Disk Scheduling Algorithms

Sanjivani, Priyamvadha, Heena, Naveen

January 7, 2014

1 Project Team Size and Members

- Sanjivani P (MT2013101) sanjivani.patil@iiitb.org
- Priyamvadha K (MT2013115) priyamvadha.115@iiitb.org
- Heena Sharma (MT2013058) heena.sharma@iiitb.org
- Naveen N (MT2013084) naveen.nallu@iiitb.org

2 Problem Background

One of the responsiblity of the OS is to use the hardware efficiently. For disk drive, this means having a fast access time and disk bandwidth

3 Problem Statement

If the desired disk drive and controller are available, request can be served immediately otherwise the new requests have to be placed on the queue of pending requests for that drive. For a multiprogramming system with many processes , the disk queue may have several pending requests. Thus the operating system has an opportunity to choose which pending request to serve next.

4 Purpose Statement(Goals)

To help OS to select the order in which track numbers have to be served, we have implemented the following Disk Scheduling Algorithms :

- FCFS(First Come First Served)
- SSTF(Shortest Seek Time First)
- SCAN
- C-SCAN
- LOOK
- C-LOOK

The goal is to choose the most efficient one from the above algorithms for a given instance and implementation using GUI.

5 Methods and approach

1. Study and analyse the examples
2. Design the algorithms
3. Start with the implementation of the algorithms in java
4. Build test cases for basic algorithms using Junit
5. Implementation also chooses the best algorithm for the given input.
6. A Consolidated graph is also implemented where the user can compare the algorithms
7. We have also facilitated the tabular view of the analysis for easier user understanding
8. The Graphical User Interface(GUI) makes it easier for the user to perform the tasks like comparing the Graphs and analysing the Algorithms

6 Sucess criteria

Based on the total head movement of each algorithm we decide the best algorithm for the given input. The output will be displayed in the form of Graphs and Tables.

7 Risk

Completion of all the implementation and testing units within the given time may be a risk. Adapting to new technologies and tools within given time.

8 Required consultation

Technical help may be required from professor and other teams using similar technology.

9 Resources

People : 4 Time : 3 months Softwares used : Java,Junit.

10 Project duration

3 months

11 Objectives/Deliverables

11.1 Requirement Gathering

We went through many examples of Disk Scheduling and reference books to understand the disk scheduling algorithms.

11.2 Assumptions and Constraints

We have assumed that the input track numbers will not change dynamically

11.3 Formal Implementation Plan

FCFS

1. Initialise headMov=0
2. Read headPos as input
3. Initialise prev=headPos
4. //Use Queue data structure for storing the requested track numbers
5. Read track numbers as input and enqueue them in the trackNo Queue
6. Once the complete input is read:
7. while trackNo queue is not empty do:
8. tNo=dequeue(trackNo)
9. headMov+=abs(prev-tNo)
10. prev=tNo
11. plot(tNo) //Plot the track No. on the FCFS graph

SSTF

1. Initialise headMov=0
2. Read headPos as input
3. Initialise prev=headPos
4. Read track numbers as input and store them in the linked list trackNo
5. Once the complete input is read:
6. while trackNo list is not empty do:
7. /* findMinDiff function returns the element in the list whose difference with prev is minimum and also deletes that element from the list */
8. tNo=findMinDiff(prev,trackNo)
9. headMov+=abs(prev-tNo)

10. prev=tNo
11. plot(tNo) //Plot the track No. on the SSTF graph

C-LOOK

1. Initialise headMov=0 //headMov is global variable
2. Create a global Stack stk
3. Read headPos as input
4. Initialise prev to headPos //Global
5. Create a BST trackNo and initialise its root-¿data to headPos
6. Read track numbers as input and keep inserting them in trackNo
7. Once the complete input is read:
8. User has the choice to move the head in any direction
9. Call inorderT(root-¿left) //inorderT function described below
10. Call plotAndCompute()
11. Call inorderT(root-¿right)
12. Call plotAndCompute()

inorderT(BST root)

1. if root is not null
2. inorderT(root-¿left)
3. stk.push(root-¿data)
4. inorderT(root-¿right)

plotAndCompute()

