xv6 system calls

Part One: System call tracing

Your first task is to modify the xv6 kernel to print out a line for each system call invocation. It is enough to print the name of the system call and the return value; you don't need to print the system call arguments.

When you're done, you should see output like this when booting xv6:

```
...
fork -> 2
exec -> 0
open -> 3
close -> 0
$write -> 1
write -> 1
```

That's init forking and execing sh, sh making sure only two file descriptors are open, and sh writing the $ prompt. (Note: the output of the shell and the system call trace are intermixed, because the shell uses the write syscall to print its output.)

Hint: modify the syscall() function in syscall.c

Part Two: ps system call

Your second task is to add a new system call to xv6. The main point of the exercise is for you to see some of the different pieces of the system call machinery.

Your new system call will get information about the processes running in the system and return it to the user program. Specifically, your new system call will have the following interface:

```
int getprocinfo(int pid, struct uproc *up);
```

Where pid is the process id of the target process, and struct uproc is a structure that describes the process, i.e., contains the following information about the process: process name, process id, parent process id, size of process memory, process state, whether process is waiting on a channel, and whether it's been killed.

You will have to define the `struct uproc` and implement the ps utility by querying the system about all processes in the system. You should create a user-level program that calls your `getprocinfo` system call; here's some source you should put in `ps.c`:

```
#include "types.h"
#include "user.h"

int main(int argc, char *argv[])
{

      // your code to print the process information in any format
           you like...

      exit();
}
```

In order to make your new `ps` program available to run from the xv6 shell, add `_ps` to the `UPROGS` definition in `Makefile`.

Since both the kernel and user level code need the definition of the `struct uproc` structure, we can define in one of the header files visible at both levels. For example lets put it in `"stat.h"`

```
struct uproc {
      // your code to define pid, ppid, sz
};
```

Your strategy for making the `getprocinfo` system call should be to clone all of the pieces of code that are specific to some existing system call, for example the "uptime" system call. You should grep for uptime in all the source files, using `grep -n uptime *.[chS]`.

To define `struct uproc` look at how the same fields are defined in xv6's `struct proc`.

When you're done, typing `ps` to an xv6 shell prompt should print all processes running in the system and information about them.

In a text file write down a few words of explanation for each of the files you had to modify in the process of creating your `getprocinfo()` system call.