|  |
| --- |
| **Subject: Operating System Sub Teacher:-Prof.S.S.Shete**  **Class :- S.E. Computer Engg Roll no :-**  **Practical No:- Date:-** |

**Title:-** Disk Scheduling Algorithms.

**Aim**:-This practical is prepared for Disk Scheduling Algorithms.

**INTRODUCTION:-**

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.

**PURPOSE:-**

The purpose of this material is to provide one with help on disk scheduling algorithms. Hopefully with this, one will be able to get a stronger grasp of what disk scheduling algorithms do.

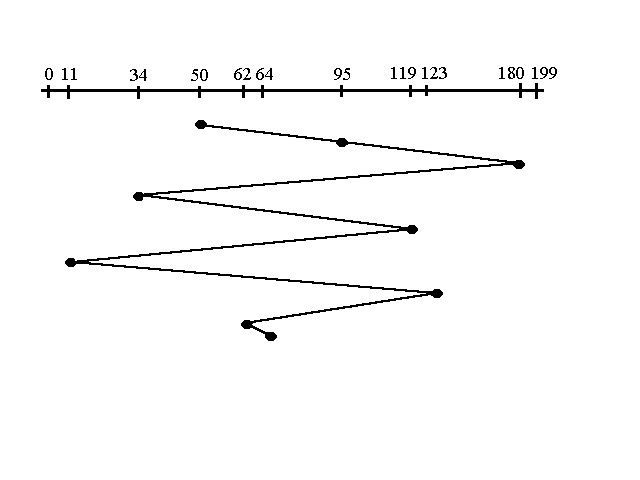
**TYPES OF DISK SCHEDULING ALGORITHMS:-**

Although there are other algorithms that reduce the seek time of all requests, I will only concentrate on the following disk scheduling algorithms:  
**1] First Come-First Serve (FCFS)  
2] Shortest Seek Time First (SSTF)  
3] Elevator (SCAN)   
4] Circular SCAN (C-SCAN)  
5] LOOK  
6] C-LOOK**These algorithms are not hard to understand, but they can confuse someone because they are so similar. What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be. I will show you and explain to you why C-LOOK is the best algorithm to use in trying to establish less seek time.

Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

1. **First Come -First Serve (FCFS)**

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. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.



**Program:-**

|  |
| --- |
| #include<conio.h>  #include<stdio.h>  int main()  {   int i,j,sum=0,n;   int ar[20],tm[20];   int disk;   clrscr();   printf("enter number of location\t");   scanf("%d",&n);   printf("enter position of head\t");   scanf("%d",&disk);   printf("enter elements of disk queue\n");   for(i=0;i<n;i++)   {   scanf("%d",&ar[i]);   tm[i]=disk-ar[i];   if(tm[i]<0)   {   tm[i]=ar[i]-disk;   }   disk=ar[i];   sum=sum+tm[i];   }   /\*for(i=0;i<n;i++)   {   printf("\n%d",tm[i]);   }   \*/   printf("\nmovement of total cylinders %d",sum);   getch();   return 0;  } |

**Output:-**

enter number of location   8

enter position of head   53

enter elements of disk queue

98

183

37

122

14

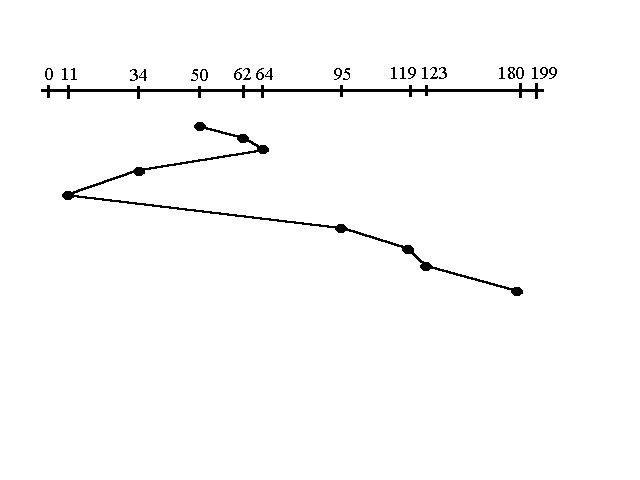
124

65

67

movement of total cylinders 640

1. **Shortest Seek Time First (SSTF)** In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to eachother the other requests will never be handled since the distance will always be greater.



