**CS450 Programming Assignment 4 Design**

**Andrew Patterson and Changyu Wu**

**Group 27**

1. **Abstract**

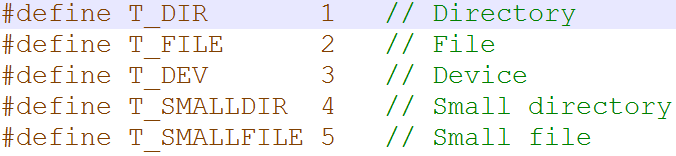
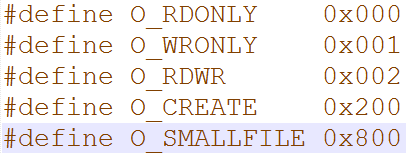
When we create a regular directory in xv6, we use system call **mkdir**, which calls **create** method (To create a new file, we call **create** too, for example, look at system call **open**, inside **“sysfile.c”**, line #299). **Create** method takes 4 arguments, the second argument is the type of the file/directory we want to create. Initially, there are 3 types of files/directories, located in **“stat.h”** file, they are, 1. **Directory (regular directory)**, **File (regular file)** and **Device**. If we want to create another type of file that’s small enough to be stored inside an inode, we’ll just need to define this type of file and make some changes to the **create** method (e.g., Pass the second argument as a small file instead of a regular file and handles this case in the function body). Also, as required, small files can only be stored in a certain type of directory, which only stores small files, with this being said, another type of directory should be defined too. Unlike small files, a small directory (a directory that only stores small files) should be almost the same as a regular directory, except for the fact that regular files are not allowed in it, which means that when we go through the **File System** in xv6, whenever we see a function that deals with regular directories, we should add something to that function to make it deal with small directories too, in exactly the same way. After we defined small file, small directory and made some small changes to the **File System** in xv6, we need to write a command that creates small directories: **mkSmallFilesdir**. And of course, another command that creates small files: **mksmf**. We need to consider the possibility that user might “accidently” store a regular file into a small directory, or link a small file to a regular file that’s stored in a regular directory, etc. These are the cases we need to handle, by making some changes to **open(), write(), read()** and **link()**

1. **Changes**

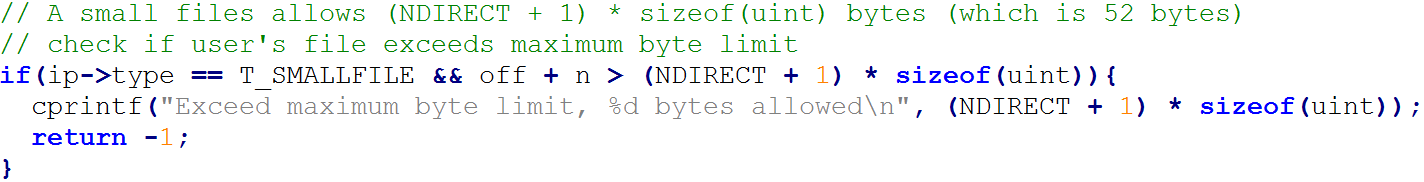
**stat.h & fcntl.h:**

Define small file and small directory in **stat.h** and define small file **flag** in **fcntl.h**

