**Test Data**

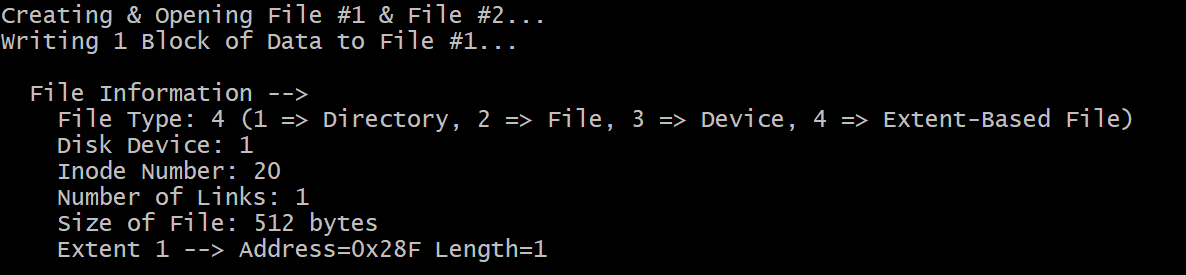
Andrew Woltman & Clive Gomes

A20350925 & A20409748

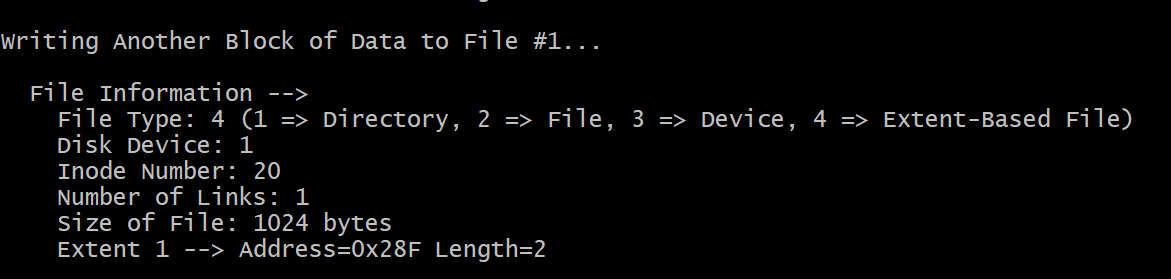
**Extent-Based File System**

To test the extent-based file system added to xv6, a test driver was created and a set of 6 tests were performed to test different scenarios which are explained in this document. Before running the tests, the max length of extents in the fs.h file was changed from 256 to 4 in order to speed up the testing. (Using the default length of 256 will cause the program to take extremely long!) The outputs at each step of testing were generated using the pstat() system call that was required to be modified to display information about file extents as well. Thus, both requirements 1 & 2 of this assignment were tested simultaneously.

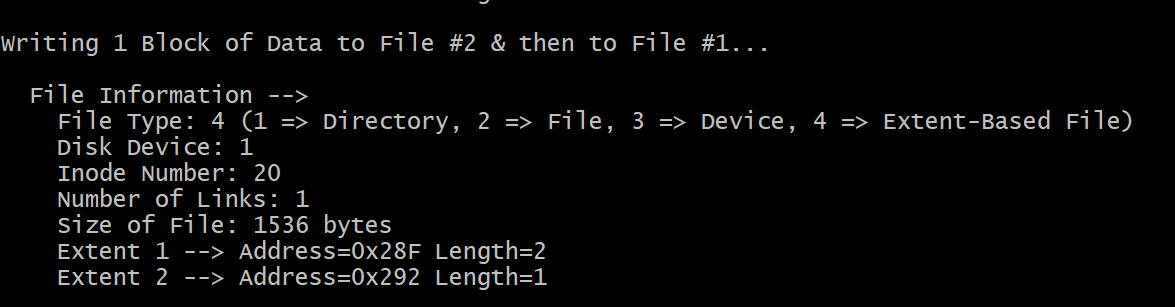
*Test Case #1:* In this simple test, a single block of data (256 bytes) was written to an empty file (File #1). We expect a single extent of length 1 to be created. This can be verified in the test results below:



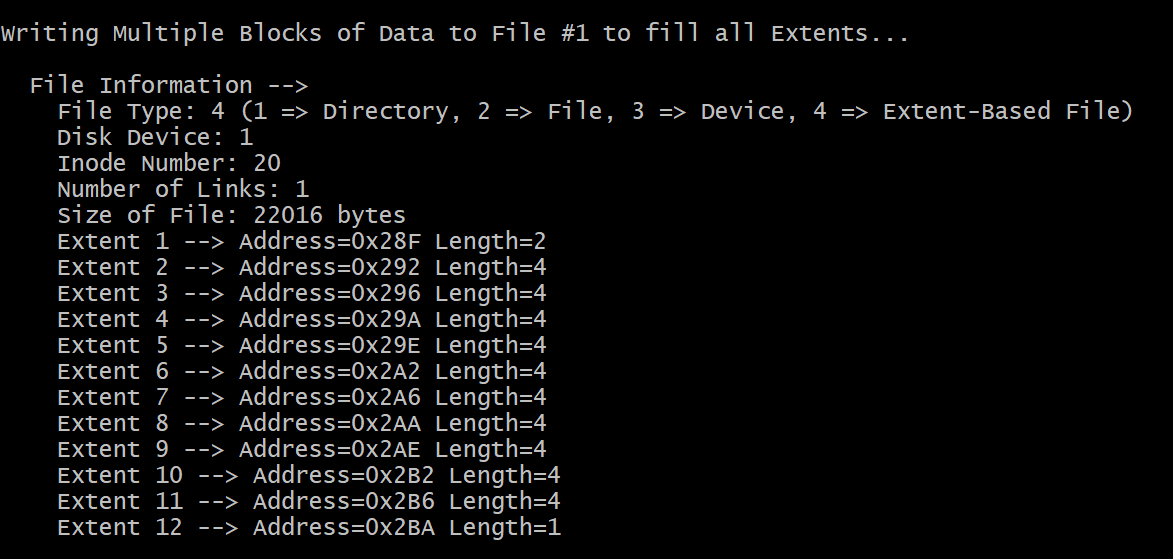
*Test Case #2:* Another block of data is written to File #1. Since there is more than 1 block for each extent (owing to its length attribute), we would expect the same extent to be used, though its length would be increased by 1. As can be seen in the test results below, the new length of 'Extent #1' is 2.



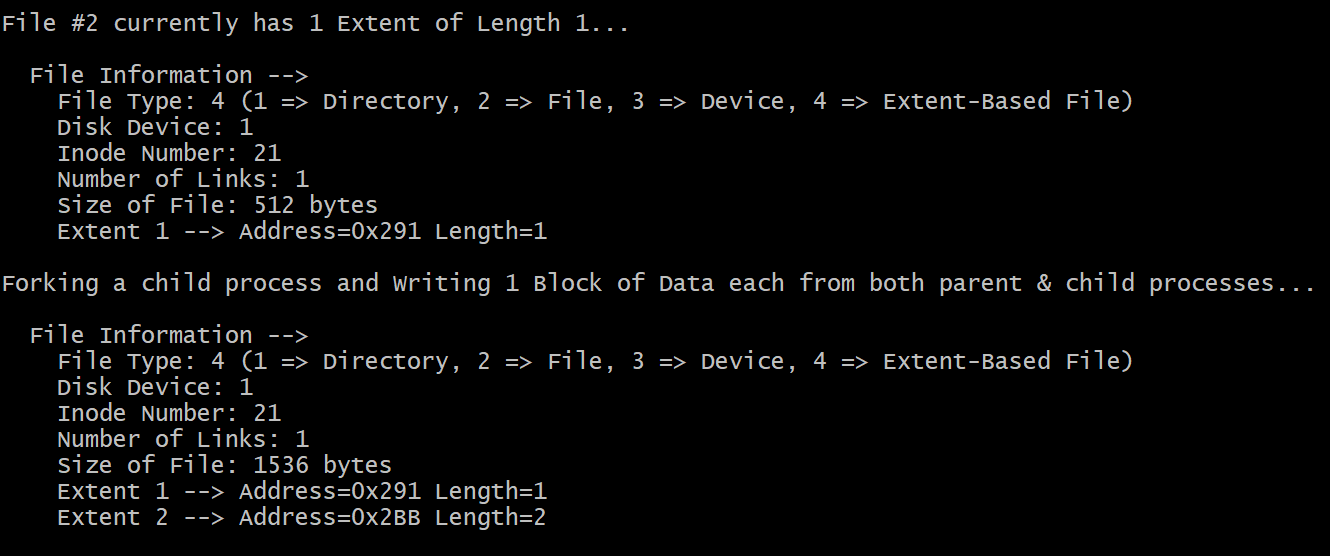
*Test Case #3:* This test checks what would happen if more than 1 file is being used by a process. Accordingly, a block of data is first written to File #2 & then to File #1. Since File #1 has already been written to before and has an extent of length 2 (which means it can take 2 more blocks of data as the max length has been set to 4), we would expect its length to increase to 3. However, the test results revealed that instead of doing so, an entirely new extent with length of 1 was created. The results below demonstrate this:

**

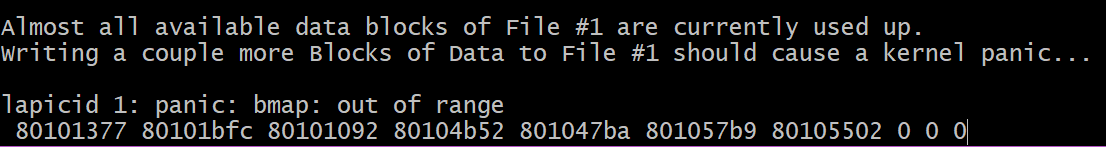
*Test Case #4:* We now write a bunch of blocks of data to File #1. We would expect consecutive extents to be used once each has a maximum length of 4. This can be verified in the test results below:



*Test Case #5:* Here, we test what would happen if multiple processes write to the same file (File #2). For this, we fork a child process, and then write a block of data each, from the two processes. We would expect a new extent of length 2 to be created, as another file was written to previously. This is seen in the test results below:



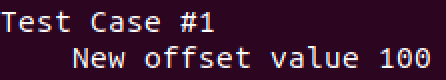
*Test Case #6:* Finally, we test to see what would happen if all blocks of data available to a file are used up & we try to write more data to it. As expected, a kernel panic occurs & xv6 crashes. This can be seen below:



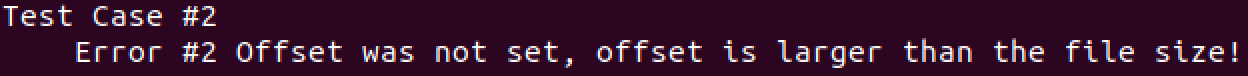
**The lseek() Syscall**

To test the lseek() system call added to xv6, a test driver was created and a set of 6 tests were performed to test different scenarios which are explained in this document. First a file of both EXTENT and NORMAL types were created and 512 bytes of data written to them. Pstat() was then modified to print either the offset that is set, an error stating the offset is too large(Error #2) or an error stating the file descriptor was invalid(Error #1). Then pstat() is called with a file and the return value of lseek() for each test case.

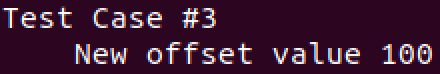
*Test Case #1:* In this simple test, 512 bytes were written to an empty EXTENT file (File #1). Then we passed the file descriptor and offset(100 bytes) to lseek. We expected the offset of the file would be set to 100. This can verified in the test below:

****

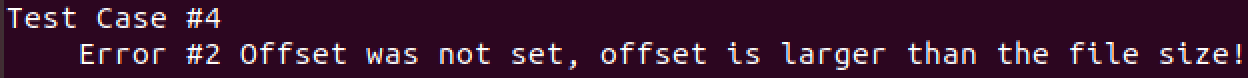
*Test Case #2:* In this simple test, 512 bytes were written to an empty EXTENT file (File #1). Then we passed the file descriptor and offset(640 bytes) to lseek. We expected the offset would give Error #2. This can verified in the test below:

****

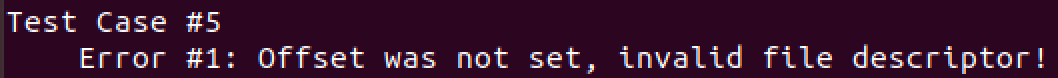
*Test Case #3:* In this simple test, 512 bytes were written to an empty NORMAL file (File #2). Then we passed the file descriptor and offset(100 bytes) to lseek. We expected the offset of the file would be set to 100. This can verified in the test below:

****

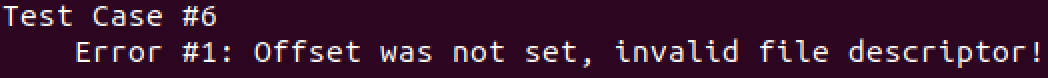
*Test Case #4:* In this simple test, 512 bytes were written to an empty NORMAL file (File #2). Then we passed the file descriptor and offset(640 bytes) to lseek. We expected the offset would give Error #2. This can verified in the test below:



*Test Case #5:* In this simple test, 512 bytes were written to an empty EXTENT file (File #1). Then we passed a negative file descriptor and offset(100 bytes) to lseek. We expected the offset would give Error #1. This can be verified in the test results below:

****

*Test Case #6:* In this simple test, 512 bytes were written to an empty EXTENT file (File #1). Then we passed a file descriptor larger than 16 (NOFILE) and offset(100 bytes) to lseek. We expected the offset would give Error #1. This can be verified in the test results below:

****