWaiting
                                        Time\n\n");
while(completed<n)
{
    count=0;
    for(i=completed;i<n;i++)
    {
        if(at[i]<=current_time)
        {
            count++;
        }
        else
        {
            break;
        }
    }
    if(count>1)
    {
        for(i=completed;i<(completed+count-1);i++)
        {
            for(j=i+1;j<completed+count;j++)
            {
                if(bt[i]>bt[j])
                {
                    temp=at[i];
                    at[i]=at[j];
                    at[j]=temp;
                    temp=bt[i];
                    bt[i]=bt[j];
                    bt[j]=temp;
```

```
                temp = process[i];
                process[i] = process[j];
                process[j] = temp;
            }
        }
    }
}
start_time = current_time;
ct[completed] = start_time + bt[completed];
tat[completed] = ct[completed] - at[completed];
wt[completed] = tat[completed] - bt[completed];
current_time = ct[completed];
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d", process[comp
        leted], at[completed], bt[completed], ct[completed],
                                tat[completed], wt[completed]);
printf("\n");
avg_tat += tat[completed];
avg_wt += wt[completed];
completed++;
}
avg_tat = avg_tat/n;
avg_wt = avg_wt/n;
printf("Average Turn around time = %f \n", avg_tat);
printf("Average waiting time = %f \n", avg_wt);
return 0;
}
```

COMPILE: gcc SJF.c
RUN : ./a.out

OUTPUT

```
Enter the number of processes: 4
Enter the arrival time of process 0: 0
Enter the burst time of process 0: 6
Enter the arrival time of process 1: 1
Enter the burst time of proces 1: 8
Enter the arrival time of process 2: 2
Enter the burst time of process 2: 1
Enter the arrival time of process 3: 3
Enter the burst time of process 3: 3
```

Process	Arrival time	Burst time	Completion time	Turn around time	waiting time
0	0	6	6	6	0
2	2	1	7	5	4
3	3	3	10	7	4
1	1	8	18	17	9

Average Turn around time = 8.750000 Average Waiting Time = 4.250000

3. PROGRAM OF ROUND ROBIN

```
#include <stdio.h>
void main()
{
    int i, num, complete=0, x, count=0, time_quantum;
    int at[25], bt[25], temp[25];
    float totWT=0, totTAT=0;
    printf("\n Enter the no: of processes:");
    scanf("%d", &num);
    x= num;
    for(i=0; i<num; i++)
    {
      printf("\n Enter arrival time of process %d:", i);
      scanf("%d", &at[i]);
      printf("\n Enter burst time of process %d:" i);
      scanf("%d", &bt[i]);
      temp[i]= bt[i];
    }
    printf("\n Enter the time quantum");
    scanf("%d", &time_quantum);
    printf("\n\n P.NO\t AT\t BT\t CT\t TA\t WT");
    for(complete=0, i=0; x!=0; )
    {
      if(temp[i]<=time_quantum && temp[i]>0)
      {
        complete+=temp[i];
        temp[i]=0;
        count=1;
      }
      else if(temp[i]>0)
      {
        temp[i]= temp[i]- time_quantum;
        complete+= time_quantum;
      }
```

```
if (temp[i]==0 && count==1)
{
    x--;
    printf("\nP%d\t %d\t %d\t %d\t %d\t %d",
        i, at[i], bt[i], complete, complete-at[i],
                                    complete-at[i]-bt[i]);
    totTAT+= complete-at[i];
    totWT+= complete-at[i]-bt[i];
    count=0;
}
if (i==num-1)
{
  i=0;
}
else if (at[i+1]<=complete)
{
    i++;
}
else
{ i=0;
}
}
printf("\n Average Turnaround Time: %f", totTAT/num);
printf("\n Average Waiting Time: %f\n", totWT/num);
}
```

OUTPUT

