Assignment 7

Advanced Embedded Software Development

with **Dan Walkes**

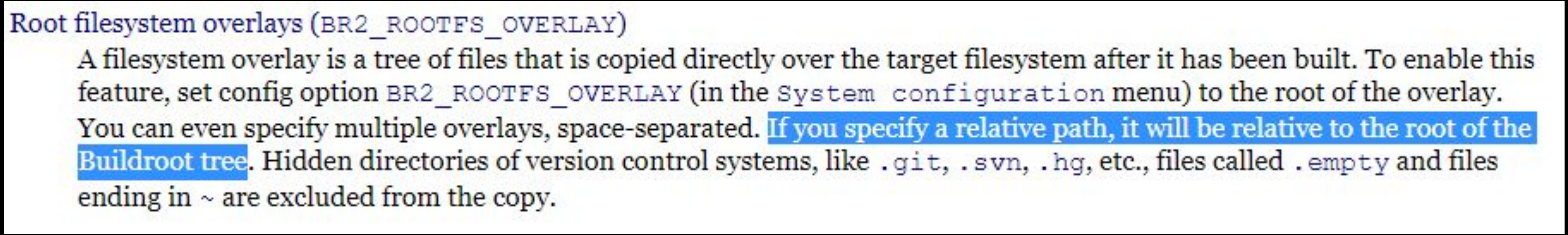
University of Colorado **Boulder**

Learning objectives:

Introduce Buildroot rootfs-overlays
Introduce Buildroot Kernel Module Support
Circular Buffer Implementation

Buildroot rootfs-overlay

- A way to add content to your root filesystem or override content from other packages
- The content you add will be placed in the rootfs at the specified path
  - Use a relative path so it works outside your implementation directory



Root filesystem overlays (BR2_ROOTFS_OVERLAY)

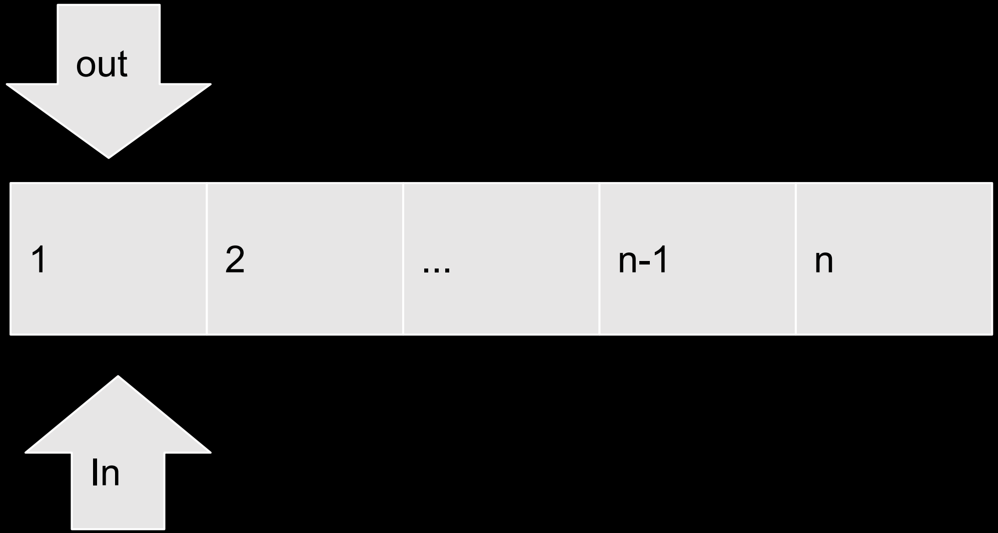
A filesystem overlay is a tree of files that is copied directly over the target filesystem after it has been built. To enable this feature, set config option BR2_ROOTFS_OVERLAY (in the System configuration menu) to the root of the overlay. You can even specify multiple overlays, space-separated. If you specify a relative path, it will be relative to the root of the Buildroot tree. Hidden directories of version control systems, like .git, .svn, .hg, etc., files called .empty and files ending in ~ are excluded from the copy.

