CS354 Final Solution, spring 2019

P1(a) 13 pts

XINU uses a linked list of index blocks to store pointers to data blocks. UNIX uses a tree structure to store pointers to data blocks.
4 pts

XINU: linked list leads to linear overhead for search.
UNIX: balanced tree structure leads to logarithmic overhead.
4 pts

Since most files are small, direct pointers allow constant search overhead to identify relevant data blocks of most files. Larger files are handled by indirect, doubly indirectly, triply indirect pointers.
3 pts

Since most files are small, a disk file system spends much of its time spinning disks to locate relevant data sectors. Hence the high seek overhead degrades disk throughput.
2 pts

P1(b) 13 pts

Tickful pro: when events are dense -- many events per unit time (e.g., msec) to process by a kernel -- then tickful can lead to less overhead. For example, if tick is 1 msec and 100 events (e.g., alarms, waking from sleep) have to be handled 10 microseconds apart, all 100 events will by handled by a single clock interrupt.
2 pts

Tickful con: if events are sparse and there are no relevant events to process for the next 100 msec, a tickful kernel with tick 1 msec will still run the interrupt handler 100 times which wastes CPU cycles and energy.
2 pts

Vice versa for tickless kernel.
2 pts

Arnold: tickless. Since emails and texts are slow apps that don't generate dense events, a tickless kernel may be more appropriate to reduce battery consumption.
2 pts

Mira: tickful. Since video chat and on-line video/audio streaming generate dense events, a tickful kernel may be more suited to reduce interrupt overhead.
2 pts

DMA.
1 pts

Without DMA, the CPU(s) would be swamped by interrupts to handle video/audio data arriving at a very fast pace which reduces CPU cycles available to execute app processes.
2 pts

P1(c) 13 pts

Internal fragmentation: suppose a minimum (and fixed) memory block size is 4 KB. Even if only 1 byte of memory is requested through a dynamic memory allocation system call, 4 KB is allocation. Most of the allocated memory (4 KB - 1 B) is wasted.
2 pts

External fragmentation: 500 MB of main memory is free in total. However, the largest contiguous memory area is 30 MB. A process requests 50 MB of memory. Even though total free memory is plentiful the system call will fail since the largest contiguous area is 30 MB.
2 pts

Indirection (or table look-up).
2 pts

Caching: Keep a address translation table, called page table, that maps page numbers to frame numbers in main memory (which is viewed as a cache). Keep part of the page table that is frequently accessed in a fast cache called TLB.
2 pts

Every machine instruction that references memory needs to undergo virtual memory to physical memory address translation. Doing the translation in software would incur exorbitant overhead.
2 pts

TLB misses can be handled by hardware or kernel. It depends.
1 pt

2^{20} which is about a million (1024 x 1024) entries.
1 pt

Most processes do not consume much memory. Hence they only utilize a small portion of their page tables. As a result, using a 2-level page table will significantly reduce the space needed to perform address translation.
1 pt

P1(d) 13 pts

Insert: use the priority (say k) of the process to be inserted as the key to go to the k'th array element. One of the two pointers will point to the end of the FIFO list. Insert process at the end of the list.
3 pts

Extract: loop down from highest priority (59) down to lowest priority (0). For the first array element that is not empty (i.e., ready processes of that priority exist), use one of the two pointers that points to the front of the list to extract the process to run next.
3 pts

Insert overhead: indexing array (of struct) by priority takes constant time. Using end-of-FIFO list pointer to insert process takes constant time.
2 pts

Extract overhead: looping from 59 down to 0 (in the worst case) takes constant time since the total priority range 60 is constant. Using the front-of-the-FIFO list pointer to extract the first process takes constant time.
2 pts

In Linux CFS, CPU usage is the main criterion for determining a process's priority. Unlike 60 levels of priority in UNIX Solaris, CPU usage has a wide range (e.g., 2^{32} if priorities are unsigned int). The best that can be done is use a heap/balanced tree structure which incurs logarithmic overhead.
3 pts

P2(a) 16 pts

The kernel buffer is managed using a counting semaphore, say s, that is initialized to 0. Semaphore s represents the number of data items (say fixed size for simplicity) in the buffer which is empty at the start.
4 pts

