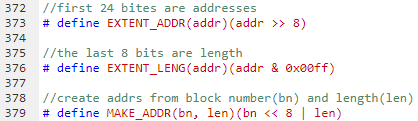
Programming Assignment 4

Question 1: Making Extent-based File System

The following files were modified in the creation of the extent-based file system in xv6:

|  |  |  |
| --- | --- | --- |
| **FILENAME in xv6** | **Added Code** | **Purpose** |
| fcntl.h | #define O\_EXTENT 0x010 | Creating the O\_EXTENT flag |
| stat.h |  | Adding support for extent files into the struct stat.   * extentinfo stores addresses and length of extent. * T\_EXTENT is a defined constant that is used to recognize if the file is extent or pointer-based. |
| fs.c | See Figures 1 - 4 for details. | Implementation of extent-based file system, while retaining backward compatibility for pointer-based files. |
| sysfile.c | See Figures 5-6 for details. | Code modified to ensure that extent-based files are recognized by the file system and action is taken accordingly. |

**Extent Files: Block Allocation**

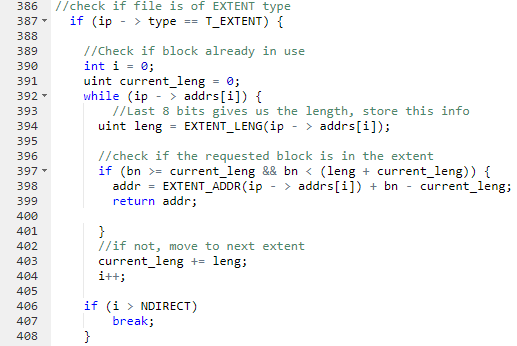


* *EXTENT\_ADDR(addr):* This stores the first 24 bits of an address. This represents the address of the extent.
* *EXTENT\_LENG(addr):* This stores the last 8 bits of an address. This represents the length of the extent.
* *MAKE\_ADDR(bn, len)*: Existing inode pointers are used to store the pointer to block number and the block length.

**Data Allocation to Extent Files**

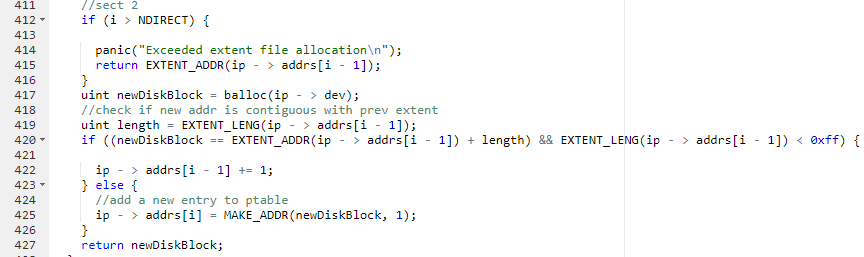
Design Goals

* Existing inode pointers are modified so that the first 24 bits store the block disk addresses while the remaining 8 bits store the length of the extent.
* To access a particular data block of a file, the function **bmap()** is invoked.Whenever a data block of a ﬁle is needed, function bmap is called. **bmap()**is an existing function in xv6, which maps file block number to disk block number.
* We have modified the **bmap()** function to include functionality for extent-based files. Existing code that provides pointer-based functionality is not changed. Thus, in this way, we are able to maintain backward compatibility for pointer-based files.
* When **bmap()** is called,first there is a check to see if the requested ﬁle block is on the disk. This is done by checking if the given block number is within the range where, length = the length of the extent, given by *EXTENT\_LENG(addr)*.
* We then repeat the aforementioned check for all the available extents in the inode. If the requested file is found, then bmap() returns the corresponding block address. Refer for Figure 1 for code.



**Figure 1: Extent-handling portion of bmap() function in fs.c**

* If the block is not found, **bmap()** allocates space for a new block. If the new block is contiguous with respect to the last extent in the inode, we increase its length. ddw
* If not, we add a new entry in the inode address table and return the disk address.
* If however the entries are greater than equal to NDIRECT, then a kernel **panic()** is thrown as this should not be possible in the first place. Refer to Figure 2 for code.

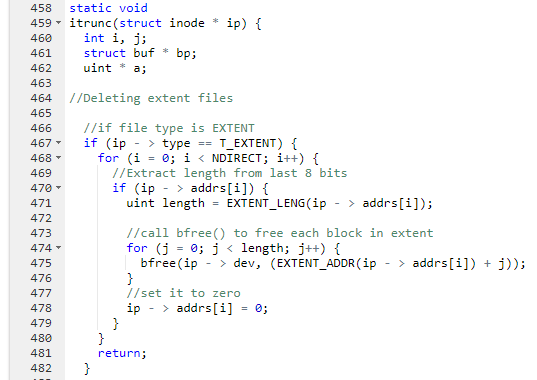


**Figure 2: Extent-handling portion of bmap() function in fs.c**

**Deleting Extent-Based Files**

Design Goals

* Functionality for deleting extent-files has been inserted into **itrunc().**
* If the type of file is T\_EXTENT, **bfree()**is called for each block in each extent.
* When a ﬁle is to be deleted, the system call **unlink()** eventually calls **itrunc()** which frees up the inodes (by calling **bfree()**).

