Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering

-Operating Systems-

Assignment #1 Report: System Call

Name: Nguyễn Hữu Trung Nhân Student ID: 1752392

Contents:

I/ Compiling Linux Kernel:

1/ Install core packages and kernel package:

I use Oracle VM VirtualBox software to create a virtual machine that runs Ubuntu 64-bit operating system to complete this assignment 1.

In order to install the core packages (gcc, make, …) and kernel-package, there are below commands:

```
$ sudo apt-get update

$ sudo apt-get install build-essential

$ sudo apt-get install kernel-package
```

Question: Why we need to install kernel-package?

The kernel-package helps us to compile and install the new kernel easier. Because the kernel-package automatically executes the routines steps that are required to compile and install the custom kernel.

2/ Download Kernel Source:

I created a new folder named "kernelbuild" at home folder. Then download the kernel source from https://cdn.kernel.org/pub/linux/kernel/v5.x/linux -5.0.5.tar.xz to the "kernelbuild" folder. I In order to extract the "linux-5.0.5.tar.xz" file, I chose "extract here" and got the "linux-5.0.5" folder.

Question: Why we have to use another kernel source from the server such as www.kernel.org, can we compile the original kernel (the local kernel on the running OS) directly?

Because compiling directly the original kernel on the running OS is extremely risky. Because this is the first time I work with the kernel, what if I did something wrong with the original kernel, the virtual machine may crash, and there is no way to go back. Using another kernel source is safer, and I can remove some kernel packages that I don't need for this assignment to make the compiling time of the kernel faster.

3/ Configuration:

I installed some essential packages in order to customize the kernel.

```
$ sudo apt-get install fakeroot ncurses-dev xz-utils bc flex libelf-dev bison openssl libssl-dev
```

Since customizing the kernel is a complicated process, I chose to copy .config file of the original kernel currently used by the virtual machine. In the folder "linux-5.0.5", I opened a terminal and entered the command:

```
$ cp /boot/config-$ (uname -r) .config
```

Then, in order to customize the kernel, I entered the command:

```
$ make menuconfig
```

To change kernel version, I chose "General setup" option. Then I chose "Local version – append to kernel release" option. Then I entered my student ID "**.1752392**". Then I pressed F6 to save and F9 to exit.

4/ Build Configured Kernel:

In the folder "linux-5.0.5", I open the terminal and entered the commands:

```
$ make -j 4

$ make -j 4 modules
```

Question: What is the meaning of these two stages, namely "make" and "make modules"? What are created and what for?

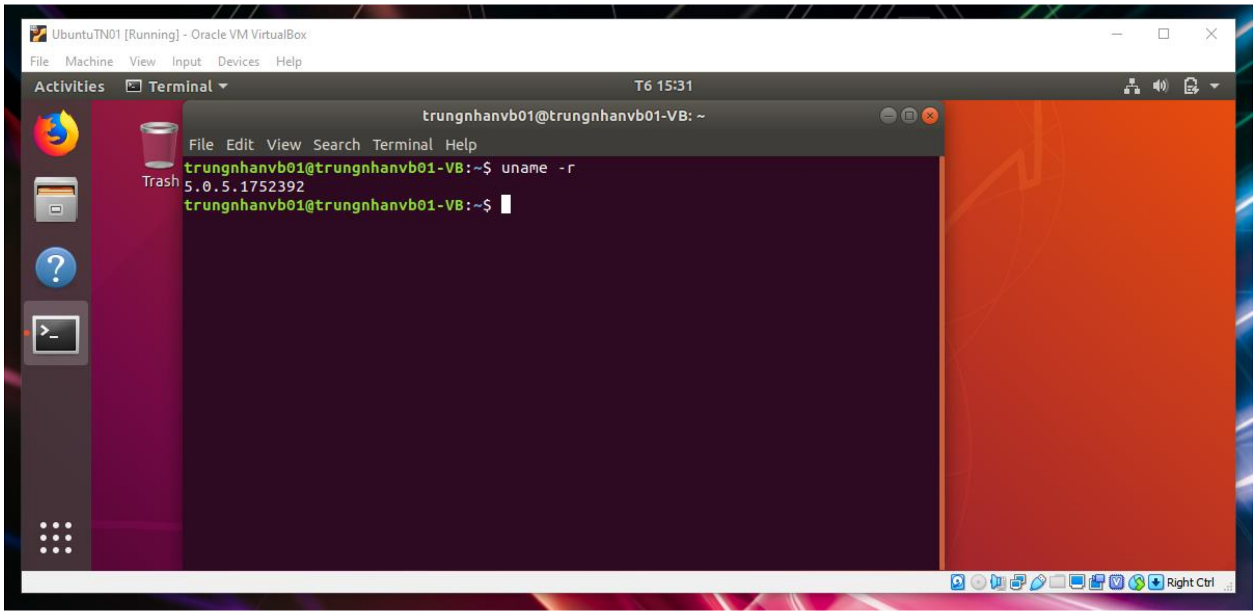
"make" compiles and links the kernel image. This step creates vmlinuz file which is the name of the Linux kernel executable. While "make modules" compiles individual files for each question you answered M during kernel config. The object code is linked against your freshly built kernel. (For questions answered Y, these are already part of vmlinuz, and for questions answered N they are skipped).

