Assignment 9 Overview

Advanced Embedded Linux Development

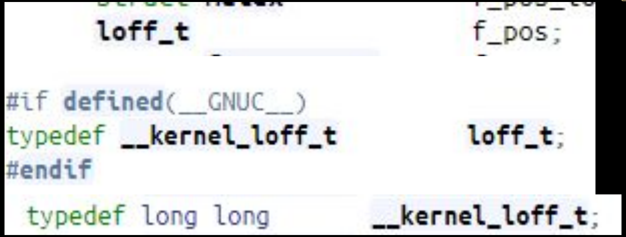
with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand Assignment 9 implementation/ioctl implementation

f_pos member



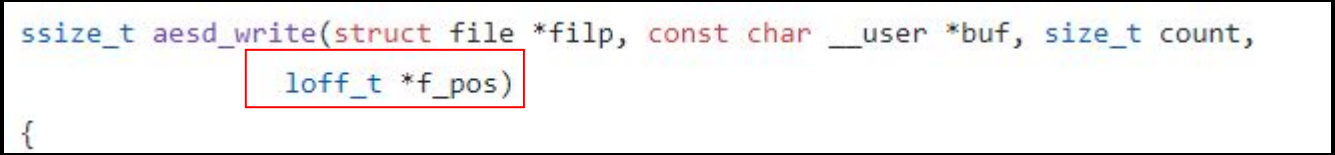
```
    loff_t                    f_pos;

#if defined(__GNUC__)
typedef __kernel_loff_t          loff_t;
#endif
 typedef long long          __kernel_loff_t;
```

- The file structure f_pos member:
  - References “The current reading or writing position”
    - 64 bit signed offset representing a byte offset in the file.

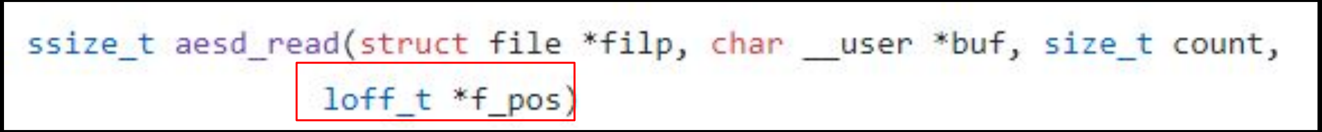
Linux Device Drivers 3rd Edition Chapter 3
https://unix.stackexchange.com/questions/274380/file-size-limit

f_pos member

- Passed as an input/output argument to read/write
    - Always 0 in write when using echo “blah” > /dev/aesdchar
        - Why?
        - > is an overwrite of the file, starts at offset 0



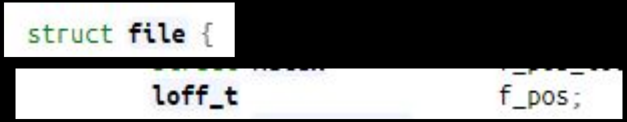
```
ssize_t aesd_write(struct file *filp, const char __user *buf, size_t count,
                loff_t *f_pos)
{
```



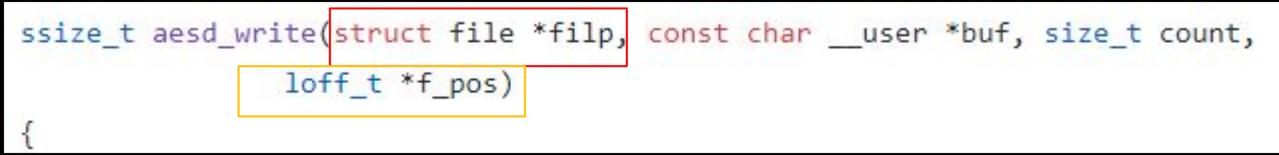
```
ssize_t aesd_read(struct file *filp, char __user *buf, size_t count,
                loff_t *f_pos)
```

f_pos member

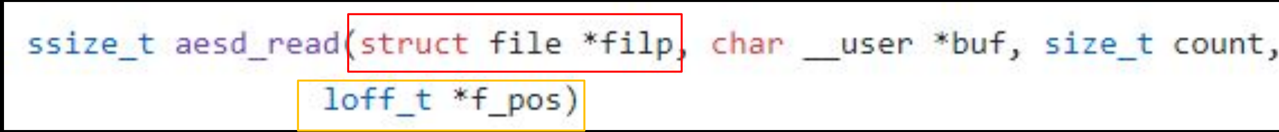
- Also included in struct file



```
struct file {
        loff_t                  f_pos;
```



```
ssize_t aesd_write(struct file *filp, const char __user *buf, size_t count,
                loff_t *f_pos)
{
```

