ECEN 449 Lab 6: An Introduction to Character Device Driver Development

03/05/2020

Jason Gilman

**Introduction:**

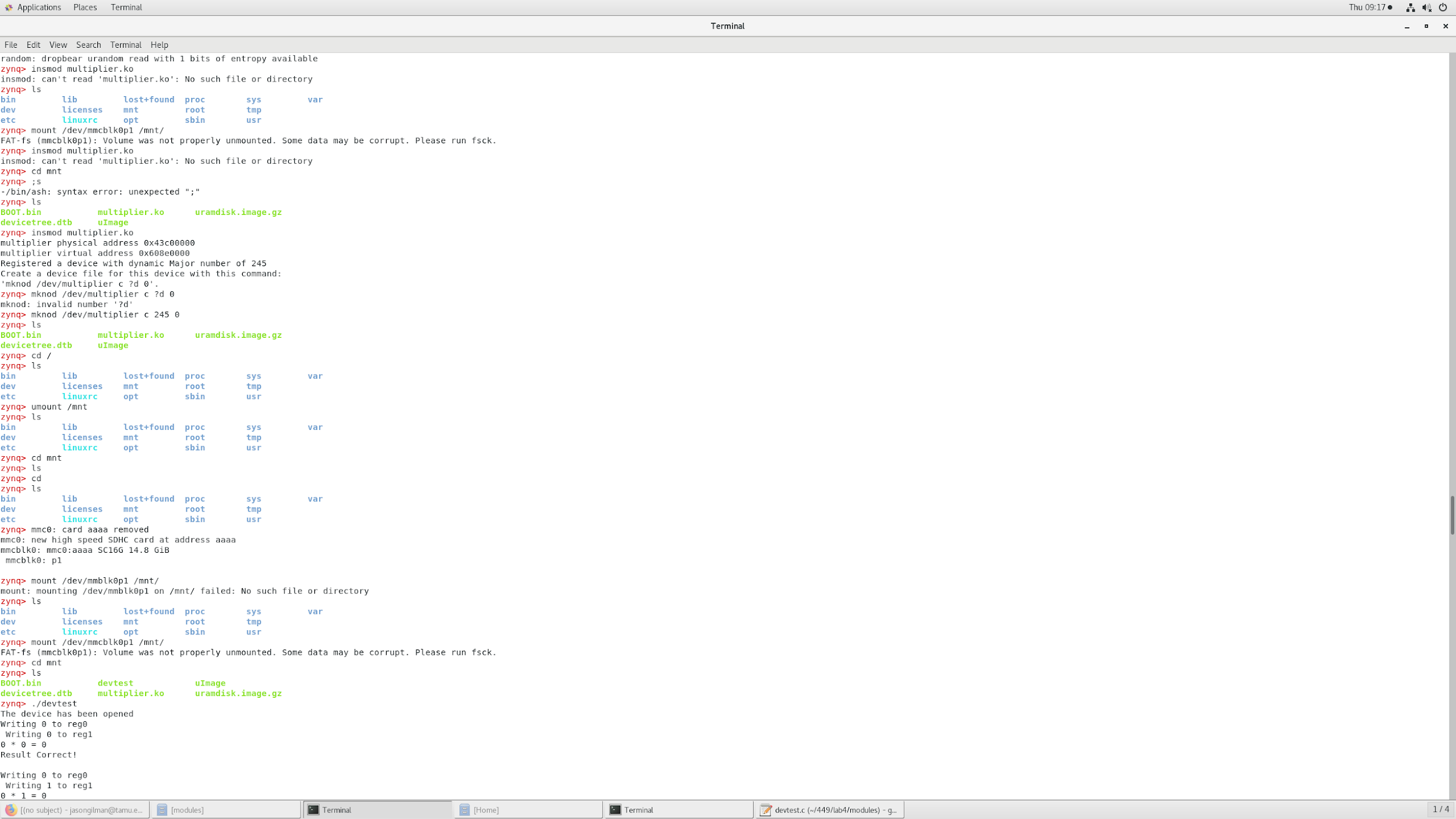
In this lab, we were tasked with creating a device driver for a simple multiplication module. The module will then be tested on the linux operating system that will run on the Zybo-10 board. This allows us to see how the software actually interacts with any user softwares.

