1. Stat64 is a function that takes in a file path as one of its parameters, and returns information about the file that is pointed to by the path. A return value of 0 0 means the file is accessed without error, whereas a return value of -1 indicates that an error has occurred. This can be due to an invalid address, an I/O error, the path prefix not pointing to a directory, among other reasons. I’m not sure what Err#2 means (and I can’t seem to locate Errno#2 in the manpages), but clearly, the first and last calls of stat64 were successful, whereas the second, third, and fourth calls of stat64 resulted in some kind of error.
2. When I copy a long file, it seems like the read\_nocancel call reads in the first 254 characters, and whilst copying, the system automatically writes successive lines without reading them first, suggesting that when the OS reads in a file, it does so iteratively, so in most cases where the user copies a file that’s greater than the number of bytes one write\_nocancel call can print, it could do so in succession without calling read\_nocancel again.
3. When cpflinebyline is called, the behavior is different to that of copyfile.py. The difference is clear – read\_nocancel and write\_nocancel has to be called in conjunction every time a new line is to be written, and this repeats till the end of the file is reach. This is dissimilar to copyfile.py, where read is only required to be called twice, and the system automatically picks this up and reads by itself. This distinction is probably made when because the two python files are coded differently. Maybe copyfile.py simply requires the OS to copy the entire file, so the OS breaks it down and implicitly does the read\_nocancel command, whereas there exists a conditional loop in cpflinebyline, and the OS isn’t intelligent enough to understand that it does the same thing, so it just follows the loop conditions and ends up performing numerous redundant steps. If the file is very large and the user only wants part of the file copied, he might want to copy line by line, such that the entire file does not get read as this might cause a blowup in time.
4. I will attempt to explain the system calls from pwd.

stat64("/Users/sgoings/OS\0", 0x7FFF5FBFF9E0, 0x7FFF5FBFFAC8) = 0 0

stat64(".\0", 0x7FFF5FBFF950, 0x0) = 0 0

Looks at the current directory to check that the current directory is allowed to be read.

getrlimit(0x1008, 0x7FFF5FBFF7F0, 0x7FFF808132EC) = 0 0

getrlimit checks to make sure that checking the working directory would not blow up the resource limit

fstat64(0x1, 0x7FFF5FBFF780, 0x7FFF5FBFF84C) = 0 0

ioctl(0x1, 0x4004667A, 0x7FFF5FBFF7CC) = 0 0

makes sure that any special characters in the file name can be read with permission

munmap(0x0, 0x1000000, 0x3, 0x1002, 0x2000000, 0x7FFF00000000) = 0x300000 0

munmap(0x100300000, 0x500000) = 0 0

munmap(0x101000000, 0x300000) = 0 0

this seems like a “clear history” command executed by the shell. This is possibly done so that the working directory remains private to the user who called this command, such that other users that should not have access to the directory would not be able to trace the directory name.

write\_nocancel(0x1, "/Users/sgoings/OS\n\0", 0x12) = 18 0

obviously, this prints the current working directory to the terminal, essentially what the pwd command is supposed to return.