COM1032 Operating Systems
Lab 10
File System Implementation

Purpose:

The purpose of this lab session is to familiarise yourself with the File System Scheduling and OS Implementation details and programming requirements.

Aim

By the end of the lab you will be able to:

- Design Disk Scheduling Algorithms
- Experiment with Java File Locking APIs

Exercise 1:

Write a Java program that simulates the disk-scheduling algorithms discussed in Section 12.4. of the Silberschatz textbook and covered in week 10 lecture (example queue tracing for all algorithms in Disk Scheduling Slides now on Surreylearn) and summarised in week 10 page:

https://surreylearn.surrey.ac.uk/d2l/le/content/188676/viewContent/1799486/View

In particular, design separate classes that implement the following scheduling algorithms:

a. FCFS
b. SSTF
c. SCAN
d. C-SCAN
e. LOOK

Each algorithm will implement the following interface:

```
public interface DiskScheduler
{
// service the requests
// return the amount of head movement
// for the particular algorithm
public int serviceRequests();
}
```

The serviceRequests() method will return the amount of head movement required by the disk-scheduling algorithm. Reference strings consisting of request for disk cylinders will be provided by the Generator class, which is available on Surrelearn. The Generator class produces random requests for cylinders numbered between 0 and 99. The API for the Generator class appears as follows:

```
// produce a default-sized list of cylinder requests
public Generator()
// produce a list of cylinder requests of size count
public Generator(int count)
// return the list of cylinder requests
public int[] getCylinders()
```

Each algorithm implementing the DiskScheduler interface must supply a constructor that is passed (1) an integer array of cylinder requests and (2) the initial cylinder position of the disk head. Assuming the FCFS class implements DiskScheduler according to the FCFS policy, an example illustrating its usage is shown below:

```
Generator ref = new Generator(1000);
int[] referenceString = ref.getCylinders();
DiskScheduler fcfs = new FCFS(referenceString, 13);
System.out.println("FCFS = " + fcfs.serviceRequests());
```

This example constructs 1,000 random cylinder requests and begins the FCFS algorithm at cylinder 13.

When you have finished, compare the amounts of head movement required by the various disk-scheduling algorithms.

Exercise 2:

Check how a file can be locked for exclusive access using the Channel Java interface:

https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/nio/channels/Channel.html

Create a class with main method, to which you pass as an argument from the command line the path to a file that you are trying to open exclusively to modify its contents and save.

You can either make your code multithreaded to test, or run multiple processes. The thread can produce an output message before trying to acquire the lock and after it has released it. Furthermore, once it has acquired the lock, thread waits ten seconds before releasing it.

On systems supporting either mandatory (i.e. Windows) or advisory (i.e. UNIX) locking, running two copies of this program illustrates how the Java file locking API works. You can do this, by making the file modification takes arguments from the user on the console. While the process stops for I/O, run another instance of the same file in another terminal window, or Eclipse run button.

Another test is to first run the Java program to acquire the file lock. Next, try opening the locked file using a standard text editor or word processor. Compare the different behaviour on mandatory and advisory systems.

Example implementation will be posted in a few days to give an opportunity to think about it.