A3 Solution Key

Part I - 20 points each

For yes/no questions, be sure to provide a brief justification.

1. A microprocessor supports a single hardware timer. Suppose instruction SET_TIMER permits one to set the timer value[1]. Does this need to be a privileged instruction?

   Operating systems should not let users access hardware directly. Therefore, any instruction that changes hardware state must be protected. For instructions, making an instruction privileged will satisfy this as execution in user mode will trigger a privilege violation interrupt.

2. When a process executes a TRAP or is interrupted, the operating system uses a separate stack located in memory unavailable to user processes to execute any operating system code rather than the stack of the current process. Why might operating systems designers select this type of implementation?

   The operating system sometimes handles sensitive information. If that information is pushed onto a user stack, those values may remain in memory when control is returned to the user process. Popping stacks usually just involved moving a stack pointer, not clearing out memory. There are other reasons to do this (e.g. avoiding corruption of a user process, or avoiding potential kernel crash in pushing information to user stack if user process doesn't allow enough stack space), but this is the primary one.

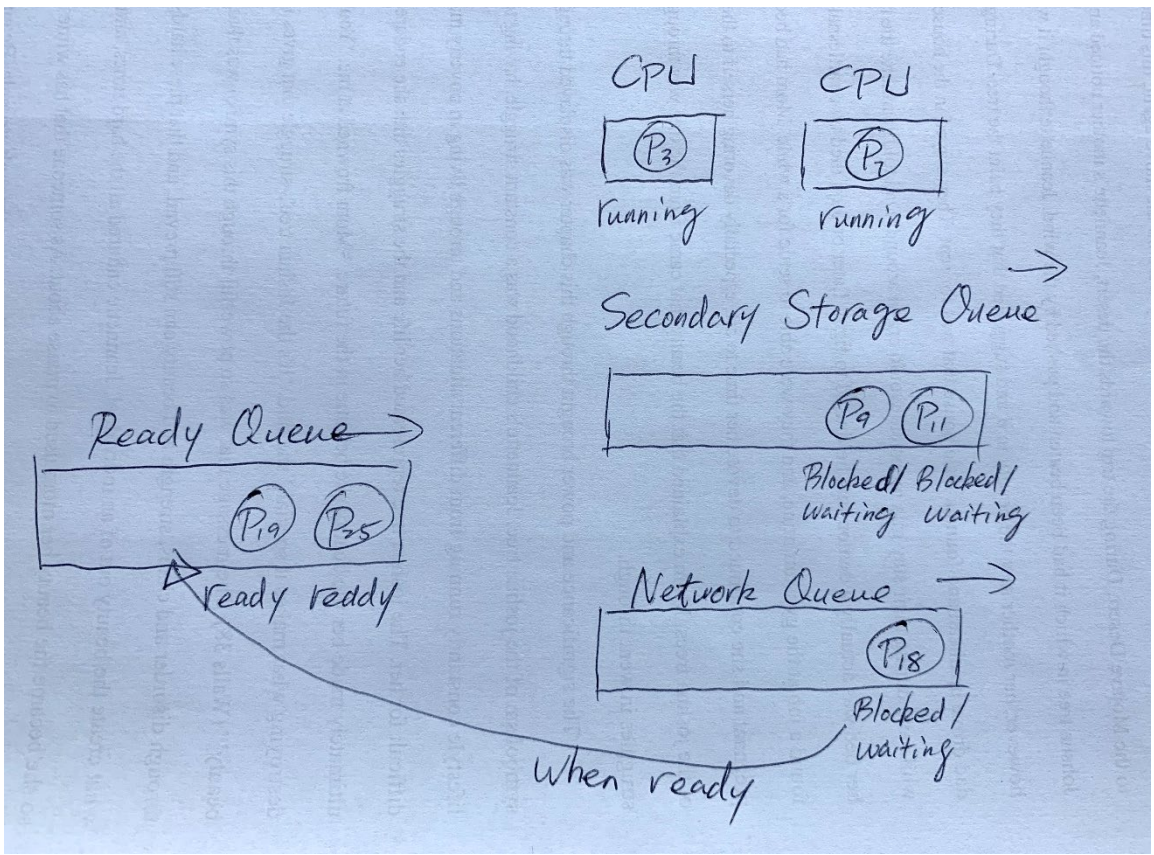
3. During an interrupt, how does the operating system select the interrupt service routine to run?

---

[1] Note that this is a conceptual question. Timers are usually controlled by memory-mapped registers, something that we have not yet studied.

When an interrupt occurs, the interrupt handler routine is invoked. It examines the interrupt number (usually given by an interrupt controller). The addresses of functions servicing each type of interrupt are stored in a table. We index this table by interrupt number and call the appropriate interrupt service routine.

4. Processes P3 and P7 are executing. The system has two types of pending I/O requests. P9 and P11 are waiting on secondary storage reads. P18 is waiting for a network write to complete. P25 and P19 are awaiting assignment of the CPU. Draw a queueing diagram for these processes. Draw an arrow showing where P18 will go once it completes. Write the state that each process is in next to the process bubble.



As we did not specify the order of multiple processes in the queues, any ordering is acceptable..

5. Process P92 has user-level threads. Process P33 has kernel-level threads.
    a. Will P33 run in supervisory mode?
    Only if P33 was started in supervisory mode. Kernel-level threads describes the switching mechanism, not whether or not the process has supervisory privileges.

    b. Which process will have faster context switches?
    P92 will have faster context switches as the overhead of shifting to supervisor mode will not occur and we are guaranteed that the next thread that runs is from the same the process.

    c. What happens when a thread in P92 or P33 block?
    When a P92 thread blocks, all threads in P92 are blocked as the operating system is not aware of them and P92 must be executing for a context switch between its threads to occur. In contrast, the OS is aware of P33's threads, so if one of them blocks, any other P33 thread in the ready state may be executed.