

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.

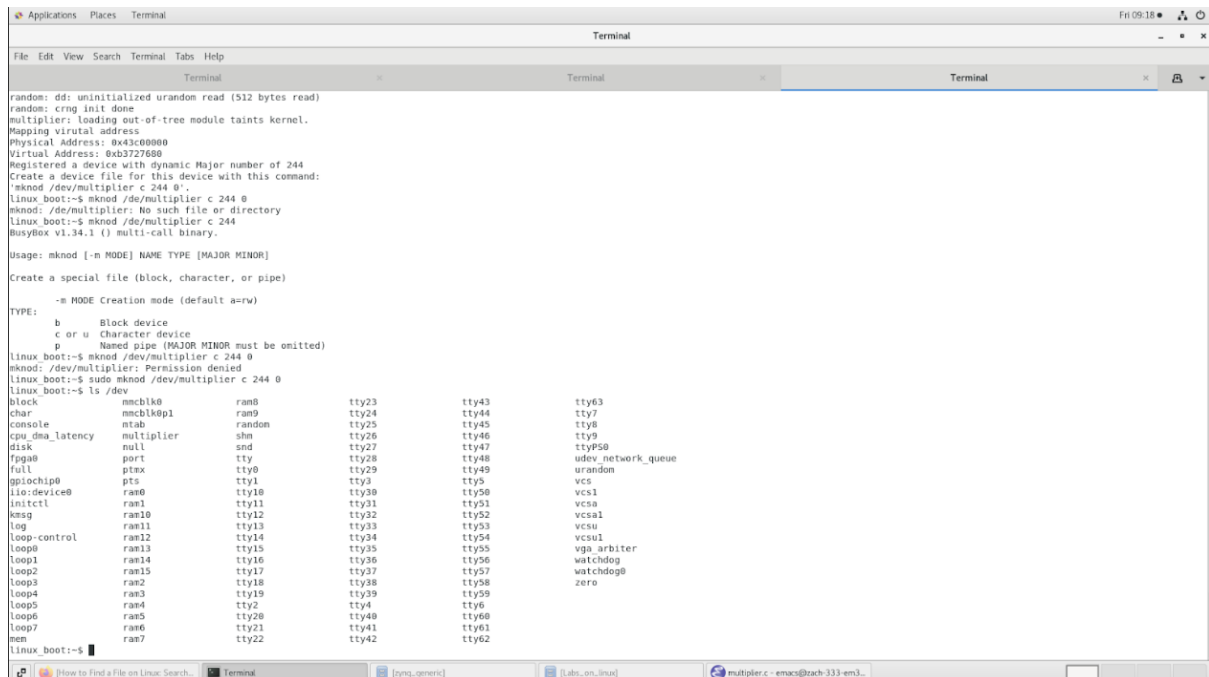
III. Results

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.



```
random: dd: uninitialized urandom read (512 bytes read)
random: crng init done
multiplier: loading out-of-tree module taints kernel.
Mapping virtual address
Physical Address: 0x43c00000
Virtual Address: 0xb3727600
Registered a device with dynamic Major number of 244
Create a device file for this device with this command:
'mknod /dev/multiplier c 244 0'.
Linux boot:~$ mknod /dev/multiplier c 244 0
mknod: /dev/multiplier: No such file or directory
Linux boot:~$ sudo mknod /dev/multiplier c 244 0
BusyBox v1.34.1 () multi-call binary.

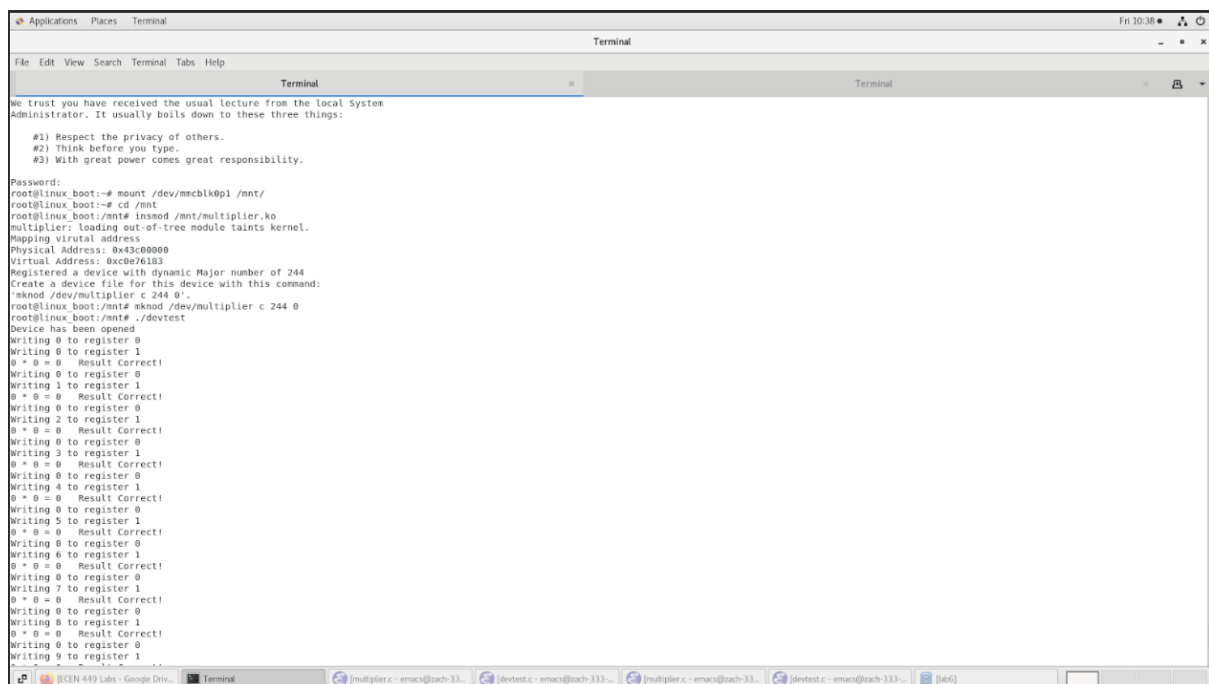
Usage: mknod [-n MODE] NAME TYPE [MAJOR MINOR]

Create a special file (block, character, or pipe)

-n MODE Creation mode (default a-rw)

TYPE:
b      Block device
c or u Character device
p      Named pipe (MAJOR MINOR must be omitted)
Linux boot:~$ mknod /dev/multiplier c 244 0
mknod: /dev/multiplier: Permission denied
Linux boot:~$ sudo mknod /dev/multiplier c 244 0
Linux boot:~$ ls -l /dev
block          mncblk0      ram8          tty23         tty43         tty63
char           mncblk0p1   ram9          tty24         tty44         tty7
console        mtab         random        tty25         tty45         tty8
cpu_dma_latency multiplier    shm           tty26         tty46         tty9
disk           null         snd           tty27         tty47         ttyP50
fpga0          port         tty           tty28         tty48         udev_network_queue
full           ptmx         tty0          tty29         tty49         urandom
gpiochip0      pts          tty1          tty3           tty5          vcs
iio:device0    ram0         tty10         tty30          tty50         vcs1
initctl        ram1         tty11         tty31          tty51         vcsa
kmsg           ram10        tty12         tty32          tty52         vcsa1
