Project 4: tester

Joel Porquet

All Sections

(FYI, there are still a few groups who haven't registered, please do it before the end of the day!)

As promised, I'm releasing a tester today. It's actually a (small) subset of the grading script, that checks a handful of test cases.

Hopefully, it should be enough to make your program segfault if you aren't handling the memory allocation properly for the various blocks you need to read from the disk (superblock, FAT blocks, root directory). Make sure that you are passing these few test cases, otherwise it is *highly* unlikely that your program will score well with the actual grading script!

You'll see that this tester only checks a few test cases around phase 1 and phase 2, but hopefully you can find ways to extend it in order to:

1. Test phase 1 and 2 even more (especially the corner cases)
2. Also test phase 3 and 4. You'll probably need to write your own C applications for these phases, in order to stress the API appropriately.

The script is located in */home/cs150/public/p4/test/* and is named *test_fs_student.sh*. I didn't initially put *fs_make.x* and *fs_ref.x* in directory *test* when releasing the project, but I think it makes sense to put them there (the script expects these executable files to be in *test*). Here is a trace showing how things are supposed to work:

```
$ ls
libfs test
$ cd test
$ tree
.
├── fs_make.x
├── fs_ref.x
├── Makefile
├── test_fs.c
└── test_fs_student.sh
$ ./test_fs_student.sh

--- Running run_fs_info ---
Info: Created virtual disk 'test.fs' with '100' data blocks
Pass: FS Info:
Pass: total_blk_count=103
Pass: fat_blk_count=1
Pass: rdir_blk=2
Pass: data_blk=3
Pass: data_blk_count=100
```

```
Pass: fat_free_ratio=99/100
Pass: rdir_free_ratio=128/128

--- Running run_fs_info_full ---
Info: Created virtual disk 'test.fs' with '100' data blocks
Info: 1+0 records in
1+0 records out
2048 bytes (2.0 kB, 2.0 KiB) copied, 0.000108704 s, 18.8 MB/s
Info: 2+0 records in
2+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 9.4056e-05 s, 43.5 MB/s
Info: 4+0 records in
4+0 records out
8192 bytes (8.2 kB, 8.0 KiB) copied, 0.000124058 s, 66.0 MB/s
Info: Wrote file 'test-file-1' (2048/2048 bytes)
Info: Wrote file 'test-file-2' (4096/4096 bytes)
Info: Wrote file 'test-file-3' (8192/8192 bytes)
Pass: fat_free_ratio=95/100
Pass: rdir_free_ratio=125/128

--- Running run_fs_simple_create ---
Info: Created virtual disk 'test.fs' with '10' data blocks
Info: 1+0 records in
1+0 records out
10 bytes copied, 6.8485e-05 s, 146 kB/s
Info: Wrote file 'test-file-1' (10/10 bytes)
Pass: file: test-file-1, size: 10, data_blk: 1

--- Running run_fs_create_multiple ---
Info: Created virtual disk 'test.fs' with '10' data blocks
Info: 1+0 records in
1+0 records out
10 bytes copied, 8.7506e-05 s, 114 kB/s
Info: 1+0 records in
1+0 records out
10 bytes copied, 7.3108e-05 s, 137 kB/s
Info: Wrote file 'test-file-1' (10/10 bytes)
Info: Wrote file 'test-file-2' (10/10 bytes)
Pass: file: test-file-1, size: 10, data_blk: 1
Pass: file: test-file-2, size: 10, data_blk: 2
```

In that trace, all the lines prefixed with "Info" (or lines that aren't prefixed with anything) are just what some programs are printing when executing. The lines prefixed with "Pass" are the lines that are actually being compared to the expected output.

As for all the projects so far, it is OK to including some debugging output in your library during the development phase, but once you submit next week, all of your printfs have to go (unless they actually part of the specifications, such as with fs_info() or fs_ls()). One good solution might be to have a macro at the beginning of *fs.c* that can easily switch between debug and release modes:

```
#if 0
#define fs_print(fmt, ...) \
       fprintf(stderr, "%s: "fmt"\n", __func__, ##__VA_ARGS__)
```

```
#else
#define fs_print(...) do { } while(0)
#endif
```

Now, if you have *#if 0* defined, then all calls to *fs_print()* will be discarded; but if you have *#if 1* defined, then all calls to *fs_print()* will actually make the messages be printed on *stderr*. The *fs_print()* macro has the same behavior as *printf*:

```
fs_print("I love %s %d!\n", "Project", 4);
```

This announcement is closed for comments

Search entries or author Unread