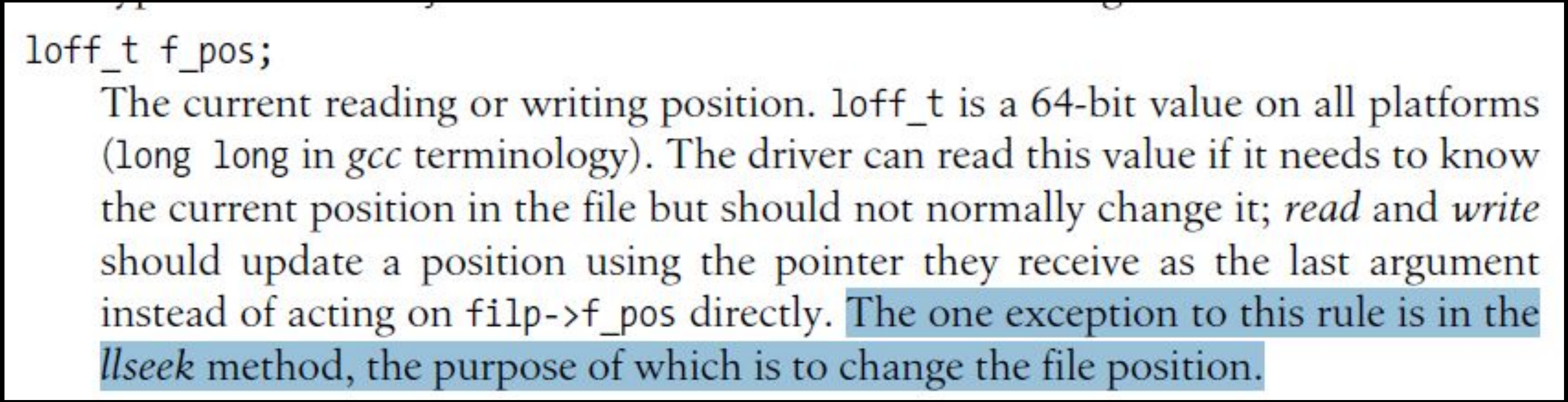


```
ssize_t aesd_read(struct file *filp, char __user *buf, size_t count,
                loff_t *f_pos)
```

- Why include in two places?
  - passed *f_pos is modified by read/write implementation
  - filp->f_pos is modified by underlying shared drivers based on read/write

Linux Device Drivers 3rd Edition Chapter 3
https://elixir.bootlin.com/linux/v5.3.8/source/include/linux/fs.h#L922

Seek Implementation



`loff_t f_pos;`

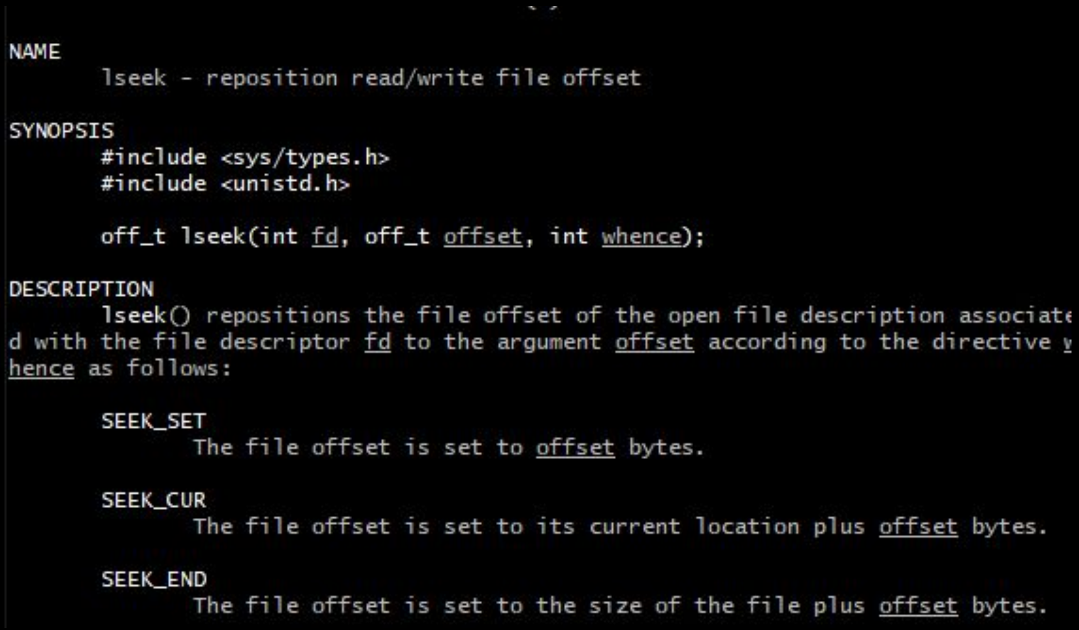
The current reading or writing position. `loff_t` is a 64-bit value on all platforms (`long long` in *gcc* terminology). The driver can read this value if it needs to know the current position in the file but should not normally change it; *read* and *write* should update a position using the pointer they receive as the last argument instead of acting on `filp->f_pos` directly. The one exception to this rule is in the *llseek* method, the purpose of which is to change the file position.

