Software Requirements Specification

for

Disk Scheduler

Version 1.0 approved

Prepared by Ricky Chon

CISC 3320 - Operating Systems

November 30th, 2020

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The goal is to implement a disk scheduler that can perform the following disk-scheduling algorithms:

- FCFS (First Come First Serve)
- SSTF (Shortest Seek Time First)
- SCAN (Elevator)
- C-SCAN (Circular SCAN)
- LOOK (Advanced Elevator)
- C-LOOK (Circular LOOK)

The program will service a disk drive with 5,000 cylinders numbered 0 to 4,999. A random series of 1,000 requests will be generated and serviced according to any of the algorithms listed above. The program can be passed the initial position of the disk head as a parameter on the command line, with its default value equaling half the number of cylinders if no initial value is passed. After performing any of the selected algorithms, the program will output the total amount of head movement that has occurred.

1.2 Document Conventions

This document follows the IEEE standard for software requirements specification.

1.3 Intended Audience and Reading Suggestions

Developers and project managers are the intended audience of this document. It is recommended to read this document from top to bottom, to go from more generic descriptions to more specific implementations.

1.4 Product Scope

The software is intended to demonstrate how disk scheduling works, and illustrate the difference in total head movement amount between different types of disk-scheduling algorithms. By executing each algorithm, developers and users can see the amount of required work the disk arm would do in order to service a list of requests.

1.5 References

2. Overall Description

2.1 Product Perspective

This product is a standalone disk scheduler, but should be built with scalability in mind for future iterations.

2.2 Product Functions

The disk scheduler must be able to do the following:

- Take in an integer parameter as the initial position of disk head in command line
- Prompt the user to select the disk scheduling algorithm of their choice
- Service a list of requests using the selected algorithm
- Output the total amount of disk head movement

An API should be available for the user to call functions to perform any of the available disk scheduling algorithms.

2.3 User Classes and Characteristics

Student Developers: other students that have this same assignment; they should be able to learn how to install the development tools to run the software, if they don't have them already

Faculty Members: professors that view this assignment and test out the software; more experience and technical expertise, should have the necessary development tools to run the software

2.4 Operating Environment

Any of the major operating systems with gcc/g++ support, as well as support for GNU Make: Windows 10, Mac OS X, Linux.

2.5 Design and Implementation Constraints

Since the goal is to create software that would be used in an operating system (disk scheduler), it is recommended to write this software in C/C++, due to its high performance. The software's code should conform to C++17 standards.

2.6 User Documentation

Provide a README markdown file containing details and instructions on running the disk scheduler for end-users, and an INSTRUCTIONS markdown file containing implementation instructions for developers. Developers can use this document alongside the instructions file for reference.

2.7 Assumptions and Dependencies

For the purpose of this assignment, assume that the software will run on any of the major operating systems with gcc/g++ support, as well as support for GNU Make: Windows 10, Mac OS X, Linux

3. External Interface Requirements

3.1 User Interfaces

A standard UNIX terminal, or any terminal that is able to run gcc/g++ and GNU Make would suffice as a user interface to test out the disk scheduler's features. The software will take in user input to select which disk-scheduling algorithm to run. There is also an option for the user to set the initial position of the disk head as a parameter when executing the program on the command line.

3.2 Hardware Interfaces

Any decent computer, laptop, remote server that can run GNU Make and compile C++ code.

3.3 Software Interfaces

3.4 Communications Interfaces

4. System Features

4.1 API - Instantiate DiskScheduler Object

4.1.1 Description and Priority
- Instantiates a DiskScheduler object to access disk-scheduling API
- High Priority

4.1.2 Stimulus/Response Sequences
- User creates new `DiskScheduler` object

- Constructor generates a list of random requests amounting to the value of `NUM_REQUESTS`, with each request value varying between 0 and `NUM_CYLINDERS - 1`.

