HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY, VNU HCM
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEM

Assignment #1: System Call

Pham Minh Tuan - MSSV: 1752595

Contents:

1 Introduction:

1.1 System Call:

In computer, a system call is controlled entry point into a kernel, allowing a process to request that the kernel perform some action on the process's behalf. The kernel will make a range of services accessible to programs via system call as application programming interface (API).

- A system call changes the processor state from **user mode** to **kernel mode**, so that CPU can access protected kernel memory.
- The set of system calls is fixed. Each system call is identified by a unique number.
- Each system call may have a set of arguments that specify information to be transferred from **user space** to **kernel space** and vice versa.

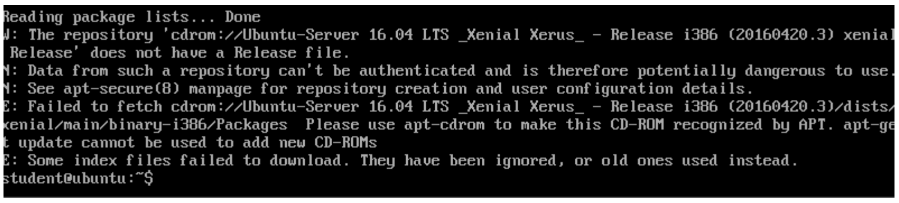
This assignment requires to create a system call named **procmem** to show users about the memory layout of specific process. I will go briefly into each step about preparing my virtual machine, compiling kernel and embedded a new system call.

2 Preparing Virtual Machine

2.1 VM Ware:

To prepare the new virtual machine for doing assignment 1, I choose to use the one given by lecturer in the link: `https://drive.google.com/file/d/15A2N4H_vx-wMkIhP2kG0fkfHE3K5MqwJ/view?usp=sharing`.

However I came with the biggest problem when trying to run update commmand due to specific errors related to current packages:



```
Reading package lists... Done
W: The repository 'cdrom://Ubuntu-Server 16.04 LTS _Xenial Xerus_ - Release i386 (20160420.3) xenial
 Release' does not have a Release file.
N: Data from such a repository can't be authenticated and is therefore potentially dangerous to use.
N: See apt-secure(8) manpage for repository creation and user configuration details.
E: Failed to fetch cdrom://Ubuntu-Server 16.04 LTS _Xenial Xerus_ - Release i386 (20160420.3)/dists/
xenial/main/binary-i386/Packages  Please use apt-cdrom to make this CD-ROM recognized by APT. apt-ge
t update cannot be used to add new CD-ROMs
E: Some index files failed to download. They have been ignored, or old ones used instead.
student@ubuntu:~$
```

It say that the I need to use the CDROM available with apt. However I came to the solution to solve this problem by upgrade it into the latest version by 18.04.2 LTS instead of the current 16.04 LTS by running upgrade command:

```
$ sudo do-release-update
```

And it also require the sudo priviledge to execute the command. After the upgrade process finished, I rebooted the virtual machine and everything seems fine since all packages is now updated automatically and up-to-date thourgh upgrading.

I followed the instructions in the assignment to install toolchains (gcc, make, etc) and the kernel-package.

QUESTION: Why do we need to install kernel package ?
Ans: We need to install kernel-package to easily manage all versions of kernel in virtual machine and install it rightfully, also it allow you to keep multiple of linux images without fuss.

2.2 Create and Download Kernel source:

The kernel compilation folder is created in *home* directory. For the source of downloading kernel: `http://www.kernel.org`, I choose to download the version of kernel exactly same as the instruction of Assignment

Moreover, the version of kernel downloaded is older than the current one running in my virtual machine since I already upgrade it to the latest as well as the kernel, so I will explain specificly about that problem after install the new kernel into the virtual machine.

QUESTION: Why do we have to use another kernel source from the server such as `http://www.kernel.org`, can we just compile the original kernel (the local kernel on the running OS) directly?
Ans: The answer is yes, you can compile directly from your current kernel. However in some situations, you need to use the other kernel instead of the current one:

- To enable experimental features that are not part of the default kernel.
- To enable support for a new hardware that is not currently supported by the default kernel.
- To debug the kernel.
- Study purpose.

