Whenever a user starts a program, the kernel allocates it a designated amount of space in physical memory. The user program is then restricted to only being able to “view” this part of memory. This is done by making the user program believe its user space starts with address 0x0000 where as in physical memory it is actually something else.

Every blue box on the left designates a different user program. Memory addresses in black are physical (Kernel) addresses and those in blue are user addresses. You'll see Kernel address 0x2000 corresponds to address 0x0000 in that user program.

So while the user sees the variable z at memory address 0x1000, the actual physical memory address of z is 0x3000. That is 0x2000 (the start of the user space) + 0x1000 (where the variable is located in that user space). If a reference to z from user space were to be dereferenced in Kernel space without conversion bad things can happen.
Copy_To_User and Copy_From_User

What this means for you is that pointers (memory addresses) from user space can not be simply used in Kernel space. That is any pointers passed in to a system call must first be “converted”.

To make this easier a couple functions have been provided, Copy_To_User and Copy_From_User.

These functions do the conversion for you. Since the operating system knows the user program that made the system call it knows where the start of the user space is.

To use them allocate space in Kernel memory for the variable (be it on the stack or in the heap using Malloc()) and call the function in the order:

Copy_To_User( ulong_t destInUser, void* srcInKernel, ulong_t size);
Copy_From_User (void * destInKernel, ulong_t scrInUser, ulong_t size);

So to use these you must know the type of the reference in order to copy the correct amount of memory.
Read(), Write(), Close()

- These system calls are made in pipe-p1.c and pipe-p2.c
- You will need to implement Sys_Read(), Sys_Write(), and Sys_Close() in geekos-project/src/geekos/syscall.c for project0
- To see how Pipe_Read(), Pipe_Write(), and Pipe_Close() will eventually be called, look in geekos-project/src/geekos/vfs.c
- Read(), Write(), and Close() in vfs.c use the struct File_Ops* of the passed in struct File* to pass control to the appropriate function