5/ Install new kernel:

```
$ sudo make -j 4 modules_install

$ sudo make -j 4 install
```

After install the kernel, I restart the virtual machine.

My Result:


UbuntuTN01 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
T6 15:31
trungnhanvb01@trungnhanvb01-VB: ~
File Edit View Search Terminal Help
Trash
Right Ctrl

```
trungnhanvb01@trungnhanvb01-VB:~$ uname -r
5.0.5.1752392
trungnhanvb01@trungnhanvb01-VB:~$ 
```

II/ Trim A Kernel:

In the “linux-5.0.5” folder, I opened the terminal and entered the command:

```
$ make localmodconfig
```

This command will create a config based on current config and loaded modules and disables any module option that is not needed for the loaded modules. When I “make localmodconfig”, I enter “no” whenever I was asked to install new module. After trimming a kernel, the compilation time of my kernel is about 10 minutes and there were only about 59 modules in my kernel.

III/ A new system call:

1/ Implementation:

In “linux-5.0.5” folder, I created “get_proc_info” folder, and inside “get_proc_info” folder, I created “sys_get_proc_info.c”:

```
#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/stat.h>

#include <linux/unistd.h>

#include <linux/sched.h>

#include <linux/sched/signal.h>
```

```
#include <linux/string.h>

#include <linux/list.h>

#include <linux/syscalls.h>

#include <linux/types.h>

#include <linux/errno.h>



struct proc_info {

	pid_t pid;

	char name [16];

};



struct procinfos {

	long studentID;

	struct proc_info proc;

	struct proc_info parent_proc;

	struct proc_info oldest_child_proc;

};
```

```
SYSCALL_DEFINE2(get_proc_info, pid_t, PID, struct procinfos*, info){


	int found_pid = 0;

	struct task_struct *task_list;

	struct task_struct *temp01;

	struct task_struct *temp02;


	info->studentID = 1752392;

	printk(KERN_INFO "My Student ID: %ld\n", info->studentID);

	if (PID == -1){

		info->proc.pid = current->pid;

		strcpy(info->proc.name, current->comm);


		info->parent_proc.pid = current->parent->pid;

		strcpy(info->parent_proc.name, current->parent->comm);


		if (list_empty(&current->children)){

			info->oldest_child_proc.pid = -1;
```

```
                strcpy(info->oldest_child_proc.name, "Nothing");

                return 0;

        }

        temp02 = list_first_entry_or_null(&current->children, struct task_struct, sibling);

        info->oldest_child_proc.pid = temp02->pid;

        strcpy(info->oldest_child_proc.name, temp02->comm);

        return 0;

}



for_each_process(task_list){

        if (task_list->pid == PID){

                found_pid = 1;

                info->proc.pid = PID;

                strcpy(info->proc.name, task_list->comm);



                info->parent_proc.pid = task_list->parent->pid;

                strcpy(info->parent_proc.name, task_list->parent->comm);
```

```
                        if (list_empty(&task_list->children)){

                                info->oldest_child_proc.pid = -1;

                                strcpy(info->oldest_child_proc.name, "Nothing");

                                return 0;

                        }

                        temp01 = list_first_entry_or_null(&task_list->children, struct task_struct,
sibling);

                        info->oldest_child_proc.pid = temp01->pid;

                        strcpy(info->oldest_child_proc.name, temp01->comm);

                }

        }



        if (found_pid == 0){

                return EINVAL;

        }

        return 0;

}
```

Also inside “get_proc_info” folder, I created “Makefile” file, inside “Makefile” file, I wrote “obj-y := sys_get_pro_info.o”

In the Makefile of “linux-5.0.5” folder, I added “ get_proc_info/”:

```
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ get_proc_info/
```

In “~/kernelbuild/linux-5.0.5/arch/x86/entry/syscalls”, at the end of “syscall_64.tbl”, I added the following line according to the syscall number:

```
548   64      get_proc_info       __x64_sys_get_proc_info
```

<u>Question:</u> What is the meaning of fields of the line that just add to the system call table (548, 64, get proc info, i.e.)