**Program :-**

|  |
| --- |
| #include<conio.h>  #include<stdio.h>  struct di  {  int num;  int flag;  };  int main()  {   int i,j,sum=0,n,min,loc,x,y;   struct di d[20];   int disk;   int ar[20],a[20];   clrscr();   printf("enter number of location\t");   scanf("%d",&n);   printf("enter position of head\t");   scanf("%d",&disk);   printf("enter elements of disk queue\n");   for(i=0;i<n;i++)   {   scanf("%d",&d[i].num);   d[i]. flag=0;   }   for(i=0;i<n;i++)   {                 x=0; min=0;loc=0;    for(j=0;j<n;j++)    {     if(d[j].flag==0)     {     if(x==0)     {     ar[j]=disk-d[j].num;     if(ar[j]<0){ ar[j]=d[j].num-disk;}     min=ar[j];loc=j;x++; }     else     {     ar[j]=disk-d[j].num;     if(ar[j]<0){ ar[j]=d[j].num-disk;}      }      if(min>ar[j]){ min=ar[j]; loc=j;}     }    }       d[loc].flag=1;       a[i]=d[loc].num-disk;       if(a[i]<0){a[i]=disk-d[loc].num;}         disk=d[loc].num;   }       for(i=0;i<n;i++)   {   sum=sum+a[i];   }         printf("\nmovement of total cylinders %d",sum);   getch();   return 0;  } |

**Output:-**

enter number of location   8

enter position of head   53

enter elements of disk queue

98

183

37

122

14

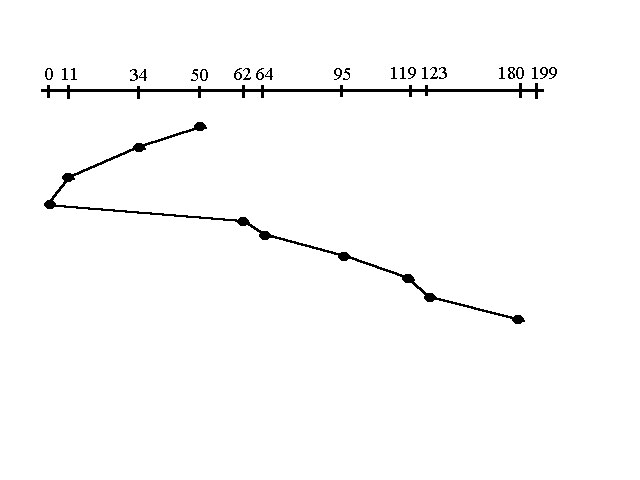
124

65

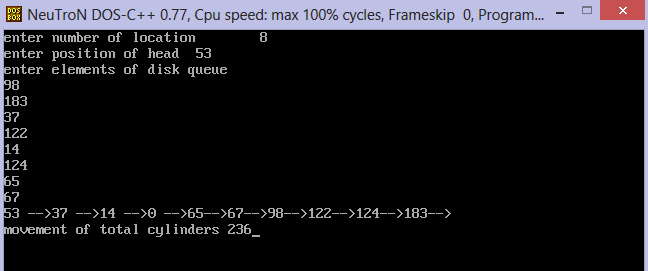
67

movement of total cylinders 236

**3.Elevator(SCAN):** :-  
This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.



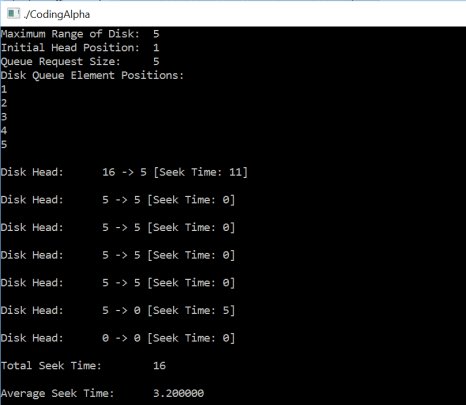
|  |
| --- |
| **Program:-**  #include<conio.h>  #include<stdio.h>  int main()  {   int i,j,sum=0,n;   int d[20];   int disk;   //loc of head   int temp,max;   int dloc;   //loc of disk in array   clrscr();   printf("enter number of location\t");   scanf("%d",&n);   printf("enter position of head\t");   scanf("%d",&disk);   printf("enter elements of disk queue\n");   for(i=0;i<n;i++)   {   scanf("%d",&d[i]);   }   d[n]=disk;   n=n+1;   for(i=0;i<n;i++)    // sorting disk locations   {    for(j=i;j<n;j++)    {      if(d[i]>d[j])      {      temp=d[i];      d[i]=d[j];      d[j]=temp;      }    }   }   max=d[n];   for(i=0;i<n;i++)   // to find loc of disc in array   {   if(disk==d[i]) { dloc=i; break;  }   }   for(i=dloc;i>=0;i--)   {   printf("%d -->",d[i]);   }   printf("0 -->");   for(i=dloc+1;i<n;i++)   {   printf("%d-->",d[i]);   }   sum=disk+max;         printf("\nmovement of total cylinders %d",sum);   getch();   return 0;  }  **Output:-** |



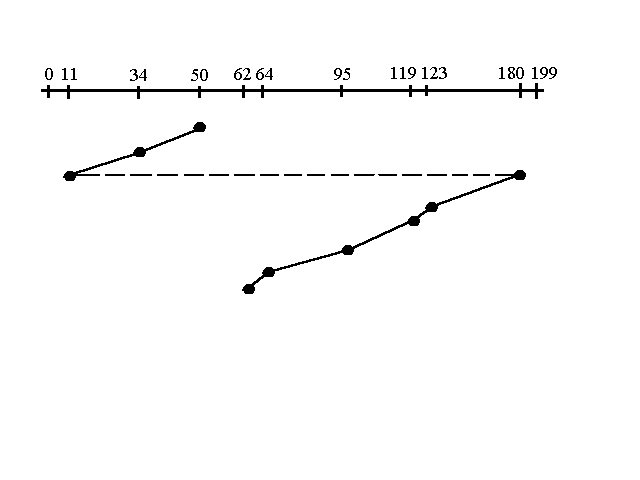
**4. Circular Scan (C-SCAN):-**  
 Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the mose sufficient.