When the receiver process P1 makes a blocking read() system call, the kernel code for read() performs wait(s). If the buffer is empty, i.e., s = 0, P1 blocks and is context-switched out. P1 waits in the semaphore queue of s. If the buffer is non-empty (s > 0), wait(s) does not block, the kernel code reads the next data item from the kernel buffer and copies it to user space buffer. P1 is the consumer or reader of the kernel buffer.
6 pts

When data arrives at the hardware buffer and triggers an interrupt, the context of the current proess P2 is borrowed to run the lower half code of the kernel. The lower half code copies the data item from hardware buffer to kernel buffer and performs signal(s).
[More precisely, the lower half checks that the kernel buffer is not full before performing the copy and calling signal(s). This detail may be ignored.]
The lower half which runs as process P2 is the producer or writer of the shared kernel buffer.
6 pts

P2(b) 16 pts

The caches used by the process to be context-switched out including TLB, L1, and L2, in general, must be flushed. When the process is context-switched in again, the aforementioned caches will encounter misses and need to be loaded again. That results in significant overhead.
2 pts

The content of ptbr (page table base register in x86) which points to the address in memory where the current process's page table begins must be saved.
2 pts

In a page fault, a process accesses an address its virtual memory but the page wherein the address lies has been evicted to disk (or SSD). The hardware generates a synchronous interrupt, i.e., exception indicating that the needed page is not in main memory (or RAM).
2 pts

Kernel handles page faults.
2 pts

Since disk (and to some extent SSD) is significantly (ballpark three orders of magnitude) slower than RAM speed, it would waste CPU cycles for the hardware to busy wait until disk I/O is completed and the missing page read into memory. Instead, the page fault causes the process that caused the fault to be context-switched out in a blocked state until the missing page has been read to RAM. When the missing page has been read in, the page faulting process is made ready and will resume when the scheduler selects it to run in the future.
2 pts

In memory thrashing, the total memory demand by processes significantly exceeds physical memory

so that a large number of pages are swapped out to disk. Despite locality of reference, a page that has been evicted to make room for a page on disk to be brought into RAM, is needed again in the near future which requires evicting another page to make room. The latter, in turn, is needed again in the near future resulting in a vicious cycle of writing to/reading from disk which slows down the system.
2 pts

The CPU(s) will be underutilized (e.g., null/idle process will be constantly running) since app processes are blocked waiting on disk I/O to complete.
2 pts

Increasing CPU speed does not alleviate the problem since the bottleneck is disk I/O and CPU is being underutilized during memory thrashing.
2 pts

P2(c) 16 pts

[This problem only considers case (ii) where a process that registered a callback function has been context-switched out because it depleted its time slice.]

A process that registered a callback function for message send is context-switched out because it depleted its time slice. This is detected by clkhandler() which is called by clkdisp() when a clock interrupt is raised. clkdisp() calls resched() which calls ctxsw() to context-switch out the process.
2 pts

When the context-switched out process receives a message, the return address of the receiver process in the stack that is used by iret in clkdisp when the receiver resumes running (reverse sequence: ctxsw() returns to resched() returns to clkhandler() returns to clkdisp() which jumps back to where the process left off when the clock interrupt occurred) must be changed to point to xruncb_lh().
4 pts

This is done in clkdisp() before executing iret by adding assembly code or calling a C function that performs this ROP manipulation. Before modifying the return address of iret, the original return address is remembered (say in a global variable) so that it can be restored later.
4 pts

When clkdisp() executes iret, the receiver process switches to user context (logically since XINU processes always run in kernel mode).
3 pts

The lower half wrapper function xruncb_lh() runs in user context. This is in keeping with isolation/protection since xruncb_lh() just calls the registered callback function which is user code. After the callback function returns, xruncb_lh() adjusts its return address in the stack to the original return address of clkdisp(). It executes ret to return to the original return address of clkdisp().
3 pts

Bonus 10 pts

Linear
2 pts

When user processes deadlock, they consume some kernel resources but the impact of the deadlock is limited to the app processes who have deadlocked. The rest of the system is not affected which is what isolation/protection is trying to achieve.
6 pts

Order semaphores in a specific manner. All processes that invoke the semaphores (using wait()) must do so by following this specific order. Not the order by which they may need to acquire semaphores which could be different.
2 pts