CS354 Midterm Solution, spring 2023

P1(a) 20 pts

Blocking system call: process is context-switched out if event (e.g., message arrival, timer expiration) has not occurred yet.
4 pts

Non-blocking system call: if event has not occurred yet, system call returns immediately with a return value indicating status of system call (e.g., failure).
4 pts

Shared memory IPC avoids copy operation from sender memory (i.e., buffer where message is located) to receiver memory. Copy overhead is linear in the size of the message, hence slow.
7 pts

Support for coordination/synchronization between processes needs to be provided to prevent data corruption in shared memory upon concurrent access.
5 pts

P1(b) 20 pts

If current process depletes its time slice, kernel considers it CPU-bound.
3 pts

If current process makes a blocking system call and voluntarily gives up CPU before depleting its time slice, kernel considers it I/O-bound.
3 pts

CPU-bound rationale: decrease priority, increase time slice.
3 pts

I/O-bound rationale: increase priority, decrease time slice.
3 pts

No. Since an I/O-bound process tends to have higher priority than a CPU-bound process, when a blocked I/O-bound process becomes ready it is able to preempt a CPU-bound process executing on a CPU (i.e., large time slice of CPU-bound process does not prevent ready I/O-bound process from promptly receiving CPU cycles).
4 pts

Yes. A process deemed I/O-bound may change its behavior by becoming CPU-bound. If so, the process will execute with high priority and large time slice which may prevent CPU-bound processes from receiving CPU cycles for a prolonged period.
4 pts

P2(a) 20 pts

Process making system call (which runs upper half code).
5 pts

Interrupt causing the current process to run lower half code.
5 pts

No. XINU processes always run in kernel mode.
5 pts

Yes. msclkcounter1 is incremented by XINU's lower half when a clock interrupt is generated every 1 msec. If the process executing main() runs with interrupts disabled, clock interrupts are disabled hence msclkcounter1 will not be incremented. Accuracy suffers.
5 pts

P2(b) 20 pts

When a process is created using system call create(), the process's stack is set up such that it mimics the content it would have had the newly created process been context-switched out (i.e., contains 8 general-purpose

registers, EFLAGS, EBP values). Therefore the same ctxsw() function will work when context-switching in a newly created process.
7 pts

Case 1: Not a newly created process. When ctxsw() executes ret it returns to its caller resched().
4 pts

Case 2: Newly created process. create() set up the stack so that ctxsw() does not return to resched() but jumps to the function pointer specified as first argument of create() (i.e., code that the newly created process is supposed to run).
4 pts

No. Since internal kernel function resched() called ctxsw() interrupts remain disabled (i.e., IF bit of EFLAGS equals 0) when ctxsw() returns to resched() (since XINU runs kernel code with interrupts disabled). Hence there is no possibility of context-switch steps being disrupted before completion.
5 pts

P3 20 pts

Synchronous IPC: Current process makes IPC system call to perform communication.
5 pts

Asynchronous IPC: Current process does not make IPC system call to read/write. Instead, a function pointer is provided to the kernel (i.e., callback function is registered) so that the kernel arranges for the callback function to be executed when the relevant event happens (e.g., message is received).
5 pts

In asynchronous IPC a kernel is tasked with executing a callback function registered by a process when a relevant event happens in the future. To preserve isolation/protection the callback function must be run in user mode and in the context of the process that registered it.
10 pts

Bonus 10 pts

If a ready process has waited for 1 sec (can be any value configured in the scheduling table) then its priority is increased (which makes it more likely to run in the near future).
5 pts

Log the time a process becomes ready. Check readylist at every clock interrupt to determine if a process has been waiting more than a threshold value. This is a simple but inefficient procedure that incurs linear overhead.
5 pts