Lab 3 Design Project

OSPFS Crash Testing

To fulfill the requirement of design project, we select to implement the OSPFS crash testing for lab 3. Implied from the name of the design project, the very first task that we need to finish is to crash the system by dysfunction the writing to disk. A global variable “nwrites\_to\_crash” (defined in ospfsmod.c) that shows up throughout all writing-related functions serves to manage the situation of crashing the system. If the value of “nwrites\_to\_crash” is -1, every writing command goes correctly, and no crashing will occur. If the variable has a value of 0, the file system will be banned from writing to disk and crash immediately as expected. However, this crashing information won’t be explicitly announced to users. If its value is greater than 0, we won’t make the system crash at this moment. Instead, we will decrease its value by one each time we attempt to write to the disk (perhaps by commands of “touch” – create new files, “cat” – read data, “ln” – create hard links, “ln – s” – create symbolic links and “echo” – write data) until the variable becomes zero. Then, the system crashes.

Specifically, we follow the several steps to achieve our goal. First of all, we include “ioctl” (its major function is to maintain or change device parameters of files) in “ospfs.h”, and define “IOC\_CRASH” so as to set up functions relevant to “ioctl”. Second, in “ospfsmod.c”, we implement a function named “ospfs\_set\_crash”, and link it with “ioctl” through a self-defined structure “ospfs\_reg\_file\_ops” so that calling “ioctl” on a regular file can set or change the value of argument passed to “nwrites\_to\_crash”. By default, the value of “nwrites\_to\_crash” is set to 0. After that, we work on the conditional codes concerning the value of “nwrites\_to\_crash” in each writing-related ospfs function. The entire set of functions that we should modify includes “ospfs\_unlink”, “ospfs\_write”, “create\_blank\_direntry”, “ospfs\_link”, “ospfs\_create”, and “ospfs\_symlink” since they all have something to do with writing to the disk. In each of them, we should check the value of “nwrites\_to\_crash” before any operations. If the value of “nwrites\_to\_crash” is less than -1, we print out the error message that announces the invalidity of its value and requires a new setup. If its value is -1, the writing command should definitely work well. However, while calling writing functions, if the value of “nwrites\_to\_crash” is 0, we simply return the variable “count”, which stores the amount of data that we intend to write to the disk. We decide not to announce the crashing of the file system explicitly but more like just to omit the writing command itself. Users will perceive that the system terminates its job. If the value of “nwrites\_to\_crash” is greater than 0, we decrease “nwrites\_to\_crash” by 1 and print out its current value (this step is trivial and helps a lot in debugging). Then, we execute the writing command.

“crashtest.sh” and “normaltest.sh” are test cases written by ourselves to find out whether our approach works well or not. In “crashtest.sh”, the value of the variable “nwrites\_to\_crash” is initially set to 4, and after we go through four input/output processes, the program won’t write anything. In “normaltest.sh”, the value of the variable “nwrites\_to\_crash” is set to -1 at the top of the file. As expected, every writing command works well and the file system never terminates if we do nothing to change “nwrites\_to\_crash”’s value.