CS 318/418/618, Fall 2017

Homework #1

1. [Anderson] For each of the three mechanisms that supports dual-mode operation – privileged instructions, memory protection, and timer interrupts – explain what might go wrong without that mechanism, assuming the system only had the other two. (In other words, if we just had memory protection and timer interrupts but not privileged instructions, what could go wrong; if we just had privileged instructions and timer interrupts, what could go wrong, etc.)

   Anwser: Many answers are possible. User-level access to privileged instructions would allow a malicious process to directly perform I/O operations that could read or modify anyone's files on disk. User-level access to physical memory would allow a process to overwrite data structures in another process. User-level ability to disable timer interrupts would allow a buggy process to loop forever, never yielding the processor.

2. [Silberschatz] What are the differences between a trap and an interrupt? What is the use of each function?

   Anwser: The CPU initiates a trap. Usually an I/O device initiates an interrupt. Traps are often used to catch arithmetic conditions (e.g. overflow, divide by zero). Interrupts are used to interrupt the CPU and run an interrupt service routine to handle some externally generated condition.

3. [Silberschatz] Which of the following instructions should be privileged?

   (a) Set value of timer

   (b) Read the clock

   (c) Clear memory

   (d) Turn off interrupts

   (e) Switch from user to monitor mode

   Anwser: All except (b).

4. [Silberschatz] Protecting the operating system is crucial to ensuring that the computer system operates correctly. Provision of this protection is the reason for dual mode operation, memory protection, and the timer. To allow maximum flexibility, however, you should also place minimal constraints on the use.

   The following is a list of instructions that are normally protected. What is the minimal set of instructions from this list that must be protected?

   (a) Change to user mode
   (b) Change to monitor mode
   (c) Read from monitor memory
   (d) Write into monitor memory
   (e) Fetch an instruction from monitor memory
   (f) Turn on timer interrupt
   (g) Turn off timer interrupt

   Anwser:

   (b). Otherwise process can get into monitor/kernel mode.
   (c). Otherwise any process can read private data (security risk).
   (d). Otherwise any process can change private data.
   (g). Otherwise any process can run forever.

5. [Crowley] Suppose the hardware interval timer only counts down to zero before signaling an interrupt. How could an OS use the interval timer to keep track of the time of day?

   Anwser: Have the interval timer reset/restart when the interrupt fires. It can do this continually to keep track of time gone by.

6. [Anderson] Suppose you have to implement an operating system on hardware that supports exceptions and traps but does not have interrupts. Can you devise a satisfactory substitute for interrupts using exceptions and/or traps? If so, explain how. If not, explain why.

   Anwser: Interrupts notify the operating system about asynchronous events. On a multiprocessor, the operating system could simulate device interrupts by using one processor to poll devices to see if an event has occurred on that device. Simulating timer interrupts – to regain control of the processor from a runaway process – is harder. The operating system would need to simulate the application execution instruction by instruction, so as to periodically check whether it needs to invoke the timer interrupt handler.

7. [Anderson] Explain the steps that an operating system goes through when the CPU receives an interrupt.

   Anwser: After the hardware saves the old stack pointer, the old program counter, and other registers, it switches to the kernel stack at the beginning of the (assembly language) interrupt handler stub. The interrupt handler stub must then save (and restore) the remaining register state, set up (tear down) the interrupt handler stack frame, and call into (return from) the interrupt handler, and execute the handler. The handler itself is device specific, but in general it will at least read control registers off the device to determine what operation has completed.

8. [Tanenbaum] For each of the following system calls, give a condition that causes it to fail: `open`, `fork`, `exec`, `unlink`.

   Anwser:

   (a) `open`:
      - The process already has the maximum number of files open.
      - The requested access to the file is not allowed.
      - The pathname was too long.

   (b) `fork`:
      - Insufficient memory to create new process.

   (c) `exec`:
      - Insufficient permission or no execute bit is set on the executable.
      - Path to executable is invalid.
      - Insufficient memory to execute.

   (d) `unlink`:
      - Path to file is invalid
      - Insufficient permission
      - File is read-only

9. The Java runtime provides a set of standard system libraries for use by programs. To what extent are these libraries similar to the system calls of an operating system, and to what extent are they different?

   Anwser: Java's system libraries are similar to the system calls of an operating system in programming interface. Programmers familiar with the C system calls for filesystems, sockets, exec, etc. will find the Java versions very similar. The Java libraries however do not represent a switch from user mode to kernel/monitor mode. Thus there is no boundary crossing.

10. List two challenges an operating system faces when passing parameters between user and kernel mode. Describe how an operating system can overcome them.

    Anwser: One is the safety challenge. User code might pass invalid parameters to kernel that would cause a kernel panic. The other is efficiency challenge. If the parameter is large, transferring the parameters might be costly. For the safety challenge, the OS needs to perform extensive sanity checks. For the efficiency challenge, the OS should avoid or reduce copies of user memory when possible.