Extenttest.c

```
*** TEST 1: Checking READ/WRITE existing file without O_EXTENT flags ***

Type of the file: T_FILE
File system's disk device: 1
Inode Number: 2
Number of links to the file: 1
Size of the file in bytes: 2170
```

For test 1, we can see that the type of file is T_FILE since we didn't specify the O_EXTENT flag. The file system's disk device is 1, the Inode number is 2, the number of links to the file is 1, adn the size of the file in bytes is 2170.

```
*** TEST 2: Checking READ/WRITE with O_EXTENT flags ***

Type of the file: T_EXTENT
File system's disk device: 1
Inode Number: 21
Number of links to the file: 1
Size of the file in bytes: 51

--- ADDRESS INFO ---
Pointer: 279
Length: 1
```

For test 2, we used the O_EXTENT flag and checked it with a file of READ/WRITE. As we expected, the type of file was shown as T_EXTENT. The file system's disk device is 1, inode number is 21, the number of links to the file is 1, and the size of the file in bytes is 51. Since the type of file is T_EXTENT, we can see its address information: a pointer of 279 and a size (length) of 1 byte.

```
*** TEST 3: Checking lseek functionality with O_EXTENT flags ***
sys_lseek SUCCESS: Offset before lseek update: 0
sys_lseek SUCCESS: Inode in-memory size before offset: 0
sys_lseek SUCCESS: Offset after lseek update: 200
sys_lseek SUCCESS: Inode in-memory size after offset: 200

Type of the file: T_EXTENT
File system's disk device: 1
Inode Number: 22
Number of links to the file: 1
Size of the file in bytes: 257

--- ADDRESS INFO ---
Pointer: 27a
Length: 1
```

As for test 3, we tested the functionality of both the O_EXTENT flag and lseek() as a function. Again, the file type here is displayed as T_EXTENT since the O_EXTENT flag was raised. The file system's disk device is 1, the inode number is 22, the number of links to the file is 1. In lseek(), we specified the offset as 200. This means that the file size should be 257 since 200 + 57 = 257. The size of the file of bytes in reality was 257, further showing the functionality of lseek(). The pointer is 27a and the length is 1 bytes.

We could say that equivalence partitioning was used for the above test cases since we divided the test cases, and so the possible outcomes, into coherent pieces by using a driver that encompasses 3 tests.

Lseektest.c

```
 * * * TEST 1: Valid File, Valid Offset * * *
File descriptor: 3
sys_lseek SUCCESS: Offset before lseek update: 0
sys_lseek SUCCESS: Inode in-memory size before offset: 0
sys_lseek SUCCESS: Offset after lseek update: 15
sys_lseek SUCCESS: Inode in-memory size after offset: 15
```

The first test case in the lseektest.c file takes in a valid file with a valid offset. The results were successful as we can see above. The value to which we set the offset to is 15 and the file descriptor used is for the file "test1". We can see that the offset and the inode in-memory size before using lseek() were both 0. After using the syscall, they went up to 15.

```
 * * * TEST 2: Valid File, Invalid Offset * * *
File descriptor: -1
sys_lseek ERROR: Input Invalid
ERROR: Couldn't change file offset
```

In the second test case, we used a valid file, test1, and an invalid offset and so we need to have an error displayed. Our expectations were right as the program returned a -1, displaying the error message that the input was in invalid and so the file offset wasn't changed.

```
 * * * TEST 3: Invalid File, Valid/Invalid Offset * * *
File descriptor: -1
sys_lseek ERROR: Input Invalid
ERROR: Couldn't change file offset
```

In the third test case, we used an invalid file, and a valid/invalid test. An error message was displayed, letting the user know that the input is invalid and the file offset couldn't be changed.

We could say that equivalence partitioning was used for the above test cases since we divided the test cases, and so the possible outcomes, into coherent pieces by using a driver that encompasses 3 tests.