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)

Timesharing (TS) process scheduling has been influenced by an approach that classifies processes as CPU- and I/O-bound, with the aim of assigning the latter higher priority. What is the rationale behind this approach? What would be the desired outcome of such an approach? Describe specific kernel mechanisms, say, in the context of XINU, that may be used to perform process classification at run-time. Assess the accuracy and overhead of the classification mechanism. Is time slicing necessary a necessary component of TS scheduling? What are the weaknesses of the classification-based approach? A couple of years ago, Linux adopted a fair scheduling implementation as its default scheduler (completely fair scheduler). How does fair scheduling deal with CPU- and I/O-bound processes, and what are the similarities/differences compared to classification-based TS scheduling? Is one superior than the other? Explain your reasoning.

PROBLEM 2 (20 pts)

Suppose you were asked to modify a UNIX/Linux kernel so that user space page fault handling is supported by extending the signal mechanism. That is, a user process registers its own page fault handling routine (i.e., callback function or signal handler) that is invoked by the kernel—after switching to user mode—when a page fault occurs. In particular, we define a new signal, SIGPGF, that is raised when a page fault occurs and a SIGPGF signal handler, registered by a user process with the kernel, is executed. In the context of x86, are there other parts of the kernel that need to be modified to guarantee isolation/protection? What are the pros/cons of the user space approach compared to kernel-based page fault handling? You may assume that persistent storage is provided by hard disk. Do the pros/cons of this approach change if persistent storage is provided by flash memory? For solid state drive persistent storage, which makes more sense in your view: page fault handling by hardware/firmware, kernel PGF handlers, or user space handlers? Explain your reasoning.