1. Modify boot.S, so that your kernel switches to EL0 (instead of switching to EL1).

**Note:** Some students may want to modify other file(s), which are not necessary. But it is fine you insist that. Just submit everything you changed. See below.

**Note:** you may find QEMU’s debug log useful, which traces the exception level.

a. (30) (QEMU users only) Attach a screenshot here showing that your code works. The screenshot may be messages printed from your kernel execution, or a qemu log.

A screen shot of a computer

Description automatically generated

b. (20) Briefly explain: what changes have you done? What register(s) do you have to touch? What values do you put in the register(s) and why?

I have changed the sysregs.h SPSR\_EL1h to (0 << 0) which would have changed the spsr\_el2 register. The value 111000000 through SPSR\_VALUE is stored into x0 initially (which eventually gets updated with the memory address of el1\_entry) and into the system register spsr\_el2.

c. (20) Upload your code a separate tarball. **Only include the file(s) you modified in the tarball**

|  |
| --- |
| How to pack a code tarball (for this & future submission)  File name: [computingid].tar.gz. e.g. lg8sp.tar.gz. All lower case. Use the command 'tar -czvf' to generate.  (Optional) A file README-cs.txt, which contains anything TA should know (e.g., any caveats of your code, extra build instructions)  Absolutely no any binaries (e.g. \*.elf, \*.o, \*.bin) or .git/ subdirectory in the tarball.  **# Sample command 1: to compress all source files under a directory**  **# Note: no trailing space after “\”**  cd YOURDIR  tar czvf ../[computingid].tar.gz \  --exclude='\*.o' \  --exclude='\*.d' \  --exclude='\*.bin' \  --exclude='\*.elf' \  --exclude='\*.img' \  --exclude='.git/\*' \  \*  **# Sample command 2: to compress specific files in a directory**  **# Note: no trailing space after “\”**  cd YOURDIR  tar czvf ../[computingid].tar.gz \  my1.c my1.h my2.c my2.h |

d. (20) Can you demonstrate that the kernel ACTUALLY reaches EL0? For instance, can you execute some instructions disallowed at EL0 before/after the switch, and reason about the results? Explain your choice and observation. Attach screenshot(s) if needed.

A screenshot of a computer program

Description automatically generated

To test whether it was in EL0, I simply added a privileged instruction in el1\_entry\_another and redirected the address to el1\_entry\_another. When running the gdb, the system freezes because the instruction runs on EL0 and that instruction is illegal. In addition, since there is no exception handling, the OS will panic. A similar instruction was executed prior to the eret which means that those instructions executed with minimal issue.

1. (30) After landing in EL0, can your kernel print out the current exception level?

If so, attach a screenshot showing the printout.

If not, explain why.

It will not be able to because in order to get the current exception level, the “get\_el()” needs to call the “mrs” instruction which is illegal in EL0.

1. (10) What does an “eret” instruction do?

The instruction “eret” returns from the exception. More specifically, it restores the ALU flags, execution state, exception level, and processor branches of SPSR\_ELn. In addition, it jumps to the address stored in ELR\_ELn by modifying the PC.

(10) Does an “eret” instruction must correspond to an earlier exception?

No, because the earlier exception could be of a higher level and the OS cannot change from a lower to a higher level. Only the other way around.

(10) What will happen if we execute an eret instruction at EL0?

It would be pointless but “eret” would return to EL0 and continue through the rest of the process. That is, moving PC to the memory address stored in ELR\_ELn.