https://buildroot.org/downloads/manual/manual.html#rootfs-custom

Buildroot Kernel Module Support

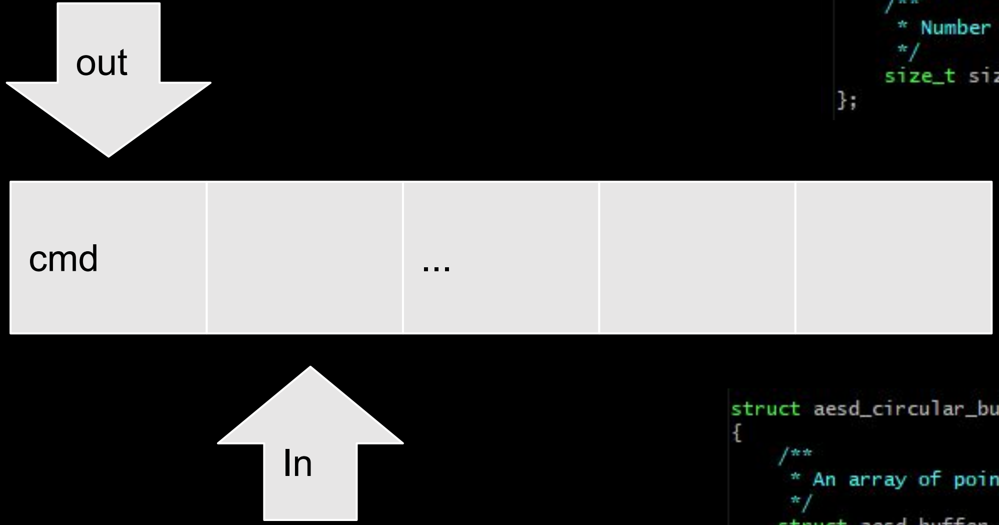
- See https://buildroot.org/downloads/manual/manual.html#_infrastructure_for_packages_building_kernel_modules
- This stackoverflow post may also be helpful: https://stackoverflow.com/a/43874273/1446624

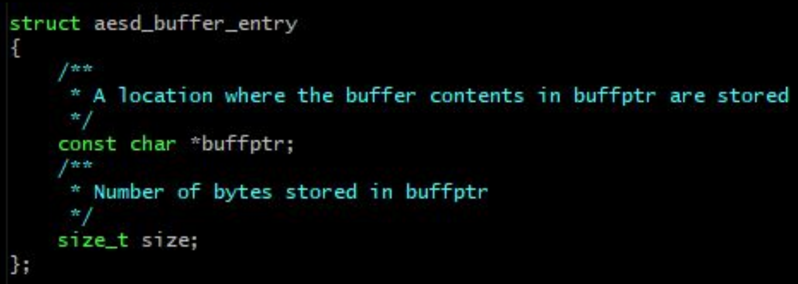
Circular Buffer



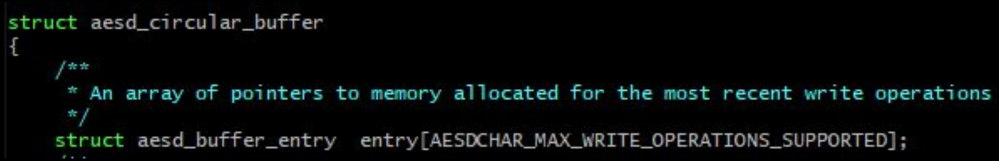
- Initial state - empty

Circular Buffer





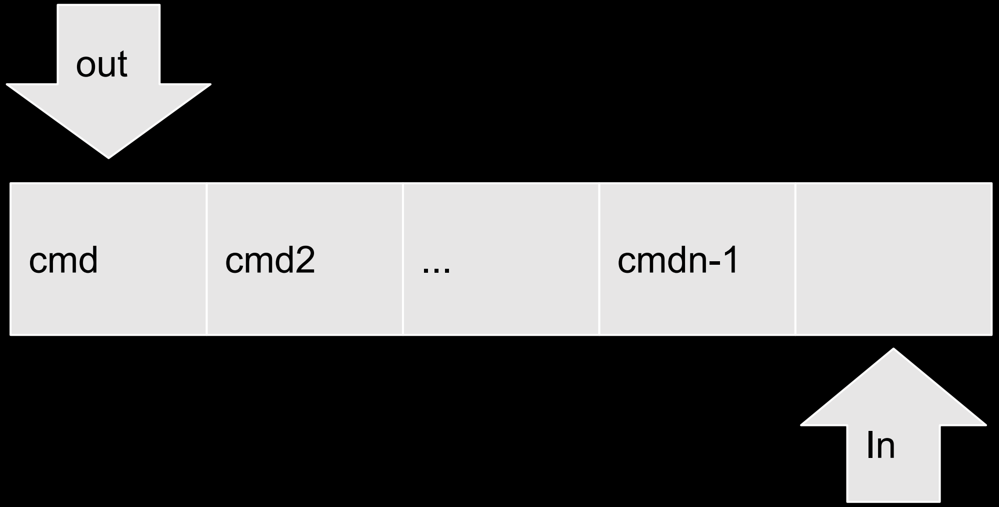
```
struct aesd_buffer_entry
{
    /**
     * A location where the buffer contents in buffptr are stored
     */
    const char *buffptr;
    /**
     * Number of bytes stored in buffptr
     */
    size_t size;
};
```



```
struct aesd_circular_buffer
{
    /**
     * An array of pointers to memory allocated for the most recent write operations
     */
    struct aesd_buffer_entry  entry[AESDCHAR_MAX_WRITE_OPERATIONS_SUPPORTED];
```