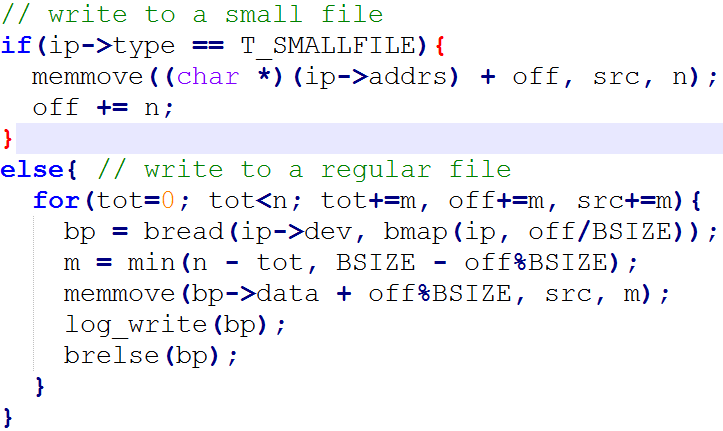
**stat.h: fcntl.h:**

**fs.c:**

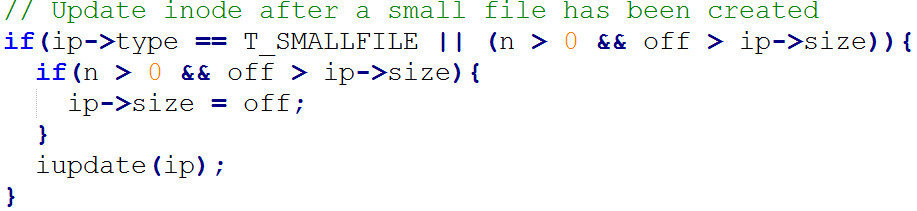
In **writei()** method at line #476. When we get **T\_SMALLFILE** flag, we know that the user wants to write to a small file. The first thing we need to check is whether the user is writing too many bytes to the file (exceeds maximum byte limit). To answer the question in the instruction, a small file must be smaller than **52** bytes, calculated by (**NIRECT + 1) \* sizeof(uint)**, where **NDIRECT = 12**, located in **fs.h** file, so at line #492, we add this statement: 

After this, it’s safe to write bytes into a small file. We check the type of file that the user wants to write bytes into and write into those files differently, at line #499:

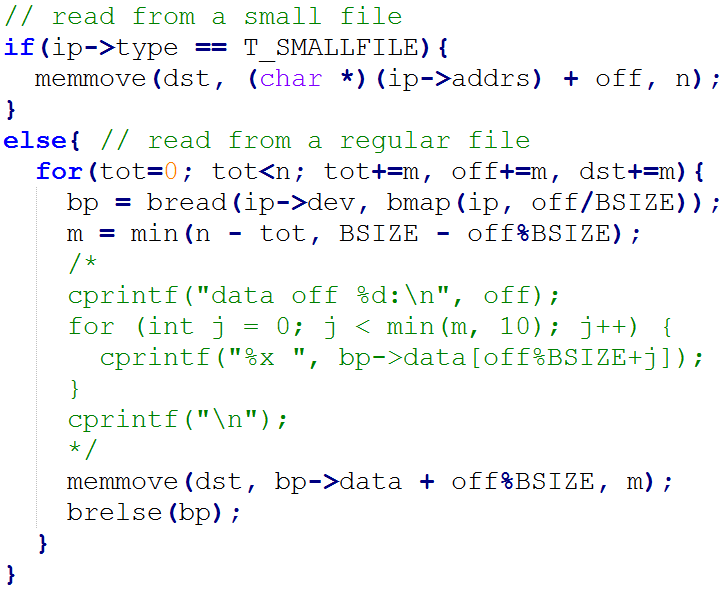


The “for” statement was in xv6 already and that writes into a regular file, we’ll just need to wrap it with an “else” block and handle writing-into-a-small-file situation in the “if” statement above

We are not done writing into a small file. This process does not allocate a block for a file, instead, it stores files inside an inode. Therefore, after creating a small file, we must update the inode where we stored the file, at line #514:



