Ritwik Divakaruni
Lin Abu-Amara

Due Date: 4/26/2020

Manual Pages

lseek()

Name

lseek - sets the current offset of open file to offset specified in the argument

Synopsis

int lseek(int fd, int offset)

Descriptions

lseek() is a system call that takes in a file descriptor and an integer type offset. If the call succeeds, the current offset of the file in place, specified by the file descriptor, is set to the offset specified in the arguments of the system call. It has the ability of working in both extend-based and pointer-based files.

Return Value

If the call succeeds, the value of the new offset of the open file is returned. If the call fails, an error message is outputted depending on what made it fail.

Bugs

The wording of the system call is vague and does not make it clear in regards to its functionality.

Colophon

This is a manual page released as part of the 4th programming assignment of CS 450 course.

Notes

This system call is designed within an xv6 environment and so its functionality is dependent on the logic found in such operating systems. Similar system calls could be found in different environments but with different syntax.

Functionality for lseek():

The **lseek()** function enables the user to set the current offset of the file open identified by its file descriptor to a new offset. It works by first checking whether the file descriptor, the corresponding struct files, and the argument are valid. In the case it is invalid, an error message is displayed letting the user know that the input chosen for the **lseek()** function is invalid. Next we check the actual validity of the offset and the file descriptor. In case one or the other is invalid, a different error message is displayed letting the user know that the offset is invalid.

If the function passes the two checks mentioned above, a third check is made to check whether the file descriptor type is of a file descriptor Inode. If this check fails, an error message is displayed, otherwise, the file descriptor of the open file is locked through **ilock()** so that the function can display to the user what the offset and Inode memory size were before any changes made by **lseek()**.

Afterwards, the offset is set to the one specified in lseek, and in case the offset is larger than the Inode in-memory size , then the size is set to the offset. **lseek()** then displayed the offset and the Inode in-memory size after the changes performed by lseek, eventually unlocking the file descriptor inode using **iunlock()** and returning the offset.

With a system call like this, driver tests are needed to show the user behind the prompt how the system call will be working. And so a driver test encompassing 3 different tests is made as follows:

lseektest.c

Test 1

This test takes in a valid file and a valid offset. The file is opened and the file descriptor is displayed. If **lseek()** fails, an error message is displayed letting the user know that it failed. The file is then closed.

Test 2

This test takes in a valid file and an invalid offset. The file is opened and the file descriptor is displayed. when **lseek()** fails, an error message is displayed letting the user know that it failed. The file is then closed.

Test 3

This test takes in an invalid file and an invalid offset. The file is opened and the file descriptor is displayed. when **lseek()** fails, an error message is displayed letting the user know that it failed. The file is then closed.

Ritwik Divakaruni PA4 Due Date: 4/26/2020
Lin Abu-Amara

O_EXTENT:

The **O_EXTENT** flag is created for the open() system call in order to make an extent based file.**O_EXTENT** flag is first defined in the *fcntl.h* file along with the other flags of **open()**. In order to create our new flag, we also need to add another flag **T_EXTENT** for a new type of file for extent which can be done in *stat.h*. The open syscall is edited to include our new flag **O_EXTENT**. If it's provided in the open syscall, a **T_EXTENT** type file is created.

printstat()

Printstat is a system call that prints out information about the file. This information includes the type of the file (T_FILE, T_DIR, T_EXTENT, T_DEV), the file system's disk device, Inode Number, Number of links to the file, and the size of the file in bytes. Through this syscall, we will be able to test our **O_EXTENT** flag properly.

With a system call like this, driver tests are needed to show the user behind the prompt how the system call will be working. And so a driver test encompassing 3 different tests is made as follows:

extenttest.c

Test 1

This test checks the read/write existing file without **O_EXTENT**. It opens a file, README, and prints information about the file using **printstat()** and **fstat()**. Since we did not use the **O_EXTENT** flag, **T_FILE** creation should bring up a success result.

Test 2

This test checks the read/write existing file with **O_EXTENT**. It opens a file, extenttest2, and prints information about the file using **printstat()** and **fstat()**.

Test 3

This test checks the **lseek()** functionality with **O_EXTENT**. It opens a file, extenttest3, and prints information about the file using **printstat()** and **fstat()**. **lseek()** is used to set the offset to 200 for the file corresponding to the file descriptor fd3. The size of the file should be 257 as opposed to 57 since the offset is set to 200.

Ritwik Divakaruni PA4 Due Date: 4/26/2020
Lin Abu-Amara

Manual Pages

printstat()

Name

printstat- displays the type of file, the file system's disk device, the inode number, the number of links to the file, the size of the file in bytes.

Synopsis

int printstat(&st)

Descriptions

printstat() is a system call that takes in a stat structure and displays information about the file. This information includes the type of the file, the file system's disk device, inode number, number of links to the file, size of the file (in bytes), and the pointer and size provided in case the files were extent based.

Return Value

The display of information through **cprintf()** as mentioned above is what the system call returns.

Bugs

The system call is case sensitive. None of the letters are capitalized in **printstat()**. Failure to adhere to this results in an error. The name of the system call is also not very clear in regards to what it is used for.

Colophon

This is a manual page released as part of the 4th programming assignment of CS 450 course.

Notes

This system call is designed within an xv6 environment and so its functionality is dependent on the logic found in such operating systems. Similar system calls could be found in different environments but with different syntax.