Debug

The faulting instruction you observed in GDB should match the one you found in #3. Now that you have determined the faulting instruction, understood the purpose of the instruction, and walked through how the kernel initializes a user process, you are in a position to modify the kernel so that `do-nothing` runs correctly.

1. Modify the Pintos kernel so that `do-nothing` no longer crashes. Your change should be in the Pintos kernel, not the userspace program (`do-nothing.c`) or libraries in `proj-pregame/src/lib`. This should not involve extensive changes to the Pintos source code. Our staff solution solves this with a single-line change to `process.c`. Explain the change you made to Pintos and why it was necessary. After making this change, the `do-nothing` test should pass but all others will likely fail. *Note: It is okay if your change seems like a hack. You will implement a better fix in the user programs project.*

2. It is possible that your fix also works for the `stack-align-0` test, but there are solutions for `do-nothing` that do not. Take a look at the `stack-align-0` test. It behaves similarly to `do-nothing`, but it returns the value of `esp % 16`. Write down what this program should return (*Hint: this can be found in `stack-align-0.ck`*) as well as why this is the case. You may wish to review stack alignment from Discussion 0. Then modify your fix if necessary so that both `do-nothing` and `stack-align-0` pass.

3. Re-run GDB as before. Execute the `loadusersymbols` command, set a breakpoint at start, and continue, to skip directly to the beginning of userspace execution. Using the `disassemble` and `stepi` commands, execute the `do-nothing` program instruction by instruction until you reach the `int $0x30` instruction in `proj-pregame/src/lib/user/syscall.c`. At this point, print the top two words at the top of the stack by examining memory (*Hint: `x/2xw $esp`*) and copy the output.

4. The `int $0x30` instruction switches to kernel mode and pushes an interrupt stack frame onto the kernel stack for this process. Continue stepping through instruction-by-instruction until you reach `syscall_handler`. What are the values of `args[0]` and `args[1]`, and how do they relate to your answer to the previous question?

Now, you can continue stepping through Pintos. Having completed running `do-nothing`, Pintos will proceed to shut down because we provided the `-q` option on the kernel command line. You can

step through this in GDB if you're curious how Pintos shuts down.

Congratulations! You've walked through Pintos starting up, running a user program to completion, and shutting down, in GDB. Hopefully this guided exercise helped you get acquainted with Pintos. **Be sure to push your code to GitHub with the small change you made in order to make the `do-nothing` and `stack-align-0` tests pass.** Check that the autograder gives you a full score on the coding portion.

Note: Your written answers will be graded by the readers, not by the autograder.
