Report on “hello” Linux Kernel Module:

* Changed MODULE\_AUTHOR to Ben Rose
* Changed the value of the name pointer to brose1
* \_\_init hello\_init(void):
* --->Added printk line to say hello to user
* \_\_exit hello\_exit(void):
* --->Added printk line to say goodbye to user
* Added module\_init(hello\_init) and module\_exit(hello\_exit) to load and unload the module, respectively

Report on “qmod” Linux Kernel Module:

* Changed qadd.c to check for argc = 2, not 1, since ./qadd is 1 argument, making ./qadd (some text) two arguments, which is the correct amount of arguments to use. Also changed all argv[0]’s to argv[1]’s, as ./qadd is argv[0] and (some text) is argv[1].
* Changed MODULE\_AUTHOR to Ben Rose
* Changed name to brose1
* Added static struct qheader \*queue as an instance of a queue for global use
* Added static DEFINE\_MUTEX(lock), which is a mutex variable called lock for locking and unlocking critical sections
* \_\_init mod\_init(void):
* --->Initialized the queue with kernel memory using kmalloc
* --->Initialized the front and last nodes of the queue to NULL
* --->Initialized the mutex variable called lock
* \_\_exit mod\_exit(void):
* --->Added mutex\_destory(&lock) to destroy the mutex variable
* --->Added kfree(queue) to destroy the queue and free the memory allocated to it
* dev\_read(struct file \*filep, char \*buffer, size\_t len, loff\_t \*offset):
* --->Begin by locking the mutex variable “lock”
* --->If the last element of the queue is NULL, i.e. the list is empty and you’re trying to remove an item, unlock the mutex, print an error message saying that you can’t take one item out of zero items, and return -1.
* --->If the last element’s next node is NULL, then the last element is equal to the first element, and so the last element and first element are set to NULL, since they are the same element.
* --->Otherwise, create a temporary node to scan over the queue for the second to front element. When we get to the second to front element, make it the new front element, as elements are removed from the front of a queue.
* --->Then unlock the mutex variable “lock”, since we’ve stopped accessing the critical section, which in this case is the queue.
* --->Copy the buffer, message, and messageSize, to the user, and set the potential error code to the error\_count variable.
* --->If the error\_count variable == 0, the sending was successful. Else, report the failure and its error number.
* dev\_write(struct file \*filep, const char \*buffer, size\_t len, loff\_t \*offset):
* --->Create the addme variable, which is the variable to be added to the queue
* --->Make addme’s message length equal to the len variable added to the function, and make its message equal to the buffer added to the function.
* --->Make a debugging variable to verify the mutex’s functionality, which is called mutex\_debug. Its result will be fed from the function mutex\_lock\_interruptible function, which will sleep until the lock is free, to be verified later.
* --->If the queue is empty, then make the next node for addme = NULL, and make the first and last node in the queue = addme, since the queue has only one element which is currently the first and last element in it.
* --->Else if the queue has one node in it, then make the next node for addme = the front element in the queue, then make the last element in the queue equal to addme, so the new last node points to the first node, as would make sense for a two-element queue.
* --->Else, the queue has two or more nodes in it, so make the next node for addme = the old last node, then make addme the new last node, which would make the second to last node the old last node, and the new last node addme.
* --->Free the allocated kernel memory for the addme node.
* --->Unlock the mutex variable “lock”, then double check that the mutex variable is unlocked, using mutex\_is\_locked, the result of which will be stored in the is\_mutex\_locked variable.
* --->Print the amount of received characters from the user.
* Added module\_init(mod\_init) and module\_exit(mod\_exit) to load and unload the module, respectively