**ECEN 449 - Lab Report**

**Lab Number:** 7

**Lab Title:** Linux Kernel: Built-In Modules

**Section Number:** 508

**Student's Name:** Samuel Fafel

**Date Due:** 04-27-2023

**TA:** Prajwal Holla

#### Purpose/Introduction:

This final lab procedure is the culmination of several of our previous labs in that we are taking the previously-built multiplier module and, rather than installing it after the linux boots, we are building it into the linux kernel itself. This allows us to boot up linux and have our module be a default part of the OS.

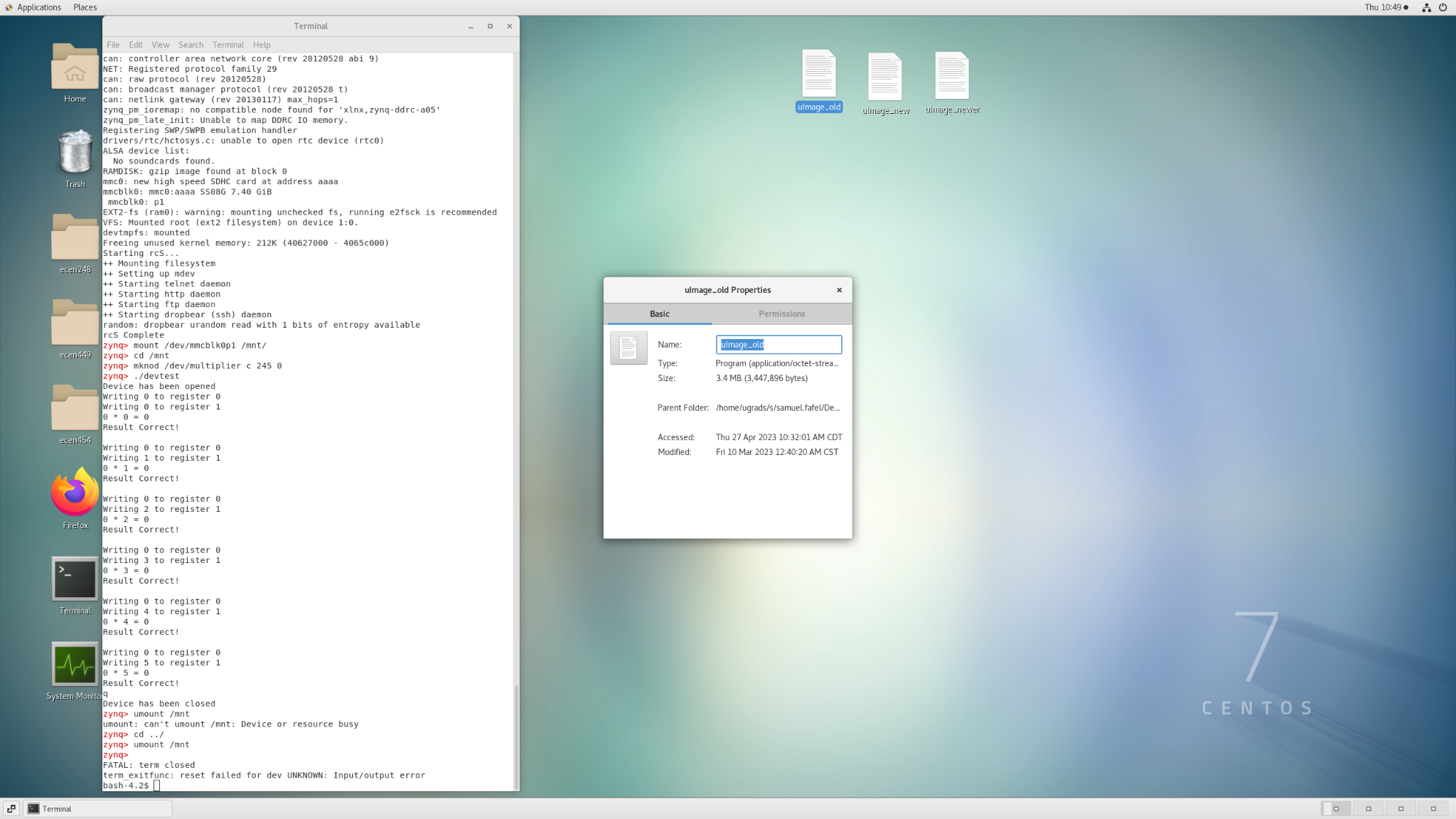
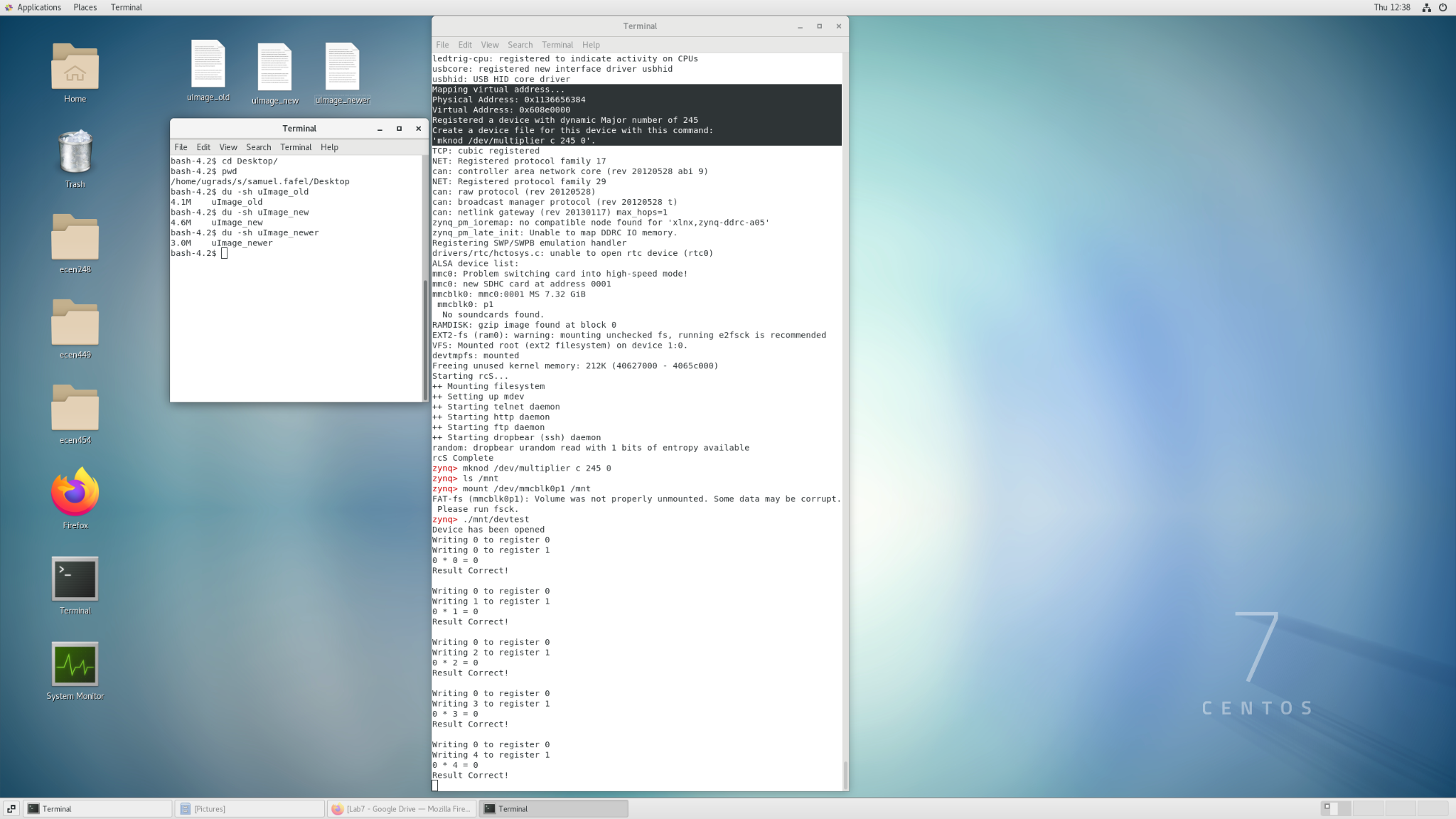
#### Procedure:

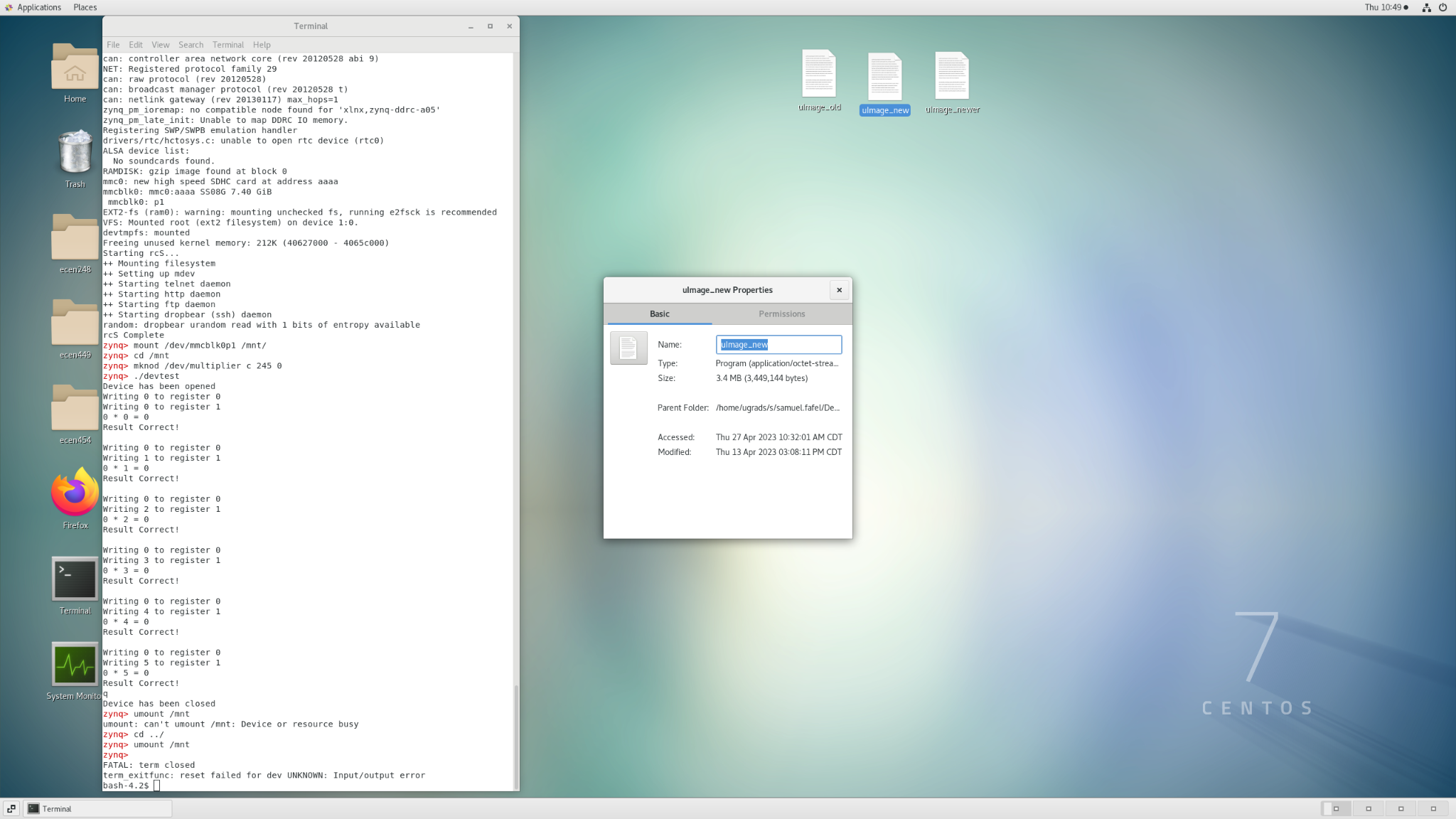
First, we backed up the existing linux kernel so that any changes/mistakes would not undo all of our previous work. Then, we became familiar with the menuconfig options upon compilation of the kernel before adding our multiply IP directly. We created the proper Makefile and Kconfig files within a subdirectory of the linux drivers, and edited the appropriate file which connected them to the device drivers. Next, we compiled the linux kernel again (and noted the size of the new uImage compared to the previous size). Lastly, we disabled some previously-included modules and again recompiled, and noted the even newer uImage size.