- We will implement llseek in assignment 9
- We will also modify filp->f_pos in an ioctl (wouldn’t typically be done this way)

Linux Device Drivers 3rd Edition Chapter 3
https://elixir.bootlin.com/linux/v5.3.8/source/include/linux/fs.h#L922

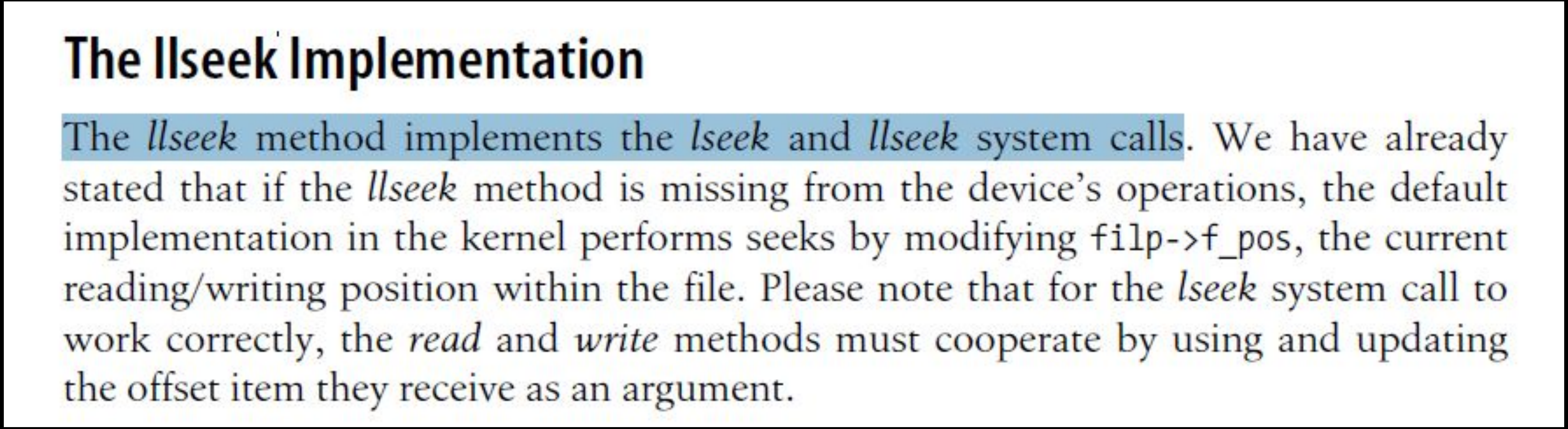
Reading Positionally

- SEEK_SET
  - Use the specified offset as the file position.
- SEEK_CUR
  - Increment or decrement file position
- SEEK_END
  - Use EOF as file position



```
NAME
       lseek - reposition read/write file offset

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       off_t lseek(int fd, off_t offset, int whence);

DESCRIPTION
       lseek() repositions the file offset of the open file description associate
d with the file descriptor fd to the argument offset according to the directive w
hence as follows:

       SEEK_SET
              The file offset is set to offset bytes.

       SEEK_CUR
              The file offset is set to its current location plus offset bytes.

       SEEK_END
              The file offset is set to the size of the file plus offset bytes.
```

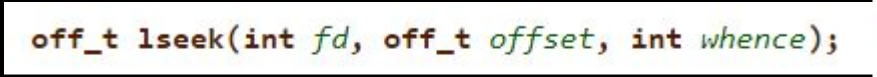
Seek Implementation



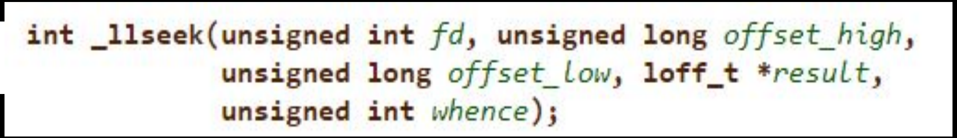
The llseek Implementation

The *llseek* method implements the *lseek* and *llseek* system calls. We have already stated that if the *llseek* method is missing from the device's operations, the default implementation in the kernel performs seeks by modifying `filp->f_pos`, the current reading/writing position within the file. Please note that for the *lseek* system call to work correctly, the *read* and *write* methods must cooperate by using and updating the offset item they receive as an argument.