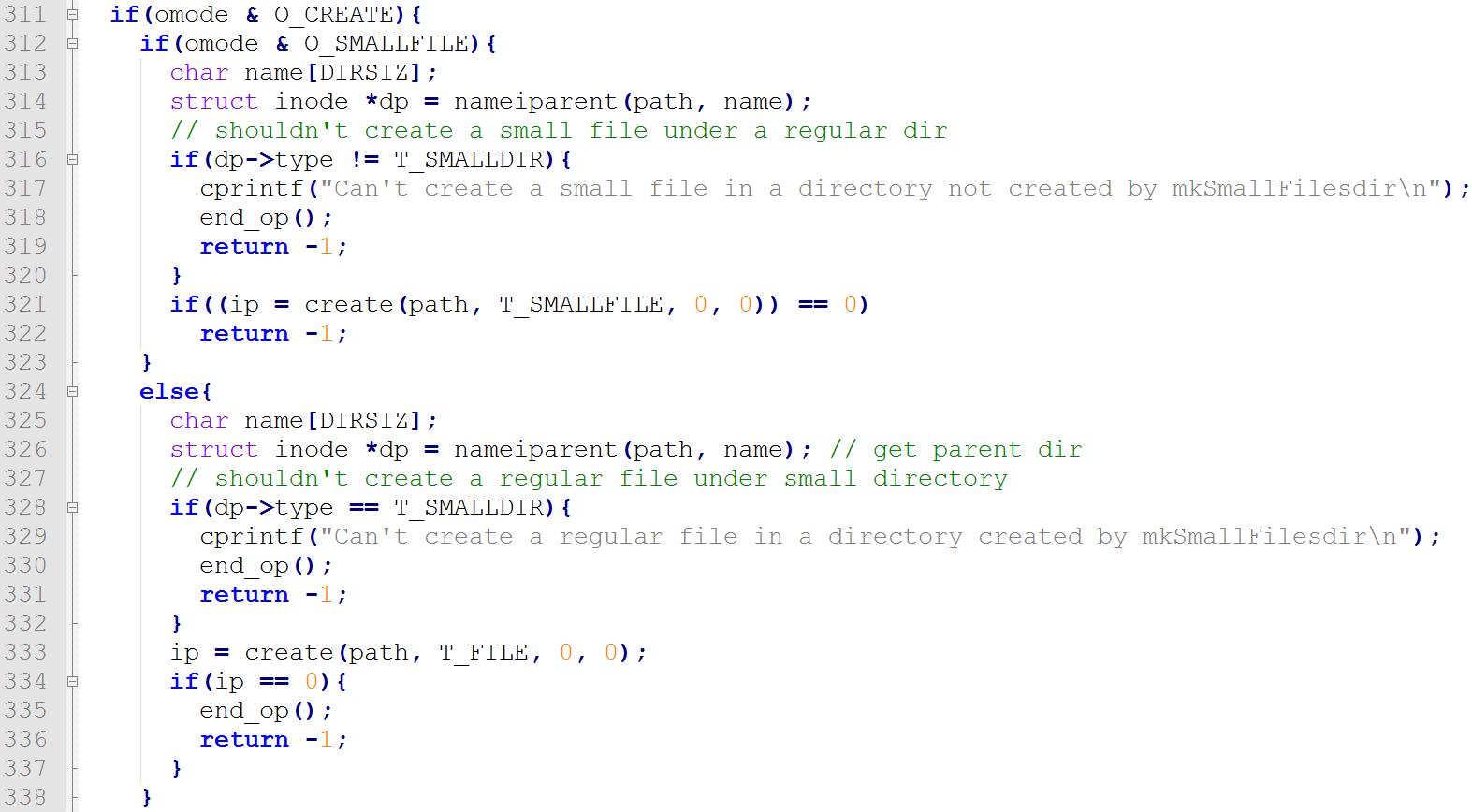
In **readi()** method at line #452, we need to handle reading-from-small-file situation. The “for” statement handles reading-from-regular-file case, we’ll just need to wrap it with an “else” statement and handles the other case in the “if” statement above, as follows:



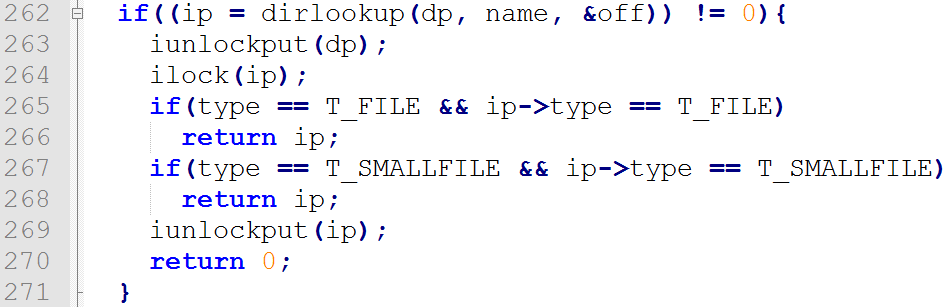
The reason why we make changes to **readi()** and **writei()** methods, instead of system calls **read()** and **write()** is that, **read()** and **write()** doesn’t really read or write to a file directly, eventually, they both call **readi()** or **writei()** methods

**sysfile.c:**

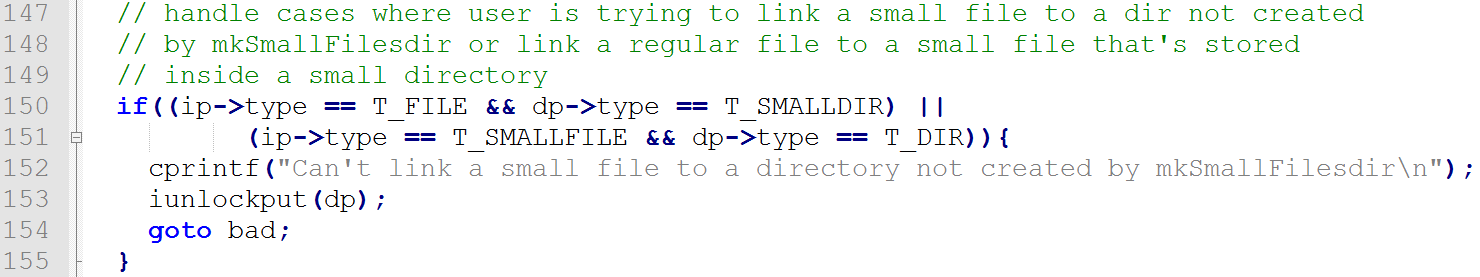
In **sys\_open()** method, we call **create()** to create a new file when target file is not present and user gives permission to do so when file is not found by passing flag **O\_CREATE**. Therefore, at line #311, where the method tries to create a new file, we need to handle cases where user wants to create a small file. Although, we can’t just create a small file on demand, we have to check if the file is being created under a regular directory, and also, if a regular file is being created under a small directory, both cases should raise some type of fault. Some explanation of the code: line #312, the “if” statement handles creating a small file. Line #313 - #320, check the type of the parent directory, whether it’s a regular directory or a small directory. Line #321, it’s safe to create a small file now, then call **create()** method to do so, and for the second argument, instead of passing **T\_FILE**, we pass **T\_SMALLFILE**. Line #324, the “else” statement handles creating a regular file. Line #325 - #332, check the type of the parent directory, if it’s a small directory, then raise a fault because we don’t want to create a regular file inside a small directory:



In **create()** method. This is where we create files and directories. At line #262, if the “if” statement returns true, we are creating a file, otherwise, we are creating a directory. So, inside this “if” block, we add some code to handle creating-small-file situation: line #267

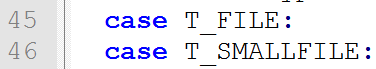


In **sys\_link()** system call. This system call links two files. We don’t want to link a regular file to a small file or a small file to a regular file, therefore, we need to check the file types before we go ahead and link them, line #147:



As of now, we’ve handled situations where we create a file (regular or small), write into a file, read from a file as well as link two files, but we haven’t made any changes to deal with creating a small directory. As mentioned in **Abstract**, a small directory should be treated in the exact same way as a regular directory, except that a small directory has type **T\_SMALLDIR** but a regular directory has type **T\_DIR**. We need to go through **File System** and add something to the code where **T\_DIR** appears. For example, in **sysfile.c**, line #133, initially, it was: , we make it . In **fs.c** file, line #541, initially, it was: , after, it should be:  **T\_DIR** appears in **fs.c** and **sysfile.c** files, we need to add **T\_SMALLDIR** to each of the places, they are, **sysfile.c:** line #133, #223, #231, #282, #345, #377 and #434, **fs.c**: line #541 and #647

In **ls.c** file. Although we made **T\_SMALLFILE** and **T\_SMALLDIR** work at this point, we can’t see it because the “ls” command doesn’t display the information about a small file/directory. Simply add the two cases to it:

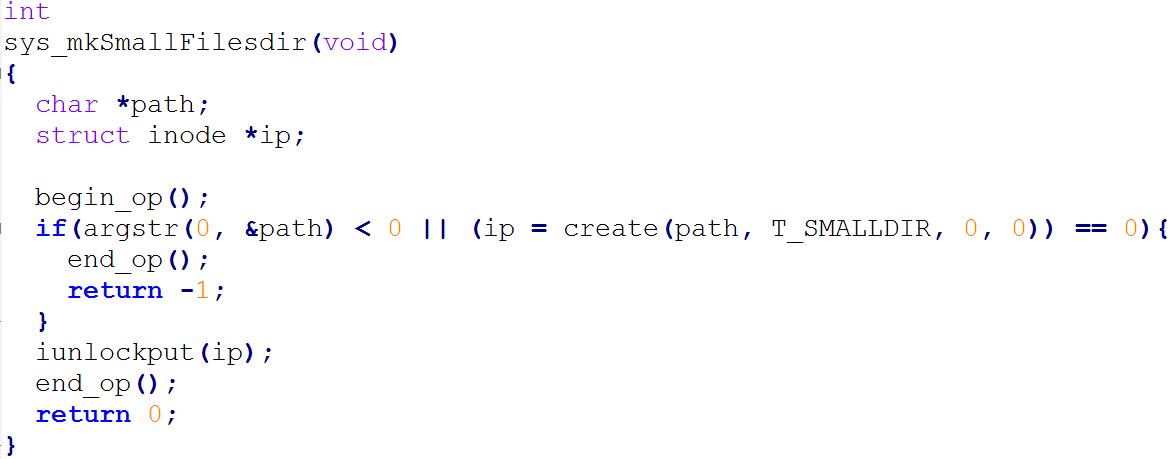
Line #45:  Line #50: 

1. **Add:**

**New System Call**

To make it easier for users to create a small directory/file, we made two commands, **mkSmallFilesdir** and **mksmf**. **mkSmallFilesdir** is also a new system call. The procedure of adding a system call is omitted but let’s take a look at the system call function body

