P1(a) 10 pts

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

Upper half: process that makes system call.
Lower half: current process whose context is borrowed to run interrupt handling code.
2 pts

The scheduler is invoked by both halves. Hence it can be viewed as residing between the two halves.
2 pts

Process makes sleepms() system call which then calls resched() after putting the process into a sleep queue.
2 pts

Clock interrupt borrows context of current process to run clock interrupt handling code which checks whether current process's time slice has depleted. If so, resched() is called.
2 pts

P1(b) 10 pts

Computation/CPU requirement per compressed video frame (i.e., period) varies by frame (i.e., not constant). Estimating CPU time required is hardware dependent (clock rate of CPU, etc.).
2 pts

System overhead such as context switch overhead, scheduling overhead, etc. need to be accounted for when performing admission control.
2 pts

Kernel must use interrupt disabling sparingly so that interrupts for real-time processes are not delayed.
2 pts

Advantage: admission control allows EDF to schedule more real-time tasks than RMS.
Drawback: higher overhead to dynamically track which real-time task has earliest deadline over time.
2 pts

No. XINU system calls disable interrupts at start and restore them only at the end. For slow system calls, this may delay interrupt processing of real-time processes.
2 pts
P1(c) 10 pts
Constant overhead since FIFO queue with dequeue at the front and enqueue at the tail suffices.
2 pts

I/O-bound processes that often block and thus relinquish CPU voluntarily will receive a smaller share of CPU cycles than CPU-bound processes that hog the CPU and deplete their time slice. This leads to a skew in CPU cycles that favors CPU-bound processes.
4 pts

By giving I/O-bound processes a priority boost relative to CPU-bound processes, their responsiveness is enhanced (helpful for interactive I/O-bound apps) which compensates for their smaller share of CPU cycles.
4 pts

P1(d) 10 pts

B busy waits repeatedly executing tset which consumes CPU cycles until its time slice depletes. Only when the scheduler eventually context switches in A, A completes its critical section and releases the lock, can B enter its critical section.
4 pts

Busy waiting using tset on a uniprocessor system is not meaningful since it only causes B to waste CPU cycles.
3 pts

In a multicore system, assuming there are no other ready processes needing CPU cycles, busy waiting will expend energy (possibly battery power) which is wasteful. However, there is a potential benefit of faster response time. That is, by not relinquishing its CPU and continually probing the tset lock, B is able to enter its critical section faster.
3 pts

P2(a) 12 pts

Pro: user app can perform its own scheduling of its threads without invoking system calls.
Con: if a user space thread makes a blocking system, it will block all other user space threads.
4 pts

Checkpoint the same machine state, i.e., process context, as during a kernel based context switch. For example, in x86 XINU, a number of registers including EFLAGS.
3 pts

Yes.
2 pts
Since the user space scheduler that checkpoints a user space thread is running in user mode, popfl that restores the bits of EFLAGS will not set IF to 0. But this is not an issue since user space threads would not be able to disable interrupts (e.g., cli is a privileged instruction), hence the sensitive instruction popfl not being able to set IF to 0 does not matter.

3 pts

P2(b) 12 pts

A process that wants to receive messages asynchronously (i.e., independent of what the process itself is doing) registers a callback function with the kernel. The kernel invokes the callback function on behalf of the process that registered it when a message arrives.

4 pts

When executing the callback, it is important to preserve isolation/protection.

2 pts

Defer execution of the callback until the receiving process is scheduled next. After context switching in the process and just after returning to user mode, insert a function call/jump to the callback at run-time.

4 pts

Drawback: running callback function is deferred until the receiving process is scheduled next which incurs delay.

2 pts

P2(c) 12 pts

Similarity: each of the guest kernels runs a process at any given time. Hence, context switching from one guest kernel to another is tantamount to switching from one process (running in one guest kernel) to another process (in another guest kernel).

3 pts

Difference: a guest kernel has much more state than a process running in that kernel. For example, states associated with system wide resources which are virtualized by the hypervisor, must be checkpointed across guest kernel switches.

3 pts

Executing cli by the guest kernel running in user mode causes a trap to the hypervisor who runs in kernel mode.

2 pts

The hypervisor does not physically set IF to 0 since that would disable interrupts for other guest kernels. Instead, the hypervisor emulates disabling interrupt by masking interrupts from the guest kernel that executed cli. That is, hardware interrupts are virtualized giving the illusion to each guest kernel owns its interrupt hardware.

4 pts
P3(a) 12 pts

Consider the following sequence of events as determined by scheduling of two concurrent processes A and B: process A acquires semaphore S; process B acquires semaphore T; process B waits on semaphore S; process A waits on semaphore T. Both processes block on semaphores that the other has acquired but not released. Hence, both A and B will block indefinitely, i.e., deadlock.

4 pts

Overhead: cycle detection in resource graph which is linear.
2 pts

Kernels, by default, choose not to provide deadlock detection service (due to its overhead) to user processes. This approach is justified since, by isolation/protection, the adverse impact of deadlocks is limited to apps whose processes deadlock.
4 pts

To prevent kernel processes or user processes running in kernel mode from deadlocking, kernel code may be written so that they follow a total order of all semaphores and acquire all semaphores in that sequence which prevents deadlocks.
2 pts

P3(b) 12 pts

Hardware support: privileged/non-privileged instructions, kernel/user mode (and a means to switch from one mode to another), memory protection.
Software support: per-process kernel stack.
4 pts

CPL is set to 00 for kernel code and 11 for user code. IOPL is set to 00 (01 will work as well). When a process tries to execute a privileged instruction, the hardware checks if CPL <= IOPL. This will only succeed for kernel code which runs in kernel mode.
4 pts

XINU system calls do not trap from user mode to kernel mode. They always run with interrupts disabled. This is not the case for UNIX/Linux/Windows, some of which (esp. slow system calls) must remain interruptable to achieve responsiveness. They incur additional overhead when performing a switch from user mode to kernel mode (e.g., through int or sysenter) and careful checking of all arguments that have been passed for soundness.
4 pts

Bonus 10 pts

No isolation/protection. System calls run with interrupts disabled.
Fixed-priority scheduler.
No multicore support.
Simple 1-word synchronous IPC.
Etc.

8 pts
[No isolation/protection is a must, but any other three will do.]

Isolation/protection is a core property required of multiprogramming operating systems.
[Other justifications, if they are meaningful and well-argued for, are fine.]

2 pts