P1(a) 13 pts

Enqueue overhead:
Use priority i as index to jump to array element i. Use pointer to end of FIFO list to create/insert new list element.
4 pts

Dequeue overhead:
Loop downward from maximum priority K-1 finding first array element that has non-empty FIFO list (or reach 0). Use pointer to beginning of FIFO list to dequeue list element. Looping over K is viewed as constant since time complexity is measured w.r.t. process count.
5 pts

Fair scheduling such as CFS uses a quantity that relates to CPU usage (actual or virtual) as the criterion for priority. Since every process may, in general, have different CPU usage (hence priority), picking a process with least CPU usage requires at least logarithmic time.
4 pts

P1(b) 13 pts

Main differences using int software interrupt:
x86 hardware consults IDT to find address of system call dispatcher code (i.e., Xtrap33) to jump to.
2 pts

int pushes EFLAGS, CS, EIP onto stack. iret pops EFLAGS, CS, EIP from stack.
1 pts

EAX is used to pass system call number to dispatcher. Dispatcher makes C function call to getpid().
2 pts

Return value from getpid() in EAX must be propagated back through Xtrap33 and igetpid() to function that called igetpid().
2 pts

No mode switch. All of XINU runs in kernel mode.
2 pts

No stack switch. Since XINU runs in kernel mode, per-process user stack can be used to run kernel code.
2 pts

Software interrupt incurred (significantly) more overhead as indicated by the additional steps above. Ignoring the need to save/restore registers which is common to both implementations, a C function call to getpid() is just pushing EIP and jumping to getpid.
2 pts

P1(c) 13 pts

\[ \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\text{RMS} & p1 & p1 & p2 & p2 & p2 & p1 & p1 \\
\end{array} \]
\[ \text{deadline violation of p2} \]
3 pts

\[ \begin{array}{cccccccc}
\text{EDF} & p1 & p1 & p2 & p2 & p2 & p1 & \ldots \\
3 pts \end{array} \]

EDF yes. RMS incurs deadline violation of p2 at time 7.
1 pts

Utilization \( \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \leq 1 \) implies admitted by EDF.
2 pts

Utilization \( \frac{2}{5} + \frac{4}{7} = \frac{34}{35} > 0.83 \) (approximate value of \( 2(2^{(1/2)}-1) \)) implies not admitted by RMS.
2 pts
Considering kernel overhead, 34/35 (high 90 percentile) may be too tight and both tasks should not be admitted.

2 pts

P1(d) 13 pts

In XINU, except for per-process run-time stacks, all code and data are shared which corresponds to the default heavy-weight/light-weight process model discussed in class. Here we are ignoring the fact that the XINU kernel is a collection of functions (upper half/lower half), not a process.

2 pts

EBP, EFLAGS, 8 general purpose registers, address of top of stack in process table

4 pts

In ctxsw() saving the state of a process is completed when ESP is saved in the saved stack pointer field (prstkptr) in its process table entry. The next instruction (EIP) proceeds to retrieve the saved stack pointer of the new process to be context switched in. Hence, when the old process is eventually scheduled and becomes new, the code to execute to retrieve its saved stack pointer is pointed to by EIP of the current process that is being context switched out. Hence, there is no need to explicitly save EIP.

3 pts

Save process state in the process table by creating additional fields (similar to saved stack pointer field).

1 pts

popfl restores the IF flag of EFLAGS which, in general, could be 1 which would re-enable interrupts before EBP has been restored which can lead to an inconsistent state.

2 pts

It was not necessary since in our version of XINU IF is guaranteed to be 0 before ctxsw() is called.

1 pts

P2(a) 16 pts

Synchronous nonblocking receive implies that an app process is employing polling to check if a message has arrived. This can waste CPU cycles. Asynchronous IPC with callback function perform receive only if there is a message waiting to be received.

3 pts

In kernels where the number of received messages is not exported to user processes, it is up to the app programmer to make sure that all queued messages have been read by using non-blocking message receive.

3 pts

To preserve isolation/protection, the callback function registered by a process must not be executed in kernel mode nor by borrowing the context of another process.

4 pts

If the current process is the one that registered the callback function, interrupt the current process to execute the callback function in its context in user mode. If not, wait until the scheduler decides to schedule the process that registered the callback function and interrupt the process after it has become current and returned to user mode to execute the callback function.

3 pts

In XINU where all processes run in kernel mode and code/data (but for stack) is shared, run the callback function in the context of the current process.

3 pts

P2(b) 16 pts

XINU uses flat addressing where the base address of all segments is set to 0 and the limit to a maximum address that no segment is allowed to exceed (i.e., effectively infinity).

4 pts

Four entries are needed at a minimum: kernel mode code segment descriptor, kernel mode data segment descriptor, user mode code segment descriptor, user mode data segment descriptor.

4 pts
In XINU, (ignoring a special entry) three entries are used: kernel mode code segment descriptor, kernel mode data segment descriptor, kernel mode stack segment descriptor.  
3 pts

The kernel stack segment descriptor is not necessary since the kernel mode data segment descriptor can be used for SS.  
2 pts

XINU's ctxsw() does not need to save CS, DS, and SS register values since all processes run in kernel mode which implies CS, DS, and SS are shared.  
2 pts

In kernels such as Linux/Windows that implement isolation/protection, if CS, DS, and SS change then they need to be saved.  
1 pts

P3 16 pts

App processes that make very few system calls that trap to the hypervisor incur minimal trap-and-emulation overhead.  
2 pts

Overhead becomes significant if apps make frequent system calls.  
2 pts

cli may be executed by a process when it makes a system call and the system call code (which is expected to be run in kernel mode) contains cli. cli may also be executed by a process in user mode since nothing prevents a programmer to write such code.  
3 pts

The hypervisor needs to ensure that the IF flag is virtualized and that cli is correctly emulated depending on whether a process in a guest kernel executes cli directly in user mode or its guest OS executes cli when a process makes a system call. In the former, correct emulation implies that the process raises an exception (cli cannot be executed in user mode) which, by default, causes it to be terminated. In the latter, the hypervisor needs to set the virtual IF flag for the guest OS to 0. To know which action to take when cli traps to the hypervisor it needs to track whether a process in the guest kernel "thinks" it is in user mode or kernel mode.  
4 pts

popf loads the EFLAGS register from a stack which affects the value of the IF flag. If a process runs in user mode, even if the bit value in the stack for the IF flag is 0, the hardware set IF to 1 so that user mode processes cannot disable external interrupts. If a process runs in kernel mode, whatever value for the IF bit is specified in the stack is used to load EFLAGS. Hence the resultant content of EFLAGS can vary depending on whether a process in a guest kernel runs in user mode or kernel mode. Since popf does not trap to the hypervisor, it has no way of knowing that popf is being executed by a process in the guest kernel and take appropriate emulation action.  
3 pts

In a simple redesign of x86, all sensitive instructions are made to trap. This ensures that the hypervisor can take appropriate trap-and-emulate actions.  
2 pts

Bonus Problem (10 pts)

Aging keeps track of how long a ready process has been waiting to receive CPU cycles.  
3 pts

The dynamic priority promotion/demotion actions of the Solaris TS scheduler are reasonable heuristic actions to affect "fairness" but there is no guarantee that one or more processes will not be starved of CPU cycles for a prolonged period.  
4 pts

In lab2, one of the problems was to monitor the waiting time of processes which is related to aging.  
3 pts