CS 354 (Park) **Final** **May 1 (Wed.), 2013**

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 (40 pts)

(a) What is a delta list, and what is its main benefit when using it to store future sleep events?

(b) In the case of processes using semaphores to mediate orderly access to shared resources, what is the simplest instance of a deadlock? How would a kernel go about detecting deadlocks, and what is the resultant overhead? What is the approach adopted by modern operating systems when it comes to dealing with deadlocks, and why is the approach viewed as viable?

(c) What is external fragmentation? What is the solution utilized by modern kernels, and why is hardware support an essential component? What is the typical role played by L1 caches in today's systems with respect to memory referencing, and why does it help mitigate the external fragmentation problem?

(d) What is the key difference between the Xinu and UNIX file systems with respect to performance? What additional optimization is implemented in UNIX file systems to address the idiosyncrasy of file sizes ("mice and elephants") in real-world file systems?

PROBLEM 2 (36 pts)

(a) In today's computing systems, TLB misses may—or may not—trigger interrupts that allows kernel code to intervene and load the missing page table entry from RAM into TLB. For example, in our `x86` machines TLB misses are handled by hardware. In contrast, page faults are implemented as interrupts that are processed by a kernel's page fault handler. First, define what a page fault is, and second, explain the rationale behind managing page faults in software, i.e., kernel, not hardware.

(b) Context switching between processes incurs significant overhead. What are the main factors that contribute to this cost? Make sure not to focus solely on overhead discussed under process management. That is, the context of a process is broader than that.

(c) In Lab 2, asynchronous nonblocking IPC with callback function was implemented in Xinu. Although Xinu does not implement isolation/protection, what design choices were followed to achieve isolation/protection which would have been necessary had the implementation been ported to Linux/Windows? Asynchronous nonblocking IPC with callback function can be adopted to device I/O where, instead of a writer process sending a message to a receiver process, an interrupt handler for a received message (e.g., Internet packet) copies the message to a receiver process. What is the reason that the technique for inter-process communication carry over to device I/O?

PROBLEM 3 (24 pts)

(a) What are the roles of the upper half and lower half of a kernel? Why is the lower half further subdivided into a top half and a bottom half? What are the two design options available for implementing a bottom half? What are their pros/cons?

(b) Suppose we have a 32-bit `x86` machine with 4 KB page size where 12 bits are used to specify the offset within a 4 KB page, and the remaining 20 address bits are translated from virtual to physical addresses by looking up the entries of a process's page table. In a 1-level page table, 2^{20} entries are required, per process, to enable address translation which consumes significant memory. What is a 2-level page table, and how does it help reduce the space requirement of a process's page table? Assume the 20 address bits are divided into two parts, 10 bits each. If a process needs only 5 pages of virtual memory to execute, how much space saving does a 2-level page table yield compared to using a 1-level page table?

BONUS PROBLEM (10 pts)

In our discussion of modern kernels, most of a kernel's services (e.g., to handle system calls, interrupts) were implemented as a library of function calls that borrow the context of the current process. We noted only two exceptions where a kernel's service was implemented as a process, i.e., kernel thread. What were these two exceptions? What is the main rationale for constructing a kernel as a library of function calls instead of a pool of kernel threads?