After the downloading is complete, you can unpack zip file. Beside, it is recommend that we can install the *openssl libssl-dev* initially.

2.3 Configuration

In this step, I just follow every instruction in the Assignment to finish the config for the downloaded kernel and change the name of the kernel following by my student ID.

3 Kernel Module:

3.1 Data Structure:

In this assignemnt, I was recommend to use the `task_struct` and `mm_struct`:

- Each process running under a Linux machine is associated with a struct `task_struct`. The source code of `task_struct` can be read here.
- Inside a `task_struct`, there is a struct `mm_struct` which stores the description of memory of the process. The source code of `mm_struct` can be read here.

3.2 Create Procmem Kernel Module:

Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They can extend the functionality of the kernel without rebooting the system.

One requirement of the assignment is to write a module to test *procmem* system call. Below is how I implement it:

```c
#include <linux/module.h>
// included for all kernel modules

#include <linux/kernel.h>
// included for KERN_INFO
#include <linux/init.h>
// included for __init and __exit macros

#include <linux/mm.h>
#include <linux/sched/signal.h>
#include <linux/delay.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("TUAN");

static int pid = 1;
static int __init procmem_init(void){
    struct task_struct *task;
    printk(KERN_INFO "Starting kernel module!\n");

    // TODO: find task_struct that is associated with the input process
    ↪  pid
    // Hint: use the for_each_process() function:
    for_each_process(task)
    {
        printk("[%d] | [%s]\n", task->pid, task->comm);
        msleep(10);

        if (task->pid == pid)
        {
            printk("Student ID: 1752595\n");

            //check if mm is NULL or not:
            if (task->mm != NULL)
            {
                printk(KERN_INFO "task->mm not null !\n");

                // TODO: show its memory layout
                printk("Code Segment start = [%lu]; end  = [%lu]\n",
                ↪  task->mm->start_code, task->mm->end_code);
                printk("Data Segment start = [%lu]; end  = [%lu]\n",
                ↪  task->mm->start_data, task->mm->end_data);
                printk("Stack Segment start = [%lu]\n",
                ↪  task->mm->start_stack);

                return 0;
            }

            else if (task->active_mm == NULL)
            {
                printk(KERN_INFO "In the thread, using active_mm\n");
            }
        }
    }
    return -1;
}

static void __exit procmem_cleanup(void){
    printk(KERN_INFO "Cleaning up module.\n");
}

module_init(procmem_init);
module_exit(procmem_cleanup);
module_param(pid, int , 0);
```

A kernel modules must have at least two functions:

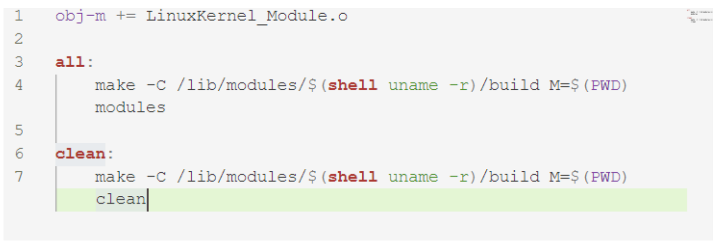
- A ”start” (initialization) function called `init_module()` which is called when the module is *insmoded* into the kernel, and an ”end” (cleanup) function called `cleanup_module()` which is called when the module is *rmmoded.*
- Moreover each kernel module needs to include *linux/module.h*, and *linux/kernel.h* to use the *printk()* log level.
- *printk()* is used to log information or give warnings, it happens to be a logging mechanism for the kernel.

3.3 Passing Command Line Arguments to a Module:

To allow arguments passed to module, we can declare the variables of the command line arguments as global and then use the `module_param()` macro. The `module_param()` macro takes 3 arguments: the name of the variable, its type and permissions bits.

