Project 5b: xv6 Small File Optimization

Notes

Start with this kernel, not the usual one: **~cs537-1/ta/xv6_p5b**

Overview

In this project, you'll be changing the existing xv6 file system to add high-performance support for small files. This type of optimization has been explored in the literature and now you can explore it, in some limited form, in this project.

The basic idea is simple: if you have a small file that can be indexed with only 13 direct data pointers, we will use the 13th pointer reserved for indirected data block as a direct data pointer, thus speeding up access to the small file, as well as saving some disk space.

To make life a little easier for you, we will create a new file type that is specifically called T_SMART. This will let your code recognize that the file being accessed is indeed a smart file and act differently than the normal case. There will also be a new flag used to create these smart files, O_SMART. More details below.

Thus, you'll have to be able to handle a new flag to open() (O_SMART), which, when passed to open(), should make sure to create the file as not a T_FILE (normal file) but rather as a T_SMART. Then, you'll have to modify the read and write paths to be able to read and write from these smart files.

When the file grows larger and cannot fit into the 13 blocks, the system should allocate the indirect pointer block and rearrange it into a normal T_FILE. As we do not have a truncate system call, the size of a file can only increase and never decrease. This means you do NOT need to worry about the case when a T_FILE becomes smaller and can fit into a T_SMART.

Details

To start, look in **include/stat.h.** You'll have to add the T_SMART in there, and define it to be 4, e.g.

```
#define T_SMART 4
```

This will let us determine if the file being accessed is indeed one of these smart files.

You also need to poke around to find where a new flag to open should be defined. You'll find that **include/fcntl.h** has these definitions. You'll need to add:

```
#define O_SMART 0x400
```

in there. Note: these must be followed exactly, or tests will not work.

Then you need to go about following the read/write paths to see where to make your changes. Start in **kernel/sysfile.c** (looking at sys_open() and sys_write(), as well as create()), and follow through the read and write paths to **kernel/file.c** and eventually **kernel/fs.c.** In that last file, make sure to

understand readi() and writei(), as well as the inode update routine iupdate().

The steps to remove a file can also be different. Note that in Linux, you are allowed to remove a file while it is being used by some other processes. The remove action only clears the directory entry and decrements the file's reference count by 1. The inode and data blocks are free'd when the count reaches zero. Take a look at how **sys_unlink** and **sys_close** work with the help of some functions in **kernel/file.c** and **kernel/fs.c** .

There is no need to do any of this for directories; they should remain as is.

The Code

The code (and associated README) can be found in **~cs537-1/ta/xv6_p5b/** . This version has some new system calls to support the auto grader. So you cannot use your old version.

Test Along The Way

It is a good habit to get basic functionalities working before moving to advanced features. You can run an individual test with the following command:

```
shell% ~cs537-1/ta/tests/5b/runtests testname
```

1. Define the two macros: `macros`
2. Create and open T_SMART: `create, create2, create3, open`
3. Write and read a small T_SMART: `write, read`
4. Grow into a normal T_FILE: `normal, write2, read2`
5. Remove a small T_SMART: `remove`