```
Enter the no:of processes:4
Enter arrival time of process 0:0
Enter burst time of process 0:6
Enter arrival time of process 1:1
Enter burst time of process 1:8
Enter arrival time of process 2:2
Enter burst time of process 2:1
Enter arrival time of process 3:3
Enter burst time of process 3:3
Enter the time quantum:4
```

P.NO	AT	BT	CT	TA	WT
P2	2	1	9	7	6
P3	3	3	12	9	6
P0	0	6	14	14	8
P1	1	8	18	17	9

```
Average Turnaround Time: 11.750000
Average waiting Time: 7.250000
```

COMPILE: gcc ROUNDROBIN.c
RUN: ./a.out

4. PROGRAM OF PRIORITY

```
#include <stdio.h>
void main()
{
  int n, i, bt[20], at[20], j, sum=0, bt1[20], wt[20], tat[20],
                  t, p[20], pro[30], index, pr=-1, temp=-1;
  float avg_wt=0, avg_tat=0;
  printf("Enter number of processes: ");
  scanf("%d", &n);
  for(i=1; i<=n; i++)
  {
     printf("\n Arrival Time of P%d= ", i);
     scanf("%d", &at[i]);
     printf("\n Burst Time of P%d= ", i);
     scanf("%d", &bt[i]);
     printf("\n Priority of P%d=", i);
     scanf("%d", &p[i]);
     pro[i]= i;
     printf("\n");
  }
  for(i=1; i<=n; i++)
   {
     for(j=i+1; j<=n; j++)
      {
        if(at[i] > at[j])
         {
            temp= at[i];
            at[i]= at[j];
            at[j] = temp;
            temp= bt[i];
            bt[i]= bt[j];
            bt[j]= temp;
            temp= p[i];
            p[i]= p[j];
            p[j] = temp;
            temp= pro[i];
```

```
                pro[i] = pro[j];
                pro[j] = temp;
            }
        }
        sum=sum+bt[i];
        bt1[i]= bt[i];
        wt[i]=0;
        tat[i]=0;
        if (temp<p[i])
            temp=p[i];
    }
    t=0;
    while (t<sum)
    {
        pr= temp+1;
        for(i=1; i<=n; i++)
        {
            if(bt[i]>0 && at[i]<=t)
            {
                if (pr>p[i])
                {
                    pr=p[i];
                    index=i;
                }
            }
        }
        bt[index]=0;
        wt[index] = t-at[index];
        tat[index] = wt[index]+ bt1[index];
        t= t+bt1[index];
    }
    for(i=1; i<=n; i++)
    {
      avg_wt= avg_wt+wt[i];
      avg_tat= avg_tat+tat[i];
    }
```

```
printf("\n Process\t Arrival Time \t Burst Time \t Waiting
                                Time \t Turnaround Time \n");
for (i=1; i<=n; i++)
 {
    printf("P%d\t %d \t %d \t %d \t %d \n", pro[i],
                                at[i], bt1[i], wt[i], tat[i]);
 }
avg_wt = avg_wt/n;
avg_tat = avg_tat/n;
printf("\n Average waiting time = %.2f \n Average
                        turnaround time = %.2f",
                                avg_wt, avg_tat);
```

OUTPUT

Enter number of processes: 5

Arrival Time of P_1 = 0
Burst Time of P_1 = 9
Priority of P_1 = 5

Arrival Time of P_2 = 1
Burst Time of P_2 = 4
Priority of P_2 = 3

Arrival Time of P_3 = 2
Burst Time of P_3 = 5
Priority of P_3 = 1

Arrival Time of P_4 = 3
Burst Time of P_4 = 7
Priority of P_4 = 2

Arrival Time of P_5 = 4
Burst Time of P_5 = 3
Priority of P_5 = 4

Process	Arrival Time	Burst Time	Waiting Time	Turnaround Time
P_1	0	9	0	9
P_2	1	4	20	24
P_3	2	5	7	12
P_4	3	7	11	18
P_5	4	3	21	24

Average waiting time = 11.80
Average turnaround time = 17.40

RESULT

Programs executed successfully & outputs are obtained.