- Why two different system calls, lseek and _llseek?
  - llseek is guaranteed to support long long offset sizes



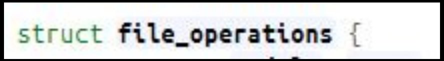
```
off_t lseek(int fd, off_t offset, int whence);
```



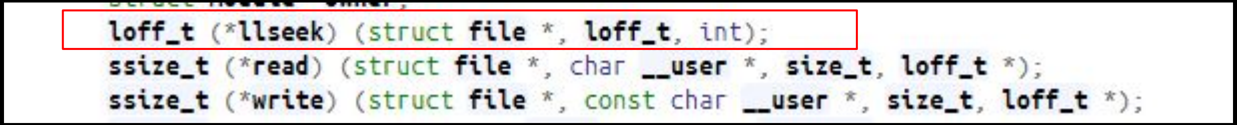
```
int _llseek(unsigned int fd, unsigned long offset_high,
            unsigned long offset_low, loff_t *result,
            unsigned int whence);
```

Linux Device Drivers 3rd Edition Chapter 3
http://man7.org/linux/man-pages/man2/llseek.2.html
http://man7.org/linux/man-pages/man2/lseek.2.html

llseek driver implementation



```
struct file_operations {
```



```
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
```

Linux Device Drivers 3rd Edition Chapter 3

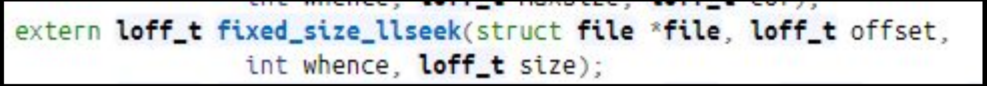
llseek driver implementation

- “If the llseek method is missing from the device’s operations, the default implementation in the kernel performs seek by modifying filp->fpos.”
- “For the lseek system call to work correctly the read and write methods must cooperate by using and updating the offset item they receive as an argument”

Linux Device Drivers 3rd Edition Chapter 3

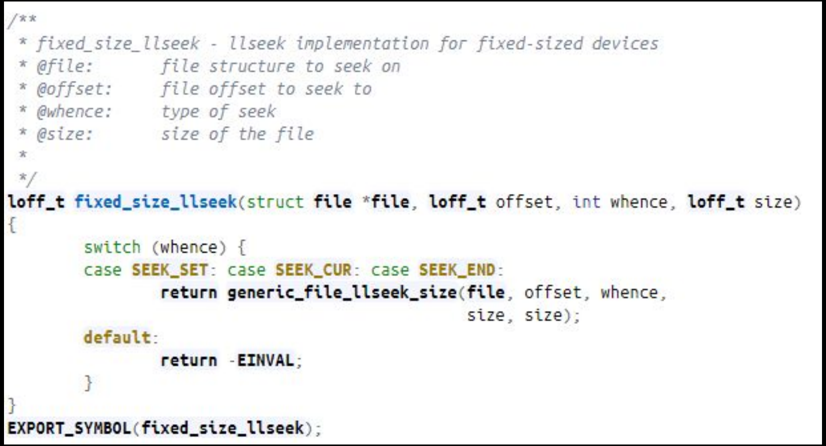
llseek driver implementation

- “If the llseek method is missing from the device’s operations, the default implementation in the kernel performs seek by modifying filp->fpos.”



```
extern loff_t fixed_size_llseek(struct file *file, loff_t offset,
                int whence, loff_t size);
```

- Several wrappers around generic llseek which seek for you which you can call from your llseek method
- This one uses a size you provide
- What should we use for size?
  - The total size of all content of the circular buffer



```
/**
 * fixed_size_llseek - llseek implementation for fixed-sized devices
 * @file:       file structure to seek on
 * @offset:     file offset to seek to
 * @whence:     type of seek
 * @size:       size of the file
 *
 */
loff_t fixed_size_llseek(struct file *file, loff_t offset, int whence, loff_t size)
{
        switch (whence) {
        case SEEK_SET: case SEEK_CUR: case SEEK_END:
                return generic_file_llseek_size(file, offset, whence,
                                                size, size);
        default:
                return -EINVAL;
        }
}
EXPORT_SYMBOL(fixed_size_llseek);
```

Linux Device Drivers 3rd Edition Chapter 3
https://elixir.bootlin.com/linux/v5.3.8/source/fs/read_write.c#L144
https://elixir.bootlin.com/linux/v5.3.8/source/include/linux/fs.h#L3078

llseek Assignment 9 Options

1. Leave llseek null and use the default llseek
   1. Would require supporting seek in write() (not assuming every write appends) which isn't an assignment requirement - I have not experimented with this myself
2. Add your own llseek function, with locking and logging, but use fixed_size_llseek for logic.
3. Implement your own llseek function separate from fixed_size_llseek handling each of the "whence" cases