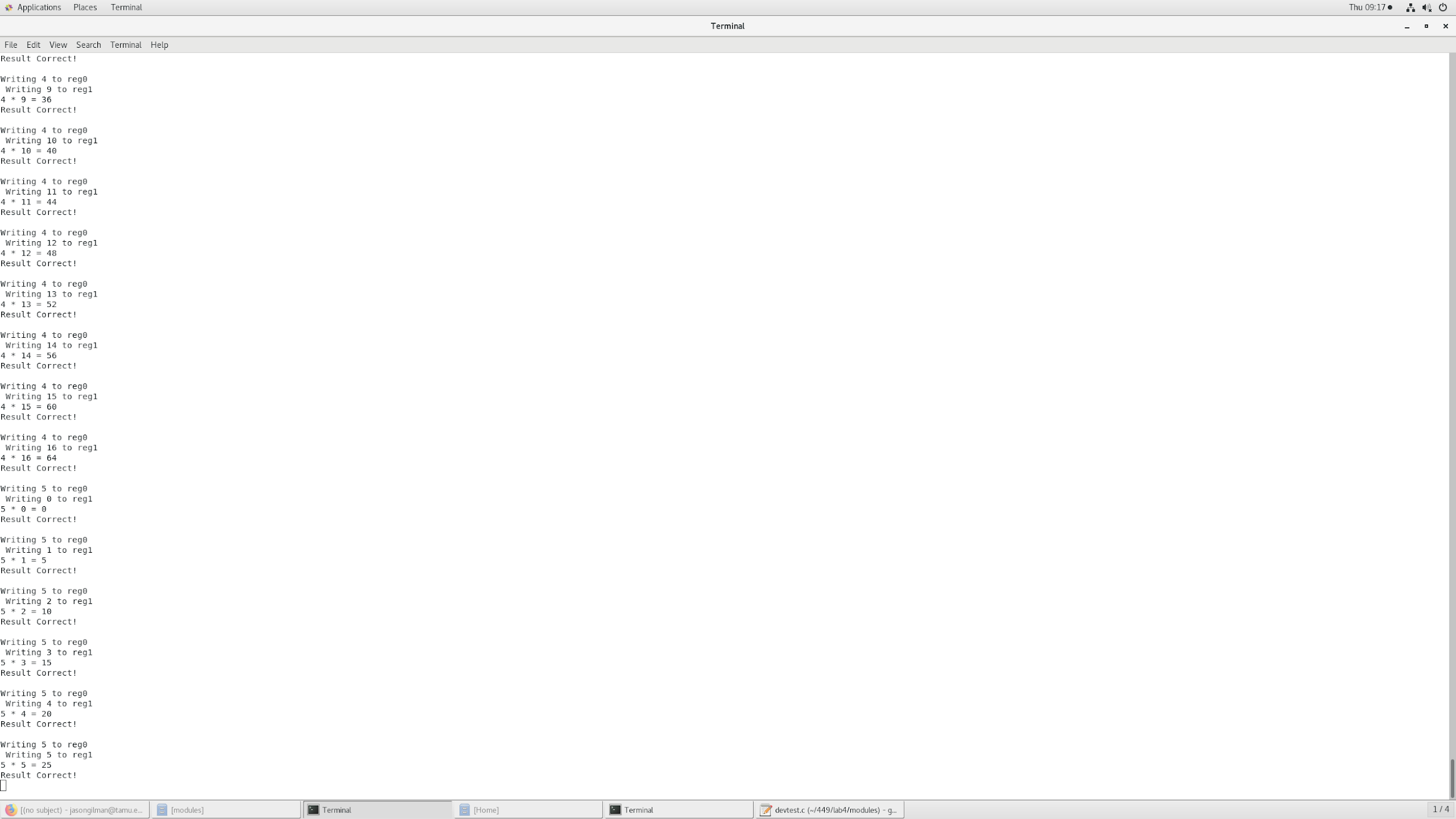
**Procedure:**

* To start the process of lab 6, a new kernel module was made, namely ‘multiplier.c’. This module needed to handle the instantiation, cleanup, and reading and writing to and from the device.
* Within the instantiation, first, the device’s memory space was determined. Next the device was registered, and the Major number associated was kept for error handling.
* In the cleanup of the module, the device was unregistered with the system, and the memory was unmapped.
* The module also handled opening and closing of the device, but only sent a message stating what was happening when one of these things was happening.
* When the device was called to read, a buffer was needed to interact with the data stored in memory. Since the data read from memory was in the kernel memory space, the data was transferred to the user space using the ‘put\_user()’ command. The number of bytes read/transferred to the user space was returned.
* When the device was called to write, another buffer was used to write into the user space. Like the previous step, just inverted, the characters written are in the user space, so by using the ‘get\_user()’ command, the data was able to be written into the kernel segment.
* After the module was compiled, and loaded into the linuxOS, the initialize function’s output was displayed to the picocom terminal.
* The next step was to create software that could interact with the module, and cause it to read and write.
* The software was written such that the device was opened, and numbers from 0-16 were written into memory, taken out, and then the product was found.
* To do this, the device was opened using ‘open()’ command, handling any errors if the device could not be opened.
* Next, numbers from 0-16 were put into an integer buffer and converted to char so that they could be loaded into the memory space using the ‘write()’ command.
* After that, the numbers were read from memory using ‘read()’ and converted from char to int using a series of buffers.
* Once the integers were obtained from memory, the product was computed and the result was checked for its correctness.
* Finally, the resources used, such as buffers, and the open file were all closed.

