**Implementation Details**

Andrew Woltman & Clive Gomes

A20350925 & A20409748

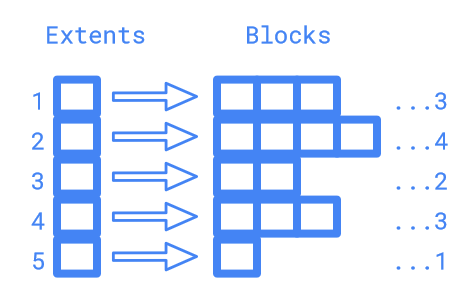
*Extent-Based File System →*

To add extent-based files to xv6, a new file type (T\_EXTENT = 4) was first added to the stat.h file. Additionally, a new flag (O\_EXTENT) was added to the fcntl.h file in order to represent extent-based files. (This flag was also included in the open() system call.) With these in place, the existing bmap() and itrunc() functions in fs.c were modified to work with both regular & extent-based files. Further explanations on these are given below. (Note: For minor modifications to files, refer to the changeLog documentation).

**bmap():** An if statement was added to check if the file is an extent-based file. The existing xv6 code was tucked away in the else statement for backward compatibility.

For extent-based files, the code first checks if an extent exists. If not, a new block is allocated, and its address and length=1 were saved in the inode — 3 byte address + 1 byte length = 4 byte inode addr. The block address is returned.

If an extent exists, the bnth block must first be located. The way this was done is best explained by the following example. Consider a file with these extents:



Suppose bn = 5. A block count n was used to keep track of the blocks being checked. All extents were looped through, one at a time. For each extent, its length is first added to n. For the first extent in the example, n = 0 + 3 = 3. This was then compared with bn. Since n < bn (3 < 5), the required block is not in the current extent, and the code, therefore, moves on to the next extent. Again, the process is repeated. This time, n = 3 + 4 = 7, which is greater than 5 (bn). This means that the required block is in the current extent. Its position in the current extent (5-3 = 2) is then calculated, and its address is computed by adding that number to the address contained in the extent (first block), which is then returned.

During the loop, if an empty extent is found, no further extents are checked because extents are always allocated consecutively. Accordingly, a new block is allocated, and its address is returned (after storing the address & length=1 in the inode).

**itrunc():** An if statement was added to check if the file is an extent-based file. The existing xv6 code was tucked away in the else statement for backward compatibility. For extent-based files, the code loops through all extents of the given file. If an extent exists, it checks the length of that extent and loops through those many adjacent blocks in memory (starting at the address given in the extent), freeing them one at a time.

To dump information about each extent of extent-based files through the fstat() system call, first, two arrays (extentAddr & extentLen) were added to the stat struct in the stat.h file. The stati() function in fs.c was then modified to populate these arrays with appropriate information about the required file. This was done by looping through all extents of the file, checking if they contain valid information, and if so, saving their values in the arrays.

*The lseek() Syscall →*

In order to set a files offset, lseek() takes a file descriptor and offset value. The file struct is obtained using argfd(), the file descriptor and offset are obtained using argint(). Once the variables are obtained, The file descriptor is checked to ensure a valid file descriptor by making sure the value is positive and smaller than NOFILE(16, the size of a file struct's ofile array). If this fails, an error will be triggered and -2 will be returned. Then the offset is checked to ensure a valid offset value, for the offset to be valid it must be positive and smaller than the size of the file. The files size is obtained by using the files structure to obtain its inode structure, from there the size of the file is obtained. If this fails, an error will be triggered and -3 will be returned. If both the file descriptor and offset check are passed, the file structures offset value will be set and also returned.