ECEN 5013

Assignment 9: Advanced Char Driver Operations

Github Classroom Link

https://classroom.github.com/a/J8YjyTm9

Suggested Reading:

1. Lecture 19.
2. Linux Driver Development Chapters 3-6

Github Classroom Start Instructions

You will start with an empty project submission repository, and fill it with the content of your assignment 8 local repository using these commands on your local buildroot-assignments repository

1. git remote rename origin assignment-8-remote
2. git remote add origin <your github classroom submission repo created with the link above>
3. git push origin master

Implementation:

1. Pick the Buildroot or Yocto implementation from Assignment 8 to use as the basis for your assignment and push to your submission repository. You can use either Buildroot or Yocto for this assignment.
2. Using your assignments 3 and later repository, complete the following steps to pull in content from the aesd-assignments repository assignment-9 branch

```
git fetch aesd-assignments
git merge aesd-assignments/assignment9
```

3. Update your “aesd-char-driver” implementation in your assignments 3 and later repository to add the following features:
    a. Add custom seek support to your driver using the llseek file_operations function. Your seek implementation should:

i. Calculate which of the most recent 10 commands correspond to the byte offset represented by the seek command. All positional types (SEEK_SET, SEEK_CUR, and SEEK_END) should be supported. The byte offset should be considered to correspond to a specific write if the offset points to a character in any position within the string.
    1. For instance, if your driver is storing writes of 5, 7, and 9 bytes each, an offset of 0-5 would represent a byte within the first write, an offset of 6-13 would represent a byte within the second write, and an offset of 14 through 23 would a byte within represent the 3rd write.

Eg: Consider the below content in your driver

Grass
Sentosa
Singapore

		Byte											
Command	Length	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Seek Offset Start of First Byte**	**Seek Offset Start of Last Byte**
1	5	G	r	a	s	s	\n	▨	▨	▨	▨	0	5
2	7	S	e	n	t	o	s	a	\n	▨	▨	6	13
3	9	S	i	n	g	a	p	o	r	e	\n	14	23

An offset of 14 would set the pointer to the start of the word “Singapore”.

ii. Use the current file offset to determine the content to return on the next read. (Write operations are not modified to use this offset and always append).
    1. For instance, in the example above, if the seek offset was calculated as position 7, the first character of the second write would not be returned in the next read command. The next read command would return “entosa\nSingapore\n”.

b. Add an ioctl command AESDCHAR_IOCSEEKTO which is passed as a buffer from user space containing two 4 byte values.
    i. The first value represents the command to seek into, based on a zero referenced number of commands currently stored by the driver in the command buffer.

            1. For instance, in the example above a value of 0 refers to the command “Grass\n”
        ii. The second value represents the zero referenced offset within this command to seek into. All offsets are specified relative to the start of the request.
            1. For instance, if the offset was 2 in the command “Grass”, the seek location should be the letter “a”
        iii. The two parameters above will seek to the appropriate place and update the file pointer as described in the seek support details within the previous step.
        iv. If the two parameters are out of range of the number of write commands/command length, the ioctl cmd should return -EINVAL

4. Add special handling for socket write commands to your aesdsocket user space utility
    a. When the string sent over the socket equals “AESDCHAR_IOCSEEKTO:X,Y” where X and Y are unsigned decimal integer values, the X should be considered the write command to seek into and the Y should be considered the offset within the write command.
        i. These values should be sent to your driver using the AESDCHAR_IOCSEEKTO ioctl described earlier. The ioctl command should be performed before any additional writes to your aesdchar device.
        ii. Do not write this string command into the aesdchar device.
        iii. Ensure the read of the file and return over the socket uses the same (not re-opened) file descriptor used to send the ioctl, to ensure your file offset is honored for the read command.

Validation:

1. Your qemu instance should pass the modified sockettest script which includes
2. Your implementation should not crash or have memory leaks.
3. Your implementation should use proper locking methods to ensure safe access from multiple threads or processes.
4. Your driver should include custom seek and ioctl commands as described in the implementation section.
    a. When behavior in a specific scenario is not specified in your driver requirements, you should use any best practices mentioned in the Linux Device Drivers book.
5. Your implementation should pass when running with the drivertest-assignment-9.sh script
6. Your aesdsocket application should include support for custom socket commands as described in the implementation section.
7. Your implementation should pass when running with the sockettest-assignment-9.sh script

8. Your project should contain build.sh, clean.sh, and runqemu.sh scripts which function as specified in previous assignments, and can successfully clean, build, and run your project without user interaction for automated build testing.

Submission:

1. Your submission repository should contain a working buildroot or yocto implementation.
2. Your buildroot or yocto implementation should reference a commit in your assignments 3 and later repository containing the code added for this assignment.