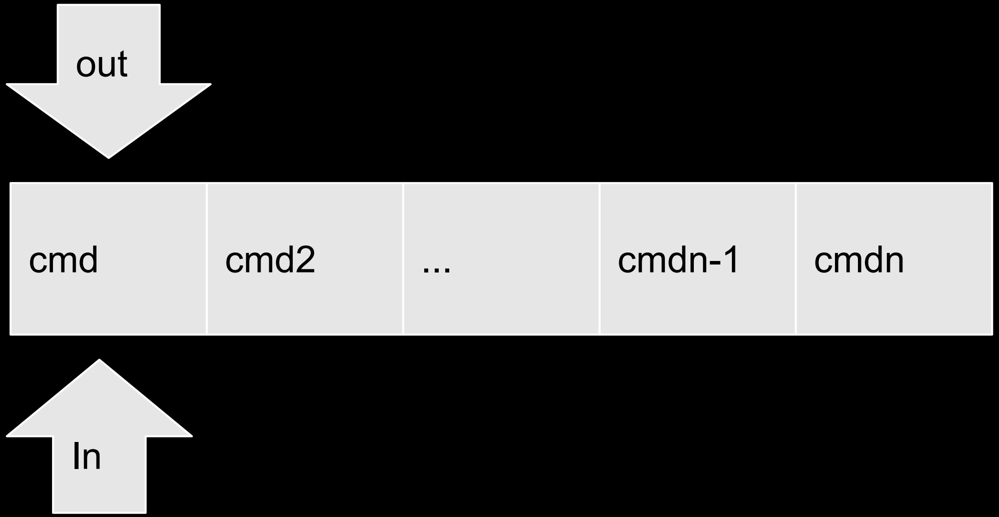
- One command written

Circular Buffer



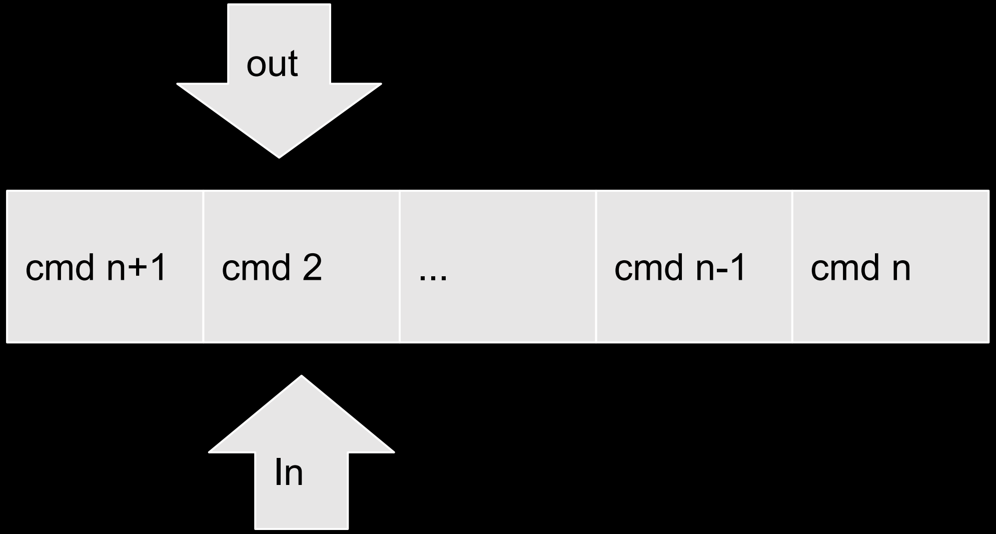
- Room for one item left in buffer

Circular Buffer



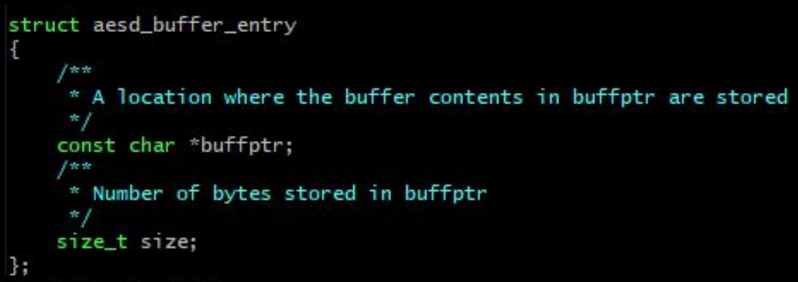
- Buffer is full

Circular Buffer

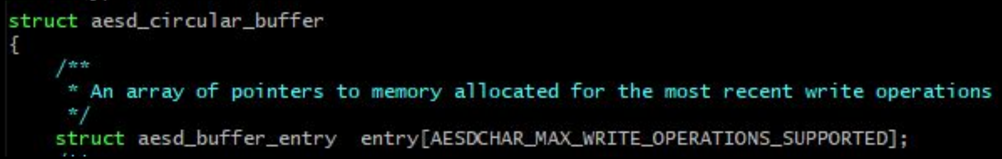