**Result/Output:**

The following was the output of the multiplier device. When loaded in, the device output the address space that it occupied, and the major number that was assigned to the device upon instantiation. Finally, when the device was ran, the two numbers being multiplied were outputed when being written into memory, and then the computation was also shown with the result being checked for correctness.





**Conclusion:**

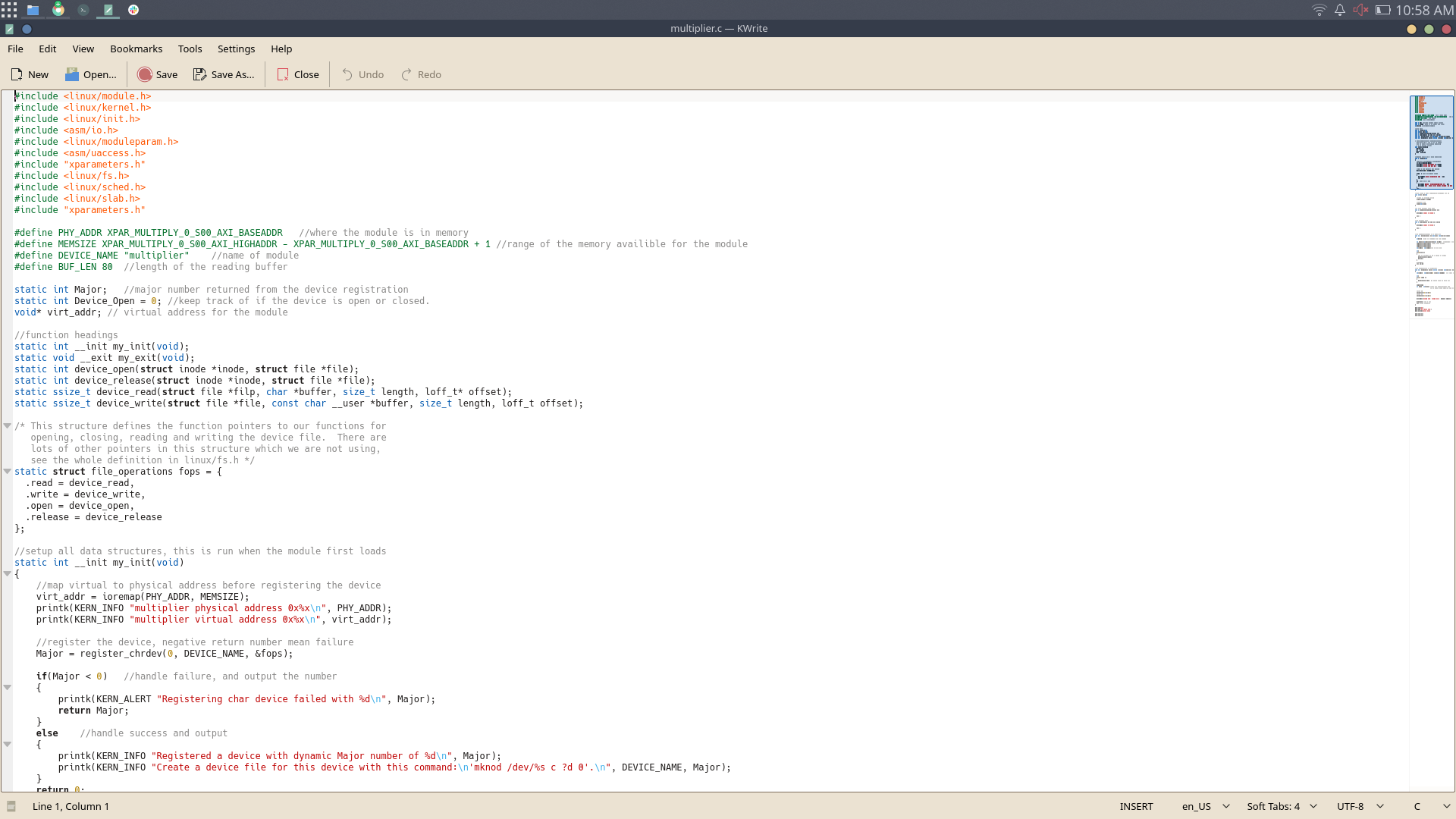
By instantiating a device and writing software to interact with the device, I was able to write values into the device’s memory space, read them out, and compute the product of the values.