1. while the Stack stk is not empty do
2. tNo=stk.pop()
3. headMov+=abs(prev-tNo)
4. prev=tNo
5. plot(tNo) //Plot the track No. on the C-LOOK graph

C-SCAN

1. Initialise headMov=0 //headMov is global variable
2. Create a global Stack stk
3. Read headPos as input

4. Initialise prev to headPos //Global
5. Create a BST trackNo and initialise its root-¿data to headPos
6. Insert 0 in the trackNo
7. Read track numbers as input and keep inserting them in trackNo
8. Once the complete input is read:
9. if direction is left
10. Call inorderT(root-¿left)//inorderT function described below
11. Call inorderT(root-¿right)
12. Call plotAndCompute()
13. else
14. Call inorderT(root-¿right)
15. Call inorderT(root-¿left)
16. Call plotAndCompute()

inorderT(BST root)

1. if root is not null
2. inorderT(root-¿left)
3. stk.push(root-¿data)
4. inorderT(root-¿right)

plotAndCompute()

1. while the Stack stk is not empty do
2. tNo=stk.pop()
3. headMov+=abs(prev-tNo)
4. prev=tNo
5. plot(tNo) //Plot the track No. on the C-SCAN graph

SCAN

1. Initialise headMov=0 //headMov is global variable
2. Create a global Stack stk
3. Read headPos as input
4. Initialise prev to headPos //Global
5. Create a BST trackNo and initialise its root-¿data to headPos

6. Insert 0 in the trackNo
7. Read track numbers as input and keep inserting them in trackNo
8. Once the complete input is read:
9. Call inorderT(root-¿left) //inorderT function described below
10. while the Stack stk is not empty do
11. Call plotAndCompute(stk.pop())
12. Call inorderTRight(root-¿right)

inorderT(BST root)

1. if root is not null
2. inorderT(root-¿left)
3. stk.push(root-¿data)
4. inorderT(root-¿right)

inorderTRight(BST root)

1. if root is not null
2. inorderT(root-¿left)
3. plotAndCompute(root-¿data)
4. inorderT(root-¿right)

plotAndCompute(tNo)

1. headMov+=abs(prev-tNo)
2. prev=tNo
3. plot(tNo) //Plot the track No. on the SCAN graph

LOOK

1. Initialise headMov=0 //headMov is global variable
2. Create a global Stack stk
3. Read headPos as input
4. Initialise prev to headPos //Global
5. Create a BST trackNo and initialise its root-¿data to headPos
6. Read track numbers as input and keep inserting them in trackNo
7. Once the complete input is read:
8. Call inorderT(root-¿left) //inorderT function described below

9. while the Stack stk is not empty do
10. Call plotAndCompute(stk.pop())
11. Call inorderTRight(root-¿right)

inorderT(BST root)

1. if root is not null
2. inorderT(root-¿left)
3. stk.push(root-¿data)
4. inorderT(root-¿right)

inorderTRight(BST root)

1. if root is not null
2. inorderT(root-¿left)
3. plotAndCompute(root-¿data)
4. inorderT(root-¿right)

plotAndCompute(tNo)

1. headMov+=abs(prev-tNo)
2. prev=tNo
3. plot(tNo) //Plot the track No. on the SCAN graph

11.4 Hand Offs and Recommendations to other teams

This project can be extended by implenting N-Scan and F-scan.

12 Testing Strategy

We have used Junit for testing. Input is track Numbers, Initial head position, Max Head Position and Direction. Output being tested is total head movement

12.1 Input

- Track Nos: 23, 89, 132, 42, 187
- Start Head Pos: 100
- Max Head Position: 199
- Direction: Up