Each system call in “syscall_64.tbl” file has the format <number> <abi> <name> <entry point>:

<number>: is the number order of the system call, system calls are called from user space through this number.

<abi>: the application binary interface, there are three types 64, x32 and common.

<name>: the name of the system call.

<entry point>: the name of the function that handles the system call.

Then, I added the prototype to the file “syscalls.h” in the directory: ~/kernelbuild/linux-5.0.5/include/linux

And I added the following lines at the end of the file before “#endif”:

```
struct proc_info;
struct procinfos;
asmlinkage long sys_get_proc_info(pid_t pid, struct procinfos* info);
```

<u>Question:</u> What is the meaning of each line above?

Line 1 and line 2 declare new struct which are used by my new system call.

Line 3 declares new syscall function.

After that, I recompile the kernel and restart the virtual machine.

2/ Testing:

I followed the C program provided in “Assignment_1_OS_2019_V1.pdf” to test my new system which has been integrated into the kernel:

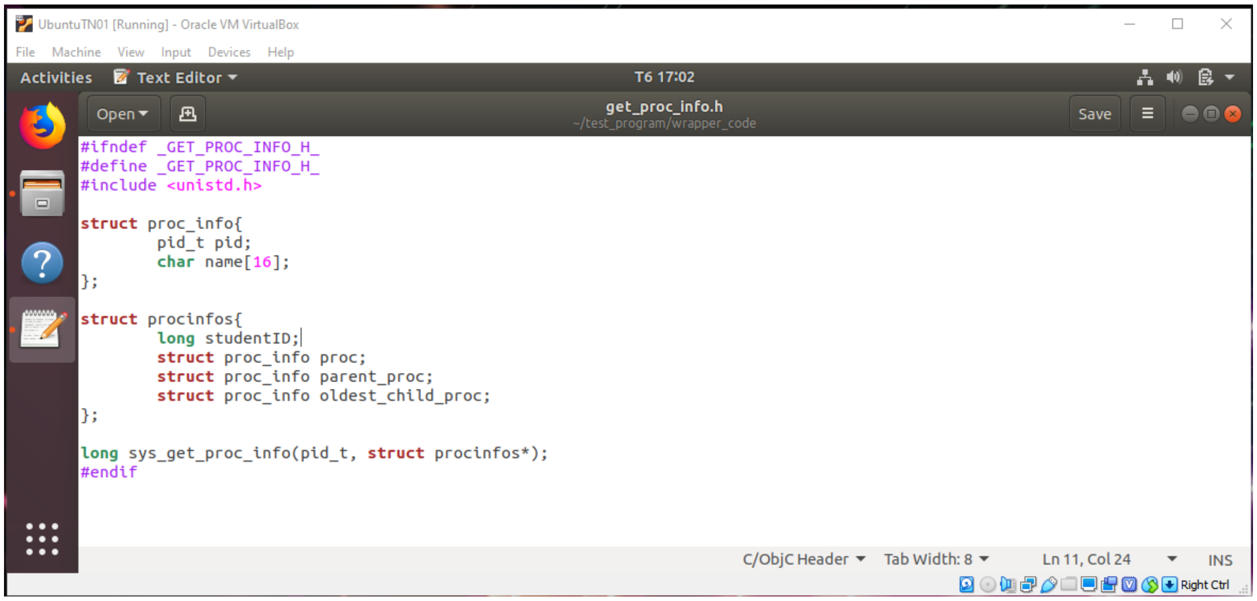
```c
#include <sys/syscall.h>
#include <stdio.h>
#include <unistd.h>

#define SIZE 200

int main(){
    long sys_return_value;
    unsigned long info[SIZE];
    sys_return_value = syscall(548, -1, &info);
    printf("My student ID: %lu\n", info[0]);
    return 0;
}
```

Question: Why this program could indicate whether our system call works or not?

