CS354 Final Solution, spring 2018

P1(a) 12 pts

Most files are small, a few are large; or 90/10 rule (90% of files
are small and take up 10% of space, 10% of files are large and take
up 90% of the space).
4 pts

Although traditional/UNIX file systems use indirect (single, double,
triple) pointers to index data blocks using a tree of logarithmic
depth for large files, a number of direct pointers (10 in the pdf
slides) are kept in the i-node to facilitate direct access to data
blocks of small files without traversing the tree.
4 pts

XINU's file system uses a linked list of index blocks each containing
a fixed number of pointers to data blocks. As such, the overhead of
finding a data block is linear in the size of the file.
4 pts

P1(b) 12 pts

Belady: evict the page that will not be accessed for the longest time
(i.e., farthest in the future).
3 pts

LRU selects a page that has not been accessed for the longest time
(i.e., farthest in the past) to evict. Under locality of reference,
pages in the recent past and near future will significantly overlap.
Hence both LRU and Belady tend to evict pages that do not satisfy
locality of reference. In this sense, LRU approximates Belady using
information available from the past.
2 pts

Global clock uses periodic sweeping across pages to give modified
pages two chances before being selected for eviction and pages that
have been only read a single chance. Global clock can be viewed as
approximating LRU by using the time to make a full sweep as a measure
of recency. Pages that have not been accessed (read or written)
during this time are marked for eviction.
2 pts

Assuming persistent storage that is significantly slower than main
memory such as hard disks, the time needed to load a missing page
into RAM is very slow. Hence, to not idle a CPU for an extended
period, the process that page faulted is context switched out and
a process that can make use of the CPU context switched in. Scheduling
and context switching take many instructions to complete and are
the purview of software (i.e., kernel).
3 pts

To reduce the overhead/delay incurred when a page fault occurs, it
is preferable to evict pages via periodic sweeping (asynchronous to
page faults) so that some number of free frames are available. By
doing so, when a page fault occurs, only reading in of a missing page
is required (but not determining a page to evict or writing back a
page if it has been modified).
2 pts

P1(c) 12 pts

Two processes p1 and p2, two semaphores s1 and s2.
Sequence of events: p1 runs and acquires s1; p2 runs and acquires s2;
p1 runs and tries to acquire s2 which causes it to block; p2 runs and
tries to acquire s1 which causes it to block.
3 pts

Cycle detection in resource graph comprised of nodes (processes and
semaphores) and directed links (has-semaphore and want-semaphore edges).
3 pts

Cycle detection takes linear time (e.g., breath-first expansion).

3 pts

The consequence of deadlock is limited to processes that deadlock, i.e., applications that have a deadlock problem. Outside of consuming some system resources (e.g., process table entries, swap space), the impact does not spread to the rest of the system which is in keeping with the principle of preserving isolation/protection.
3 pts

P1(d) 12 pts

In memory thrashing, a memory-intensive process (or processes) requires memory (i.e., number of pages) that far exceeds physical memory. As a consequence, even with locality reference, pages that have been recently evicted are needed again in the near future which leads to a cycle of elevated page faults.
3 pts

Main symptoms: high page fault rate (also disk I/O rate), low CPU utilization (i.e., null/idle process keeps running on CPU).
3 pts

Increasing CPU power does not help since during thrashing a CPU is idle most of the time (unless there are unrelated CPU-intensive processes to run) waiting for disk I/O to complete to load a missing page.
3 pts

Increasing memory helps since this will decrease the likelihood of a page fault. If memory demand is still excessive compared to physical memory, this will serve to delay the onset of thrashing.
3 pts

P1(e) 12 pts

In XINU where upper and lower halves run with interrupts disabled, the design goal was to insert run-time hooks so that a callback function was executed in the context of the process that registered the callback function after it resumes and just after interrupts are restored/enabled.
4 pts

If the process has low priority, it may take a while for the scheduler to select this process to run. Since the callback function is executed in the context of the process that registered it, actual message receive by the process may be significantly delayed.
4 pts

There is no fundamental difference with XINU. Under the assumption that priority is low, significant delay may be incurred.
[One difference is that in Solaris I/O-bound process tend to get a priority boost so the assumption of low priority may not hold.]
4 pts

