Christian Shadis

Dr. Alsallami

CS-373

Due 5/9/2021

**Device Handler Seek Strategy Comparison Application**

For my final project in CS-373 I opted to implement various Device Handler Seek Strategies in Java and develop an Android application that provides a concise numerical performance summary of the various algorithms. I coded the app in Java in Android Studio, and all algorithm implementations are written in Java. The app has only one screen and limited functionality but utilizes several Android UI elements and seven different device scheduling algorithms.

The motivation for this project was In-Class Activity 6, in which we coded the First-Come, First-Served algorithm and attempted to code LOOK. That activity proved challenging for my team and me, so I decided to pursue it further. I decided to implement First-Come First-Served (FCFS), Shortest Seek Time First (SSTF), SCAN, LOOK, C-SCAN, C-LOOK, and N-Step SCAN in Java, and run a set of requests through every algorithm. I would then calculate the mean, standard deviation, and variance of each request for each algorithm and display those results neatly.

The structure of the application consists of nine Java classes and one XML stylesheet. There is a Requests.java class with attributes ‘Arrival Time’, ‘Track Requested’, and ‘Sector Requested’. The main activity consists of the app’s title, a discrete slider, two buttons, a TableView, and a TextView. The buttons, Run Default and Randomize, respectively, determine what Request objecst are run through the algorithms. The Randomize button generates a list of requests, the length of which is determined by the value of the slider when Randomize is pressed. The Run Default button generates a default 10-element list of requests. The summary statistics are displayed in the TableView, and the requests run through the algorithms are displayed in the TextView.

Implementing the various algorithms in Java was the most difficult part of the project. Each separate class consisted of its own algorithm, a simulation of which is called with the runAlgorithm() method. This method returns a list of string objects containing the average, variance, and standard deviation of the elapsed times. The algorithms were programmed using specifications for seek, search, and transfer times given in Activity 6. Seek time was calculated as 12ms plus 0.1ms for every track traversed. Search time was calculated as 0.5ms for every sector traversed. Transfer time was a constant 1.2ms per request. Additional parameters include the number of tracks, 256, and the number of sectors, 10. Using these parameters and an ArrayList of Request objects, I simulated the movement of the read/write head and stored the time elapsed for each request in an array to calculate the summary statistics. The most challenging aspect of the coding was trying to account for the arrival times of the requests: the head cannot retrieve it if it has not arrived. I approached this challenge by implementing a ‘distances’ helper method that would return a specific value if the arrival time were greater than the current time. If the ‘distance’ were that specific value, the request could not be serviced. Another challenge in implementing the algorithms was the ‘looking ahead’ aspect of algorithms like LOOK. This was also addressed by creating the ‘distances’ method which accounts for the direction of the read/write head’s movement.

While most of my time was spent implementing the algorithms, I also struggled a bit with the various layout containers in Android Studio. I wanted the slider to have much higher values available, but I was unsuccessful in implementing a ScrollView, which would allow for the display of more than 15 Requests in the TextView. I also faced bugs in the user interface because of the app’s Constraint Layout being configured improperly – few of the elements in the app were constrained, which caused every element to overlap or be pushed off the screen.

While there are few requests sent through the simulations, I noticed that LOOK and SSTF tended to be the best-performing algorithms when given a small number of requests. They tended to have the lowest variance and average time. FCFS seemed to be the slowest of the algorithms, and CSCAN and N-Step SCAN seemed to have the highest variance.

If I had more time to work on the project, I would first run many randomized requests through the algorithms to determine with more certainty which ones are the best-performing. I would add a third button to the layout, which would run thousands of requests through the algorithms for the most accurate summary statistics possible. I also would have created a Custom Requests activity in which the user would be able to set the parameters for every request that is sent through the algorithm. They would be able to configure any number of requests and evaluate each algorithm’s performance. Finally, I would have explored finding a way to display the motion of the read/write head to get a more accurate perspective on the efficiency of each algorithm.

I learned a few important concepts in the completion of this project. I learned how to use TableView to arrange UI elements in a grid-shaped pattern, I learned how to implement the Comparable interface to add sorting functionality to custom objects, I learned how to use a discrete slider and improved my understanding of buttons and OnClickListeners, and I improved my Java programming skills.

As an avid Android user, coding applications in Java has real-world benefits: I can learn how to program apps with more real-world usability, make devices interact with each other, and even dabble in the Internet of Things. Java and Android open opportunities to program smart devices other than phones and have innumerable possible applications. While this specific project does not have much use for the general public, it is important for computer scientists to understand the technical aspects of the operating system and how information is actually retrieved from the disk. Coding this project increased my understanding of these concepts and my Android development skills.

Research Materials:

McHoes, A. M., & Flynn, I. M. (2018). Chapter 7: Device Management. In *Understanding operating systems*. Boston, MA, USA: Cengage Learning.

Documentation:

Android: <https://developer.android.com/docs>

Java: https://docs.oracle.com/javase/8/docs/