Because I called my new system call (syscall (548, -1, &info))in the kernel table, so if the system call returns “0” and the program printed my student ID to the terminal and return 0 then my system call works and if the program printed the trash value in the terminal, this means that the program does not understand my new system call then my new system call has not been added to the kernel successfully.

3/ Wrapper:

I created “get_proc_info.h”:

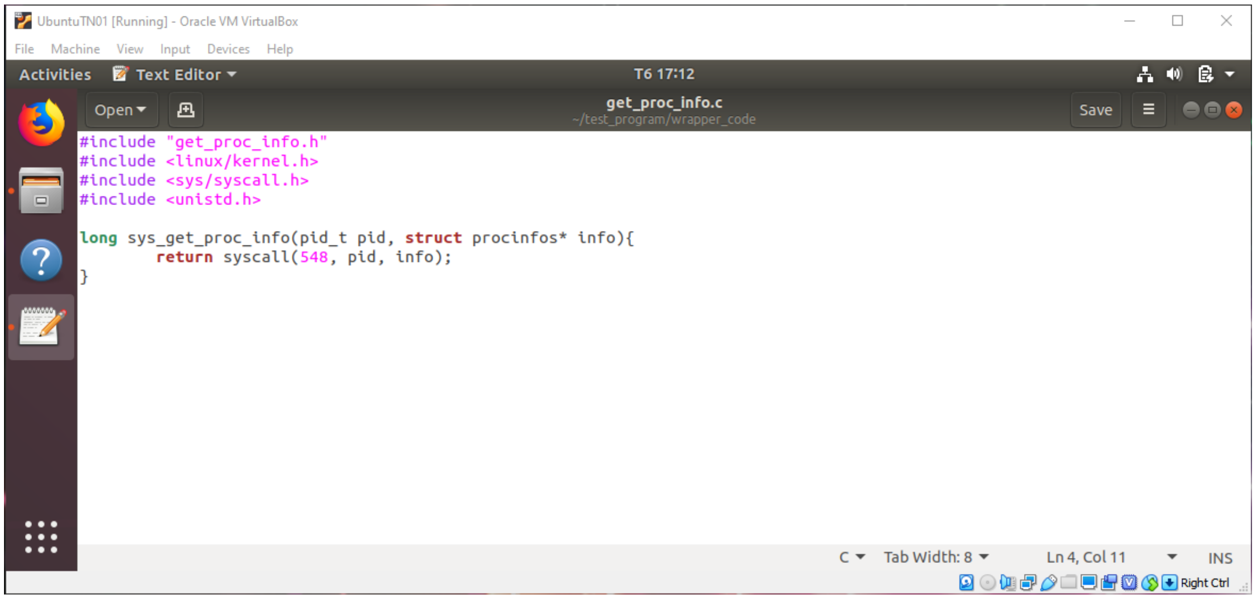

UbuntuTN01 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Text Editor
T6 17:02
Open
get_proc_info.h
~/test_program/wrapper_code
Save
C/ObjC Header
Tab Width: 8
Ln 11, Col 24
INS
Right Ctrl

```c
#ifndef _GET_PROC_INFO_H_
#define _GET_PROC_INFO_H_
#include <unistd.h>

struct proc_info{
        pid_t pid;
        char name[16];
};

struct procinfos{
        long studentID;
        struct proc_info proc;
        struct proc_info parent_proc;
        struct proc_info oldest_child_proc;
};

long sys_get_proc_info(pid_t, struct procinfos*);
#endif
```

Question: Why we have to redefine procinfos and proc info struct while we have already defined it inside the kernel?

Because I have just defined the structs in syscalls.h which located in the kernel space. In order to make my system call “friendly user” and easier to call it, I have to redefine these structs and function again in user space for the user to use my new system call easier.

And then, I created “get_proc_info.c”:


UbuntuTN01 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Text Editor
T6 17:12
Open
get_proc_info.c
~/test_program/wrapper_code
Save
C
Tab Width: 8
Ln 4, Col 11
INS
Right Ctrl

```
#include "get_proc_info.h"
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

long sys_get_proc_info(pid_t pid, struct procinfos* info){
        return syscall(548, pid, info);
}
```

4/ Validation:

I then copied the header “get_proc_info.h” to the header directory /usr/include:

```
$ sudo cp <my own path to get_proc_info.h> /usr/include
```

Question: Why root privilege (e.g. adding sudo before the cp command) is required to copy the header file to /usr/include?

