College Of Engineering Trivandrum

System Software Lab

Mohammed Rabeeh
S5 CSE Roll No:35

TVE18CS036

Department of Computer Science

September 9, 2020

Contents

CS331 - System Software · 2020 ·

Cycle 1

Exp No 2

1 Aim

Simulate the following disk scheduling algorithms.

(a) FCFS

(b) SCAN

(c) C-SCAN

2 Algorithms

2.1 FCFS Scheduling

1. Let 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.

2.2 SCAN Scheduling

1. Let 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 direction represents whether the head is moving towards left or right. In the direction in which head is moving service all tracks one by one.
3. Calculate the absolute distance of the track from the head.
4. Increment the total seek count with this distance.
5. Currently serviced track position now becomes the new head position.
6. Go to step 3 until we reach at one of the ends of the disk.
7. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

2.3 C-SCAN Scheduling

1. Let 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. The head services only in the right direction from 0 to size of the disk.
3. While moving in the left direction do not service any of the tracks.
4. When we reach at the beginning(left end) reverse the direction.https://www.overleaf.com/project/5f58cf4a7480a900019f

5. While moving in right direction it services all tracks one by one.
6. While moving in right direction calculate the absolute distance of the track from the head.
7. Increment the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach at right end of the disk.
10. If we reach at the right end of the disk reverse the direction and go to step 3 until all tracks in request array have not been serviced.

3 Program Code

3.1 FCFS Disk Scheduling

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int diskQueue[20], n, i, seekTime = 0, diff;
  printf("Enter the size of Queue: ");
  scanf("%d", &n);
  printf("Enter the Queue: ");
  for (i = 1; i <= n; i++)
  {
    scanf("%d", &diskQueue[i]);
  }
  printf("Enter the initial head position: ");
  scanf("%d", &diskQueue[0]);
  for (i = 0; i < n; i++)
  {
    diff = abs(diskQueue[i + 1] - diskQueue[i]);
    seekTime += diff;
    printf("\n%d -> %d - Time: %d", diskQueue[i], diskQueue[i + 1], diff);
  }
  printf("\n\nTotal Seek Time: %d", seekTime);
  printf("\nAverage Seek Time = %f", (float)seekTime / n);
  printf("\n");
  return 0;
}
```

3.2 SCAN Disk Scheduling

```
#include <stdio.h>
#include <stdlib.h>

void scan(int Ar[20], int n, int start);
void bubble_sort(int Ar[20], int n);

int main()
{
  int diskQueue[20], n, start, i;
  printf("Enter the size of Queue: ");
  scanf("%d", &n);
  printf("Enter the Queue: ");
  for (i = 1; i <= n; i++)
  {
    scanf("%d", &diskQueue[i]);
  }
```

```
  printf("Enter the initial head position: ");
  scanf("%d", &start);
  diskQueue[0] = start;
  ++n;
  bubble_sort(diskQueue, n);
  scan(diskQueue, n, start);

  return 0;
}

void scan(int Ar[20], int n, int start)
{
  int i, pos, diff, seekTime = 0, current;
  for (i = 0; i < n; i++)
  {
    if (Ar[i] == start)
    {
      pos = i;
      break;
    }
  }
  for (i = pos; i < n - 1; i++)
  {
    diff = abs(Ar[i + 1] - Ar[i]);
    seekTime += diff;
    printf("\n%d -> %d - Time: %d", Ar[i], Ar[i + 1], diff);
  }
  current = i;
  for (i = pos - 1; i >= 0; i--)
  {
    diff = abs(Ar[current] - Ar[i]);
    seekTime += diff;
    printf("\n%d -> %d - Time: %d", Ar[current], Ar[i], diff);
    current = i;
  }
  printf("\n\nTotal Seek Time: %d", seekTime);
  printf("\nAverage Seek Time = %f", (float)seekTime / (n - 1));
}

void bubble_sort(int Ar[20], int n)
{
  int i, j, tmp;
  for (i = 0; i < n - 1; i++)
  {
    for (j = 0; j < n - 1 - i; j++)
    {
      if (Ar[j] > Ar[j + 1])
      {
        tmp = Ar[j];
        Ar[j] = Ar[j + 1];
        Ar[j + 1] = tmp;
      }
    }
  }
}
```

3.3 C-SCAN Disk Scheduling

