Laboratory Exercise #6

An Introduction to Character Device Driver Development

Name: Daniel Horan

UIN: 527005307

ECEN 449-504

I. Introduction

The focus of this lab is to develop a character device driver for the ZYBO Z7-10 board, using the embedded Linux environment set up in previous labs. This driver is intended to build upon our kernel module from Lab 5, enabling user applications to interact with the multiplication peripheral designed earlier. The primary goal is to enhance our technical skills in kernel programming and device driver development, which are crucial for effective hardware-software integration in microprocessor system design. Through this lab, we will directly engage with the Linux kernel, gain hands-on experience in driver development, and solidify our understanding of user-kernel space interactions.

II. Procedure

The laboratory procedure begins with the development of a character device driver, titled 'multiplier.c', within the PetaLinux project framework. Utilizing the foundational work established in Lab 5, this task focuses on enabling access from user-space to the multiplication hardware peripheral. The process initiates with the insertion of a USB drive containing the PetaLinux installation and the creation of the device driver in the designated project directory. This stage includes careful mapping of the multiplication peripheral's physical address to virtual memory and registering the device with the Linux kernel, with particular attention to the dynamic allocation of a major number.

Subsequent steps involve the coding of the driver's primary functions: open, close, read, and write. These are integral for facilitating user-space interactions with the hardware, with the read and write functions being especially critical for data transmission. Methodical programming is employed to ensure data handling is performed using 'put_user' and 'get_user' system calls, which are essential for kernel-to-user space communication.

Once the driver is implemented, the procedure advances to the modification of the 'multiplier.bb' file to integrate any additional header files. The 'multiplier.ko' module is then compiled using 'petalinux-build', and loaded onto the ZYBO Z7-10's Linux operating system. The module's functionality is confirmed through 'dmesg' logs, followed by the establishment of the '/dev/multiplier' device node, enabling user applications to interact with the new driver.

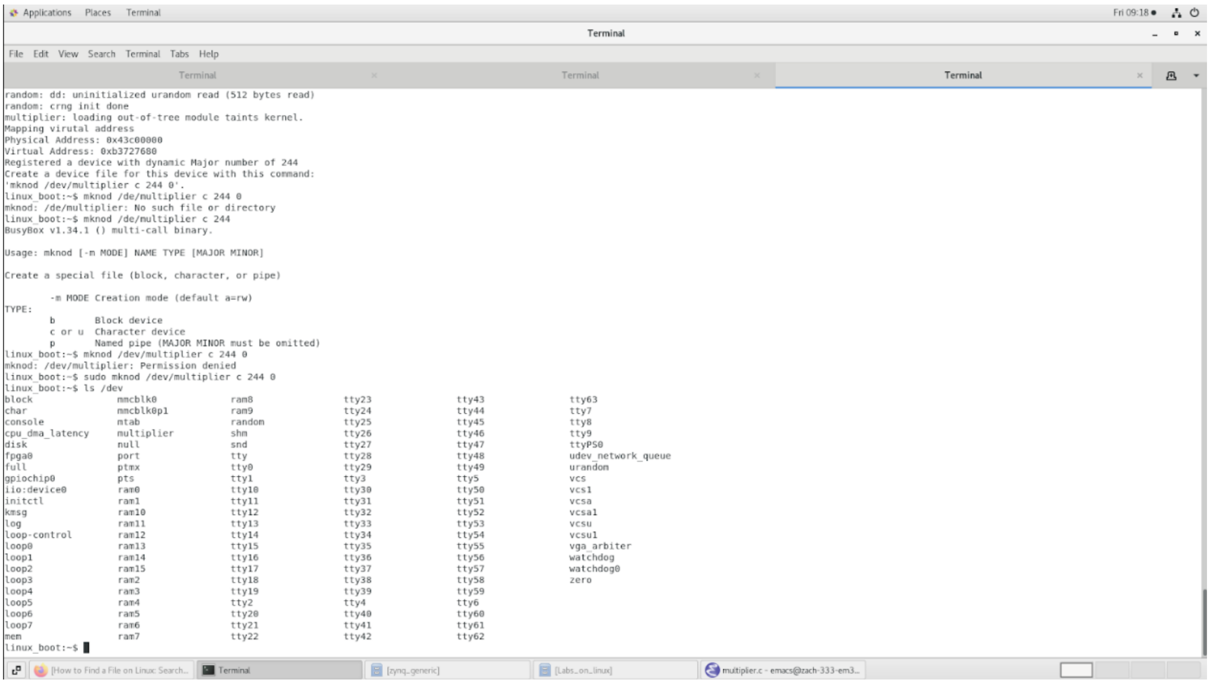
The final phase of the lab involves the composition of a user-space application, 'devtest.c', which communicates with the '/dev/multiplier' device. The application, developed in C, leverages standard system calls for performing read and write operations with the device driver. This executable is cross-compiled, transferred to an SD card, and subsequently run on the ZYBO Z7-10 board. The operation of the character device driver is tested through this application, ensuring that the interaction with the multiplication peripheral is as intended and the device driver operates correctly within the system's context.

