Section 2
October 5th

Pipes:

What you can expect from the client:

- The client will never write more than `512 bytes` if the client ever does, you may consider this an error.
- For `lab1`: Read will never read more bytes than have been written (this prevents blocking) For `lab2` onward, it may be the case that read asks for more bytes than have been written.

Question for you: What should happen if the write end or the read end is closed? When can you free the buffer of the pipe?

Some room for you to ponder

Look at the Piazza post: An argument for pipes
This gives an example of how pipes are used in bash. Note the usage of `dup` and `close` to setup the file table for the program that is `exec'`ed.

Allocate struct pipe at runtime: You probably need a `struct pipe` to track meta data of the pipe and also the buffer. To do that, you need to define `struct pipe` and when there is `sys_pipe`, you should `kalloc()` to get a memory page and treat that entire page as the `struct pipe`.

What information do "struct pipe" has to track: offset of the reader, offset of the writer, whether read side has closed, whether write side has closed

How to make "pipe" behave as a file: In your `struct file`, define an enum to indicate whether it is a pipe or an inode. Also in your file struct, you need metadata to track if the file is readable or writable. When a pipe is constructed, you need to construct two file structures, one side is readable but not writable and the other is writable but not readable. In your file structure, set a pointer to the pipe struct. Upon read/write to the file, check if it is a pipe and then call your pipe-related functions.