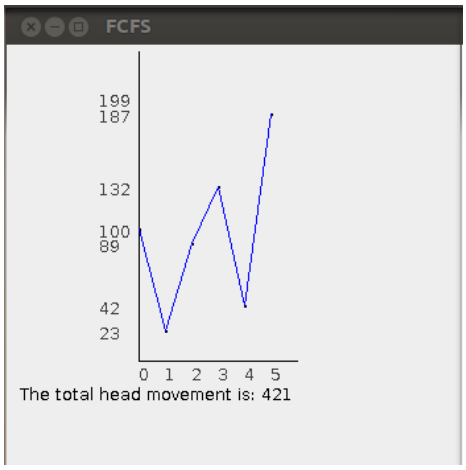

Figure 1: FCFS

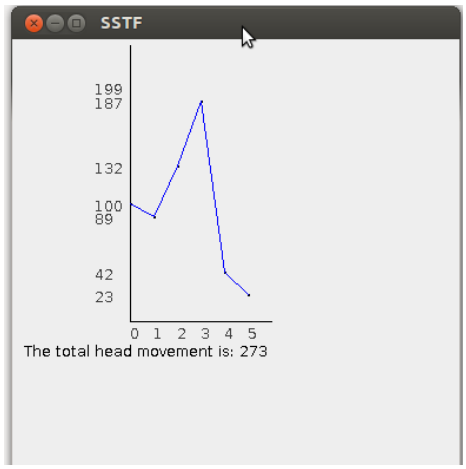

Figure 2: SSTF

12.2 Output

The total head movement is:

1. FCFS: 421
2. SSTF: 273
3. SCAN: 275
4. CSCAN: 341
5. LOOK: 251
6. CLOOK: 317

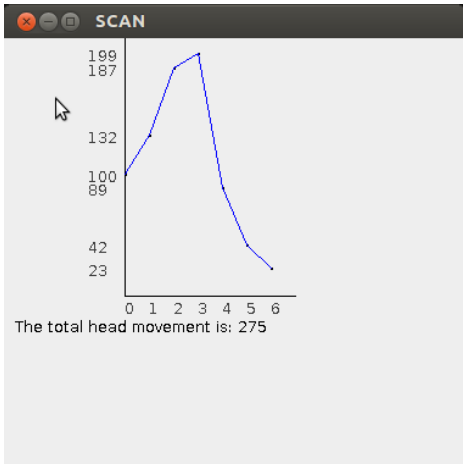

Figure 3: SCAN

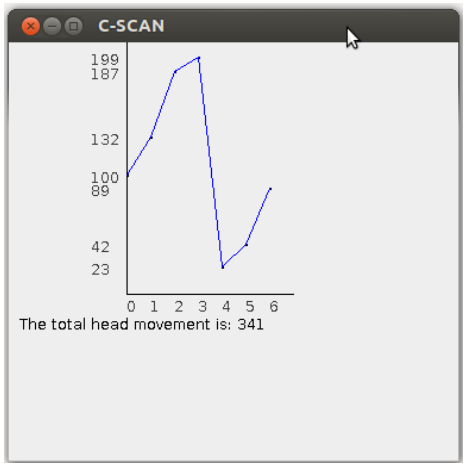

Figure 4: C-SCAN

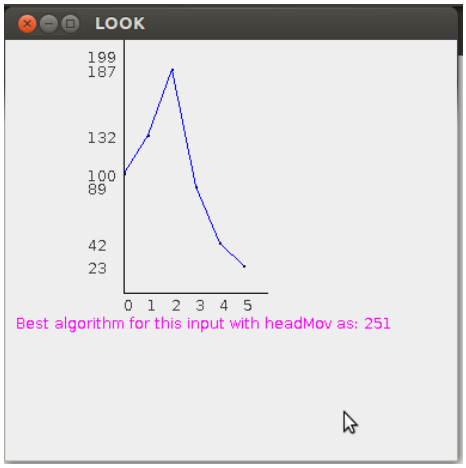

Figure 5: LOOK

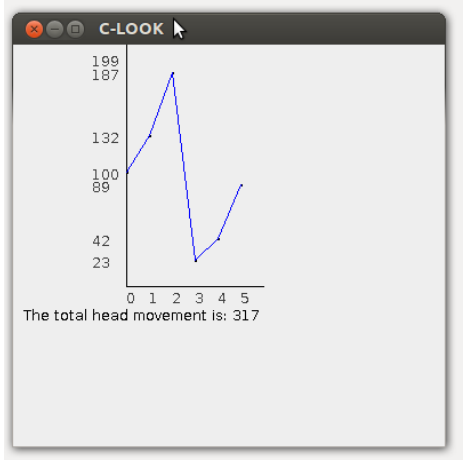

Figure 6: C-LOOK