III. Results

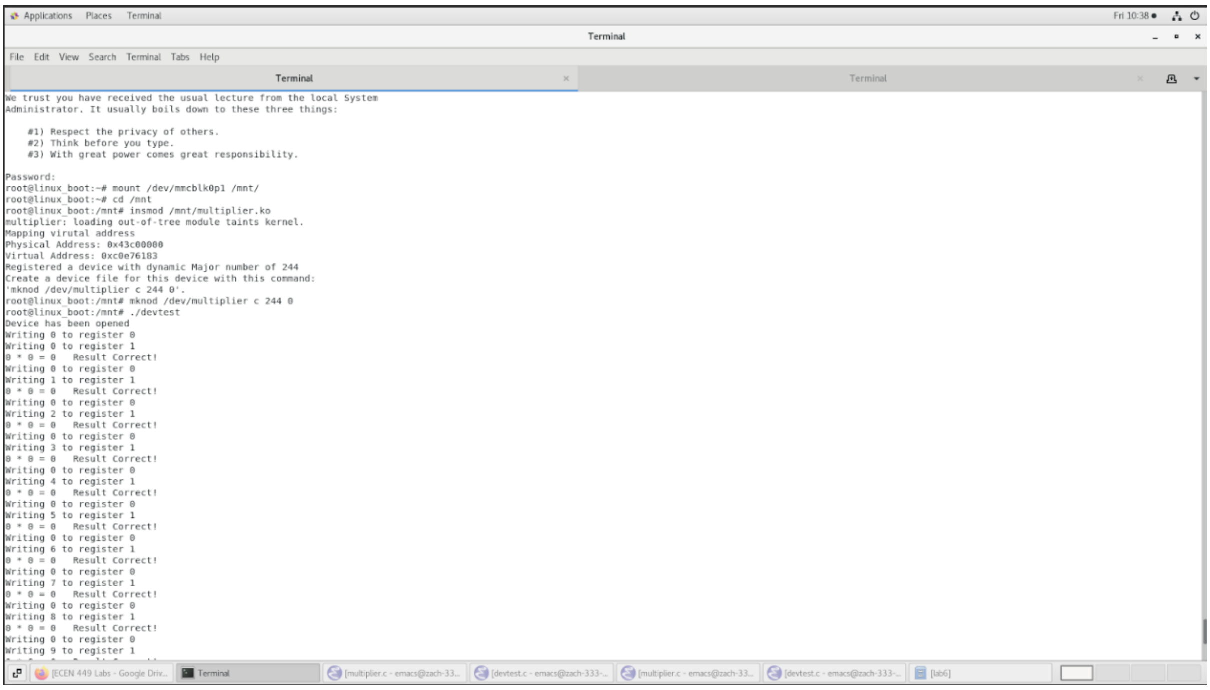
In Lab 6, our primary objective was to develop a character device driver, 'multiplier.c', for the ZYBO Z7-10 board and integrate it within the Linux kernel. The lab began smoothly with the initial setup and creation of the device driver in the PetaLinux environment. However, during the development phase, I encountered a significant issue that necessitated rebuilding the entire project. Despite this setback, I was able to troubleshoot and resolve the problems, successfully completing the driver.

The 'multiplier' device driver was then effectively loaded into the Linux system on the ZYBO Z7-10 board. To test its functionality, I developed and executed the user application 'devtest.c', which interacted with the '/dev/multiplier' device. This application played a crucial role in demonstrating the successful operation of the character device driver, confirming that it could correctly perform read and write operations with the multiplication peripheral.

A final demonstration to the TA validated the successful completion of the lab. The character device driver and user application operated as intended, showcasing the effective interaction with the hardware. This lab provided practical experience in embedded Linux development and highlighted the importance of problem-solving and persistence in complex system programming.



Running *multiplier.c*



Running *devtest*

IV. Conclusion

In this lab, we navigated the complexities of developing a character device driver for the ZYBO Z7-10 board within an embedded Linux environment. This experience underscored the critical role of device drivers in bridging hardware and software. The lab began with foundational tasks, progressing to more challenging aspects, including a significant issue that required rebuilding the project. This process highlighted the intricacies of embedded system programming and the importance of problem-solving skills.

Successfully integrating the 'multiplier.c' device driver and testing it with the 'devtest.c' user application, we demonstrated the practical application of theoretical concepts in microprocessor system design. The lab concluded with a successful demonstration to the teaching assistant, affirming our ability to effectively manage kernel-space and user-space interactions. This experience not only met the objectives of the lab but also enhanced our understanding of device driver development in the context of embedded Linux systems.

V. Questions

a) The 'ioremap' command is necessary because it maps the physical address of the multiplier hardware into the virtual address space that the kernel operates in. This allows the kernel to communicate with the hardware, which is crucial because the processor cannot directly access physical addresses in a protected memory system. Furthermore, 'ioremap' ensures that the mapped memory is non-cacheable, preventing the CPU from caching its contents, which is important for I/O operations where the hardware state might change independently of the processor's actions.

b) The multiplication in part 3 of this lab would likely be faster than in Lab 3 because it uses a dedicated hardware peripheral for the operation, eliminating the software overhead and enabling parallel processing alongside the ARM processor.

c) This lab uses a custom hardware peripheral for multiplication, offering faster execution but at the cost of increased development complexity and resource use. Lab 3's software-based method is more flexible and easier to update but is generally slower and less efficient for compute-intensive tasks.

d) Device registration should be the last step in the initialization routine to ensure that all prerequisites of the device driver, such as memory allocation and hardware setup, are complete before the kernel and userspace can interact with it. This prevents access to the device before it is fully ready, which can cause errors or system crashes.

   Conversely, device un-registration must happen first in the exit routine to immediately stop new interactions with the device from userspace or the kernel while it is being removed. This ensures that all operations are halted before the driver begins to free resources and de-allocate memory, maintaining system stability and preventing data corruption or undefined behavior.