Because those folders are at the root directory, I must switch to the super user mode (sudo) in order to change any thing inside the root directory.

I then compiled the source code file as a shared object:

```
$ gcc -shared -fpic get_proc_info.c -o libget_proc_info.so
```

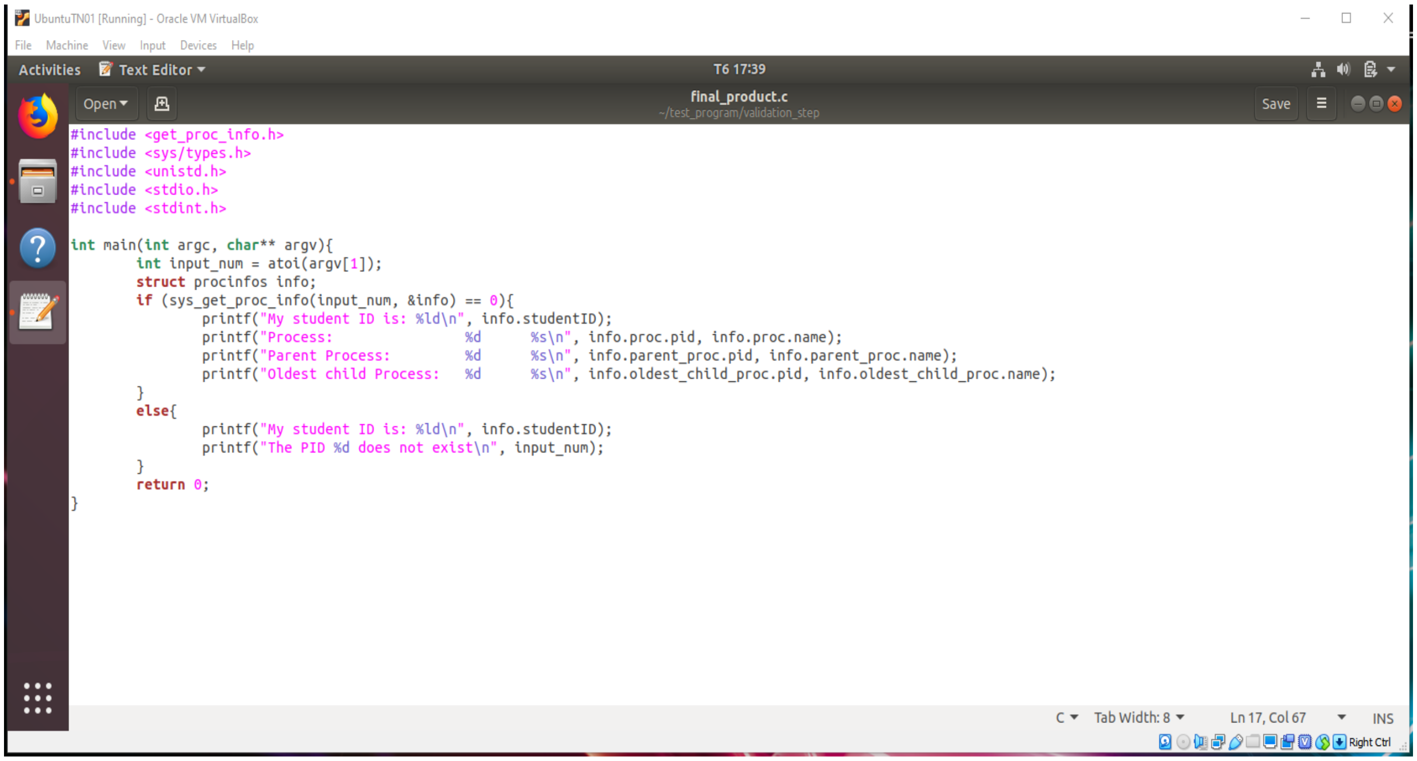
And then I copied the shared object file to the library directory:

```
$ sudo cp <my own path to .so file> /usr/lib
```

<u>Question:</u> Why we must put -share and -fpic option into gcc command?