4.1.3 Functional Requirements

- REQ-1: number of cylinders in disk drive must be 5,000
  - `#define NUM_CYLINDERS 5000`
- REQ-2: number of requests to service must be 1,000
  - `#define NUM_REQUESTS 1000`

4.2 API - FCFS (First Come First Serve) Algorithm

4.2.1 Description and Priority

- Simplest of all disk-scheduling algorithms. Requests are addressed in the order they arrive in the disk queue.
- High Priority

4.2.2 Stimulus/Response Sequences

- User calls the function `void fcfs(int head)`
- Function loops through all requests
- Distance is calculated between current track and head, and added to total head movement
- Current track becomes new head
- Total amount of head movement is printed
- Seek sequence is printed if there are 25 requests or lower

4.2.3 Functional Requirements

- REQ-1: number of cylinders in disk drive must be 5,000
  - `#define NUM_CYLINDERS 5000`
- REQ-2: number of requests to service must be 1,000
  - `#define NUM_REQUESTS 1000`

4.3 API - SSTF (Shortest Seek Time First) Algorithm

4.3.1 Description and Priority

- The tracks closest to the current disk head position are serviced first in order to minimize seek operations
- High Priority

4.3.2 Stimulus/Response Sequences

- User calls the function `void sstf(int head)`
- Function loops through all requests
- Find the request that closest to head and keep track of its index and distance amount
- Distance between head and closest request is added to total head movement

- A flag is set to signify that the request has been serviced, to avoid servicing it again accidentally
- The closest request to old head becomes new head
- Total amount of head movement is printed
- Seek sequence is printed if there are 25 requests or lower

4.3.3 Functional Requirements

- REQ-1: number of cylinders in disk drive must be 5,000
  - `#define NUM_CYLINDERS 5000`
- REQ-2: number of requests to service must be 1,000
  - `#define NUM_REQUESTS 1000`

4.4 API - SCAN (Elevator) Algorithm

4.4.1 Description and Priority

- Head starts from one end of the disk and moves towards the other end, servicing requests in between one by one until the head reaches the end. Then, the direction of the head is reversed and the process continues as the head continuously scans back and forth across the disk.
- High Priority

4.4.2 Stimulus/Response Sequences

- User calls the function `void scan(int head)`
- Initial scan direction is set
- One of the end values of the disk is appended to its respective vector
  - If the direction is left, append 0 to left vector
  - If the direction is right, append `NUM_CYLINDERS - 1` to the right vector
- Function loops through all requests
  - Values smaller than initial head are appended to the left vector
  - Values greater than initial head are appended to the right vector
  - Both vectors are then sorted in ascending order
- A loop is run exactly twice to service requests on the two vectors
  - If scan direction is left
    - Loop through left vector in descending order
    - Sequentially service each track
    - Calculate distance between current track and head
    - Increment distance to total head movement
    - Current track becomes new head
    - When the head reaches the beginning of the disk, reverse direction
  - If scan direction is right
    - Loop through right vector in ascending order
    - Sequentially service each track
    - Calculate distance between current track and head
    - Increment distance to total head movement
    - Current track becomes new head

        - When the head reaches the end of the disk, reverse direction
    - Total amount of head movement is printed
    - Seek sequence is printed if there are 25 requests or lower

4.4.3 Functional Requirements

- REQ-1: number of cylinders in disk drive must be 5,000
    - `#define NUM_CYLINDERS 5000`
- REQ-2: number of requests to service must be 1,000
    - `#define NUM_REQUESTS 1000`

4.5 API - C-SCAN (Circular SCAN) Algorithm

4.5.1 Description and Priority

- Head starts from one end of the disk and moves towards the other end. Unlike SCAN, C-SCAN doesn't reverse the head direction, but goes to the other end of the disk and starts servicing the requests from there.
- Medium Priority

4.5.2 Stimulus/Response Sequences

- User calls the function `void cscan(int head)`
- Both end values of the disk are appended to their respective vectors
    - Append 0 to left vector
    - Append `NUM_CYLINDERS - 1` to the right vector
- Function loops through all requests
    - Values smaller than initial head are appended to the left vector
    - Values greater than initial head are appended to the right vector
    - Both vectors are then sorted in ascending order
- Service the requests to the right of head
    - Loop through right vector in ascending order
    - Sequentially service each track
    - Calculate distance between current track and head
    - Increment distance to total head movement
    - Current track becomes new head
- When head reaches the end of the disk, jump to the beginning (head = 0)
- Service the requests to the left of head
    - Loop through left vector in ascending order
    - Sequentially service each track
    - Calculate distance between current track and head
    - Increment distance to total head movement
    - Current track becomes new head
- Total amount of head movement is printed
- Seek sequence is printed if there are 25 requests or lower

