Effective Disk Management

Report 2

Prepared by

Arpit Jain | 2015047
Gautam Yadav | 2015093

Under guidance of

Dr. M.K. Bajpai
Assistant Professor
IIITDM Jabalpur

29/09/2017

Implementation Phase

A New Optimized Real-Time Scheduling Algorithm

Paper Authors

Nidhi

Madan Mohan Malaviya University of Technology, Gorakhpur (UP)-273010, India

Dayashankar Singh

Madan Mohan Malaviya University of Technology, Gorakhpur (UP)-273010, India

Abstract

In this proposed algorithm, initially the disk head is at the disk start position and has the direction towards the final disk position. It means initial head position and direction of head is always same. First we sort all the cylinders input blocks by using any sorting algorithm. Initially the head is at position 0 and sequentially moves and reached from this block to the highest input block number, servicing all the input request blocks in front of the head immediately.

Algorithm

1. Assume a[] is an array containing track numbers and x is the position of last input block.
2. Inialize Head position is at 0. Assume h denotes the current head position.
3. Sort input blocks of cylinder number in ascending order with the help of any sorting algorithm.
4. Initially head position h is 0.
5. for(i=0; i<=x; i++)
6. Service the input request in front of head immediately.
7. Total_head_movements = x;
8. Return total_head_movements;

