CS354 Midterm Solution, spring 2019

P1(a) 13 pts

Respond to system calls.
3 pts

XINU system calls do not trap from user mode to kernel mode.
3 pts

XINU system calls disable external interrupts.
3 pts

The first impacts reliability/security.
2 pts

The second impacts responsiveness.
2 pts

P1(b) 13 pts

Respond to interrupts.
3 pts

Clock (or system timer) interrupts handling.
3 pts

Manage time slice count down.
1 pts

Manage sleep queue dequeue/extract.
1 pts

Update system uptime clktime.
1 pts

Manage time slice: when time slice reaches 0, resched() is called.
2 pts

When a process's sleep time has expired, it is moved from the sleep queue to
the ready list and resched() is called.
2 pts

P1(c) 13 pts

On a single processor system, test-and-set consumes CPU cycles while repeatedly
executing tset until time slice expires. This wastes CPU cycles. Interrupt disabling
does not.
4 pts

Interrupt disabling prevents important events such as clock interrupts that manage
time slice from being carried out during a critical section (CS). If a CS takes
significant time to execute, this can adversely impact system responsiveness.
Counting semaphores only disable interrupts during wait() and signal() which take
incur constant overhead. During a CS, interrupts remain enabled.
4 pts

read(), a slow system call, can take significant time to complete depending on the
amount of data to be copied from kernel to user memory.
2 pts

Because the primitives wait() and signal() require hardware support (interrupt
disabling) to work correctly.
3 pts

P1(d) 13 pts

Restoring EFLAGS, if the IF bit is 1, may re-enable interrupts at which point
ctxsw() may be preempted before completion by another process which may lead to
corrupted kernel state.
4 pts

In XINU this is not necessary since all system calls disable interrupts. That is,

when resched() calls ctxsw(), the IF flag in EFLAGS is guaranteed to be 0. The same goes for XINU's interrupt handler where clkdisp disables interrupts before calling clkhandler() which may call resched().
4 pts

When a process is context-switched out, saving its state is completed when ctxsw() executes an instruction to copy ESP to the saved stack pointer field in its process table entry. The next instructions of ctxsw() copy the saved stack pointer of the process to be context-switched back into ESP before restoring the 8 general-purpose registers, EBP, and EFLAGS. Thus when a process is context-switched in, EIP of the process being context-switched out will point to the instruction in ctxsw() that copies the saved stack pointer to ESP. There is no need to save EIP.
5 pts

P2(a) 16 pts
Use a FIFO as ready list with a fixed time slice. Every process gets serviced round-robin and gets equal share of CPU time (in the long run). Scheduling overhead is constant since enqueue/insert uses a pointer to the head of the FIFO list, and dequeue/extract uses a pointer to the end of the list.
4 pts

Under mixed CPU- and I/O-bound processes, I/O-bound processes that frequently block will, by their very nature, receive less CPU time compared to CPU-bound processes. On the other hand, I/O-bound processes benefit from faster response times. Hence a dual goal is to achieve fair sharing of CPU cycles to the extent possible and rewarding I/O-bound processes with faster response times which do not consume much CPU time.
4 pts

When a process depletes its time slice, decrease its priority. When a process blocks before depleting its time slice, increase its priority.
2 pts

I/O-bound process that block before depleting their time slice are rewarding by promoting their priority which improves their responsiveness relative to CPU-bound processes.
2 pts

Yes. For example, in a population of many I/O-bound processes and a few CPU-bound processes, even though each I/O-bound process does not use much CPU time, as a group another I/O-bound process is likely to replace an I/O-bound process when it blocks. Thus as a whole I/O-bound processes may starve the CPU-bound processes.
2 pts

Kernel keeps track of how long a process has been ready and waiting for CPU cycles. If the wait time exceeds a threshold (e.g., 1 second), the process is given a significant priority boost. Hence a process cannot starve for too long.
2 pts

P2(b) 16 pts

One of the semaphores (psem) indicates the amount of free space in the queue. It is initialized to the size of the queue. The other semaphore (csem) represents the amount of data in the queue. It is initialized to 0.
4 pts

wait(psem)
2 pts

signal(csem)
2 pts

The writer blocks when calling wait(psem) with psem equal to 0. This means the queue is full.
2 pts

Protecting a producer/consumer queue with a single counting semaphore to achieve mutual exclusion results in more overhead compared to using two counting semaphores. For example, in the mutual exclusion approach, suppose the writer has written 5 bytes into the buffer and in the midst of its critical section (before calling signal()) is context-switched out. The reader is context-switched in, however, when executing wait() it will block and be context-switched out even though there are 5 bytes of data that it could read. Since context-switch overhead is significant, this leads to significant performance degradation.

3 pts

With two semaphores, when the writer is context-switched out, psem = N - 5 and csem = 5 where N is the total queue size. When the reader is context-switched in and performs wait(csem) it will not block and proceed to read 5 bytes without corrupting the shared data structure. This reduces preventable context-switch overhead.
3 pts

P2(c) 16 pts

uchprio() contains the trap instruction int. Extended in-line assembly is used to pass the two arguments of uchprio() in EBX, ECX and return its value in EAX.
3 pts

_Xint36 saved any registers that needed to be saved, received the two arguments from uchprio(), pushed the arguments on the stack following CDECL, called chprio(), and restored registers that were saved.
3 pts

chprio() performed the actual changing of priority and returned the old priority value to _Xint36.
3 pts

Since the return value from chprio() is passed through EAX by CDECL, if _Xint36 saves EAX using pushal after trapping and restores EAX using popal before returning to uchprio(), the return value from chprio() would be lost.
3 pts

Not save EAX since it did not need to be saved by _Xtrap36.
2 pts

Since XINU always runs in kernel mode, i.e., a process does not switch from user mode to kernel mode upon trapping to the kernel since it is already in kernel mode, there is no need to maintain a separate per-process kernel stack and switch to it upon trapping.
2 pts

Bonus 10 pts

Each process has a potentially difference CPU usage value. With up to n (number of processes) different priority values, using a heap/balanced tree data structure of logarithmic depth is the best that can be done w.r.t. insert/extract overhead.
4 pts

An I/O-bound process that consumed little CPU time, slept for a prolonged period (say 10 minutes), woke up, and then behaves like a CPU-bound process would starve CPU-bound for a prolonged period.
3 pts

Rule (i) prevents a newly created process which has no CPU usage from starving existing processes that have accumulated significant CPU usage. The large gap between CPU usage of new and old processes is eliminated by rule (i). Similarly, the potentially large gap between CPU usage of a I/O-bound processes that sleep for a prolonged period and CPU-bound processes that do not is eliminated (significantly reduced) by rule (ii). In that sense, both (i) and (ii) are priority gap reducing measures.
3 pts