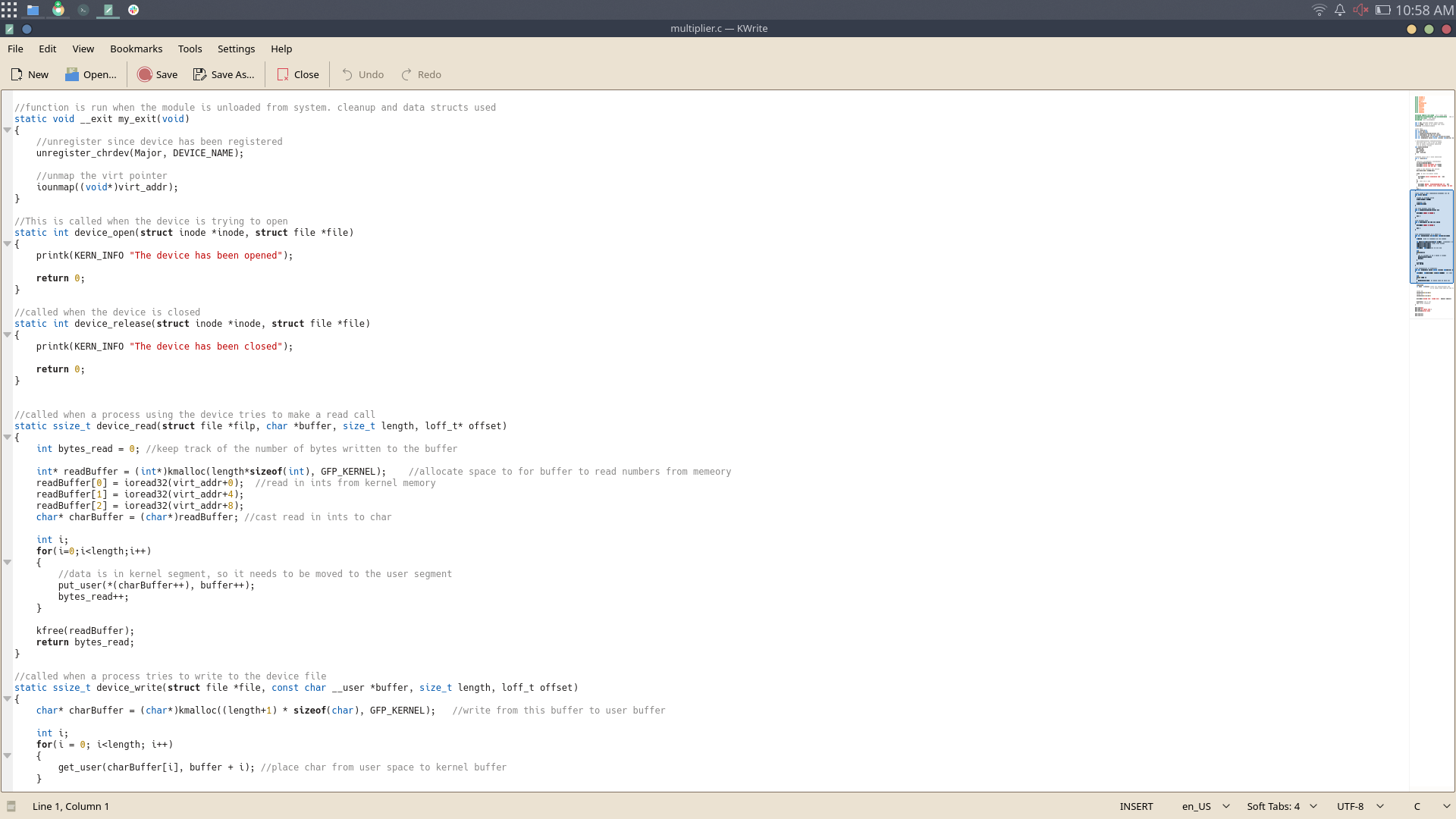
**Post Lab Questions:**

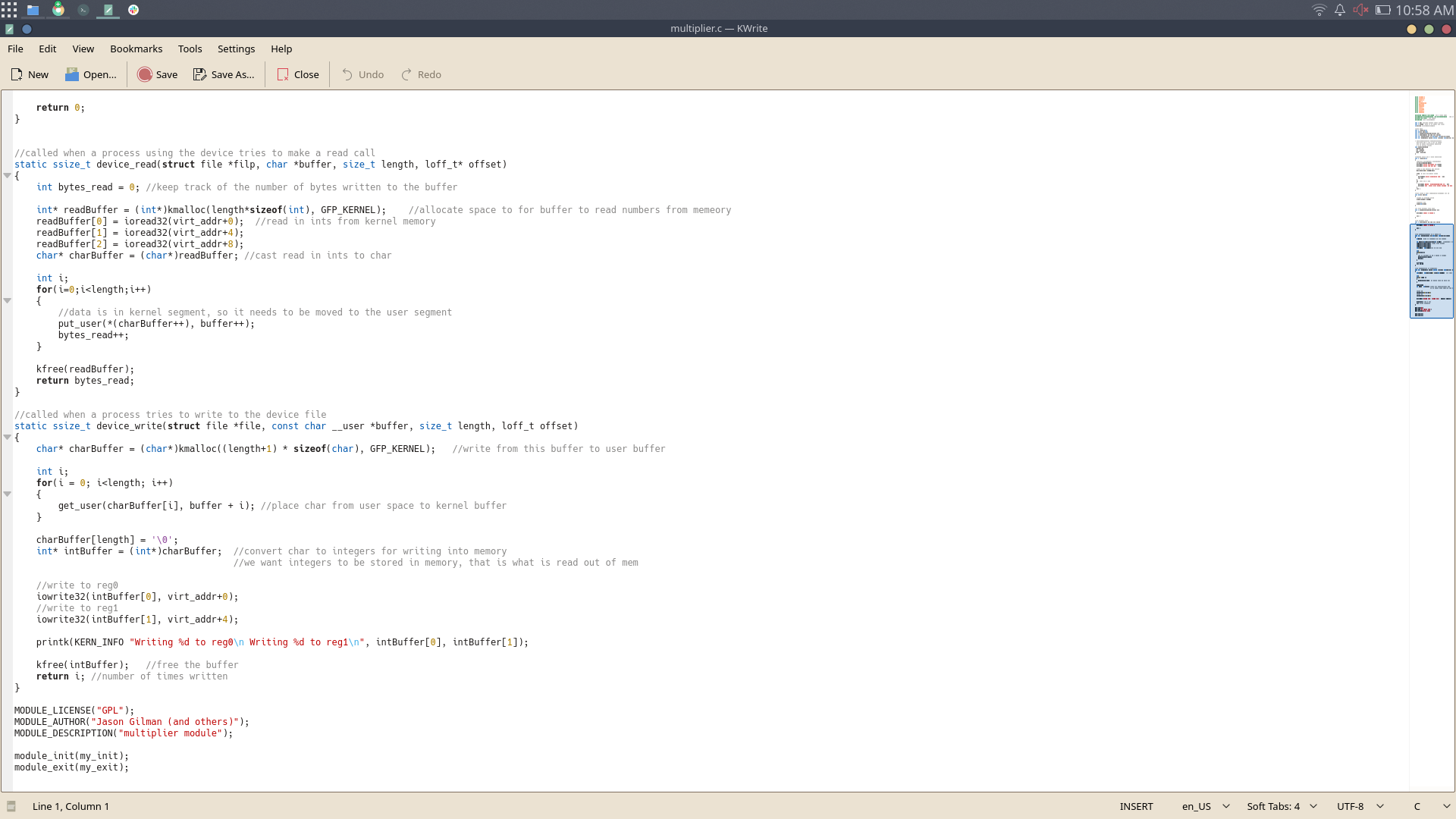
1. We used the ‘ioremap()’ command because we did not access the memory space by using the physical addresses, we used virtual addresses. The ioremap command allows us to use the virtual address to manipulate the memory space, to access specific data.
2. The implementation of lab3 was quicker because the software interacts directly with the processor used. The lab6 implementation had to go through the linuxOS inorder to communicate with the memory space and send commands to the processor.
3. In lab6 we used device drivers on linuxOS to interact with the hardware, this made it easier to create constructs and interface with the hardware, because linux has libraries in order to do so. In lab3, there was no support system in the form of an operating system, so we were interacting directly with the hardware. Using the OS, the operations are a bit slower, but it is easier to interface with the hardware.
4. When you first register a device, the device may be accessed immediately, so it is important to set up the appropriate buffers and address spaces. If we set up the resources after registering the device, there would be a conflict between the OS and the device. Likewise, when unregistering the devie, you do this first, so that when you deallocate any resources, there is no conflict between the OS and the recently deactivated device. If you deallocated resources first, there would be an issue within the driver because the needed resources would not be available.

**Appendix:**

Multiplier.c







Devtest.c

