CS354 Midterm Solution, spring 2018

P1(a) 15 pts

Provide reliability or robustness to computing systems (against
software bugs and malicious program behavior).
4 pts

User mode/kernel mode
Privileged/non-privileged instructions
Trap instruction (to enter kernel mode)
Memory protection
4 pts

Per-process kernel stack.
3 pts

XINU chooses not to provide isolation/protection. (The entire
system runs in kernel mode on x86 backends where code and data are
shared. Per-process stack is private but not protected.)
4 pts

P1(b) 15 pts

Interrupt disabling
Tset instruction
6 pts

Interrupt disabling has a negative effect on system performance
since important kernel events (clock ticks, packet arrivals, etc.)
are ignored while critical section code is executing.
Tset is preferable since although it wastes CPU cycles due to
busy waiting, it does not result in the system being put in lockdown
due to interrupt disabling.
6 pts

Yes. A producer/consumer queue is a shared data structure. As
such, techniques that provide mutual exclusion will shield the
shared data structure from being accessed concurrently and hence
from potential corruption. It is sufficient (but not necessary).
3 pts

P1(c) 15 pts

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

Upper half: sleepms() calls resched().
Lower half: clock interrupt handler (e.g., clkhandler() in XINU)
calls resched().
7 pts

P1(d) 15 pts

Interrupt disabling.
Interrupt restoring/reenabling.
4 pts

These actions are performed to achieve mutual exclusion while XINU's
upper half executes.
4 pts

Since interrupts are disabled, important kernel events such as clock
ticks and packet arrivals may be missed. In the case of sleepms(),
inserting a process into a sleep queue may take a while (linear overhead).

During this time, CPU usage monitoring and time slice management by
the lower half's clock interrupt handler are disabled.
4 pts

System calls perform sanity checks in the form of detailed verification
that arguments are being correctly used.
3 pts

P2(a) 20 pts

Save/push BP (more precisely EBP).
Save FLAGS.
Save 8 registers (including AX, BX, CX, DX, SP, BP).
Save SP in the saved stack pointer field (prstkptr) of the current/old process.
8 pts

Since XINU stored the new process's SP in the saved stack pointer field of its
process table entry, it can use this address to access the information saved in
the process's stack.
6 pts

resched() is written in C and compiled by gcc. ctxws() is written in assembly.
Therefore when writing the code of ctxsw() we need to understand how gcc produces
the caller part of caller-callee function call management so that the callee
part in ctxsw() works correctly.
6 pts

P2(b) 20 pts

XINU uses the ready list which is a priority queue/list sorted (in nonincreasing
order) by process priority as the key. Dequeue is constant but enqueue is linear
in the number of processes.
6 pts

In fair scheduling we need to find a ready process that has the smallest CPU usage.
Since every process's CPU usage, in general, may be different, we need to use a
data structure such as a heap or balanced search tree. Thus for both enqueue and
dequeue, the overhead in general is proportional to the depth of a balanced tree
which is logarithmic in the number of processes.
6 pts

Dequeue in multilevel feedback queue: loop at most 60 iterations to find the list
of the highest priority processes. Dequeue the first element in the list. Both
operations incur constant overhead since they don't depend on the number of processes
in the system.
Enqueue in multilevel feedback queue: Using the priority of the process to be enqueued
as index, go to the list of processes of equal priority. Use the tail pointer of the
list to insert the process at the end of the list. Both operations incur constant
overhead.
8 pts

Bonus 10 pts

Kernels are reactive in the sense that (for the most part) they spring into action
when (top-down) system calls are made by processes or (bottom-up) interrupts are
raised
by hardware that need to be serviced.
5 pts

No. Kernels are for the most part library functions that are executed when invoked by
system calls or interrupts. The basic method employed is to borrow the context of the
current process to execute the kernel's upper and lower half code.
5 pts