4.5.3 Functional Requirements

- REQ-1: number of cylinders in disk drive must be 5,000
    - `#define NUM_CYLINDERS 5000`

- REQ-2: number of requests to service must be 1,000
    - `#define NUM_REQUESTS 1000`

4.6 API - LOOK (Advanced Elevator) Algorithm

4.6.1 Description and Priority

- Services requests similar to SCAN, but it also looks ahead to see if there are more tracks that need to be serviced. If there are no pending requests in the moving direction, the head reverses the direction and starts servicing requests in the opposite direction.
- Medium Priority

4.6.2 Stimulus/Response Sequences

- User calls the function `void look(int head)`
- Initial scan direction is set
- Function loops through all requests
    - Values smaller than initial head are appended to the left vector
    - Values greater than initial head are appended to the right vector
    - Both vectors are then sorted in ascending order
- A loop is run exactly twice to service requests on the two vectors
    - If scan direction is left
        - Loop through left vector in descending order
        - Sequentially service each track
        - Calculate distance between current track and head
        - Increment distance to total head movement
        - Current track becomes new head
        - When the head reaches the beginning of the disk, reverse direction
    - If scan direction is right
        - Loop through right vector in ascending order
        - Sequentially service each track
        - Calculate distance between current track and head
        - Increment distance to total head movement
        - Current track becomes new head
        - When the head reaches the end of the disk, reverse direction
- Total amount of head movement is printed
- Seek sequence is printed if there are 25 requests or lower

4.6.3 Functional Requirements

- REQ-1: number of cylinders in disk drive must be 5,000
    - `#define NUM_CYLINDERS 5000`
- REQ-2: number of requests to service must be 1,000
    - `#define NUM_REQUESTS 1000`

4.7 API - C-LOOK (Circular LOOK) Algorithm

4.7.1 Description and Priority

- A combination of C-SCAN and LOOK. Prevents unnecessary traversal to both ends of the disk if there are no requests. The head will jump to the other end of disk if it reaches the final request in each direction.
- Medium Priority

4.7.2 Stimulus/Response Sequences

- User calls the function `void clook(int head)`
- Function loops through all requests
    - Values smaller than initial head are appended to the left vector
    - Values greater than initial head are appended to the right vector
    - Both vectors are then sorted in ascending order
- Service the requests to the right of head
    - Loop through right vector in ascending order
    - Sequentially service each track
    - Calculate distance between current track and head
    - Increment distance to total head movement
    - Current track becomes new head
- When head reaches the end of the disk, jump to the beginning of the left vector (head = left[0])
    - Also add the distance between previous head and left[0] to total head movement
- Service the requests to the left of head
    - Loop through left vector in ascending order
    - Sequentially service each track
    - Calculate distance between current track and head
    - Increment distance to total head movement
    - Current track becomes new head
- Total amount of head movement is printed
- Seek sequence is printed if there are 25 requests or lower

4.7.3 Functional Requirements

- REQ-1: number of cylinders in disk drive must be 5,000
    - `#define NUM_CYLINDERS 5000`
- REQ-2: number of requests to service must be 1,000
    - `#define NUM_REQUESTS 1000`

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.2 Safety Requirements

5.3 Security Requirements

5.4 Software Quality Attributes

Maintainability: easy to understand, repair, and optimize without any breaking changes
Scalability: easy to add new features or deprecate old ones
Reliability: make sure the implemented features always work as intended
Testability: easy to test features without extra hassle
Ease of use: easy for developers to pick up and use
Well documented: the software API should be well documented to eliminate any confusion on each feature

5.5 Business Rules

6. Other Requirements

The software should be licensed under the MIT License.

Appendix A: Glossary

API: Application Programming Interface
IEEE: Institute of Electrical and Electronics Engineers

Appendix B: Analysis Models

Appendix C: To Be Determined List