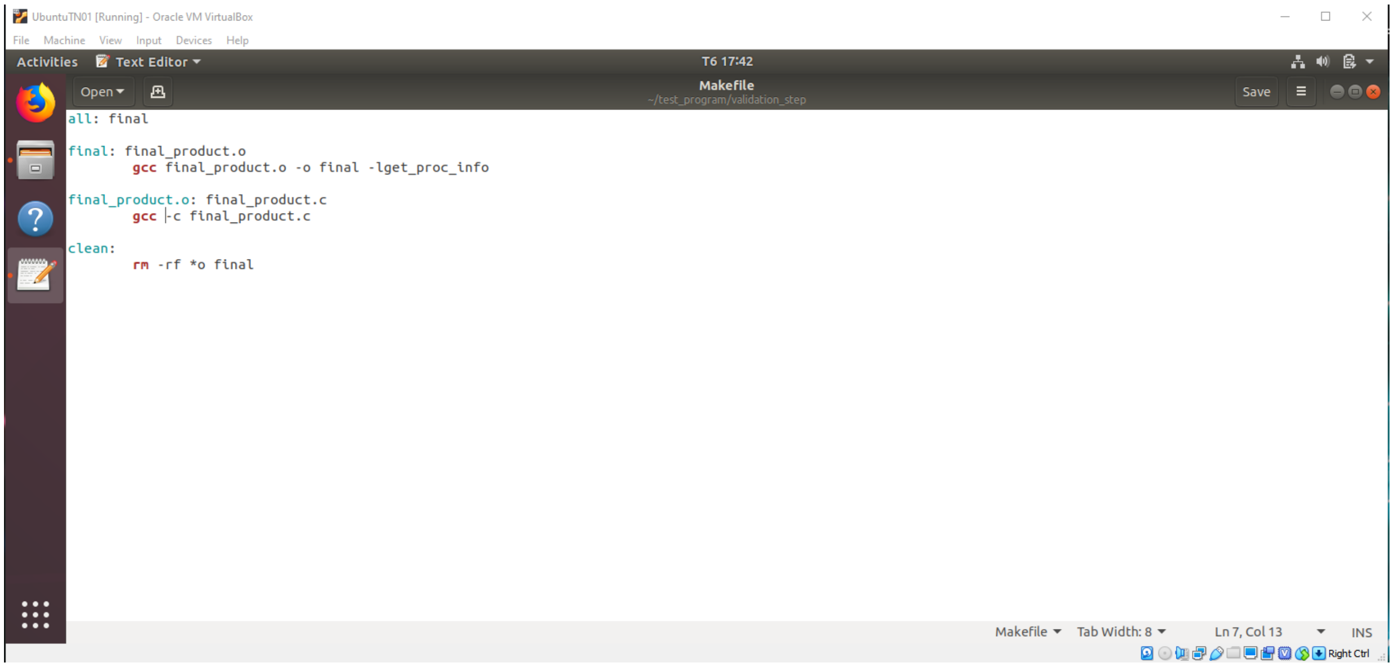
"-shared" produces a shared object which can then be linked with other objects to form an executable. "-fpic" means that the generated machine code does not depend on being located at a specific address in order to work.

I finally wrote my own C program to check my new system call:

