【OS】Day44(2)

Class	Operating System: Three Easy Pieces
Date	@February 28, 2022

【Ch39】Interlude: Files and Directories

39.7 Writing Immediately With fsync()

Most times when a program calls `write()`, it is just telling the file system: please write this data to persistent storage, at some point in the future.

The file system, for performance reasons, will buffer such writes in memory for some time; at that later point in time, the writes will actually be issued to the storage device.

From the perspective of the calling application, writes seem to complete quickly, and only in rare cases(e.g. the machine crashes after the `write()` call but before the write to disk) will data be lost.

However, some applications require something more than this eventual guarantee. For example, in a database management system(DBMS), development of a correct recovery protocol requires the ability to force writes to disk from time to time.

To support these types of applications, most file systems provide some additional control APIs. In the UNIX world, the interface provided to applications is known as `fsync(int fd)`. When a process calls `fsync()` for a particular file descriptor, the file system responds by forcing all dirty(i.e. not yet written) data to disk, for the file referred to by the specified file descriptor.

Here is a simple example of how to use `fsync()`. The code opens the file `foo`, writes a single chunk of data to it, and then calls `fsync()` to ensure the writes are forced immediately to disk.

```
int fd = open("foo", O_CREAT|O_WRONLY|O_TRUNC, S|IRUSR|S_IWUSR);
assert(fd > -1);
```

```
int rc = write(fd, buffer, size);
assert(rc == size);
rc = fsync(fd);
assert(rc == 0);
```

39.8 Renaming Files

When typing at the command line, this is accomplished with `mv` command; in this example, the file `foo` is renamed `bar`:

```
prompt> mv foo bar
```

Using `strace`, we can see that mv uses the system call `rename(char *old, char *new)`, which takes the original name of the file(`old`) and the new name(`new`).

Imagine that we are using a file editor, and we insert a line into the middle of a file. The way the editor might update the file to guarantee that the new file has the original contents plus the line inserted is as follows.

```
int fd = open("foo.txt.tmp", O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR|S_IWUSR);
write(fd, buffer, size);
fsync(fd);
close(fd);
rename("foo.txt.tmp", "foo.txt");
```

The editor write out the new version of the file under a temporary name(`foo.txt.tmp`), force it to disk with `fsync()`. Then, when the application is certain the new data is on the disk, rename the temporary file to the original file's name. This last step atomically swaps the new file into place, while concurrently deleting the old version of the file.