EEL 5737 HW #4 Part A

Part A a) Problem Set 9 b) Problem Set 10 c) Exercise 6.4

a Problem Set 9

Q9.1) n is the gate that the service call is intending to enter kernel mode through, which is saved in a designated register so that the kernel can read the intended gate name.

Q9.2) A, C, and D

The kernel changing pmar changes the current address space. Changing the user mode bit switches from user to kernel or vice versa. Any application attempting to write to the pmar will cause an exception which changes to kernel mode.

Q9.3) C

It will return where it left from which will be in YIELD() after the SVC call

13

HW #4 (cont.)

C (Q9.4) A and B because address space separation is always an enforcement of modularity, and the user-mode bit stops the applications from being able to modify their address spaces so they cannot exploit other program's memory.

(Q9.5) C and D because now that we have a timer interrupt, the user could've entered kernel mode from any instruction location when the timer triggered the interrupt.

(Q9.6) A, B, and C A and B the same as in Q9.4, and now C because a thread cannot hold the processor if it does not call YIELD, which would cause other threads to starve.

(Q9.7) A, B, and C

A - T1 could read 7 in R0 then get preempted, then end up adding 1+2 giving it 3.
B - if either thread is interrupted after loading into R0, then comes back, in the case of 4+8=12
C - This is expected as long as nothing is preempted between loads.

B

HW #4 (cont.)

c Q9.8) C Now that the code executes atomically, only powers of 2 will print.

Q9.9) A Since 100 is in the range 100-112 is the interrupt triggers inside one thread running those instructions, it will get its upc set to 100 while the other thread enters that range.

Q9.10) A and C

A - this is how it will run with no preemption

C - If preempted after instruction 104 finishes, a will have b saved in it, then when the section restarts, a=2 then we will get b also equals 2.

B+D cannot happen as a will always change no matter how many times the thread is preempted and subsequently restarted.

NS

HW #4 (cont.)

b Problem Set 10

Q10.1) A Virtual addresses are what is used to indicate the location of instructions.

Q10.2) A With this implementation, the only values that can be on the stack are return addresses for the next return instruction.

Q10.3) B If when we return here, we return to the instruction at VA 5, then we are currently in PRINT_MSG

Q10.4) A If we have I/O messages that are written to a specific address, that is, by definition, memory-mapped I/O.

Q10.5) A When YIELD returns in this implementation, the next instruction should always be continue at VA 34.

Q10.6) B We see here that within each thread stack that the only address values that can be present are those to which a called procedure will return

$\Rightarrow$

HW #4 (cont.)

b Q10.7) A This is the case as we should only preempt a thread if it is in the WAITING state.

Q10.8) A and B For thread 0 to be preempted to run thread 1, thread 0 would have to call YIELD AND thread 1 would have to be RUNNABLE and therefore input device 1 would have to have called NOTIFY

Q10.9) A The values that can be on the stack now will be any address for an instruction except an address for anything in the device interrupt handler.

Q10.10) A and C For A, this causes the device to be runnable then set to waiting and will then wait forever. For C, this causes the device to wait forever for input_available[0] to change, but it cannot as interrupts are disabled while DEVICE(n) runs.

→

HW #4 (cont.)

C Exercise 6.4) With only one page table in hardware, page faults that have to access the disk more for multiple applications, some suggestions just for this part are get a faster disk to reduce page fault penalty, add more page tables to hardware, or add more RAM. Respectively, reducing page fault penalty improves his performance, more page tables would reduce the overhead on performance for context switches, and more RAM allows for total page faults. He could also just add processors in parallel and pipeline operations, schedule preemptively to keep any stalled threads from hogging the CPU, or many other things.