I suggest option 2

Linux Device Drivers 3rd Edition Chapter 3
https://elixir.bootlin.com/linux/v5.3.8/source/fs/read_write.c#L144
https://elixir.bootlin.com/linux/v5.3.8/source/include/linux/fs.h#L3078

read/write method and f_pos

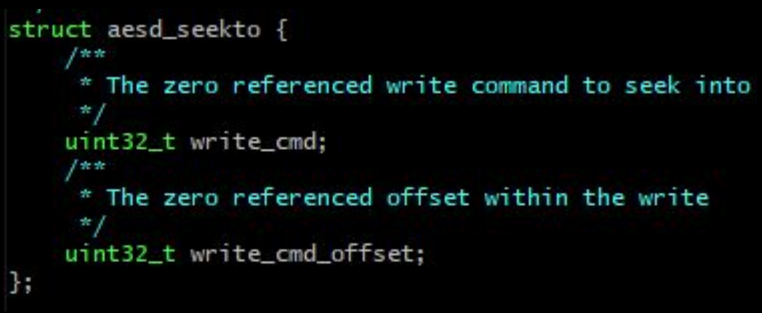
- “For the lseek system call to work correctly the read and write methods must cooperate by using and updating the offset item they receive as an argument”
  - read function:
    - Must set *f_pos to *f_pos + retcount where retcount is the number of bytes read
  - write function:
    - Must set *f_pos to *f_pos + retcount where retcount is the number of bytes written

Linux Device Drivers 3rd Edition Chapter 3
https://elixir.bootlin.com/linux/v5.3.8/source/fs/read_write.c#L144
https://elixir.bootlin.com/linux/v5.3.8/source/include/linux/fs.h#L3078

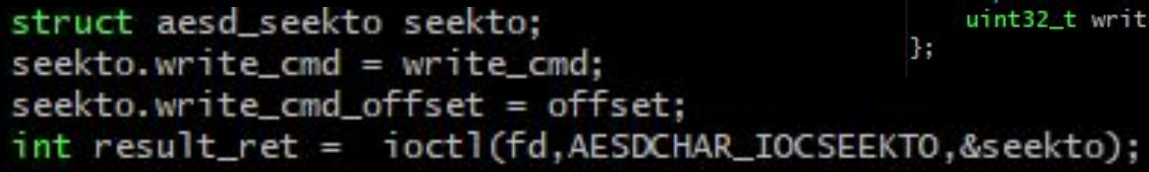
ioctl implementation

- Add an aesd_ioctl.h file you can share with your aesdsocket implementation
- See provided aesd_ioctl.h file at https://github.com/cu-ecen-aeld/aesd-assignments/blob/assignment9/aesd-char-driver/aesd_ioctl.h

ioctl user space implementation - aesdsocket

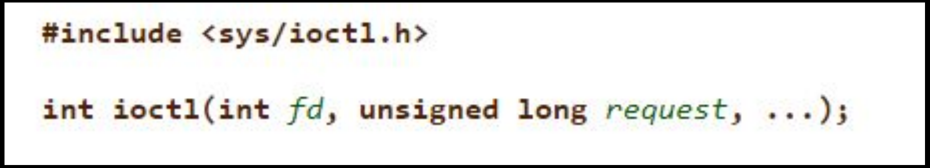


```
struct aesd_seekto {
    /**
     * The zero referenced write command to seek into
     */
    uint32_t write_cmd;
    /**
     * The zero referenced offset within the write
     */
    uint32_t write_cmd_offset;
};
```



```
struct aesd_seekto seekto;
seekto.write_cmd = write_cmd;
seekto.write_cmd_offset = offset;
int result_ret =  ioctl(fd,AESDCHAR_IOCSEEKTO,&seekto);
```

- #include the aesd_ioctl.h file
- Use the ioctl command with fd representing the driver
- Pass the filled in structure to the driver via ioctl

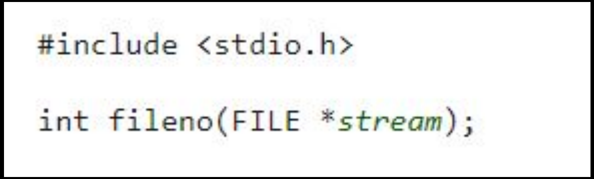


```
#include <sys/ioctl.h>

int ioctl(int fd, unsigned long request, ...);
```