3.4 Compiling the Kernel Modules:

The way to compile the kernel is different from the user applications. I follows the instruction through `http://www.tldp.org/LDP/lkmpg/2.6/html/x181.html` to use *Makefile* to compile a kernel module:



```
obj-m += LinuxKernel_Module.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Figure 1: Makefile

Building and Running Module: Build the module by simply calling *make* in command line.

After building the module, the information of the module by running:

```
$ modinfo LinuxKernel_Module.ko
```

To insert the module into kernel:

```
$ sudo insmod ./LinuxKernel_Module.ko
```

To check if the module is inserted succesfully:

```
$ lsmod
$ cat /proc/modules
```

To remove the module from the kernel:

```
$ sudo rmod LinuxKernel_Module
```

We can the look at */var/log/messages* to see that it is logged in system log file.

4 Procmem System Call

4.1 Implementation:

I follow the instructions to implement the system call since I use the given VMWare virtual machine and the system architecture is x86.

QUESTION: What is the meaning of other components, i.e. `i386`, `procmem`, and `sys_procmem`?
ANS:

- `i386` is the ABI (Application Binary Interface) which is the interface between two program modules usually are libraries and operating system architecture. Beside i386, there are also x64 and x32.
- `procmem` is the name of system call.
- `sys_procmem` is the entry point, a set of functions to call system call. The name of the entry point often have the prefix `sys_`.

```
struct proc_segs;
asmlinkage long sys_procmem(int pid, struct proc_segs *info);
```

QUESTION: What is the meaning of each line above?
ANS:

- The first line is to call the definition of struct `proc_segs`.
- `asmlinkage` is a #define for some gcc magic that tells the compiler that the function should not expect to find any of its arguments in registers (a common optimization), but only on the CPU's stack.

For the code implement of the `sys_procmem.c` is required to embedded with this report. However, the way I implement it is similar to the TODO task in the kernel module.

After the implmentation completes, we need to inform the compiler of the new source file when we rebuild the kernel. I just follow the instruction.

5 Compiling Kernel

5.1 Build the configured Kernel:

I run the Makefile in the `/kernelbuild/linux-4.4.56/` to compile the kernel and create `vmlinuz`. However I think to prevent from taking too much time to compile, all the *make* command should be run at most 4 core-processors. To compile the kernel:

```
$ make -j 4
```

To build the loadable modules:

```
$ make -j 4 modules
```

QUESTION: What is the meaning of these two stages, namely “make” and “make modules”?
ANS:

- I run ”make” inside the kernel folder to compile and link the kernel images. After the compilation completes, there will be the file named ”vmlinuz”.
- I run ”make modules” to compile the modules, leaving the compiled binaries in the build directory. However, when I search in the Internet about the difference between ”make modules” and ”make `modules_install`” they recommend to use ”make `modules_install`” if there is no modules need to be built.

5.2 Install the new Kernel:

Install the modules:

```
$ sudo make -j 4 modules_install
```

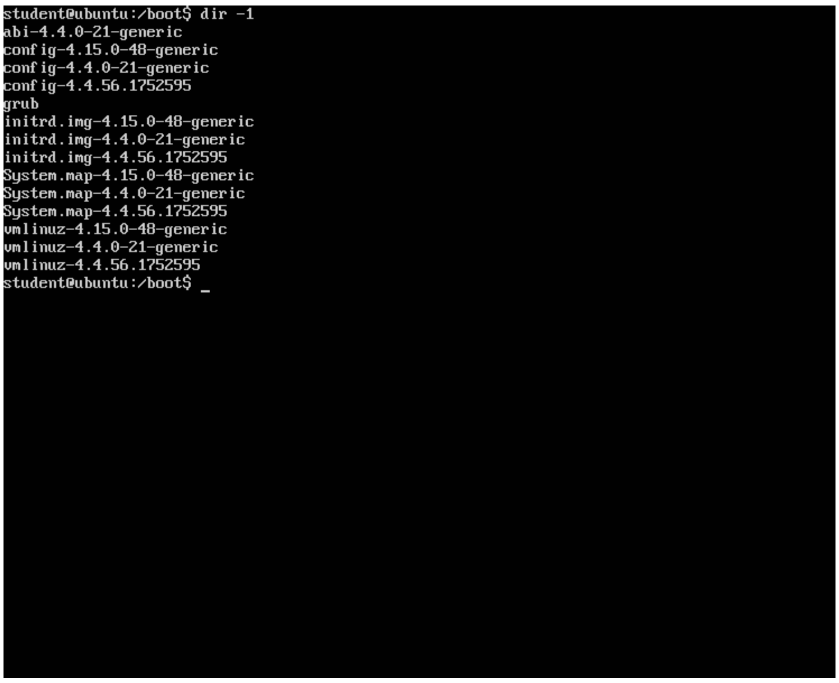
Then, if there is no errors, I can start installing new kernel:

```
$ sudo make -j 4 install
```

After the installation is successfull, reboot the virtual machine.

And there is a problem here, after the reboot I run ”uname -r” and the current kernel is still here not the new one. After searching for issues in the Internet, I came with the issues that they recommend to download and install the latest kernel into the system, and the current one is newer than the one I installed, so the system choose to boot into the default kernel.

I have already checked the /boot directory and everything is fine. So the problem here is how to make the system boot into the old kernel.



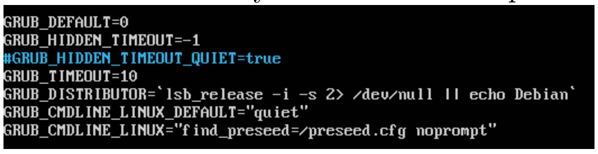
```
student@ubuntu:/boot$ dir -1
abi-4.4.0-21-generic
config-4.15.0-48-generic
config-4.4.0-21-generic
config-4.4.56.1752595
grub
initrd.img-4.15.0-48-generic
initrd.img-4.4.0-21-generic
initrd.img-4.4.56.1752595
System.map-4.15.0-48-generic
System.map-4.4.0-21-generic
System.map-4.4.56.1752595
vmlinuz-4.15.0-48-generic
vmlinuz-4.4.0-21-generic
vmlinuz-4.4.56.1752595
student@ubuntu:/boot$ _
```

I change the GRUB system file to modify the way to boot into the system.

- First I locate where the GRUB:

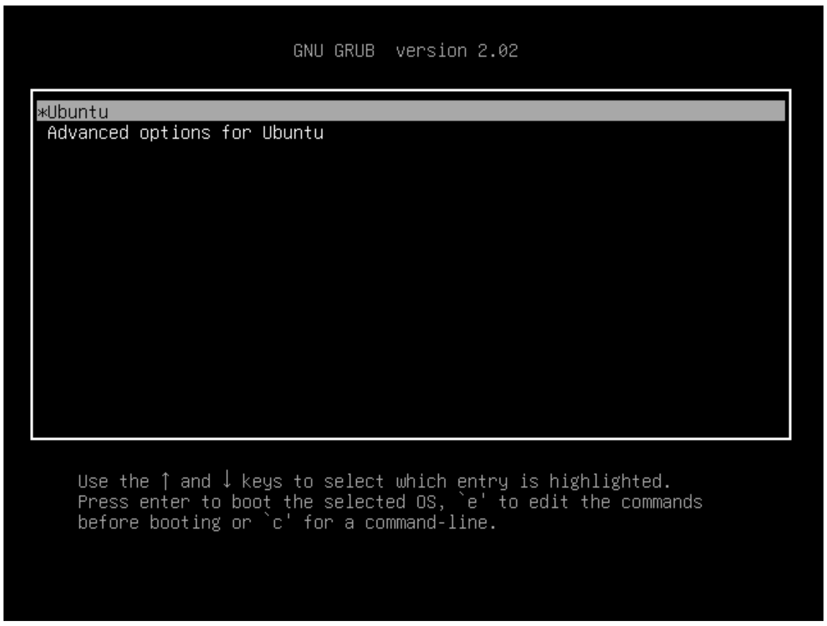
  ```
  $ whereis grub
  ```

- I use nano to modify the GRUB in the picture below:



```
GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=-1
#GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX="find_preseed=/preseed.cfg noprompt"
```

- After that I reboot system again:
- Now we have the grub menu as below; Choose "Advaned options for Ubuntu"

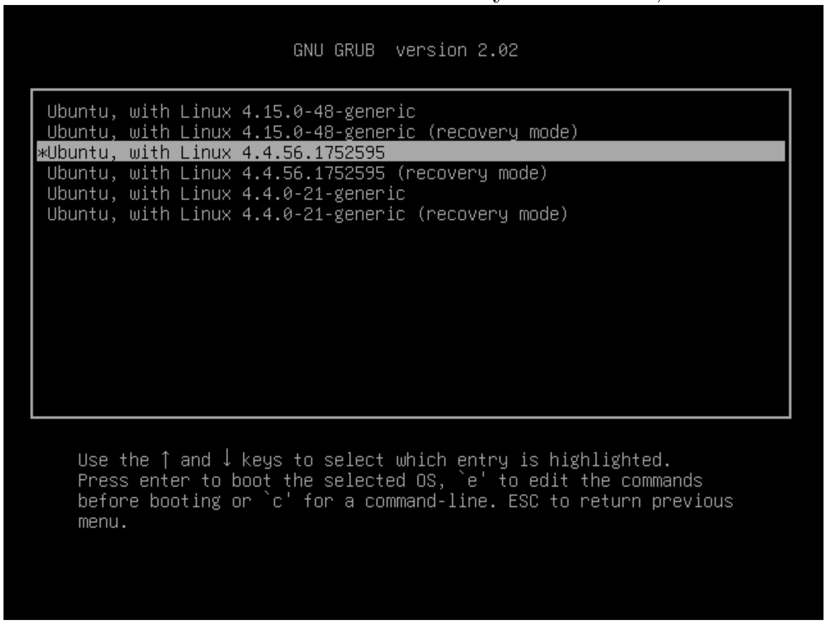


```
                GNU GRUB  version 2.02

*Ubuntu
 Advanced options for Ubuntu

   Use the ↑ and ↓ keys to select which entry is highlighted.
   Press enter to boot the selected OS, `e' to edit the commands
   before booting or `c' for a command-line.
```

- Choose to boot into the kernel with my student ID, that is the new kernel:



```
                GNU GRUB  version 2.02

 Ubuntu, with Linux 4.15.0-48-generic
 Ubuntu, with Linux 4.15.0-48-generic (recovery mode)
*Ubuntu, with Linux 4.4.56.1752595
 Ubuntu, with Linux 4.4.56.1752595 (recovery mode)
 Ubuntu, with Linux 4.4.0-21-generic
 Ubuntu, with Linux 4.4.0-21-generic (recovery mode)

   Use the ↑ and ↓ keys to select which entry is highlighted.
   Press enter to boot the selected OS, `e' to edit the commands
   before booting or `c' for a command-line. ESC to return previous
   menu.
```

5.3 Testing

I implement and compile the test code in the instructions and the result is my student ID. The system call works well. The `[number_32]` is 377 in my `syscall_32.tlb`

QUESTION: Why could this program indicate whether our system call works or not?
ANS: In the test program, it uses syscall() to call the system call with specific number(377) and pass the outputs into the array INFO. Then it call print to print the first element of INFO to console which is the student ID.

6 Wrapper

QUESTION: Why do we have to re-define `proc_segs` struct while we have already defined it inside the kernel?
ANS:

- It will give the error ”module is in use”.
- The linux kernel is only unloaded kernel when the `module_exit` function returns successfully. So, we cannot access to the struct also its parameter.
- To test it in the wrapper file, we need to re-define it again with the parameter and its name as exactly as the struct we defined in the kernel.

7 Validation

After writing the wrapper with the distinct header file and implement file, we can validate our work.

QUESTION: Why is root privilege (e.g. adding sudo before the cp command) required to copy the header fle to `/usr/include`?
ANS: We need to root privilege to copy the file since the `/usr` is belong to the system file, so it cannot be easily modified except the administrator, so we need to use `sudo` command to execute that command.

I compile the source code as shared library with the given command in instruction.

QUESTION: Why must we put `-share` and `-fpic` options into the gcc command?
ANS:

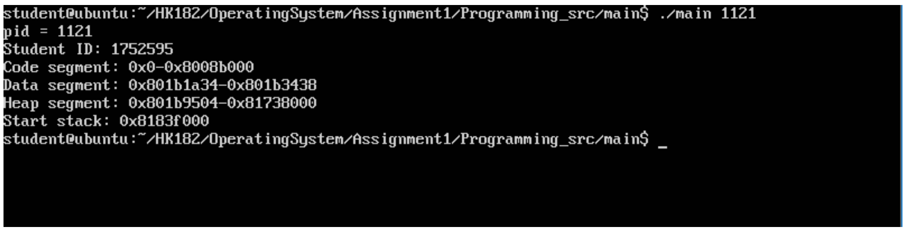
- -share: produce a shared object which can then be linked with other objects to form an executable.
- -fpic: to give the predictable result.

Valiate the work with the main.c file in the instruction. Run the compile command to compile it:

```
$ gcc -c main.c -o main.o -lprocmem
$ gcc -o main main.o -lprocmem
```

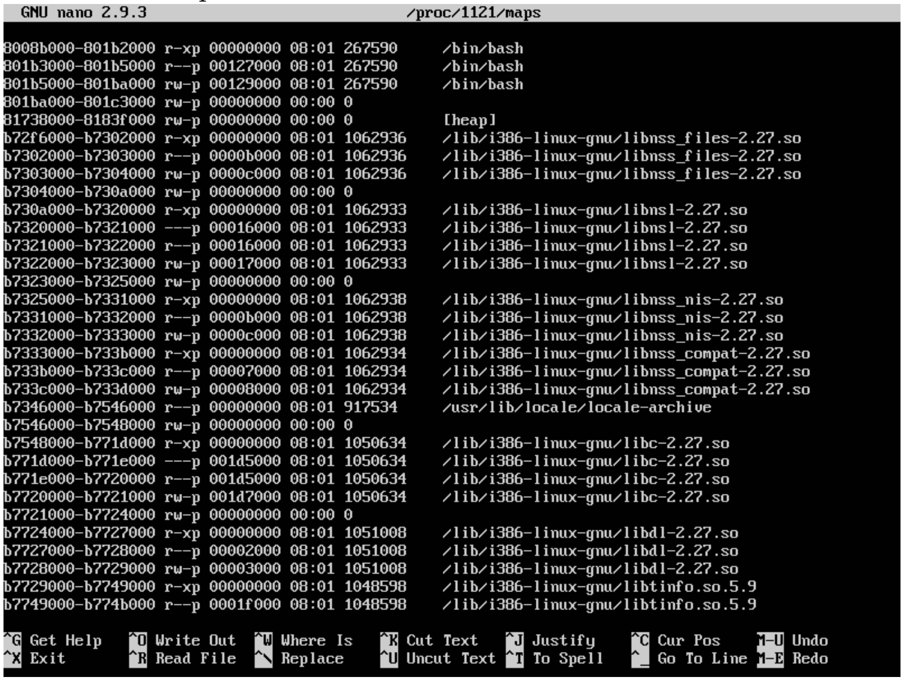
However, I got the error ”stack smashing detected ¡unknown¿ terminated Aborted”, and it is due to the struct needed to define as the pointer in the function procmem. The code in the instruction is totally wrong, and it takes me a whole day to realize it.

After everything run perfectly, we can check the result with the given one in /proc/¡pid¿/maps file.



```
student@ubuntu:~/HK182/OperatingSystem/Assignment1/Programming_src/main$ ./main 1121
pid = 1121
Student ID: 1752595
Code segment: 0x0-0x8008b000
Data segment: 0x801b1a34-0x801b3438
Heap segment: 0x801b9504-0x81738000
Start stack: 0x8183f000
student@ubuntu:~/HK182/OperatingSystem/Assignment1/Programming_src/main$ _
```

This is how maps file look like:



```
  GNU nano 2.9.3                          /proc/1121/maps

8008b000-801b2000 r-xp 00000000 08:01 267590     /bin/bash
801b3000-801b5000 r--p 00127000 08:01 267590     /bin/bash
801b5000-801ba000 rw-p 00129000 08:01 267590     /bin/bash
801ba000-801c3000 rw-p 00000000 00:00 0
81738000-8183f000 rw-p 00000000 00:00 0          [heap]
b72f6000-b7302000 r-xp 00000000 08:01 1062936    /lib/i386-linux-gnu/libnss_files-2.27.so
b7302000-b7303000 r--p 0000b000 08:01 1062936    /lib/i386-linux-gnu/libnss_files-2.27.so
b7303000-b7304000 rw-p 0000c000 08:01 1062936    /lib/i386-linux-gnu/libnss_files-2.27.so
b7304000-b730a000 rw-p 00000000 00:00 0
b730a000-b7320000 r-xp 00000000 08:01 1062933    /lib/i386-linux-gnu/libnsl-2.27.so
b7320000-b7321000 ---p 00016000 08:01 1062933    /lib/i386-linux-gnu/libnsl-2.27.so
b7321000-b7322000 r--p 00016000 08:01 1062933    /lib/i386-linux-gnu/libnsl-2.27.so
b7322000-b7323000 rw-p 00017000 08:01 1062933    /lib/i386-linux-gnu/libnsl-2.27.so
b7323000-b7325000 rw-p 00000000 00:00 0
b7325000-b7331000 r-xp 00000000 08:01 1062938    /lib/i386-linux-gnu/libnss_nis-2.27.so
b7331000-b7332000 r--p 0000b000 08:01 1062938    /lib/i386-linux-gnu/libnss_nis-2.27.so
b7332000-b7333000 rw-p 0000c000 08:01 1062938    /lib/i386-linux-gnu/libnss_nis-2.27.so
b7333000-b733b000 r-xp 00000000 08:01 1062934    /lib/i386-linux-gnu/libnss_compat-2.27.so
b733b000-b733c000 r--p 00007000 08:01 1062934    /lib/i386-linux-gnu/libnss_compat-2.27.so
b733c000-b733d000 rw-p 00008000 08:01 1062934    /lib/i386-linux-gnu/libnss_compat-2.27.so
b7346000-b7546000 r--p 00000000 08:01 917534     /usr/lib/locale/locale-archive
b7546000-b7548000 rw-p 00000000 00:00 0
b7548000-b771d000 r-xp 00000000 08:01 1050634    /lib/i386-linux-gnu/libc-2.27.so
b771d000-b771e000 ---p 001d5000 08:01 1050634    /lib/i386-linux-gnu/libc-2.27.so
b771e000-b7720000 r--p 001d5000 08:01 1050634    /lib/i386-linux-gnu/libc-2.27.so
b7720000-b7721000 rw-p 001d7000 08:01 1050634    /lib/i386-linux-gnu/libc-2.27.so
b7721000-b7724000 rw-p 00000000 00:00 0
b7724000-b7727000 r-xp 00000000 08:01 1051008    /lib/i386-linux-gnu/libdl-2.27.so
b7727000-b7728000 r--p 00002000 08:01 1051008    /lib/i386-linux-gnu/libdl-2.27.so
b7728000-b7729000 rw-p 00003000 08:01 1051008    /lib/i386-linux-gnu/libdl-2.27.so
b7729000-b7749000 r-xp 00000000 08:01 1048598    /lib/i386-linux-gnu/libtinfo.so.5.9
b7749000-b774b000 r--p 0001f000 08:01 1048598    /lib/i386-linux-gnu/libtinfo.so.5.9

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo
```

The first column is the address corresponding to the last column which is the pathname, you can check if the output of the main.c source code is valid with the address given.

The second column is the perms field which is the set of the permission.

- r = read.
- w = write.
- x = execute.
- s = shared.
- p = private.

The offset field is located in the third column

The next column is dev that is the device.

The next one is inode on that device 0 indicates that no inode is associated with the memory region.

8 References:

[1] FAQ, "Using the gnu compiler collection gcc," https://gcc.gnu.org/onlinedocs/gcc-4.3.2/gcc/Link-Options.html#Link-Options.

[2] gjaingjain 2 and dirkgentlydirkgently 91.9k15114176, "What is the 'asm-linkage' modifier meant for?" https://stackoverflow.com/questions/10459688/what-is-the-asmlinkage-modifier-meant-for.

[3] M. Kerrisk, *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press, 2012.

[4] O. Pomerantz, *Linux kernel module programming guide*. iUniverse.com, 2000.

[5] "Shared libraries with gcc on linux," https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html.

[6] S. T. Tajiryan, "How can i boot with an older kernel version?" https://askubuntu.com/questions/82140/how-can-i-boot-with-an-older-kernel-version.