The Null Pointer Dereferencing

In this project, you'll be changing xv6 to support a feature virtually every modern OS does. The task is causing an exception to occur when your program dereferences a null pointer.

Details

In xv6, the VM system uses a simple two-level page table. As it currently is structured, user code is loaded into the very first part of the address space. Thus, if you dereference a null pointer, you will not see an exception (as you might expect); rather, you will see whatever code is the first bit of code in the program that is running.

Task 1:

Thus, the first thing you might do is create a program that dereferences a null pointer. Then run it on Linux as well as xv6, to see the difference.

Task 2:

Made changes to xv6 code to cause an exception when it dereferences a null pointer.

Hint:

1. Figure out how xv6 sets up a page table.
2. Look at how `exec()` works to better understand how address spaces get filled with code and in general initialized. That will get you most of the way.
3. The rest of your task will be completed by looking through the code to figure out where there are checks or assumptions made about the address space.
4. Think about what happens when you pass a parameter into the kernel, for example; if passing a pointer, the kernel needs to be very careful with it, to ensure you haven't passed it a bad pointer.
5. Look at the xv6 makefile as well. In there user programs are compiled so as to set their entry point (where the first instruction is) to 0. If you change xv6 to make the first page invalid, clearly the entry point will have to be somewhere else (e.g., the next page, or 0x1000). Thus, something in the makefile will need to change to reflect this as well.

Final output:

Make a program that dereferences a null pointer. You should be able to demonstrate what happens when user code tries to access a null pointer. If you do this part correctly, xv6 should trap and kill the process without too much trouble on user part.