CS354 Midterm Solution, spring 2022

P1(a) 20 pts

Privileged vs. non-privileged instructions.
Kernel mode vs. user mode.
Trap (and untrap) instruction.
Memory protection.
8 pts

Not sufficient as utilizing user run-time stack while running upper
half or lower half kernel code in kernel mode results in compromise
of isolation/protection.
2 pts

Need (per-process) kernel stack to run kernel code in kernel mode.
2 pts

x86 XINU's GDT has three relevant entries: kernel code to run in kernel
mode, kernel data to access in kernel mode, kernel stack to access in
kernel mode.
3 pts

In x86 Linux and Windows, GDT contains 4 relevant entries: kernel code
to run in kernel mode, kernel data to access in kernel mode, user code
to run in user mode, user data to access in user mode. There is no
need to separate stack entries since there is no distinction between
data and stack.
3 pts

XINU's GDT is fundamentally different since XINU does not implement
isolation/protection, i.e., all processes from start until end run in
kernel mode.
2 pts

P1(b) 20 pts

FIFO enqueue: constant since pointer to the last element of the queue
can be used to insert a process.
3 pts

FIFO dequeue: constant since pointer to the first element of the queue
can be used to extract a process.
3 pts

Round-robin is not considered adequate since real-world processes include
I/O-bound processes that inherently require less CPU time than CPU-bound
processes.
3 pts

Modified goal is to provide enhanced responsiveness (i.e., response time)
to I/O-bound processes when they become ready (since many are interactive
processes).
3 pts

UNIX Solaris assigns processes that behave in a I/O-bound manner higher
priority compared but smaller time slice compared to processes that behave
in a CPU-bound manner. This is captured in a table comprised of 60 rows
(60 priority levels).
2 pts

Solaris uses a multi-level feedback queue to implement priority and
time slice updates based on whether a process deplete its time slice or
voluntarily relinquishes a CPU. This data structure is an array of 60
rows, each implementing a FIFO queue for round-robin scheduling of
ready processes of the same priority value.
2 pts

To dequeue, a loop from priority 59 down to 0 is performed to find the
highest priority at which there is at least one ready process. The process
at the front is dequeued (since FIFO) which incurs constant overhead.
In the worst-case, 59 iterations in the loop are performed which is also
constant independent of the number of processes in the multi-level
feedback queue.
2 pts

To enqueue a process of priority k, the k'th row is visited and the
processes added at the end of the FIFO queue which incurs constant overhead.
2 pts

P2(a) 20 pts

When there is a no message, a process making a receive() system call is
context-switched out, i.e., blocks. (When there is a message, receive()
returns immediately with the message.) A blocked process is unblocked
(i.e., inserted into ready list) when a message arrives in the future.
6 pts

No, receive() is only an upper half system call.
6 pts

Yes. A sleeping process is woken up (i.e., made ready) by the clock
handler in the lower half of the kernel. Then the scheduler is invoked
which may schedule the unblocked process depending on its priority.
When the woken up process become current, it continues executing sleemps()
after it was context-switched out. That is, resched() returns to
sleepms(), and sleepms() returns to the function that called it. Hence
sleepms()'s operation depends both on upper and lower half functionalities
of the kernel.
8 pts

P2(b) 20 pts

First software layer: system call wrapper function.
For example, calling the wrapper xresume(), it executes int to trap to the
systel call dispatcher (second software layer).
4 pts

Second software layer: system call dispatcher.
The operand of the int instruction in the wrapper function specifies an
entry in x86 IDT which invokes the system call dispatcher in the kernel's
lower half. The system call dispatcher then calls the kernel function to
carry out the task of resume().
4 pts

Third software layer: kernel function carrying out requested task.
In XINU, this would be the function resume().
4 pts

One, we did not implement user mode, i.e., a process starts and ends its
life in kernel mode.
4 pts

Two, we did not switch to a per-process kernel stack after trapping to
kernel code.
4 pts

P3 20 pts

Case (ii) impacted ctxsw() in that we first restored EBP before restoring EFLAGS.
8 pts

This is needed so that when EFLAGS is restored and IF flag equals 1 in the
new process, an interrupt cannot context-switch in another process before
the new process has completed its context-switch in procedure.
2 pts

create() sets up a new process's stack so that its content gives the illusion
that the process ran earlier and was context-switched out by ctxsw()
(although it never ran).
6 pts

This means, one, pushing initial EBP, EFLAGS, and 8 general purpose
register values onto the new process's stack.
2 pts

Before EBP, EFLAGS, and 8 general purpose register values are pushed the
function pointer of the new process is pushed so that ctxsw(), instead
of returning to resched(), jumps to the first instruction of the new process.
2 pts

Bonus 10 pts

There are so many I/O-bound processes so that even when each I/O-bound process
blocks before depleting its time slice, there is another I/O-bound ready
process to take its place (i.e., to be context-switched in). Thus CPU-bound
processes with lower priority may not get a chance to run.
5 pts

Solaris mitigates starvation by monitoring how long a process has been
waiting in ready to receive CPU time. If this exceeds a threshold (default
1 second), then the process's priority is significantly increased so that
it is likely to be scheduled in the near future.
5 pts