Design Document

Question 1 (file descriptors):
How are file descriptors implemented? What kernel data structures were created to manage file descriptors? Briefly describe the implementation of open, close, read, and write.

File descriptor is a structure containing a vnode pointer "v", an integer "flg", and an off_t (64 bit integer) "os". v is the abstract representation of the file using by the file descriptor; flg indicates how to open the file, such as read-only which is represented by 0 and write-only which is represented by 1; os is the offset within the file that the file descriptor associated with.

A new field fdt (which is a fd_table structure) is added to the thread structure. Each fd_table contains a pointer to an array (array structure defined by kernel) of file descriptors called "fds" and an integer named "num_fd". fds contains all the file descriptors of the thread, where their index in the array is their ID. Three file descriptor are created when a fdt is initialized. They are standard input, standard output, and standard error (fd 0, 1, 2). The I/O are operated through console device (con:). The fd_num keeps track of the number of fd opened for the thread to prevent from open too many files.

Open:
Open includes a char pointer "filename", an integer "flags", an integer mode, and an integer pointer "return_val". After check for valid filename and flags, vfs_open is called to open the file/device/other kernel object named by the pathname filename in mode specified by flags, and store contents into a vnode "v". If success, vfs_open will return 0. fd_create will then be called with vnode v, flag flags, and an offset of 0, which will create a file descriptor "fd". fd will then be added to current thread's fdt, appending to the end of the fds array. This is done by calling fd_table_add_fd, which returns the added fd's ID upon success and returns -1 when failed. If added successfully, the fd ID will be stored at the address pointed to by return_val, and 0 will be returned to indicate a successful Open of a file. Current thread's fd_num field will increase by 1, to avoid open more file than the limit number of files sys161 can open. Whenever an error happens, the open function will return -1, and the err number indicating the type of error is stored in the address pointed by return_val.

Close:
Close includes an integer "fd" and an integer pointer "return_val". It first checks for valid input of fd, such as deadbeef or a fd ID bigger than the length of array fds. Then fd_table_rm_fd will be called, which "removes" the fd from

current thread's fdt's fds by calling vfs_close and setting fd to null in the array. The fdt's num_fd is decreased by 1 every time a fd is removed. If success, returns 0; if not, returns 1, and the close function will return -1 and store the error number in the address pointed by return_val. If the close function succfully close the file descriptor, it will return 0.

Read:
Read takes in an integer "fd", a void pointer "buf", a size_t buflen, and an integer pointer "return_val". It checks for valid address, valid fd, and make sure the function is not write-only. An uio structure is initialized to store data read by VOP_READ from the given fd's vnode. If any of the above condition failed, function will return -1 and and store the error number in the address pointed to by return_val. fd's offset will be set to u's offset to indicate the number of byte read already. If u's uio_resid equals 0, no more bytes are needed to read, and the number of bytes read already (which equals to buflen) will be stored in the address return_val points to. If there is still bytes left to be read, amount of bytes left to be read (buflen-u.uio_resid) will be stored. Upon success, Read function returns 0.

Write:
Write takes in an integer "fd", a const void pointer "buf", a size_t nbytes, and an integer pointer "return_val". It checks for valid address, valid fd, and make sure the function is not read-only. An uio structure is initialized to store data to be write by VOP_WRITE to the given fd's vnode. If any of the above condition failed, function will return -1 and and store the error number in the address pointed to by return_val. fd's offset will be set to u's offset to indicate the number of byte wrote already. If u's uio_resid equals 0, no more bytes are required to write, and the number of bytes wrote already (which equals to nbytes) will be stored in the address return_val points to. If there is still bytes left to be write, the amount (nbytes-u.uio_resid) will be stored. Upon success, Write function returns 0.