P1(a) 13 pts

Two processes p1, p2; two binary semaphores s1, s2. p1 has s1 but wants s2.
p2 has s2 but wants s1.
3 pts

Cycle detection in resource graph comprised of nodes (processes or semaphores) and links
(has or wants) requires linear time.
2 pts

Kernels do not consider providing deadlock detection service to user processes their
responsibility. The rationale is that if user processes deadlock the impact is limited
to app processes that deadlock. Beyond some consumption of system resources, there
is no system-wide impact.
3 pts

The impact of deadlock is limited to the processes that deadlock without impacting the
rest of the system.
2 pts

Impose linear (i.e., total) order on semaphores and require all processes to acquire semaphores
in the same order.
3 pts

P1(b) 13 pts

Allocating part of virtual address space to themselves allows user processes to use refer to
kernel addresses in a consistent manner during system calls. Hence when transition from user mode
to kernel, or even between context switches, kernel address references remain valid and need not
be flushed from TLB.
3 pts

In 32-bit x86 Linux: allocate high 1 GB memory for kernel so that a user process can use only
0 - 3 GB of virtual memory address space.
2 pts

Physical memory (3 GB or 4 GB) has no bearing on the usable address space (i.e., virtual address
space) of a process.
2 pts

Meltdown involves a rogue process reading kernel memory using a side channel attack. Targeting
kernel memory for read is possible because the kernel has mapped itself in the user process's
virtual address space.
2 pts

A different page table in kernel mode during system call means that there is no fixed mapping of
the kernel in the virtual address space of user processes. That is, as with user processes where
the same virtual address A may contain different bits depending on which process is running (i.e.,
which page table is used to translate A into a physical address thus permitting aliasing), kernel
address space is not accessible via user page tables.
2 pts

When hardware performs out-of-order instruction execution for optimization, make sure that checking
whether an instruction involves privileged operation is done before executing an instruction.
2 pts

P1(c) 13 pts

Context-borrowing
Pro: lower overhead
Con: cannot make blocking calls
Vice versa in the case of kernel thread implementation
4 pts

The consumer/reader of the shared kernel buffer was not able to keep up with the speed at which
data was being written to the buffer.
3 pts

The reason that the reader was, at times, unable to keep up with pace of the kernel buffer filling
up is because the reader (system call return from read) is an app process that perform compression.
When compression is able to compress an image (in a video stream) by a lot, the resultant compressed
image is small, hence copy overhead to move it around (e.g., to send out as network packets) takes
little time. When an image does not compress much, the resultant linear copy cost is correspondingly
higher during which time reading from the shared kernel buffer slacks off. Hence when an image not
compressed much, kernel buffer builds up, and vice versa.
4 pts

Implementing the bottom half as a kernel process worsens performance due to increased overhead (see
pro/con) and uncertainty introduced by scheduling. That is, a kernel scheduler does not take into
consideration what processes do such as who reads what from whom. Hence when the reader of the shared
kernel buffer falls behind and buffer fills up, that is not a factor considered by the scheduler.
2 pts

P1(d) 13 pts

Memory thrashing is a phenomenon where physical memory is too small relative to memory demand of processes that leads to a high page fault rate.
In more detail, referenced pages are not in memory which requires evicting resident pages. Those pages, in turn, are referenced again which leads to a vicious cycle where the system spends most of its time dealing with
3 pts

Symptoms: high page fault rate, low CPU utilization (i.e., idle process uses up CPU), frequent disk I/O
3 pts

Increasing CPU speed does not help since CPU is not the bottleneck.
Increasing memory helps since supply is increased which reduces page fault rate.
Increasing disk space does not help since disk space is not the bottleneck.
3 pts

For page fault rate to increase, a process has to run so that it can trigger page faults. During thrashing, processes block on disk I/O and are not running. Thus there is a self-limiting which bounds page fault rate.
2 pts

There are few good options. One is to swap out an entire process — one that consumes significant memory (elephant) — which frees up significant memory for other processes. Another is to reallocate some of kernel memory (i.e., buffer space) to user space.
2 pts

P2(a) 16 pts

Hardware buffer: when interrupt is raised, top half borrows current process’s context to read/copy data from hardware buffer to kernel buffer. The process performing the operation is the consumer/reader of the hardware buffer.
2 pts

Kernel buffer: The producer/writer is the process running the top half. The consumer/reader of the kernel buffer is the app process that performed a read system call and ran the kernel's upper half.
2 pts

User buffer: The producer/writer is the app process that performed a read system call that ran the kernel's upper half which copied data from kernel buffer to user buffer.
2 pts

The hardware buffer does not need protection since it is being read by the top half when an interrupt is raised.
2 pts

