Siran Shan (304283584)

Li-Wei Tseng (304289977)

Lab 3 Design Project: OSPFS Crash Testing

For lab 3 design project, we chose OSPFS crash testing. There are two major parts in the project. First, we would like to crash the system (mainly by silently disable writes to disk data and leave it at an incorrect state). In order to do this, we used a per-OSPFS variable “nwrites\_to\_crash” to control when to crash the system. Also, we wrote a C-program “crash\_setter.c” to set “nwrites\_to\_crash.” The effects of “nwrites\_to\_crash” are as following: when “nwrites\_to\_crash” is set to -1, which is also the default value, OSPFS would behave normally. When “nwrites\_to\_crash” is set to 0, OSPFS would crash. In this design project, when OSPFS “crashes,” all writes to disk data would fail quietly. That is, whenever data is written to disk, the write is simply ignored. However, as “nwrites\_to\_crash” is set to greater than 0, “nwrites\_to\_crash” would decrement by one each time we write data to disk. Second, we wrote a shell script “crash\_tester.sh” to demonstrate a bug with our OSPFS implementation. Essentially, we would first set “nwrites\_to\_crash” to be greater than zero by invoking “crash\_setter” with desired value (./crash\_setter num) which is compiled from “crash\_setter.c”. Then we made a series of system, such as creating a file (“touch”), writes data (“echo”), link files (“ln”), read files (“cat”) and so on. Each system call would decrement “nwrites\_to\_crash” by one. Thus, when “nwrites\_to\_crash” reaches 0, OSPFS would crash. That is, further system call would not work until “nwrites\_to\_crash” is set to -1 or larger than 0. This passage concludes a complete specification of the feature’s behavior.

The following paragraph will describe the details of implementation and how the codes are organized. First, in “ospfs.h,” we have defined several MACROs. We first included “<linux/ioctl.h>” to have “ioctl.” Also, “IOC\_MAGIC,” the major number of device, is defined as “k”. We later defined a “ioctl” call named IOCTL\_CRASH as “\_IO(IOC\_MAGIC, 0),” The design problem is broken into two modules: “test” and “find bugs.” The logic of “test” is contained in “ospfsmod.c,” Each system call, such as “ospfs\_unlink,” “ospfs\_write,” “create\_blank\_direntry,” “ospfs\_link,” “ospfs\_create,” “ospfs\_symlink,” checks the value of “nwrites\_to\_crash.” The checking algorithm is as following: If the variable is less than -1, we would call “eprintk(“Invalid nwrites\_to\_writes\n.”).” If the variable is larger than 0, we would decrement “nwrites\_to\_crash” by 1 and call “eprintk(“nwrites\_to\_crash = %d\n”, nwrites\_to\_crash).” The purpose of calling “eprintk” is to inform users the current value of “nwrites\_to\_crash” and also makes the debugging process easier. When “nwrites\_to\_crash” is 0, we would return “count,” which is the amount of data to write and thus causes the write to fail “silently” -- simply ignores writing data to disk. We also implement a function called “ospfs\_set\_crash.” As we could tell by its name, this function would set “nwrites\_to\_crash.” We make the system call called “ioctl” linked to “ospfs\_set\_crash” in the static struct called “osfps\_reg\_file\_ops” (which means it works on regular files). “ioctl” is called on a regular file in the main function in “crash\_setter.c” to change the value of “nwrites\_to\_crash.” This concludes the “test” module. As for the “find bugs” module, we have a script file “check-lab.sh” which would call “crash\_setter” to change the value of “nwrites\_to\_crash.” The effect and the logic of the file are included in the first paragraph, so we do not explain it again here.

The result satisfies the specification of the feature. OSPFS would behave differently according to various values of “nwrites\_to\_crash.” The main obstacle of this design project is to figure out the implication and the usage of “ioctl,” which manipulates the device parameters of special files. Also, writing test cases is also tricky at first because we need to understand the design project fully to know what the correct outputs are. From this assignment, we learn how to add additional functionality to an existing project with relatively ambiguous specification. We also learn how to correctly work with system call and shell script. Simon Shen mainly work on codes, while Li-Wei Tseng on reports.