CS 354 (Park) **Final** **May 2 (Tue.), 2017**

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 a delta list? What is its time complexity? (Consider enqueue and dequeue operations separately.) Why is a delta list well suited for managing kernel timer events including sleep events? Which part of a kernel handles sleep dequeue events, which handles enqueue events? In XINU, what specific kernel function calls are invoked by the two parts? How is time slice bookkeeping (i.e., countdown) of the current process managed in XINU? What is its overhead? How can time slice bookkeeping be carried out by using a unified event queue that keeps track of both sleep and time slice depletion events via a single delta list? In lab6, a signal called XINUSIGXTIME that represents a process's wall time exceeded event was added to XINU's signal handling subsystem. How can XINUSIGXTIME be integrated into the unified event queue?

(b) Page tables, in general, can exert significant memory pressure in demand paging kernels. What does memory demand of real-world processes look like, and how does this help reduce page table memory pressure? In a 32-bit architecture with 4 KB pages, describe how a 2-level page table helps reduce memory consumption of page tables. To speed up virtual-to-physical address translation using page table look-up, what hardware support is provided and who—kernel or hardware—manages this support? What is a page fault and who (kernel or hardware) handles it in modern kernels? What is the rationale for this design? Suppose a page fault occurs and memory is completely filled, i.e., there are no free frames. If a kernel had a crystal ball and could see into the future, which page would be optimal to evict? If this page had been modified (i.e., written to) while in resident in RAM, would it still be the best page to evict to disk or SSD? Explain your reasoning.

PROBLEM 2 (50 pts)

(a) What is the motivation for diving the lower half of a kernel into two subsystems—top and bottom halves—in today's operating systems? What are the tasks assigned to a top half? What are the chores assigned to a bottom half? What are the two design choices for implementing a bottom half? What are their pros/cons? Why is hardware support in the form of DMA important for delivering adequate performance in web cam streaming applications? How does the role of the top half change when DMA is used to assist in video streaming? Suppose a process makes a read() system call to read data from an interface (e.g., USB) which is to be copied to a user space buffer. Assume that at the moment the read() system call is invoked, the requested data has not arrived and the kernel buffer allocated for the device is empty. Since read(), by default, is blocking, the process that invoked it is context switched out. At some point in the future, the requested data arrives at the device interface which triggers an interrupt. D escribe the subsequent sequence of events, including who the producers and consumers are of kernel and user space buffers, that results in delivery of the data to user space buffer.

(b) Suppose an app programmer is tasked with writing an app that sends out data packets through a network interface using a system call, sendpacket(), where packets are to be spaced 0.5 msec apart. This should result in 2000 packets being sent out during a 1 second time interval. Assume CPU and network interface speeds are more than adequate for handling this load. The app programmer proceeds to write code whose structure is a loop wherein sleepmicro()—a system call that sleeps for a given number of microseconds—is called with argument 500 after a packet is sent out using sendpacket(). The app runs on a tickful Linux kernel with the tick value set to 1 msec (as in XINU). What approximate data rate in units of packets per second (the goal is 2000) does the app actually generate when it is executed? Explain how you arrive at your answer. Suppose a colleague suggests to this programmer that configuring Linux as a tickless kernel may help achieve the desired data rate. Why is this a meaningful solution? Given the benefit that a tickless kernel may bring, what is its potential drawback? What is an alternate solution that preserves the default tickful mode of Linux but reconfigures it? What is its potential drawback?

BONUS PROBLEM (10 pts)

What are three real-world properties (two pertain to processes) discussed in class that operating systems exploit to significantly improve performance? What would happen for kernel design and system performance if we lived in a world where these properties did not hold? Which of them is the most important, and why?