The kernel buffer needs protection using counting semaphore (mutual exclusion via binary semaphore or interrupt disabling could be used but is not considered an effective solution) so that concurrent read/write operations by lower and upper halves do not corrupt it.
2 pts

User buffer needs protection if read is performed asynchronously using callback function. Otherwise, a synchronous read system call serializes read/write access to user buffer.
2 pts

In read, the shared kernel buffer is guarded by a counting semaphore that is initialized to 0 and represents the amount of data (in some unit) present in the buffer. Since the lower half cannot be block when context-borrowing, it uses signal() which is nonblocking to increment the counting semaphore. The upper half, which can block, uses wait() to block until data is present. In write, the counting semaphore represents free space in the buffer so that the lower half calls signal() before writing to the buffer and the upper half calls signal() before writing to the buffer. The upper half, after writing to the buffer, generates a software interrupt which causes the lower half to run. Hence, even though the lower half makes a blocking wait() call before reading from the kernel buffer, since it only runs after the upper half has written to the buffer and generates a software interrupt, wait() is guaranteed not to block.
4 pts

P2(b) 16 pts

The setup with backing store is fundamentally different as a page that requires reading from backing store results in context switch.
3 pts

Overall structure: when backing store is activated with a server running on a frontend machine, a process (rdsprocess) is created on a backend machine which actually handles interaction with the frontend backing store through network communication. A page faulting process interacts with rdsprocess to receive an evicted page from the backing store server.
4 pts
The read_bs() (and other function calls in paging/) that is invoked by the page fault handler running in the context of the process that page faulted ultimately performs a receive() IPC call. read_bs() calls read() on device RDISK which through XINU's device switch table leads to calling rdsread() in device/rds. rdsread() then calls receive() which returns a pointer to a message buffer. receive() is a blocking call and the sender process, rdsprocess, informs the page faulting process after it has communicated with the backing store server and received the requested page.

6 pts

After the page faulting process has been context-switched out due to calling receive() which, in turn, calls resched(), rdsprocess interacts with the backing store server to fetch the missing page. Sending a message to the page faulting process unblocks it from receive() which allows resched() to eventually run it in the future. At that point, the page fault handler uses the pointer returned by receive() to access the content of the missing page, updates kernel data structures and returns using iret. This restarts the instruction whose memory reference triggered the page fault.

3 pts

P3 16 pts

Tickless:
Pro: Clock interrupt is raised only when there are events to process. Hence saves energy/battery power which is especially relevant for mobile devices.
Con: When the number of events to process is dense (e.g., tens of events per 1 millisecond), programming an interval timer to interrupt after several microseconds incurs significant interrupt processing overhead.

Tickful:
The opposite of tickless.

4 pts

Delta list update in lower half: constant (only the front element needs to be decremented).
1 pts

Delta list enqueue: linear since an event may end up at the end of the delta list.
1 pts

Delta list dequeue: in general (i.e., worst-case) linear since all events may have the same expiration time. If the expiration times are different then constant overhead may be achievable.
1 pts

A unified event queue manages not just sleep events of other timing related events such as timers/alarms, time slice expiration, time-of-day updates, etc., using a common linked list.
1 pts

It simplifies event processing since a single list can be inspected to determine if it is time to take actions such as waking up a sleeping process, raising an alarm signal, preempting a process that has used up its time slice, etc.
1 pts

Other events: time-of-day updates, bound on wall clock time usage by a process (similar to XCPU), etc.
1 pts

Instead of configuring the system timer to go off every 1 msec (tick), the clock interrupt handling code will program the interval timer to go off at a designated time. The designated time is obtained from a unified delta list that contains all events to be processed in the future. They include sleep events, alarm events, time slice expiration events, etc. Thus all parts of XINU that relate to the events in the unified event queue must be updated. The front element of the delta list will specify with what value to program the interval timer when the clock interrupt handler is invoked.
3 pts

No fundamental change in kernel overhead since multiple event queues are integrated into one.
1 pts

In the tickless design, specify a minimum time interval (e.g., 0.05 msec) so that events below this resolution are round up the minimum. The benefit of tickless design — no unnecessary clock interrupt processing — can be combined with not overly dense timer events by grouping events with time resolution less than the minimum and handling them as a block.
2 pts

Bonus Problem (10 pts)

Systems:
Incorporate system overhead such as context-switch overhead when making scheduling decisions.
2 pts
Incorporate timer inaccuracy when making scheduling decisions.
Implement kernel design so that interrupt disabling is minimized to facilitate responsiveness.

Other:

Find effective means to estimate the computation time requirement of a process per period which is hardware dependent.

Etc.