

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 (50 pts)

(a) What is the approach followed by operating systems such as Solaris for scheduling time shared processes? How do different scheduling classes (e.g., TS, SYS, RT in Solaris) interact and co-exist? How can one implement EDF in Solaris without modifying the kernel?

(b) Based on our coverage of Xinu, what parts of Xinu are hardware dependent requiring assembly coding, and what parts can be written in C? How would one go about modifying Xinu so that it provides protection/isolation? Since we haven't discussed high-level memory management, memory protection issues can be ignored.

(c) Why are operating systems organized into upper and lower halves? What kernel services only require the upper half (i.e., the lower half is not involved at all)? Why is the lower half further subdivided into top and bottom halves? How is Xinu's lower half architected? Is a top/bottom half division needed for the `tty` device driver?

(d) What are internal and external fragmentation in the context of Xinu's low-level memory management? What is the cost of memory compaction? Is compaction a feasible solution? How is the external fragmentation problem handled in modern operating systems? Is hardware support optional or required?

(e) What is zero-copy I/O? What major savings in overhead does it enable, especially for file servers, when compared to operating systems that do not support zero-copy I/O? In particular, compare the savings against file server implementations using memory-mapped (i.e., shared memory) files supported by most operating systems.

PROBLEM 2 (30 pts)

(a) What is the main advantage of implementing multithreading support entirely in user space as compared to kernel level support? What is the main disadvantage? How can it be overcome? What is Xinu's process model and how does it relate to multithreading? If you were to extend Xinu to a multi-user operating system, how would you go about modifying its process model?

(b) Discuss how real-time video streaming from a DV camcorder attached to a FireWire (or USB 2.0) interface on x86 PCs is handled in operating systems such as Linux and Windows with DMA support. Describe the roles played by the major subsystems such as DMA controller, interrupt controller, top half, bottom half, and upper half. Where is the typical performance bottleneck in Linux? What differences exist for Windows? For isochronous video streaming over FireWire/USB, does it make sense for DMA to copy the data directly to user space? How about the case where the source of the video stream is an Ethernet interface carrying UDP or TCP packets (e.g., the camcorder is a networked webcam)? Explain your reasoning.

PROBLEM 3 (20 pts)

What are the challenges associated with providing accurate real-time scheduling support (e.g., RMS) as part of a kernel for killer apps such as real-time compressed video playback and encoding? Separate the challenges into three categories: application dependent, kernel dependent, and hardware dependent. Discuss how the challenges may be addressed. In the kernel dependent case, relate your solution to how modern kernels are architected (i.e., interrupt processing). Give an evaluation of which solutions/challenges are more or less promising. Can the "Ostrich solution" that we discussed in the context of how kernels deal with deadlock detection/prevention play a useful role for real-time multimedia scheduling support? Explain your reasoning.

BONUS PROBLEM (10 pts)

An important discovery in systems is the peculiar nature of process lifetimes which has implications to CPU scheduling and load balancing. Describe what this discovery is and why it is relevant to systems engineering.