P2(a) 20 pts

Lower half kernel chores triggered by device interrupts need to be promptly completed. Some devices inherently require more processing by the lower half than others. By dividing the lower half into top (fast) and bottom (less fast) halves, the top half performs urgent chores that don't take much time such as copying bits from device buffer to kernel memory with interrupts disabled (to prevent data structure corruption). The bottom half performs slower chores with interrupts enabled so that when future interrupts are raised they can preempt the bottom half so that responsiveness is preserved.
4 pts

Example: TCP/IP packet processing. Packets are copied by top half (assuming no DMA support) from device buffer to kernel memory by top half. Further processing is required by the bottom half to strip headers, calculate checksums, etc. to ready the payload of TCP for copying to user space buffer.
2 pts

Bottom half options:
(i) Borrow context of the current process.
(ii) Use a separate kernel process/thread to execute bottom half code.

4 pts

Pro of bottom half: lower overhead than context switching to a kernel process.
Con of bottom half: cannot make blocking call.
Vice versa for kernel process implementation.
4 pts

In isochronous mode used for video streaming, an interrupt is trigged every 125 microsecond when a packet containing part of image data arrives. Without special hardware support (i.e., DMA), a CPU would be overwhelmed executing lower half code to service interrupts. DMA shields the CPU from interrupts by performing data copy from device buffer to memory on its own which allows the CPU to perform other tasks. After a certain number interrupts stemming from streaming packets have been copied does the DMA let the next interrupt through so that the CPU runs its lower half which instructs the DMA what to do next (usually to handle the next N packet interrupts). Thus interrupt load of the CPU is significantly reduced.
4 pts

The top half with DMA does not perform data copy from device buffer to memory since this is done by the DMA. Instead, it instructs the DMA what to do which includes where in memory to copy data and how much.
2 pts

P2(b) 20 pts

We discussed using counting semaphores to achieve sharing of producer/consumer queues for IPC between a writer/producer and reader/consumer (i.e., two processes).
2 pts

When performance device I/O using read(), a process p1 makes a read() system call which is handled by the upper half of the kernel. In general, data requested by read() may not be available in which case p1 is context switched out and another process p2 is context switched in. Suppose while p2 is running, data arrives at a device interface which raises an interrupt.
2 pts

The lower half of the kernel (ignoring the details of top and bottom halves) copies data from device buffer to the kernel FIFO queue that is shared between upper and lower halves of the kernel. That is, it acts as the writer/producer of the shared FIFO queue.
4 pts

The lower half then readies the p1 which was blocked. Key point: the lower half kernel code that writes to the shared FIFO queue is process p2.
4 pts

At some future time, the scheduler selects p1 to run. p1 resumes where it left off -- in the upper half of the kernel performing the read() system call -- which entails copying the data from the shared FIFO queue to a user buffer whose pointer is provided by read(). That is, the upper half is the reader/consumer of the shared kernel buffer and writer/producer of the user buffer.
4 pts

The process that performs said reading is p1. Hence in this example of read() device I/O, process p2 which executes the code of the lower half writes to the kernel FIFO queue and process p1 which executes the code of the upper half reads from the kernel FIFO queue. That is, at the end of the day, two processes are communicating using a shared buffer and the counting semaphore techniques of IPC carry over.
4 pts

Bonus 10 pts

Tickful: a fixed tick value which determines a periodic interval at which system timer clock interrupts are raised.
Tickless: no such fixed tick value, i.e., tick can vary over time.
4 pts

If a mobile device is not actively used, it may not have enough future events to handle. In a tickful kernel, e.g., with tick value 1msec (as in XINU), the mobile would repeatedly run lower half code every 1msec for no useful purpose which expends energy and reduces battery charge. By using a tickless kernel, system timer clock interrupts are raised only when there is a chore to be performed by the kernel's lower half.
3 pts

For sleep and time slice events, manage them using a unified delta list. In the case of

time slice management, XINU uses a global variable preempt to decrement it by 1msec each
time the clock interrupt handler runs. Instead, insert into the unified delta list when
a process is context switched in an element that specifies that it's a time slice event
with the time value (i.e., key of the delta list) being the initial quantum. All the other
elements in the unified delta list are sleep elements. XINU programs an interval timer
to raise an interrupt after X msec specified in the first member of the delta list. The
first clock interrupt occurs after X msec at which time XINU processes the delta list element
at the head (and any element following it with 0 offset). Then XINU programs the interval
timer to raise an interrupt after Y msec where Y is the positive time offset of the next
delta list element.
3 pts