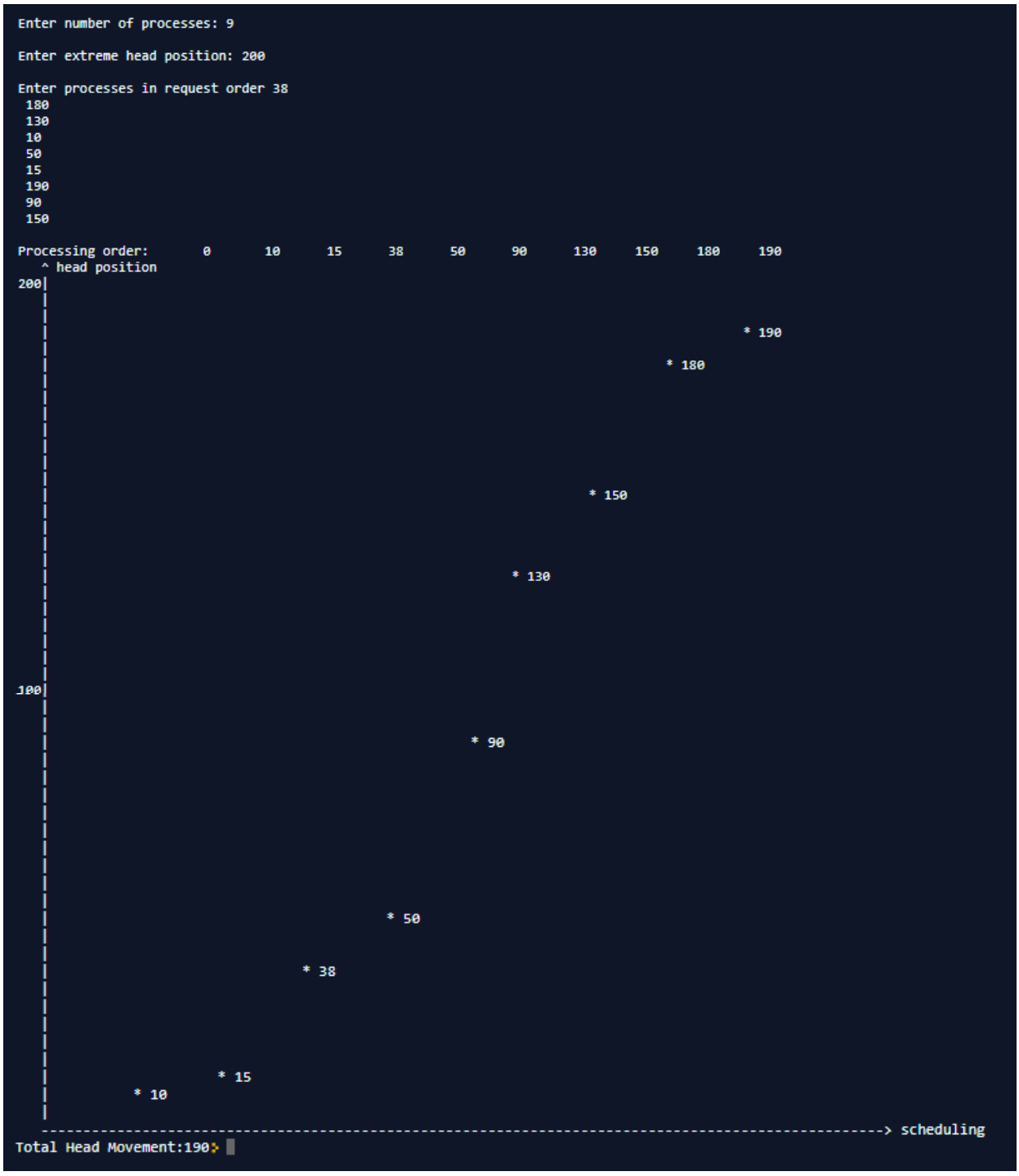
Simulation Code

Language Used : C

```c
#include<stdio.h>

// A function to implement swapping of two variables

void swap(int *xp, int *yp)
{
  int temp = *xp;
  *xp = *yp;
  *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)

{
  //i and j are dummy variables

  int i, j;
  for (i = 0; i < n-1; i++)
    // Last i elements are already in place
      for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
          swap(&arr[j], &arr[j+1]);
}

int main()
{
  //n denotes total number of processes
  //h denotes total head Movement
  //i,k,j denote dummy variables
  //x denotes extreme head position

  int n,i,h=0,x,k,j;

  printf("\nEnter number of processes:");
  scanf("%d",&n);
  printf("\nEnter extreme head position:");
  scanf("%d",&x);

  int a[n+1];
  a[0] = 0;

  //taking input of processes track location from user

  printf("\nEnter processes in request order");
  for(i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
    }

  //sorting and calculating total head movements
```

```
  bubbleSort(a, n+1);

  for(i=0;i<n;i++)
    h+=(a[i+1]-a[i]);

  //output of the order in which Processing is done

  printf("\nProcessing order:");
  for(i=0;i<=n;i++)
    printf("\t%d",a[i]);

  //display the graph

  printf("\n   ^ head position\n");
  for(i=x/4;i>0;i--)
  {
    if(i==x/4)
      printf("%d|",x);
    else if(i==x/8)
      printf("%d|",x/2);
    else
      printf("   |");
    for(j=0;j<=n;j++)
    {
      if(a[j]/4==i)
      {
        for(k=0;k<j*10;k++)
        {
          printf(" ");
        }
        printf("* %d",a[j]);
      }
    }
    printf("\n");
  }
  printf("   ");
  for(i=0;i<x/2;i++)
  {
    printf("-");
  }
  printf("> scheduling");

  printf("\nTotal Head Movement:%d",h);

  return 0;
}
```

Output


Enter number of processes: 9
Enter extreme head position: 200
Enter processes in request order 38
180
130
10
50
15
190
90
150
Processing order: 0 10 15 38 50 90 130 150 180 190
^ head position
200
100
* 190
* 180
* 150
* 130
* 90
* 50
* 38
* 15
* 10
scheduling
Total Head Movement:190

Conclusion

In this paper, a new real-time optimized disk scheduling has been implemented which imposes almost no performance penalty over the non-real time optimal schedulers, when have sufficient slack time. With the help of our simulation and comparison of this proposed algorithm with existing algorithms, it is clear that the proposed algorithm reduces the total head movements. In this algorithm, sometimes the number of head movements is equal to SSTF or LOOK scheduling but it occurs very rarely. Worst case occurs when all the input blocks are concentrated near the extreme position or at the extreme position.

In this paper a lot of efforts have been done to improve the performance of disk I/O access, even there are tremendous scope for the improvement of disk I/O access.

References

- http://research.ijcaonline.org/volume93/number18/pxc3896046.pdf