ECE 2420 Programming Exercise #6

(Deadlock Prevention)

# Overview

In this exercise we will study a hypothetical magnetic hard drive, (no real basis in reality), but as close to a real-world problem as I can dream up. Suppose that an engineering firm has created a novel way to increase hard drive read performance by adding additional read/write heads (not far from reality). However, as the system is undergoing tests, it has become clear that when two adjacent heads try to read or write at the same time, the data becomes corrupted. In this exercise, the coder will implement an algorithm to allow the read/write heads to be as busy as possible, but not corrupt the data from the adjacent head.

# Multi-Head Hard Drive Architecture

As shown in the figure below, a notional hard drive has been developed with *N* heads. However, it is a requirement no two adjacent heads can be reading or writing at the same time due to interference issues. And, as usual, rather than fix the hardware, it has fallen on you as the computer engineer to work around the issue in your embedded code. For this exercise, the student need not worry about read/write ordering nor how the read/write actually happens, (that is all well-developed code). The only thing the coder is concerned with is ensuring that no two **adjacent** heads are engaged at the same time. Your system must also never deadlock. Don’t worry about head starvation or make any other attempt at fairness. That will be handled by the algorithm that calls into your code.

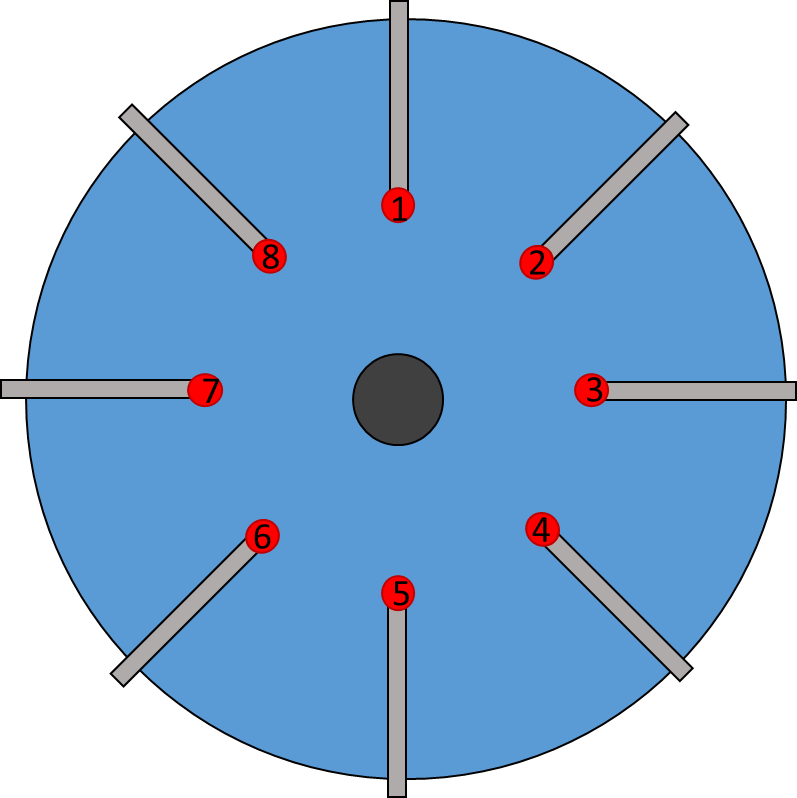


Figure 1. Notional Hard Drive

# Programming Concepts

This exercise covers many programming concepts including deadlock, threads, reference counting pointers, templates, STL library, mutexes, abstract classes and inheritance, lambdas, closures, and functors.

# System Requirements

See grading rubric for grade values associated with specific system requirements. Additional requirements/hints are shown below:

1. The design must use the provided header verbatim. This will allow automated testing of the design you produce.
2. You may (should) create derived classes of your choosing
3. Implementation of functions in abstract base class is acceptable
4. Your main should not be in the same implementation file as your other classes
5. Consider a rigorous main program that adequately tests your deadlock-free algorithm

# Turn-in Procedures

Turn in all source code and required analysis via git commit/push by 11:59p.m. on 19 October.

Grading Rubric

(ECE 2420 PEX6)

|  |  |  |
| --- | --- | --- |
| Requirement / Criteria | Available Points | Student’s Score |
| Implementation is deadlock free | 20 |  |
| Implementation allows maximum concurrency | 20 |  |
| Static factory method is correct; does something smart to link right and left head objects for locking | 10 |  |
| Action is correctly called | 10 |  |
| Rigorous main program tests efficiency and effectiveness | 10 |  |
| Total | **70** |  |