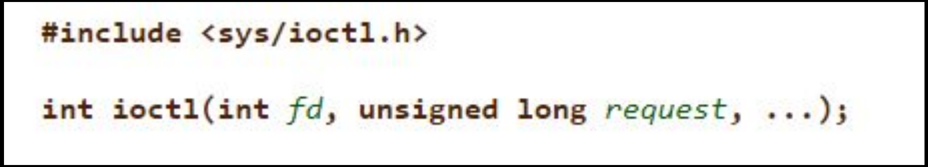
http://man7.org/linux/man-pages/man2/ioctl.2.html
https://github.com/cu-ecen-aeld/aesd-assignments/blob/assignment9/aesd-char-driver/aesd_ioctl.h

ioctl implementation - aesdsocket

- My aesdsocket uses a FILE* to access my driver. How do I get the fd used for ioctl?
  - Use fileno()



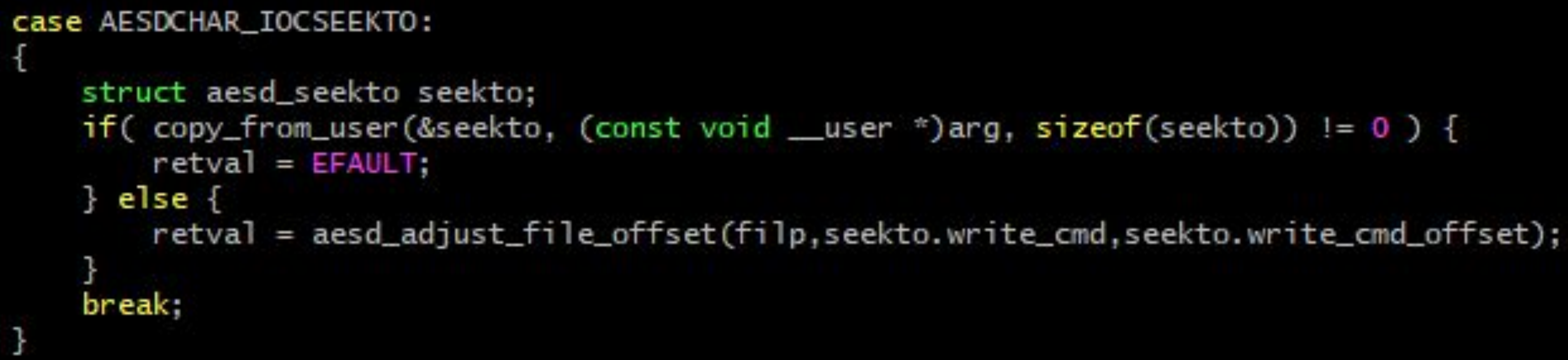
```
#include <stdio.h>

int fileno(FILE *stream);
```



```
#include <sys/ioctl.h>

int ioctl(int fd, unsigned long request, ...);
```

http://man7.org/linux/man-pages/man3/fileno.3p.html

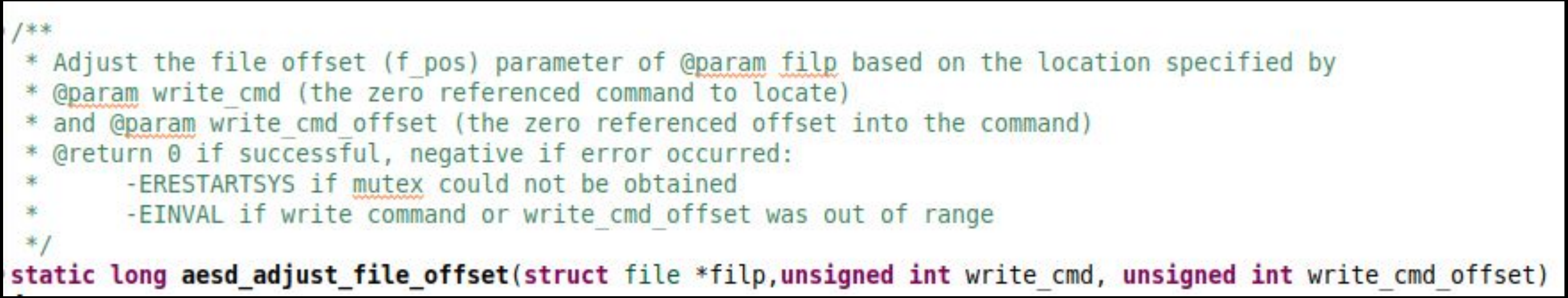
ioctl implementation - driver



```c
case AESDCHAR_IOCSEEKTO:
{
    struct aesd_seekto seekto;
    if( copy_from_user(&seekto, (const void __user *)arg, sizeof(seekto)) != 0 ) {
        retval = EFAULT;
    } else {
        retval = aesd_adjust_file_offset(filp,seekto.write_cmd,seekto.write_cmd_offset);
    }
    break;
}
```

