Name: Talari Madhu

Reg no.: 11804587

Section: K18AW

Roll no.:49

Email id: [madhuthalari321@gmail.com](mailto:madhuthalari321@gmail.com)

**Question :10**

Write a C program to solve the following problem: Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO

Order is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders)that the disk arm moves to satisfy all the pending requests for each of the FCFS disk-scheduling algorithms?

**Description**

FCFS disk-scheduling algorithms requests in the order they arrive in the disk queue. In this algorithm all requests are serviced sequentially but generally, it does not provide the fastest service.

In operating systems, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down. Disk Scheduling Algorithms are used to reduce the total seek time of any request.

All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results

**Algorithm:**

1. 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. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

**Coding part**

#include<stdio.h>  
#include<conio.h>  
int main()  
{  
    int head,i,n1;  
    printf("\nEnter head position");  
    scanf("%d",&head);          //head points to current position  
    printf("\nEnter number of requests");  
    scanf("%d",&n1);  
    int req[n1];                //req is the request list  
    for(i=0;i<n1;i++)  
    {  
            scanf("%d",&req[i]);  
    }  
    int diff=req[0]-head;  
    if(diff<0)  
    {  
              diff=diff\*-1;  
    }  
    for(i=1;i<n1;i++)  
    {  
            if((req[i]-req[i-1])>0)  
               diff=diff+(req[i]-req[i-1]);  
            else  
               diff=diff+(req[i-1]-req[i]);  
    }  
    printf("Seek time = %d\n",diff);  
    getch();  
}

**Explanation:**

The head moments that occur while serving these requests are:

(125-86) +(1470-86) +(1470-913) +(1774-913) + (1774-948) +(1509-948) +(1509-1022) +(1750-1022) +(1750-130)

=>7063

Complexity: n\*n

**Question: 17**

1. Design a scheduling program to implements a Queue with two levels:

Level 1 : Fixed priority preemptive Scheduling

Level 2: Round Robin Scheduling

For a Fixed priority preemptive Scheduling (Queue1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2. All the processes in second level queue will complete their execution according to round robin scheduling.

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.

2. Priority of Queue 2 has lower priority than in Queue 1.

**Description:**

Fixed-priority preemptive scheduling is a scheduling system commonly used in real-time systems. With fixed priority preemptive scheduling, the scheduler ensures that at any given time, the processor executes the highest **priority** task of all those tasks that are currently ready to execute.

In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

Round robin is a pre-emptive algorithm

The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.

The process that is preempted is added to the end of the queue.

Coding part

#include<stdio.h>

#define N 10

typedef struct

{

      int pid, at,bt,pp;

      int q, ready;

} process\_structure;

int Queue(int t1)

{

      if(t1==0 || t1==1 || t1==2 || t1==3)

      {

            return 1;

      }

      else

      {

            return 2;

      }

}

int main()

{

      int limit, count, temp\_process, time, j, y;

      process\_structure temp;

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

      scanf("%d",&limit);

      process\_structure process[limit];

      for(count=0;count<limit;count++)

      {

            printf("\nProcess ID:\t");

            scanf("%d", &process[count].pid);

            printf("Arrival Time: ");

            scanf("%d", &process[count].at);

            printf("Burst Time: ");

            scanf("%d", &process[count].bt);

            printf("Process Priority: ");

            scanf("%d", &process[count].pp);

            temp\_process = process[count].pp;

            process[count].q = Queue(temp\_process);

            process[count].ready = 0;

      }

      time = process[0].bt;

      for(y=0;y<limit;y++)

      {

            for(count=y;count<limit;count++)

            {

                  if(process[count].at < time)

                  {

                        process[count].ready = 1;

                  }

            }

            for(count=y;count<limit-1;count++)

            {

                  for(j=count+1; j<limit; j++)

                  {

                        if(process[count].ready == 1 && process[j].ready == 1)

                        {

                              if(process[count].q == 2 && process[j].q == 1)

                              {

                                    temp = process[count];

                                    process[count] = process[j];

                                    process[j] = temp;

                              }

                        }

                  }

            }

            for(count=y;count<limit-1;count++)

            {

                  for(j=count+1;j<limit;j++)

                  {

                        if(process[count].ready==1 && process[j].ready==1)

                        {

                              if(process[count].q==1 && process[j].q==1)

                              {

                                    if(process[count].bt > process[j].bt)

                                    {

                                          temp=process[count];

                                          process[count]=process[j];

                                          process[j]=temp;

                                    }

                                    else

                                    {

                                          break;

                                    }

                              }

                        }

                  }

            }

            printf("\nProcess[%d]:\tTime:\t%d To %d\n", process[y].pid, time, time + process[y].bt);

            time = time + process[y].bt;

            for(count=y; count<limit; count++)

            {

                  if(process[count].ready==1)

                  {

                        process[count].ready=0;

                  }

            }

      }

      return 0;

}

**Explanation:**

Libraries used

<stdio>: this library is used for input and output.

<stdlib>: this library is used for string.

Fixed priority preemptive scheduling is a scheduling system used in real time systems.

**Complexity** : nlogn