```
#include <stdio.h>
#include <stdlib.h>
```

```
void cscan(int Ar[20], int n, int start);
void sort(int Ar[20], int n);

int main()
{
  int diskQueue[20], n, start, i, max;
  printf("Enter the size of Queue: ");
  scanf("%d", &n);
  printf("Enter the Queue: ");
  for (i = 1; i <= n; i++)
  {
    scanf("%d", &diskQueue[i]);
  }
  printf("Enter the initial head position: ");
  scanf("%d", &start);
  diskQueue[0] = start;
  ++n;
  sort(diskQueue, n);
  cscan(diskQueue, n, start);

  return 0;
}

void cscan(int Ar[20], int n, int start)
{
  int i, pos, diff, seekTime = 0, current;
  for (i = 0; i < n; i++)
  {
    if (Ar[i] == start)
    {
      pos = i;
      break;
    }
  }

  for (i = pos; i < n - 1; i++)
  {
    diff = abs(Ar[i + 1] - Ar[i]);
    seekTime += diff;
    printf("\n%d -> %d - Time: %d", Ar[i], Ar[i + 1], diff);
  }
  current = 0;

  printf("\n%d -> %d - Time: %d", Ar[i], current, 0);

  for (i = 0; i < pos; i++)
  {
    diff = abs(Ar[i] - current);
    seekTime += diff;
    printf("\n%d -> %d - Time: %d", current, Ar[i], diff);
    current = Ar[i];
  }
  printf("\n\nTotal Seek Time: %d", seekTime);

  printf("\nAverage Seek Time = %f\n", (float)seekTime / (n - 1));
}

void sort(int Ar[20], int n)
{
  int i, j, tmp;
  for (i = 0; i < n - 1; i++)
```

```
  {
    for (j = 0; j < n - 1 - i; j++)
    {
      if (Ar[j] > Ar[j + 1])
      {
        tmp = Ar[j];
        Ar[j] = Ar[j + 1];
        Ar[j + 1] = tmp;
      }
    }
  }
}
```

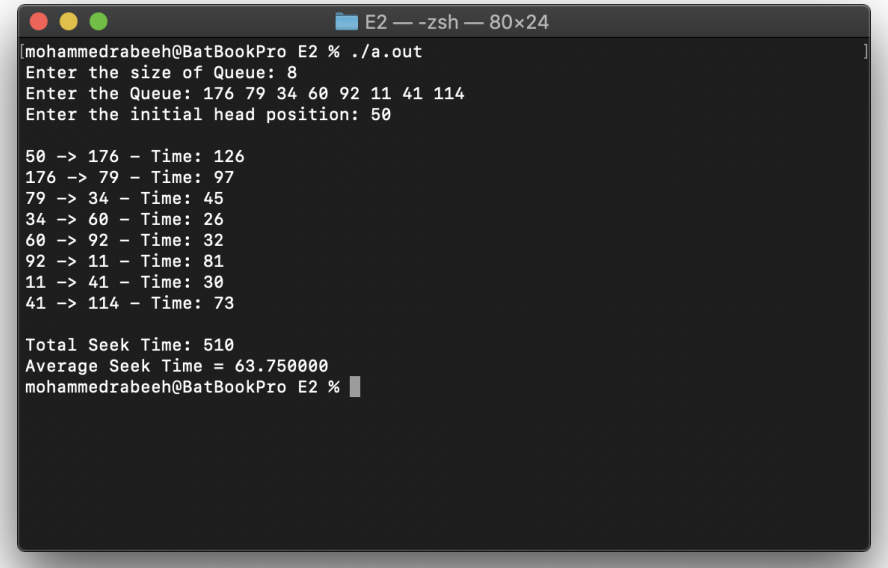
4 Program Output

Input

