# Project #4

## 1. Before starting

Please start your project from fresh copy of xv6. After download `fspatch.patch` from KLMS, apply it using following command (run on root of xv6 – where the README file exists): `patch -p0 < fspatch.patch`

## 2. Small File Optimization

In this project, you'll be changing the existing xv6 file system to add high-performance support for small files. The basic idea is simple: if you have a small file (just a few bytes in size), instead of allocating a data block for it, just store the data itself inside the inode. This optimization can reduce disk space as well as speeding up the small file accesses. The inode has some number of slots for block addresses (specifically, `NDIRECT` for direct pointers and one more for an indirect pointer), you should be able to store `(NDIRECT + 1) * 4` bytes in the inode for these small files (52 bytes or less, in current xv6's implementation).

For easier implementation, we will create a new file type called `T_SMALLFILE`. This will let your code recognize that the file being accessed is indeed a small file and act differently than the normal (`T_FILE`) case. There will also be a new flag used to create these small files, `O_SMALLFILE`.

Thus, you will have to be able to handle a new flag `O_SMALLFILE` to `open()` system call. When `O_SMALLFILE` is passed to `open()`, you should create the file as NOT a `T_FILE` but rather as a `T_SMALLFILE`. Then, you will have to modify the read and write paths to be able to read and write from these small files. You will also have to check for errors; for example, you'll have to return errors when the user tries to make the small file bigger than the `(NDIRECT + 1) * 4` bytes.

### 2.1. Details

To start, look in `include/stat.h`. You'll have to add the `T_SMALLFILE` in there, and define it to be 4:

- `#define T_SMALLFILE 4`

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

Next, in `include/fcntl.h`, add definition of `O_SMALLFILE` as `0x400`.

- `#define O_SMALLFILE 0x400`

**Important!!** These must be followed EXACTLY, or test scripts will not work.

Then, you need to look around how current xv6 handles read and write to normal files. Start in `kernel/sysfile.c` (for `sys_open()`, `sys_read()`, `sys_write()` and `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()`.

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

Note that a real file system likely wouldn't create a new file type (`T_SMALLFILE`) for this type of optimization; rather, it would store small files' data in inode, and then, when the file grew beyond what fits in the inode, it would allocate data blocks and put all the data in there. In this project, small files always just use the inode to store data, and are NOT allowed to grow beyond what fits in there. This simplification is in place to make your life easier.

## 2.2. Hints

You need to modify/add codes mostly in the following files:

- `include/stat.h`
- `include/fcntl.h`
- `kernel/sysfile.c`
- `kernel/sysfile.h`
- `kernel/file.c`
- `kernel/file.h`
- `kernel/fs.c`
- `kernel/fs.h`

## 3. Deliverable

Implement small file optimization on current xv6 file system. Specifically, add a new file type called `T_SMALLFILE` and handle such special file properly. Other types of file (e.g., normal file or directory) should be handled as is.

## 4. Grading Criteria

We will use 20 scripts for grading. You can check the descriptions below.

**macros (5):** Make sure the macros are the right value.

**fulldisk (10):** First uses all free blocks on disk for normal files and then makes sure small files can still be created.

**create (10):** Creates a small file and makes sure the type is properly set.

**create2 (10):** Should not create a small file if `O_SMALLFILE` is passed in but not `O_CREATE`.

**open (10):** `open()` called a second time without `O_CREATE` and `O_SMALLFILE`. Checks file's type to make sure it is `T_SMALLFILE` and that read and write still work.

**read (10):** Tries to read more than possible and makes sure only amount available is read.

**read2 (10):** Reads one byte at a time from small file.

**write (10):** Tries to write more than possible to a small file and makes sure only 52 bytes are written.

**write2 (10):** Repeatedly opens, writes, closes a small file. The last write should be the one that is persisted.

**write3 (10):** Repeatedly opens, writes to the end, closes a small file until file is full.

**write4 (10):** Small write followed by a big write to a small file should truncate the big write up to max small file size.

**readonly (10):** Makes sure small files can be opened as read only and writes would fail.

**writeonly (10):** Makes sure small files can be opened as write only and reads would fail.

**remove (10):** Makes sure removing a small file does not corrupt the file system by freeing blocks.

**stress (10):** Creates and deletes a lot of small files one at a time to make sure nothing breaks under a heavy load.

**stress2 (10):** Creates and deletes a lot of small files 100 of them at a time to make sure nothing breaks under a heavy load.

**smallbig (10):** Creates bunch of small files; then creates bunch of big files to make sure normal files are not affected.

**inode (10):** Uses all inodes in the file system; then repeatedly deleted and creates a file and makes sure the correct inode is reused for the new file.

**inode2 (10):** Uses all inodes in the file system; then repeatedly deletes bunch of files and then creates that many files and makes sure the correct inodes are reused for the new files.

**usertests (10):** Running standard tests to make sure you did not break anything for normal files.

After finishing your implementation, compress your xv6 source codes using the commands below. Then, submit the compressed file on KLMS.

```
#First, move to a directory containing the xv6 directory
mv <DIRECTORY_CONTAINING_THE_XV6_DIRECTORY>

#Then, compress your xv6 directory
tar  -czvf  <Student ID>_proj4.tar.gz  xv6
```