- Buffer is full, output pointer also needs to be advanced.

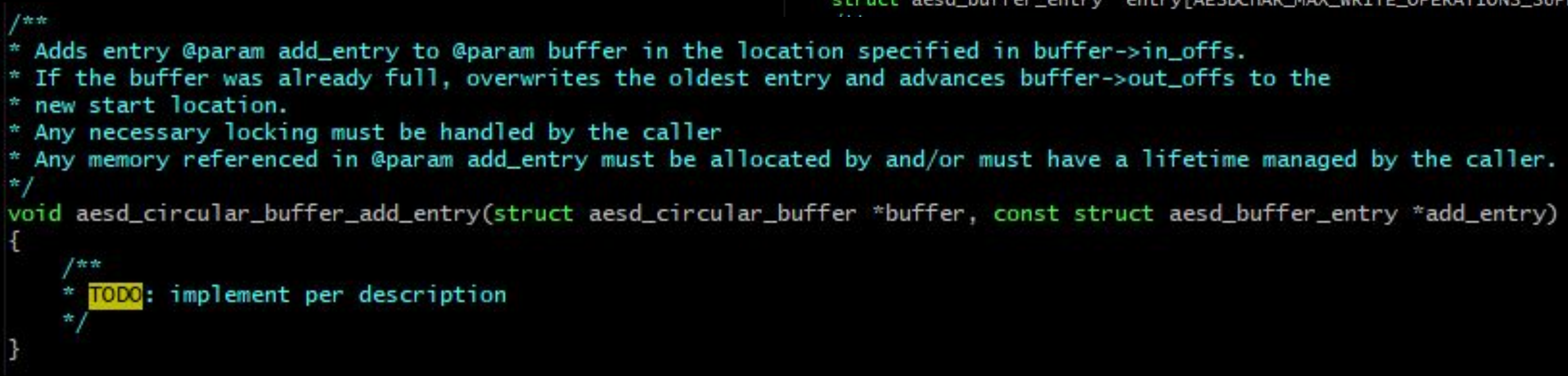
Circular Buffer Add



```
struct aesd_buffer_entry
{
    /**
     * A location where the buffer contents in buffptr are stored
     */
    const char *buffptr;
    /**
     * Number of bytes stored in buffptr
     */
    size_t size;
};
```



