```
CS354 Midterm Solution, fall 2023

P1(a) 20 pts

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

sleepms().
// Others covered thus far: kill(), receive(), recvtime(), suspend(), yield().
4 pts

clkhandler().
4 pts

Current (i.e., old) process is strictly highest priority process.
// Assumes clkhandler() called resched() because of time slice depletion.
4 pts

P1(b) 20 pts

abc() jumps to userret() (macro INITRET which is userret() has been set up
as return address of abc() by create()) which calls kill() to terminate the
process that executed abc().
4 pts

create() sets of up the stack of abc() so that it appears to have run before
and context-switched out by calling ctxsw().
4 pts

First, create() sets up the return address to be abc, not the address of
resched() after calling ctxsw().
3 pts

Second, create() sets up the stack to contain saved EBP, EFLAGS, and 8 general
purpose registers.
6 pts

Third, the IF bit of the saved EFLAGS value is set to 1.
// So that when ctxsw() jumps to user code abc() the process runs with
// interrupts enabled.
3 pts

P2(a) 20 pts

A multilevel feedback queue is a data structure that is a (1-D) array
of elements where each array element implements a FIFO queue. The length
of the array is the number of priority levels.
6 pts

Enqueue: the priority of the ready process to be inserted serves as the
index of the 1-D array. Each array elements contains a pointer to the
end of the FIFO queue. The ready process is added to the end of the list.
Both operations incur constant overhead.
5 pts

Dequeue: loop from highest priority (59) down to lowest priority (0) to
find first array element that is not empty. Since each array element
implements a FIFO queue, use the pointer that points to the ready process
at the front to extract it. Both operations take constant time (independent
of the number of ready processes in the multi-level feedback queue).
5 pts

Linux CFS uses a quantity related to CPU usage as a process's priority to
select one who has received least CPU usage to run next. Since CPU usage
is not bounded by a fixed priority range (e.g., 0-59 in Solaris UNIX), a
balanced tree data structure with logarithmic depth is used for insertion
and extraction.
4 pts

P2(b) 20 pts

Blocking means that when receive() is called and there is no message in
```

the receiver's buffer the scheduler is called to context-switch out the
current process.
4 pts

send() is nonblocking which means that if send() fails (because the
receiver's buffer is full) it returns immediately.
2 pts

sendb() blocks by calling resched() if the receiver's buffer is full.
4 pts

Before calling resched(), sendb() must temporarily store its message. This
could be in a new process table field of the sender process or a separate
message buffer.
5 pts

The sending process who blocks must be put in a (by default) FIFO queue
associated with the receiver. Multiple senders that block when attempting to
send to the same receiver would be placed in the FIFO queue of the receiver.
// In some cases a priority queue may be used to favor processes of higher
// priority.
5 pts

P3 20 pts

System call wrapper. Executes trap instruction int to trap to kernel code,
communicates with system call dispatcher (using extended inline assembly).
4 pts

System call dispatcher. Acts as gateway to system calls. Disables interrupts,
determines which system call was called, switches stacks, sets up content
of kernel stack, calls upper half kernel function.
4 pts

Upper half kernel function. Internal upper half kernel function to carry
out the requested task.
4 pts

We reused legacy XINU system calls as upper half kernel function.
2 pts

Clobber register list obviated the need to save/restore registers in system
call dispatcher code.
2 pts

We copied the values of EFLAGS, CS, EIP to kernel stack for safe keeping
since the user stack, in general, may be modified by user processes. In
particular, this may lead to corruption of the return address.
4 pts

Bonus 10 pts

x86 Linux and Windows set up 4 entries: kernel code segment, kernel
data/stack segment, user code segment, user data/stack segment.
4 pts

XINU sets up 3 entries: kernel code segment, kernel data segment, kernel
stack segment.
2 pts

CS register is changed to point to kernel code segment of GDT (before it
was pointing to the user code segment of GDT).
2 pts

CS does not change since XINU processes always run in kernel mode.
2 pts