CIS 415 Operating Systems: Worksheet 5 - Semaphore Basics - Solutions

1. Indicate whether synchronization is needed in the following areas of OS resource management. Is semaphores the appropriate mechanism for each area?
   a. management of dedicated devices such as printer.

   Synch needed. Semaphores not appropriate at level of user processes because they don't share variables. Single printer server can serve to coordinate use.

   b. movement of PCBs among blocked, ready, and running queues -- i.e. doing the linked list manipulations.

   Synch needed. Movement of PCBs must be atomic to keep queues in order. Semaphores might be used. Also masking of timer and I/O interrupts while the queue is being modified.

   c. managing one buffers for disk I/O; managing two buffers for disk I/O.

   Synch needed. Producer-consumer issues regardless of one buffer or more. Semaphores work for this.

   d. coordinating two processes that are currently executing the Unix ls command on the same directory.

   No synch needed. Reading the directory doesn't require synch.

   e. coordinating read and write access to an array of integers shared between a parent and a child process.

   Yes synch needed. Classic semaphore situation.

   f. coordinating students who want to come to Ginnie's office hours when she only has four chairs.

   Yes sink needed, to wash dishes after snacking in Ginnie’s office.

2. Look at each sample code below and indicate whether it is possible for processes A and B to violate mutual exclusion. CS = critical section.

   a. semaphore mutex = 1;
      A: P(mutex); CS; V(mutex);
      B: P(mutex); CS; V(mutex);  OK, standard semaphore solution.

   b. A: disable interrupts; CS; enable interrupts;
      B: disable interrupts; CS; enable interrupts;  Works on uniprocessor but too low level. May not work on multicore.

   c. semaphore mutex = 1;
      A: disable interrupts; CS; enable interrupts;
      B: P(mutex); CS; V(mutex);  Violates M.E. Only B is using the semaphore!

   d. semaphore mutex = 1; semaphore cutex = 1;
      A: P(mutex); CS; V(mutex);
      B: P(cutex); CS; V(cutex);  A and B are not using same semaphore

      Violates M.E.
e. semaphore mutex = 2;
   A: P(mutex); CS; V(mutex);
   B: P(mutex); CS; V(mutex);

   Violates M.E.
   mutex = 2 -> both allowed in

f. semaphore mutex = 1;
   A: V(mutex); CS; P(mutex);
   B: V(mutex); CS; P(mutex);

   Violates M.E.
   improper use of P and V

/* test-and-set atomically reads flag and sets it to not-clear */
g. flag = clear;
   A: while (test-and-set(flag) != clear) noop; CS; flag = clear;
   B: while (test-and-set(flag) != clear) noop; CS; flag = clear;

   Works on all systems; standard solution.

h. flag = clear;
   A: while (flag != clear) noop; flag = not-clear; CS; flag = clear;
   B: while (flag != clear) noop; flag = not-clear; CS; flag = clear;

   Violates M.E. Same problem as COUNT++/--

i. flag = 1;
   A: P(flag); CS; V(flag);
   B: P(flag); CS; V(flag);

   Compiler error. flag is not a semaphore.