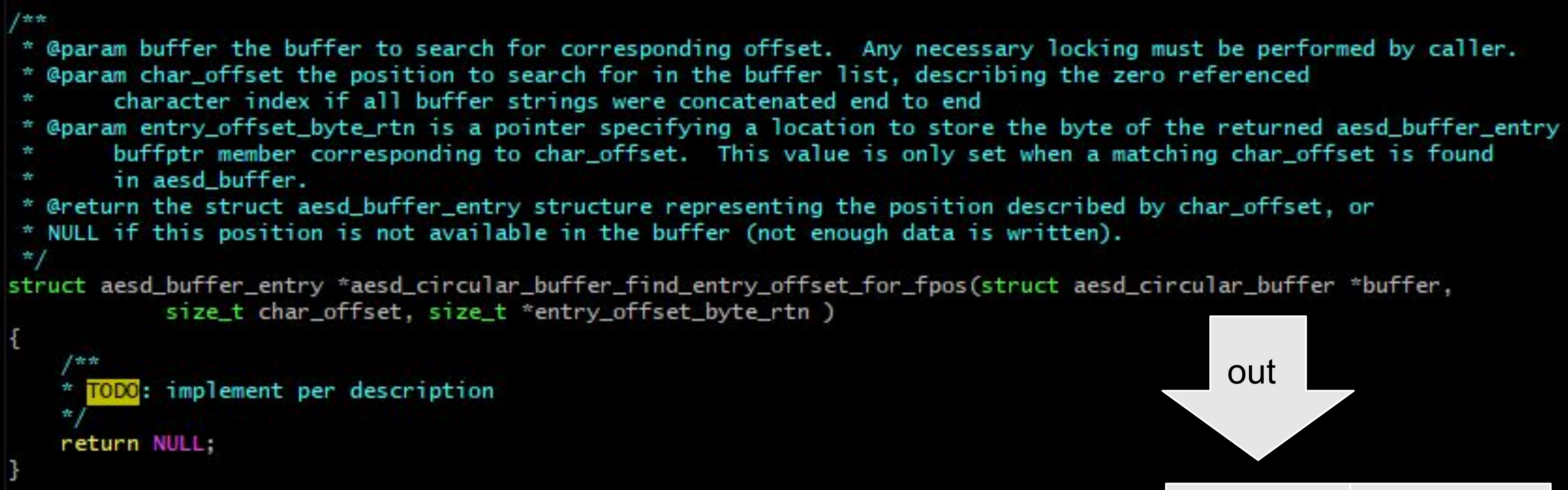
```
struct aesd_circular_buffer
{
    /**
     * An array of pointers to memory allocated for the most recent write operations
     */
    struct aesd_buffer_entry  entry[AESDCHAR_MAX_WRITE_OPERATIONS_SUPPORTED];
```



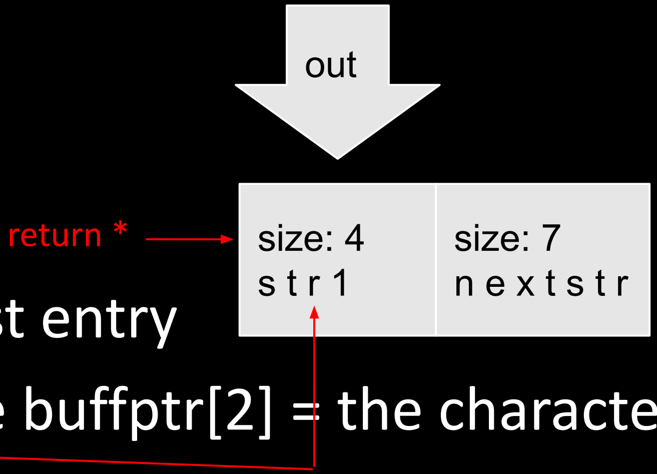
```
/**
* Adds entry @param add_entry to @param buffer in the location specified in buffer->in_offs.
* If the buffer was already full, overwrites the oldest entry and advances buffer->out_offs to the
* new start location.
* Any necessary locking must be handled by the caller
* Any memory referenced in @param add_entry must be allocated by and/or must have a lifetime managed by the caller.
*/
void aesd_circular_buffer_add_entry(struct aesd_circular_buffer *buffer, const struct aesd_buffer_entry *add_entry)
{
    /**
    * TODO: implement per description
    */
}
```

- Add to the circular buffer until full
- Then replace the oldest entry with new entry

