Student Name: Anubhav Verma

Student ID: 11702195

Email Address: anubhavverma990@gmail.com

Github Link: https://github.com/anu9901998

DESCRIPTION

In this the problem many processes come at same time and each process allocated CPU not more than 10 time unit. CPU execute process which have shortest job and if process burst is greater than 10 unit it will start executing next job. CPU must idle for 3 time unit after the arrival of the processes .it is required to find their average waiting time and average turnaround time. The scheduling algorithm which is required to be implemented is SJF (Shortest Job First) as well as RR (Round Robin).

LRTF SCHDEDULING ALGORITHM (solution)

- Create a structure Process containing all necessary fields like at (Arrival Time), bt (Burst Time), rt(Remaining Time), tat (Turn Around Time), wt (Waiting Time).
- Sort each process according to burst time;
- Find the process having shortest job and execute for each single unit. Increase the total time by 1 and reduce the Burst Time .
- Process having shortest job will execute first and if its burst time is greater than 10 unit CPU pre-empt to next shortest job.
- find tat and wt of each process.
- tat (Turn Around Time): Completion Time – Arrival Time
- wt (waiting time): Turn Around Time – Burst time.
- Finally, the average Turnaround time and average waiting time is calculated

SOLUTION CODE, COMPLEXITY AND DESCRIPTION

Step 1. Taking input(number of processes(n),arrival time(at) ,burst time(bt))

```
{
    int i,j,time=0,sum_bt=0,tq;
    char c;
    float avgwt=0;
    float avgtat=0;

     printf("\nEnter Number of Processes:\n");
     scanf("%d",&n);
     printf("Arrival time for all process must be same:\n");
     for(i=0,c='A';i<n;i++,c++)
     {
         p[i].name=c;
         printf("\n Process %c\n",c);
         printf("\n Arrival Time :");
         scanf("%d",&p[i].at);
         printf(" Burst Time :");
         scanf("%d",&p[i].bt);
         p[i].completed=0;
         sum_bt+=p[i].bt;
     }
```

Complexity: O(N)

Description: It initializes the Process structure by user input.

Step 2: Sorting

```
    for(i=0;i<n;i++)
    {
        for(j=i;j>=1;j--)
        {
            if(p[j].at==p[j-1].at)
            {                                   /*sorting according to
                                                 burst time so first sorted
                                                 job will execute*/
                if(p[j].bt<p[j-1].bt)
                {
                    temp=p[j-1];
                    p[j-1]=p[j];
                    p[j]=temp;
                }
            }
        }
    }
    for(i=0;i<n;i++)
```

Complexity :$O(N^2)$

Description: It sort the processes according to their burst time, it uses selection sort for the same.

Step 3: Enque and Deque the process in and from the queue respectively.

```
int n;
int q[10];  //queue
int front=-1,rear=-1;
void enqueue(int i)
{
    if(rear==10)
        printf("overflow");
    rear++;
    q[rear]=i;
    if(front==-1)
        front=0;

}

int dequeue()
{
    if(front==-1)
        printf("underflow");
    int temp=q[front];
    if(front==rear)
        front=rear=-1;
    else
        front++;
    return temp;
}
```

Complexity: O(1)

Description: enque the process in the queue and deque from the queue for scheduling.

Step 4:checking whether process is in queue or not

```
    return temp;
}
int isInQueue(int i)
{
    int k;
    for(k=front;k<=rear;k++)
    {
        if(q[k]==i)
        return 1;
    }
    return 0;
}
```

Complexity: O(N)

Description: checking process is in the queue or not if yes return 1 else return 0.

Step 5: Calculating Turn Around Time and Waiting Time

```
        time+=p[i].rt;
        p[i].rt=0;
        p[i].completed=1;
        printf(" %c ",p[i].name);
        p[i].wt=time-p[i].at-p[i].bt;
        p[i].tt=time-p[i].at;
        for(j=0;j<n;j++)     /*enqueue the processes which have come while scheduling */
        {
            if(p[j].at<=time && p[j].completed!=1&& isInQueue(j)!=1)
            {
                enqueue(j);
            }
        }
```

Complexity: O(N)

Description: It calculates the TAT and WT for each process.

Step 6: Displaying the value

```
    }
}

printf("\nName\tArrival Time\tBurst Time\tWaiting Time\tTurnAround Time");
for(i=0;i<n;i++)
{
    avgwt+=p[i].wt;
    avgtat+=p[i].tt;
    printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d",p[i].name,p[i].at,p[i].bt,p[i].wt+3,p[i].tt);
}
printf("\nAverage waiting time:%f\n",avgwt/n);
printf("\nAverage turnarround time:%f\n", avgtat/n);
```

Complexity: O(N)

Description: It displays the value of each process.

Overall complexity of solution: $O(N^2)$

CONSTRAINTS AND BOUNDARY CONDITIONS

Constrain 1: Processes must enter at same time (arrival time must be same) , this will sort according to shortest job first.

```
    }
        for(i=0;i<n;i++)
    {
        for(j=i;j>=1;j--)
        {
            if(p[j].at==p[j-1].at)
            {                                   /*sorting according to
                                                 burst time so first sorted
                                                 job will execute*/
                if(p[j].bt<p[j-1].bt)
                {
                    temp=p[j-1];
                    p[j-1]=p[j];
                    p[j]=temp;
                }
            }
        }
    }
    for(i=0;i<n;i++)
```

This is done before selecting the next student to execute.

Constrain 2: executing each process for a fixed time quanta(here t=10)

TEST CASES

TEST CASE NAME	INPUT	OUTPUT
TEST CASE #1	No of Processes 3 Process arrivaltime bursttime A 0 2 B 0 3 C 0 4	Average TAT: 5.333333 ms Waiting Average: 2.333333 ms Gantt chart A B C
TEST CASE #2	No of Processes 5 Process arrivaltime bursttime A 0 15 B 0 13 C 0 9 D 0 4 E 0 3	Average TAT: 21.799999 ms Waiting Average: 13.000000 ms Gantt chart E D C B A B A
TEST CASE #3	No of Processes 5 Process arrivaltime bursttime A 4 4 B 4 5 C 4 6 D 4 8 E 4 9	Average TAT:16.600000 ms Waiting Average :10.200000 ms Gantt chart A B C D E
TEST CASE #4	No of Processes 3 Process arrivaltime bursttime A 5 19 B 5 15 C 5 15	Average TAT: 41.333332 ms Waiting Average :25.000000 ms Gantt chart B C A B C A

GitHub Repository

No of commits: 13