log            ram11        tty13         tty33          tty53         vcsu
loop-control   ram12        tty14         tty34          tty54         vcsu1
loop0          ram13        tty15         tty35          tty55         vga_arbiter
loop1          ram14        tty16         tty36          tty56         watchdog
loop2          ram15        tty17         tty37          tty57         watchdog0
loop3          ram2         tty18         tty38          tty58         zero
loop4          ram3         tty19         tty39          tty59
loop5          ram4         tty2          tty4           tty6
loop6          ram5         tty20         tty40          tty60
loop7          ram6         tty21         tty41          tty61
mem            ram7         tty22         tty42          tty62
```

Running *multiplier.c*



```
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.

Password:
root@linux boot:~# mount /dev/mncblk0p1 /mnt/
root@linux boot:~# cd /mnt
root@linux boot:/mnt# insmod /mnt/multiplier.ko
multiplier: loading out-of-tree module taints kernel.
Mapping virtual address
Physical Address: 0x43c00000
Virtual Address: 0xc0e76183
Registered a device with dynamic Major number of 244
Create a device file for this device with this command:
'mknod /dev/multiplier c 244 0'.
root@linux boot:/mnt# mknod /dev/multiplier c 244 0
root@linux boot:/mnt# ./devtest
Device has been opened
Writing 0 to register 0
Writing 0 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 1 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 2 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 3 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 4 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 5 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 6 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 7 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 8 to register 1
0 * 0 = 0 Result Correct!
Writing 0 to register 0
Writing 9 to register 1
```

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.