Circular Buffer Find Offset

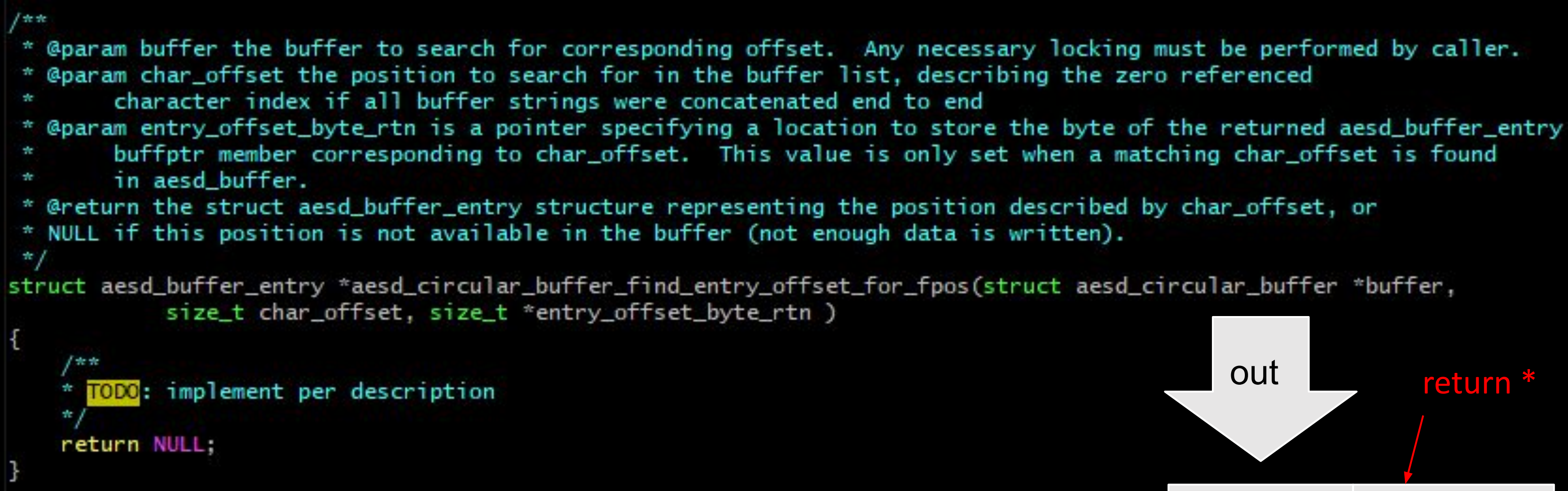


```c
/**
 * @param buffer the buffer to search for corresponding offset.  Any necessary locking must be performed by caller.
 * @param char_offset the position to search for in the buffer list, describing the zero referenced
 *      character index if all buffer strings were concatenated end to end
 * @param entry_offset_byte_rtn is a pointer specifying a location to store the byte of the returned aesd_buffer_entry
 *      buffptr member corresponding to char_offset.  This value is only set when a matching char_offset is found
 *      in aesd_buffer.
 * @return the struct aesd_buffer_entry structure representing the position described by char_offset, or
 * NULL if this position is not available in the buffer (not enough data is written).
 */
struct aesd_buffer_entry *aesd_circular_buffer_find_entry_offset_for_fpos(struct aesd_circular_buffer *buffer,
            size_t char_offset, size_t *entry_offset_byte_rtn )
{
    /**
    * TODO: implement per description
    */
    return NULL;
}
```

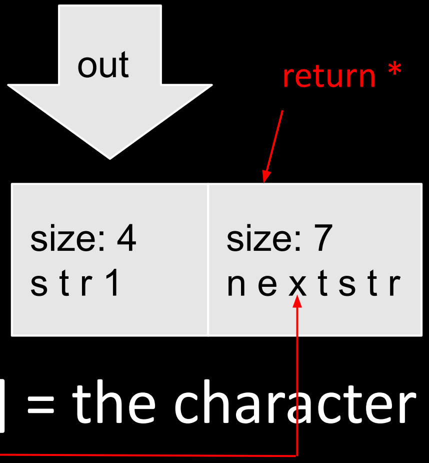


- when char_offset is 2:
    - aesd_buffer_entry points to first entry
    - entry_offset_byte_rtn is 2 since buffptr[2] = the character at zero referenced location 2 (r) in “str1nextstr”

Circular Buffer Find Offset



```
/**
 * @param buffer the buffer to search for corresponding offset.  Any necessary locking must be performed by caller.
 * @param char_offset the position to search for in the buffer list, describing the zero referenced
 *      character index if all buffer strings were concatenated end to end
 * @param entry_offset_byte_rtn is a pointer specifying a location to store the byte of the returned aesd_buffer_entry
 *      buffptr member corresponding to char_offset.  This value is only set when a matching char_offset is found
 *      in aesd_buffer.
 * @return the struct aesd_buffer_entry structure representing the position described by char_offset, or
 * NULL if this position is not available in the buffer (not enough data is written).
 */
struct aesd_buffer_entry *aesd_circular_buffer_find_entry_offset_for_fpos(struct aesd_circular_buffer *buffer,
            size_t char_offset, size_t *entry_offset_byte_rtn )
{
    /**
    * TODO: implement per description
    */
    return NULL;
}
```



- when char_offset is 6:
  - aesd_buffer_entry points to second entry
  - entry_offset_byte_rtn is 2 since buffptr[2] = the character at zero referenced offset 6 (x) for combined “str1nextstr”