CS 3423 Operating Systems
Fall Semester 2017
Prof. Pai H. Chou

Assignment 12

Due Date: Sunday, December 17, 2017, 11:59pm
Up to one-day late submission without penalty
Up to one-week late submission with 20% penalty
Submit electronically on iLMS

What to submit: One zip file named <studentID>`-hw12.zip` (replace <studentID> with your own student ID). It should contain four files:

- one PDF file named **hw12.pdf.** Write your answers in **English, and elaborate in order to receive full credit**. Check your spelling and grammar. Include your name and student ID!
- No programming for this assignment.

[100 points] Problem Set

1. [20 points] (from Chapter 12) Explain why SSTF scheduling tends to favor middle cylinders over the innermost and outermost cylinders.

2. [20 points] **12.8** Requests are not usually uniformly distributed. For example, we can expect a cylinder containing the file-system metadata to be accessed more frequently than a cylinder containing only files. Suppose you know that 50 percent of the requests are for a small, fixed number of cylinders.
    a. Would any of the scheduling algorithms discussed in this chapter be particularly good for this case? Explain your answer.
    b. Propose a disk-scheduling algorithm that gives even better performance by taking advantage of this "hot spot" on the disk.

3. [40 points] **13.3** Consider the following I/O scenarios on a single-user PC:
    a. A mouse used with a graphical user interface
    b. A tape drive on a multitasking operating system (with no device preallocation available)
    c. A disk drive containing user files
    d. A graphics card with direct bus connection, accessible through memory-mapped I/O

    For each of these scenarios, would you design the operating system to use buffering, spooling, caching, or a combination? Would you use polled I/O or interrupt-driven I/O? Give reasons for your choices.

4. [20 points] **13.7** Typically, at the completion of a device I/O, a single interrupt is raised and appropriately handled by the host processor. In certain settings, however,

the code that is to be executed at the completion of the I/O can be broken into two separate pieces. The first piece executes immediately after the I/O completes and schedules a second interrupt for the remaining piece of code to be executed at a later time. What is the purpose of using this strategy in the design of interrupt handlers?