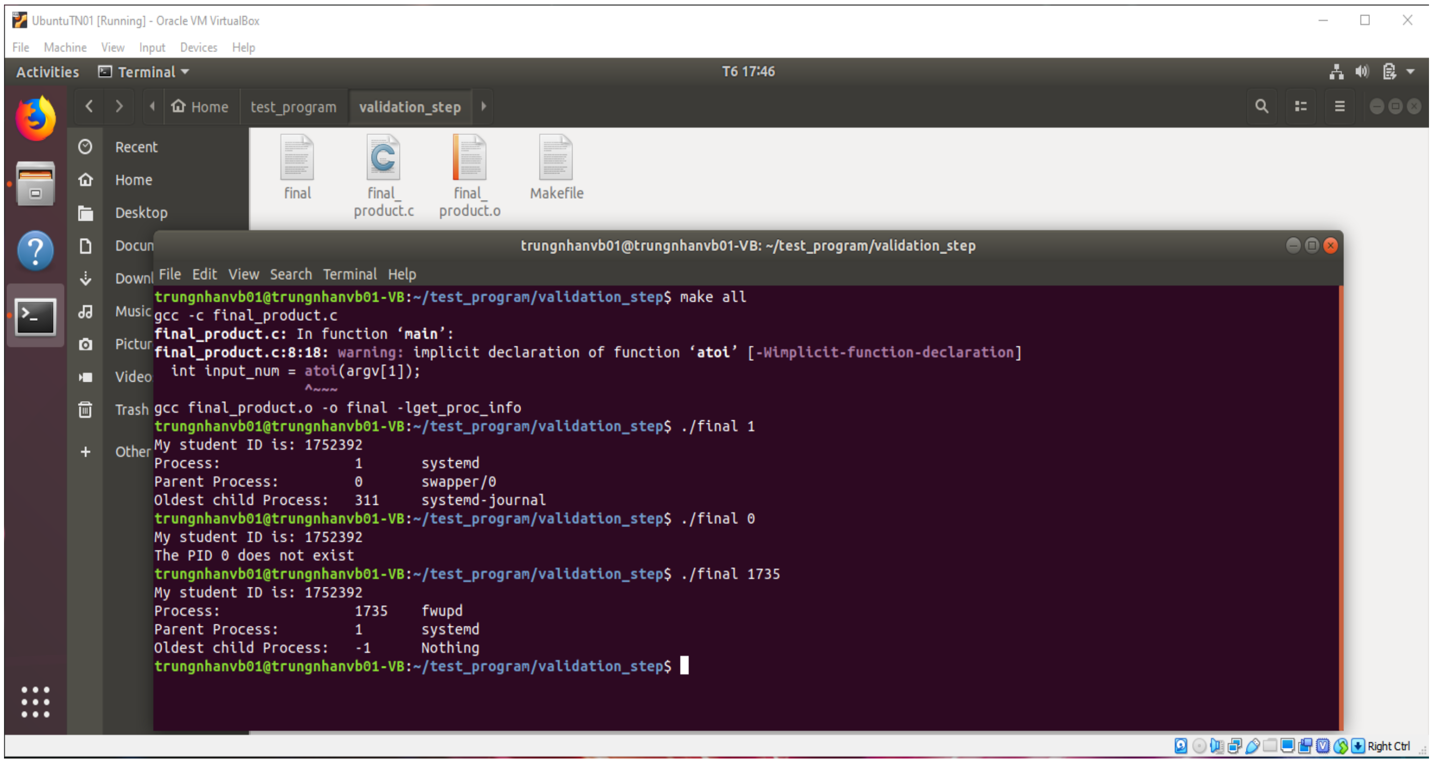


```c
#include <get_proc_info.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdint.h>

int main(int argc, char** argv){
        int input_num = atoi(argv[1]);
        struct procinfos info;
        if (sys_get_proc_info(input_num, &info) == 0){
                printf("My student ID is: %ld\n", info.studentID);
                printf("Process:                %d      %s\n", info.proc.pid, info.proc.name);
                printf("Parent Process:         %d      %s\n", info.parent_proc.pid, info.parent_proc.name);
                printf("Oldest child Process:   %d      %s\n", info.oldest_child_proc.pid, info.oldest_child_proc.name);
        }
        else{
                printf("My student ID is: %ld\n", info.studentID);
                printf("The PID %d does not exist\n", input_num);
        }
        return 0;
}
```

This is my Makefile file of my C program:


```
all: final

final: final_product.o
	gcc final_product.o -o final -lget_proc_info

final_product.o: final_product.c
	gcc -c final_product.c

clean:
	rm -rf *o final
```

This is my final result:


```
trungnhanvb01@trungnhanvb01-VB:~/test_program/validation_step$ make all
gcc -c final_product.c
final_product.c: In function ‘main’:
final_product.c:8:18: warning: implicit declaration of function ‘atoi’ [-Wimplicit-function-declaration]
  int input_num = atoi(argv[1]);
                  ^~~~
gcc final_product.o -o final -lget_proc_info
trungnhanvb01@trungnhanvb01-VB:~/test_program/validation_step$ ./final 1
My student ID is: 1752392
Process:                1       systemd
Parent Process:         0       swapper/0
Oldest child Process:   311     systemd-journal
trungnhanvb01@trungnhanvb01-VB:~/test_program/validation_step$ ./final 0
My student ID is: 1752392
The PID 0 does not exist
trungnhanvb01@trungnhanvb01-VB:~/test_program/validation_step$ ./final 1735
My student ID is: 1752392
Process:                1735    fwupd
Parent Process:         1       systemd
Oldest child Process:   -1      Nothing
trungnhanvb01@trungnhanvb01-VB:~/test_program/validation_step$
```