- #include the aesd_ioctl.h file
- Use copy_from_user to obtain the value from userspace

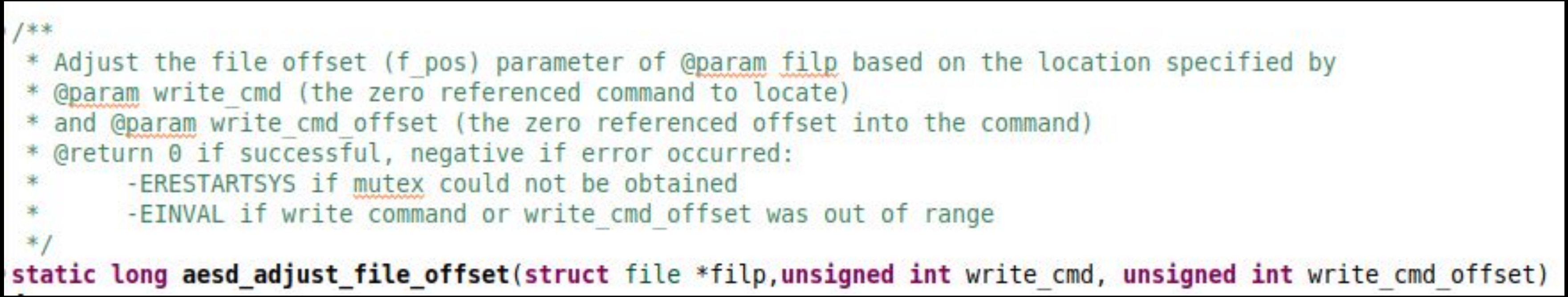
Adjusting the file offset from ioctl



```
/**
 * Adjust the file offset (f_pos) parameter of @param filp based on the location specified by
 * @param write_cmd (the zero referenced command to locate)
 * and @param write_cmd_offset (the zero referenced offset into the command)
 * @return 0 if successful, negative if error occurred:
 *      -ERESTARTSYS if mutex could not be obtained
 *      -EINVAL if write command or write_cmd_offset was out of range
 */
static long aesd_adjust_file_offset(struct file *filp,unsigned int write_cmd, unsigned int write_cmd_offset)
```

- Check for valid write_cmd and write_cmd_offset values
- Calculate the start offset to write_cmd
- Add write_cmd_offset
- Save as filp->f_pos

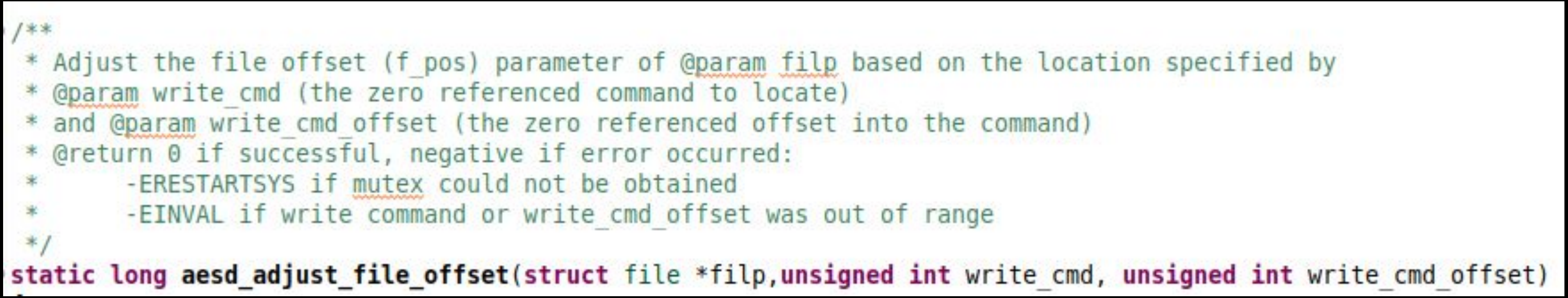
Adjusting the file offset from ioctl



```
/**
 * Adjust the file offset (f_pos) parameter of @param filp based on the location specified by
 * @param write_cmd (the zero referenced command to locate)
 * and @param write_cmd_offset (the zero referenced offset into the command)
 * @return 0 if successful, negative if error occurred:
 *      -ERESTARTSYS if mutex could not be obtained
 *      -EINVAL if write command or write_cmd_offset was out of range
 */
static long aesd_adjust_file_offset(struct file *filp,unsigned int write_cmd, unsigned int write_cmd_offset)
```