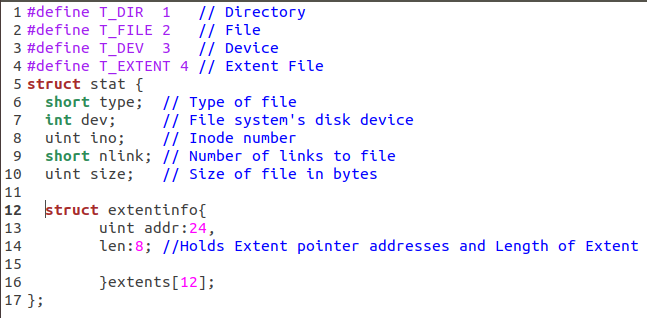


**Figure 3: Shows extent-files are deleted/de-allocated**

Question 2: Modifications to fstat()

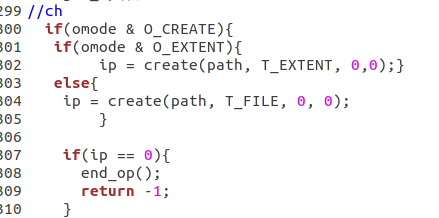
The following files are modified to implement extent-based file system in xv6. Note that fstat() is not directly changed, rather the functions and structs that fstat() uses, has been changed to include extent functionality.

* **fs.c:** **stati()** function has been changed.

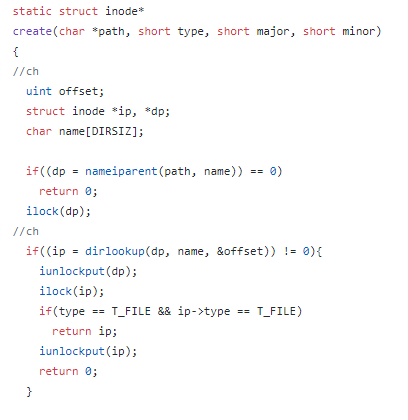


**Figure 4: The struct stat is shown here. Notice in Line 4, T\_EXTENT definition has been added. Also, in Lines 12-16, extent functionality has been added.**

* **sysfile.c:** The following code has been changed in the sys\_open() function. Modifying sys\_open() in this way ensures that extent-based file functionality can be added to xv6 while also ensuring backward compatibility.

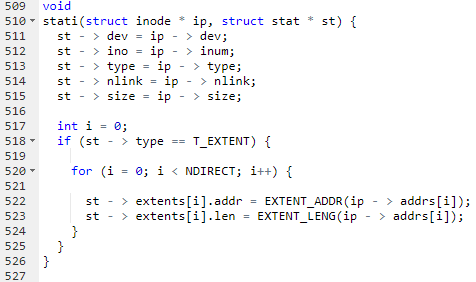


**Figure 5: The function sys\_open() has been changed so that extent-based files can be created.**

****

**Figure 6: The function create() has been changed so that extent-based files can be created.**

* **stat.h**: The struct **stat** has been changed to give it extent functionality.

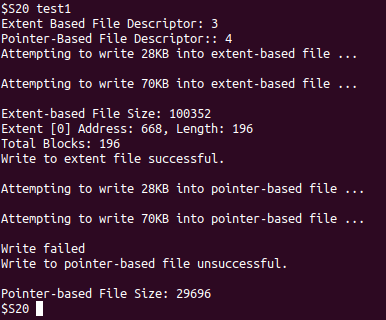


**Figure 7: The function stati has been shown here. Lines 517-525 have been added to give xv6 extent-based file system functionality.**

**Testing Extent-Based Files in xv6**

test1.c: This test attempts to write 98KB into an extent-based file and a pointer-based file separately. We know that pointer-based files can store upto 70KB of data in them. Thus, a successfully executed extent-based file should be able to store 98KB of data in it, whereas a pointer-based file would not be able to. This test also makes use of the **fstat()** function so this serves as a test to see if fstat() is running successfully too.

Results:



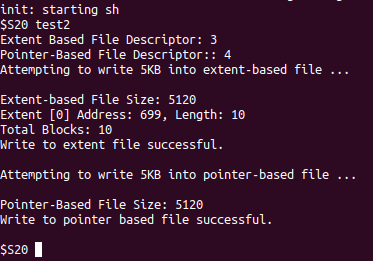
**Figure 8: Test 1 results**

Attempt to write 98KB into an extent file should be successful.

Attempt to write 98KB into a pointer-based file should be unsuccessful.

test2.c: This test attempts to write 5KB into an extent-based file and a pointer-based file separately. It should be possible to write 5KB into both types of files successfully and that is what is observed when test2 is run. This test also makes use of the **fstat()** function so this serves as a test to see if fstat() is running successfully too.

Results:



**Figure 9: Test 2 results**

Attempt to use fstat() for extent file: successful.

Attempt to use fstat() for pointer-based file: successful.