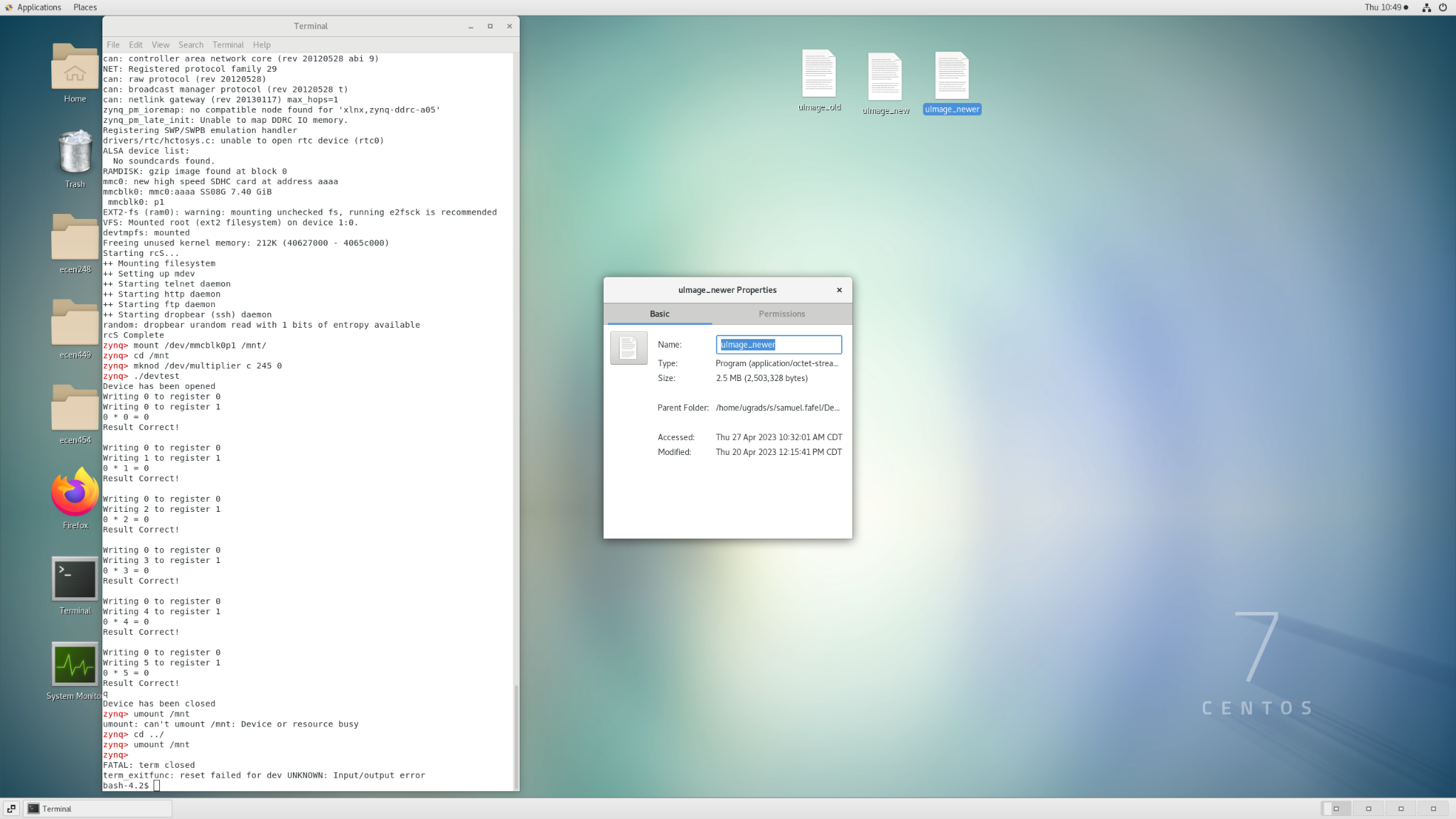
#### Results:

As a result of the above procedure, we created a uImage which contained all of the default drivers as well as our multiply driver. This uImage was (3,449,144 Bytes / 3.4 MB) as opposed to the original size of (3,447,896 Bytes / 3.4 MB), meaning that our new uImage was about (1,248 Bytes / 1.2 KB) larger than the original. Lastly, our third uImage which excluded the specified default drivers, but again included our multiply driver was (2,503,328 Bytes / 2.5 MB), considerably smaller than either of the first two. This is to be expected, as the excluded drivers were relatively large.

In the screenshots below, one can observe the output of our new linux kernel booting properly with the expected output of the multiply driver showing up during the boot sequence. Additionally, one can observe the size of each of the uImages marked "\_old" (original from Lab5), "\_new" (including multiply driver and other drivers), and "\_newer" (including multiply driver, but excluding those specified by the lab manual.







#### Conclusion:

This lab finalizes the work done in the previous labs, in that it guided us through the process of including our own built-in driver for the linux kernel. It shows us how that is done both in the directory level (making and including files in subdirectories of the kernel) as well as the command level (commands to compile and change compilation configuration).

#### Questions:

1. Advantages/Disadvantages
   1. Some advantages of loadable kernel modules are: Flexibility in features available to the OS: Using loadable kernel modules we can change the kernel without having to restart the system. Also, Faster development of modules: they can be developed and tested independently of the rest of the kernel.
   2. A disadvantage of loadable kernel modules is possible compatibility issues: In order for the module to work, it must be compatible with the kernel. If the kernel is updated, the module would need to be updated as well (asynchronously)
   3. Some advantages of built-in kernel modules are: Performance capabilities: built-in modules do not need to be loaded or unloaded from the kernel. Also, Compatibility is less of an issue since the modules are always part of the kernel image itself.
   4. Some disadvantages of built-in kernel modules are: Lack of flexibility in features: built-in modules can't be modified without rebuilding the whole kernel again. This also impacts development time for the module: maintaining, modifying, and testing the new kernel module may require rebuilding the whole kernel every time.