Remarks: Keep the answers compact, yet precise and to-the-point. Long-winded answers that do not address the key points are of limited value. Binary answers that give little indication of understanding are no good either. Time is not meant to be plentiful. Make sure not to get bogged down on a single problem.

PROBLEM 1 (20 pts)
UNIX/Linux export a comprehensive and unified “signal” abstraction to user space via system calls while Windows does not. Based on the architecture of modern kernels we discussed—upper half, lower half, asynchronous user space callback function support, kernel/user mode, isolation/protection—sketch the signal layer design of a kernel that implements a signal abstraction similar to UNIX/Linux. Describe what happens when an event is processed by the kernel that ultimately is exposed to user space as a signal event (in UNIX parlance, a signal is raised). Note that much of our discussion on asynchronous I/O with callback function support—allbeit generalized to a unified signal abstraction—applies here.

PROBLEM 2 (20 pts)
Describe how full virtualization works and why it is considered efficient. Explain the complication encountered in x86 systems with sensitive instructions and how this problem can be addressed in practice. The Android mobile operating system utilizes virtualization by running specialized Java bytecode (i.e., Dalvik) in a Java virtual machine environment (Dalvik VM) that sits on top of a Linux kernel. What are the pros/cons of such a software virtualization approach compared to full virtualization? Why might we not just port mobile operating systems such as Android to desktops and servers? What about laptops?