|  |
| --- |
| **Program:-**  #include<stdio.h>  #include<stdlib.h>   int main()  {        int queue1[30], queue2[30], queue3[30];        int limit, disk\_head, count = 0, j, seek\_time = 0, range, diff;        int t1, t2 = 0, t3 = 0;        float avg\_seek\_time;        printf("Maximum Range of Disk:\t");        scanf("%d", &range);        printf("Initial Head Position:\t");        scanf("%d", &disk\_head);        printf("Queue Request Size:\t");        scanf("%d", &limit);        printf("Disk Queue Element Positions:\n");        while(count < limit)        {              scanf("%d", &t1);              if(t1 >= disk\_head)              {                    queue1[t2] = t1;                    t2++;              }              else              {                    queue2[t3] = t1;                    t3++;              }              count++;        }        count = 0;        while(count < t2 - 1)        {              j = count + 1;              while(j < t2)              {                    if(queue1[count] > queue1[j])                    {                          t1 = queue1[count];                          queue1[count] = queue1[j];                          queue1[j] = t1;                    }                    j++;              }              count++;        }        count = 0;        while(count < t3 - 1)        {              j = count + 1;              while(j < t3)              {                    if(queue2[count] > queue2[j])                    {                          t1 = queue2[count];                          queue2[count] = queue2[j];                          queue2[j] = t1;                    }                    j++;              }              count++;        }        count = 1;        j = 0;        while(j < t2)        {              queue3[count] = queue1[j];              queue3[count] = range;              queue3[count + 1] = 0;              count++;              j++;        }        count = t2 + 3;        j = 0;        while(j < t3)        {              queue3[count] = queue2[j];              queue3[0] = disk\_head;              count++;              j++;        }        for(j = 0; j <= limit + 1; j++)        {              diff = abs(queue3[j + 1] - queue3[j]);              seek\_time = seek\_time + diff;              printf("\nDisk Head:\t%d -> %d [Seek Time: %d]\n", queue3[j], queue3[j + 1], diff);        }        printf("\nTotal Seek Time:\t%d\n", seek\_time);        avg\_seek\_time = seek\_time / (float)limit;        printf("\nAverage Seek Time:\t%f\n", avg\_seek\_time);        return 0;  } |

**Output:-**



**5.C-LOOK** :-  
 This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.

From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.   
  


**Program:-**

#include<math.h>

#include<stdio.h>

int main()

{

int i,n,j=0,k=0,x=0,l,req[50],mov=0,cp,ub,end, lower[50],upper[50],

temp,a[50];

printf("enter the current position\n");

scanf("%d",&cp);

printf("enter the number of requests\n");

scanf("%d",&n);

printf("enter the request order\n");

for(i=0;i<n;i++)

{

scanf("%d",&req[i]);

}

printf("Enter the upper bound\n");

scanf("%d",&ub);

/\*break the request array into two arrays : one with requests lower than

current and other with requests higher than current position. Also sort

these two new arrays\*/

for(i=0;i<n;i++)

{

if(req[i]<cp)

{

lower[j]=req[i];

j++;

}

if(req[i]>cp)

{

upper[k]=req[i];

k++;

}

}

//sort the lower array in reverse order

for(i=0;i<j;i++)

{

for(l=0;l<j-1;l++)

{

if(lower[l]<lower[l+1])

{

temp=lower[l];

lower[l]=lower[l+1];

lower[l+1]=temp;

}

}

}

// sort the upper array in ascending order

for(i=0;i<=k;i++)

{

for(l=0;l<k-1;l++)

{

if(upper[l]>upper[l+1])

{

temp=upper[l];

upper[l]=upper[l+1];

upper[l+1]=temp;

}

}

}

printf("Enter the end to which the head is moving (0 - for lower end

(zero) and 1 - for upper end\n");

scanf("%d",&end);

switch(end)

{

case 0:

for(i=0;i<j;i++)

{

a[x]=lower[i];

x++;

}

for(i=0;i<k;i++)

{

a[x]=upper[i];

x++;

}

break;

case 1:

for(i=0;i<k;i++)

{

a[x]=upper[i];

x++;

}

for(i=0;i<j;i++)

{

a[x]=lower[i];

x++;

}

break;

}

mov=mov+abs(cp-a[0]);

printf("%d -> %d",cp,a[0]);

for(i=1;i<x;i++)

{

mov=mov+abs(a[i]-a[i-1]);

printf(" -> %d",a[i]);

}

printf("\n");

printf("total head movement = %d\n",mov);

}

**Output:-**