- Check for valid write_cmd and write_cmd_offset values
- When would values be invalid?
  - haven't written this command yet
  - out of range cmd (11)
  - write_cmd_offset is >= size of command

Adjusting the file offset from ioctl



```
/**
 * Adjust the file offset (f_pos) parameter of @param filp based on the location specified by
 * @param write_cmd (the zero referenced command to locate)
 * and @param write_cmd_offset (the zero referenced offset into the command)
 * @return 0 if successful, negative if error occurred:
 *      -ERESTARTSYS if mutex could not be obtained
 *      -EINVAL if write command or write_cmd_offset was out of range
 */
static long aesd_adjust_file_offset(struct file *filp,unsigned int write_cmd, unsigned int write_cmd_offset)
```

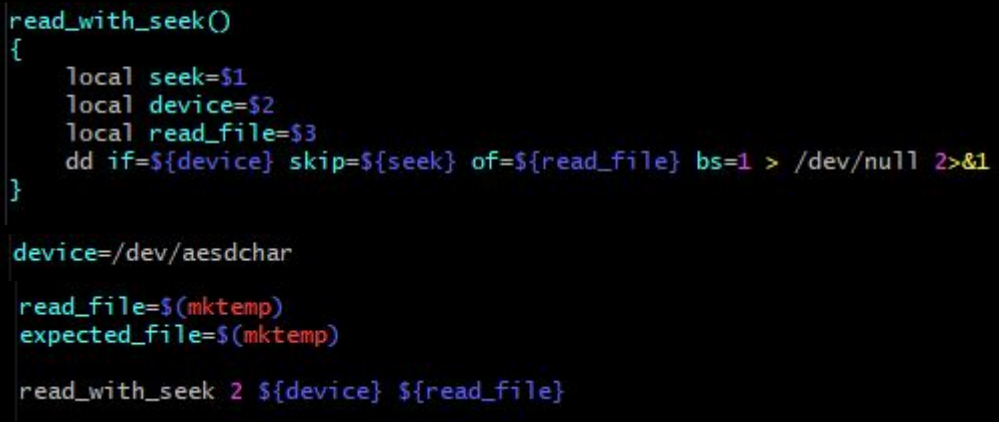
- Calculate the start offset to write_cmd
    - Add length of each write between the output pointer and write_cmd
- Add the write_cmd_offset

New Test Scripts

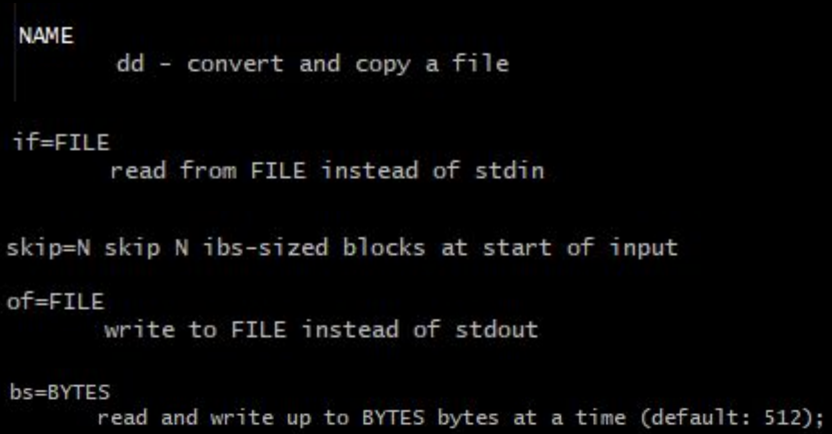
- drivertest.sh and sockettest.sh scripts should continue to pass with the changes in this assignment.
- New scripts assignment9/drivertest.sh and assignment9/sockettest.sh should also succeed.
    - You will need to stop/restart your driver when running the sockettest scripts (since we aren’t supporting command delete).

Seeking on the command line

- Use the dd utility to perform seeking on the command line



```
read_with_seek()
{
    local seek=$1
    local device=$2
    local read_file=$3
    dd if=${device} skip=${seek} of=${read_file} bs=1 > /dev/null 2>&1
}

device=/dev/aesdchar

read_file=$(mktemp)
expected_file=$(mktemp)

read_with_seek 2 ${device} ${read_file}
```



```
NAME
       dd - convert and copy a file

if=FILE
       read from FILE instead of stdin

skip=N skip N ibs-sized blocks at start of input

of=FILE
       write to FILE instead of stdout

bs=BYTES
       read and write up to BYTES bytes at a time (default: 512);
```

http://man7.org/linux/man-pages/man1/dd.1.html
https://github.com/cu-ecen-5013/aesd-assignments/blob/assignment9/aesd-char-driver/drivertest-assignment-9.sh