```
Queue = 176 79 34 60 92 11 41 114
Initial Start Position = 50
```

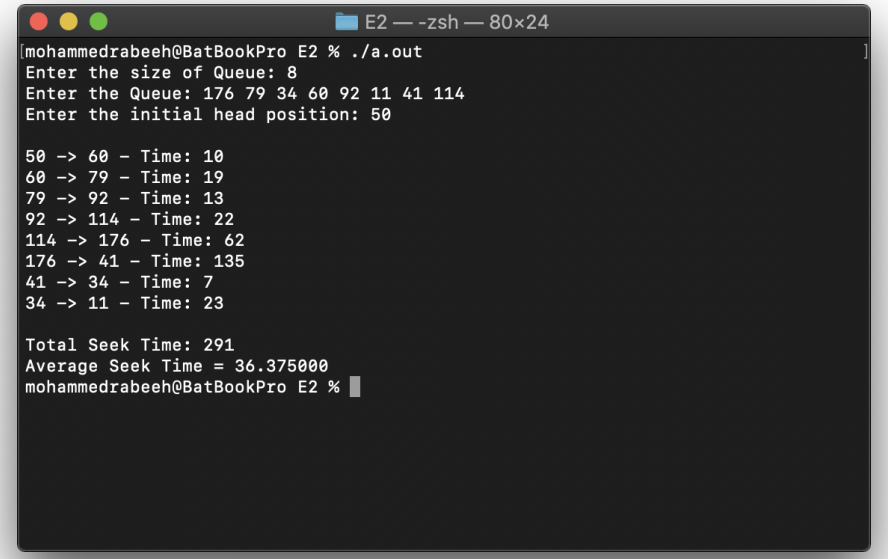
4.1 FCFS Disk Scheduling

Output



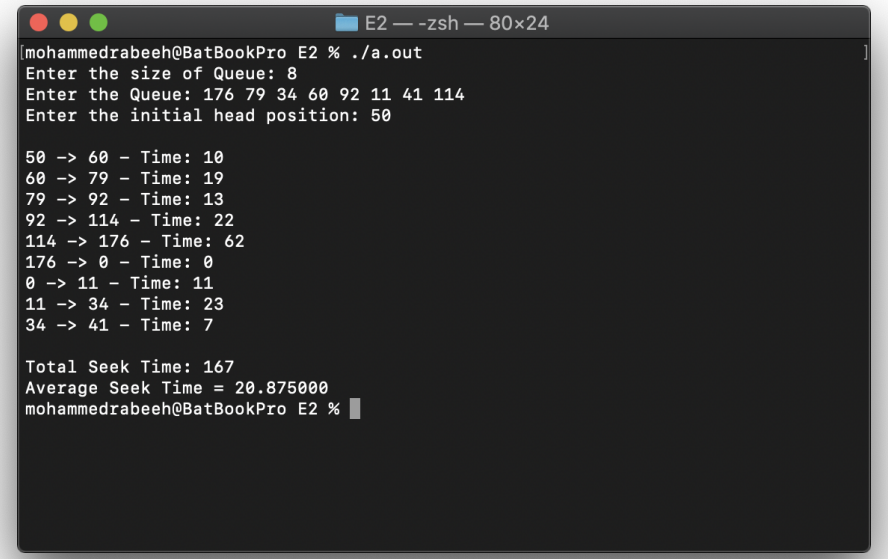
4.2 SCAN Disk Scheduling

Output



4.3 C-SCAN Scheduling

Output



5 Result

The above mentioned disk scheduling algorithms, ie, FCFS, SCAN, and C-SCAN were implemented in C Language and their Output verified. The program was executed on macOS Catalina 10.15.3 operating system using ZSH Shell.