CS354 Midterm Solution, spring 2025

P1(a) 20 pts

Incorporating the features of real-world workloads characterized by CPU- and I/O-bound processes (and hybrids) where the goal is to provide I/O-bound processes that do not significantly consume CPU cycles with improved response time.
4 pts

I/O-bound processes are assigned higher priority and smaller time slice compared to CPU-bound processes.
4 pts

Solaris monitors how long a process waits in ready state to receive CPU cycles. If wait time exceeds a threshold the process's priority is increased.
4 pts

A round-robin scheduler would be well-suited.
4 pts

A round-robin scheduler is not well-suited for mixed workloads since it does not differentiate between CPU- and I/O-bound processes, thus not improving the response time of I/O-bound processes that consume significantly less CPU cycles.
4 pts

P1(b) 20 pts

Two processes P1 and P2, two semaphores Sa and Sb. P1 runs first, acquires Sa, then is context-switched out. P2 runs next, acquires Sb, then attempts to acquire Sa which results in blocking and being context-switched out. P1 runs again, tries to acquire Sb causing blocking on Sb and being context-switched out. Neither process can become ready.
6 pts

A deadlock exists in a resource graph if there is a cycle.
// Can be detected using breadth first (or depth first) traversal.
4 pts

Cycle detection has linear overhead (i.e., time complexity).
3 pts

Deadlock detection incurs linear overhead. When a deadlock arises its impact is for the most part confined to the app whose processes/threads are stuck. The principle of underlying isolation/protection is not violated.
4 pts

Order all semaphores in a specific linear (i.e., total) order. All processes must acquire semaphores dictated by this order.
3 pts

P2(a) 20 pts

When context-switching in a process that ran before but was context-switched out, changing the order of restoring EBP and EFLAGS does not matter since ctxsw() returns to resched() where interrupts remain disabled (i.e., IF = 0).
// The above is for clarity and may be skipped as long as explanation below
// is clear.

In the case where the process being context-switched in runs for the very first time, executing ret in ctxsw() will cause a jump to the function specified in the first argument of create(). Since user code must run with interrupts enabled, restoring EFLAGS will enable external interrupts (i.e., IF = 1). Thus restoring EFLAGS before restoring EBP allows the possibility of the context-switching in procedure to be preempted before its completion where EBP is restored. This leads to an inconsistent system state.
// Inconsistent since the newly saved EBP when context-switching in is
// preempted is not the EBP of the checkpointed process. Preemption while
// context-switching in has not been completed violates XINU's kernel
// design where all kernel code (upper or lower half) completes while
// interrupts are disabled.
12 pts

When the process being context-switched in runs for the very first time, executing ret in ctxsw() causes a jump to the first instruction of the function specified as first argument of create(). Otherwise, ctxsw() returns to resched().
8 pts

P2(b) 20 pts

Interrupt disabling.
Main advantage: low overhead (and simple).
Main drawback: disruptive since all external interrupts are silenced.
4 pts

tset.
Main advantage: interrupts remain enabled.
Main drawback: meaningless on uniprocessor machine, wastes CPU cycles by busy waiting.
4 pts

Counting semaphore.
Main advantage: mitigates busy waiting of tset by blocking which context-switches out a process until a shared resource becomes available.
// No wastage of CPU cycles busy waiting.
4 pts

A producer/consumer buffer can always be protected using a single counting semaphore. Utilizing two counting semaphores allows concurrent access to the shared FIFO buffer as long as the buffer area for read and write do not overlap (i.e., there is a gap).
4 pts

Counting semaphore implemented using wait() and signal() are not pure software primitives since interrupt disabling is used to achieve atomicity of operations within wait() and signal().
4 pts

P3 20 pts

XINU system calls are regular function calls that do not contain a trap instruction.
4 pts

XINU's GDT has 3 main entries: kernel mode text, kernel mode data, kernel mode stack. Linux/Windows has 4 main entries: kernel mode text, kernel mode data (stack is treated as data), user mode text, user mode data.
4 pts

In lab2 we did not implement user mode and kernel mode separation (all processes still ran in kernel mode). We did not implement switching between user stack and kernel stack.
4 pts

In XINU, ctxsw() returns by jumping to the code of the function specified as first argument of create(). We need to change the ret instruction of ctxsw() to iret so that it untraps when jumping to user code in user mode. We also need to switch the runtime stack from kernel stack to user stack. The two steps can be facilitated by modifying create() so that it sets up the stack of a newly created process by pushing in SS, ESP, EFLAGS, CS, EIP where SS points to the GDT entry for user data/stack, ESP points to the user stack, EFLAGS has IF set to 1, CS points to GDT's user text entry, EIP is the function pointer provided as first argument of create().
// The above are the main elements. ctxsw() must be further modified so that
// when the process being context-switched in is not a newly created process
// then ctxsw() returns to resched() by executing ret. Other correct designs
// are possible too.
8 pts

Bonus 10 pts

In asynchronous IPC with callback function, a process registers user code in the form of a callback function with a kernel requesting that the callback function be executed when a specific future event occurs.

5 pts

Since the callback function is user code, a kernel must make arrangements such
that it is executed in user mode and in the context of the process that registered
it. This assures that isolation/protection is preserved.
5 pts