P1(a) 12 pts

Upper half: respond to system calls.
Lower half: respond to interrupts.
4 pts

Upper half: sleepms().
Handles sleep requests.
Lower half: clkhandler().
Handles clock interrupts at every tick..
4 pts

[Other system calls/interrupt handlers are fine too.]

The scheduler is an internal kernel function that is called
by kernel functions in both upper and lower halves. As such, it
may be viewed as lying in-between.
4 pts

[As long as the answer indicates that the scheduler is an internal
kernel function that is called by both halves, it is not necessary
to characterize it as "in-between". Just calling it part of one
half, without explaining that it can be invoked by both, is
incorrect.]]

P1(b) 12 pts

Kernel mode/user mode.
Privileged/non-privileged instructions.
Memory protection.
3 pts

Before executing an instruction, the CPU checks if CPL <= IOPL.
If not, an interrupt is generated that indicates protection
violation which terminates the offending process.
3 pts

In XINU, CPL and IOPL are set to 0 for all processes which
neutralizes the hardware's CPL <= IOPL check.
3 pts

In kernels such as Linux/Windows that provide isolation/protection,
CPL is set to 3 and IOPL to 0 for all user mode processes. This

prevents them from executing privileged instructions.
3 pts

P1(c) 12 pts

Interrupt disabling (e.g., through cli in x86) prevents handling of interrupts (e.g., clock, packet arrival) in a timely manner, especially when the critical section is comprised of many instructions. Counting semaphores, except during the wait() and signal() calls, allow interrupts to preempt the current process.
6 pts

When the critical section code is short.
3 pts

Counting semaphore make use of hardware support in the form of interrupt disabling during wait() and signal() system calls. In this sense, they are not a purely software feature.
3 pts

P2(a) 20 pts

Processes that deplete their quantum are demoted w.r.t. priority. The lower priority is associated with a larger time slice.
4 pts

Processes that voluntarily relinquish the CPU are promoted w.r.t. their priority and the associated quantum is decreased.
4 pts

The above increases the relative priority of processes that voluntarily give up the CPU (i.e., IO-bound) over those that are CPU-bound. This tends to improve equitable sharing of CPU cycles and reward IO-bound processes, when they become ready, with faster response times.
3 pts

To prevent starvation, Solaris monitors how long a ready process has been waiting to receive CPU cycles. If the wait time exceeds a threshold (e.g., 1 second), the process's priority is promoted. Eventually, this leads the process to reach the highest TS priority which, by round-robin, assures CPU cycles will be received.
3 pts

It differed in its policy to assign larger time slices to processes with
higher priorities.
3 pts

Some app process may make consecutive blocking calls that voluntarily
relinquish CPU thus amplifying its priority and increasing its quantum.
Subsequently, it may hog the CPU which prevents lower priority processes
from receiving CPU cycles for a prolonged period.
3 pts

P2(b) 20 pts

1. Save (i.e., push) ebp.
2. Save EFLAGS
3. Save 8 general purpose registers.
4. Save process's stack pointer.

8 pts

After restoring the new process's stack pointer and general purpose registers
(steps 4 and 3), ctxsw() restores ebp first before restoring EFLAGS (steps 1 and 2).
6 pts

Since EFLAGS contains the new process's IF (i.e., interrupt) flag, if IF happens to
be 1 (enabled), then before ebp is restored an interrupt may occur which causes
an interrupt handling routine to run in the context of the new process (it borrows
the new process's context).
3 pts

In XINU, when an interrupt handling routine such as clkhandler() runs, it is
ultimately a function call which saves/restores and uses ebp and esp assuming that
both ebp and esp are those of the interrupted process. esp has been retored by
ctxsw(), hence contains the stack pointer of the new process. However, ebp has not,
i.e., contains the frame pointer of the old process, which results in a kernel bug.
3 pts

P3 24 pts

Call the two user mode processes that share memory p1, p2. Suppose p1 runs first
and it makes a system call to sleep() (a counter part of sleepms() in XINU) in a
kernel that provides isolation/protection. The kernel code that implements sleep()
eventually calls the kernel's scheduler function (e.g., resched() in XINU) which
follows typical caller-callee function call conventions (e.g., CDECL) that pushes
the return address of calling functions into p1's user space run-time stack.

8 pts

[In the above, any other blocking call that triggers rescheduling and context switch will do.]

Suppose the scheduler picks p2 to run next. p2, because it is able to read/write to p1's memory, modifies the last return address of p1's run-time stack to point to user code (i.e., malware). When eventually p1 resumes running, the last kernel function that implemented the context switch will return to the kernel function that called it.
8 pts

However, because p2 modified this address, the code that gets executed upon return is not that of the calling kernel function but user malware. The malware code runs in kernel mode, which means the system has been compromised.
8 pts

[The mechanism used is a variation of the hijacking method of lab2. Only through memory protection (hardware support) and using a separate kernel space run-time stack when executing system calls (software support) can this hijacking problem be prevented.]

Bonus

Two semaphores, call them csem and psem, are initialized to 0 and the size of the buffer (in bytes, packets, or other meaningful unit). csem represents the size of the content stored in the buffer and psem represents how much free space is available. Initially there is nothing in the buffer and, hence, the entire buffer is free.
5 pts

Using a single counting semaphore to regulate access to a producer/consumer buffer through mutual exclusion prevents concurrent access by a producer (i.e., writer) and consumer (i.e., reader), even when concurrent access would not cause a problem because producer and writer are accessing different, non-overlapping parts of the buffer. Using two counting semaphores as discussed allows concurrent access as long as it does not corrupt the shared data structure.
5 pts