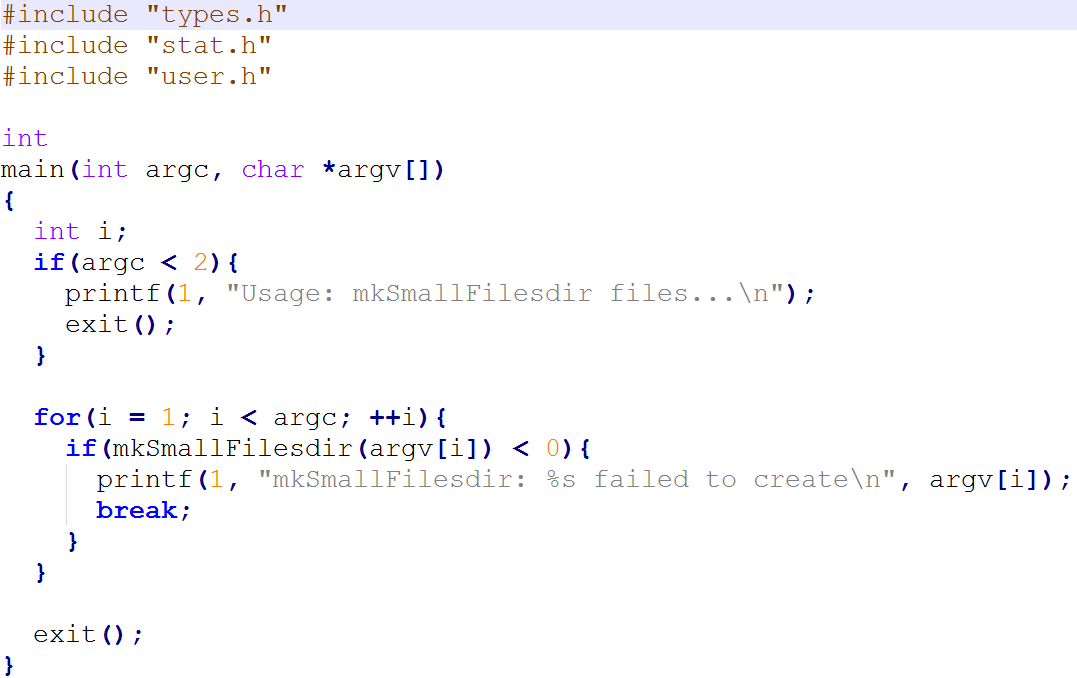
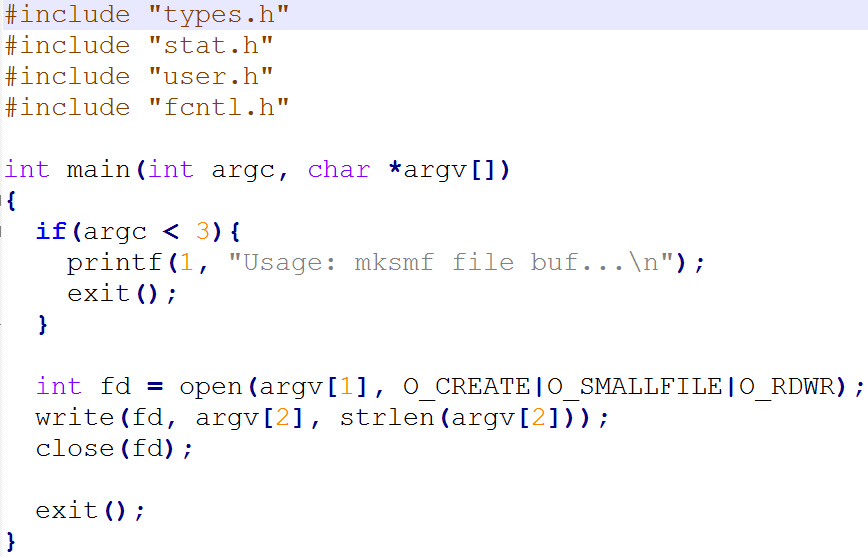
Inside **sysfile.c**, line #386:



The system call above it is **mkdir**. The two system calls are almost the same except for the “if” statement, **mkdir** passes **T\_DIR** but **mkSmallFilesdir** passes **T\_SMALLDIR**

**New Command:**

**mkSmallFilesdir in mkSmallFilesdir.c** and **mksmf in mksmf.c**

**mkSmallFilesdir.c** is not much different than **mkdir.c**, one calls **mkSmallFilesdir** system call and the other calls **mkdir** system call

**mksmf.c** calls system call **open()** to create a new file and passes **O\_SMALLFILE** to it. After it’s created, write to the small file