**HCMC University Of Technology**

**Faculty Of Computer Science & Engineering**


# Course: Operating System

# Assignment #1 – System Call

**TA:** Vũ Văn Thống

**Student:** Đặng Nhật Quân

**Student ID:** 1813694

*Ho Chi Minh, 5/2020*

# Contents

# 1. Compiling the new kernel

First thing we have to do is Set up Virtual machine. We have to update and install core packages by command line:

```
$ sudo apt-get update
$ sudo apt-get install build-essential
$ sudo apt-get install kernel-package
```

**QUESTION:** Why we need to install kernel-package ?

**ANSWER:** Because kernel-package is a package contains many kernel version, one of them will be suitable for your OS's configuration. Besides that, kernel-package is also used to compile and install a custom kernel. In this assignment, we gonna create a new system call and add it into kernel, make this kernel become unique.

Next, we create a kernel dirctory and dowload linux-4.4.21.tar.xz into it and decompression:

```
$ mkdir ~/kernelbuild                                          //Create dirctory
$ cd ~/kernelbuild                                             //Go inside dirctory
$ wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux -4.4.21.tar.xz   //Dowload linux
$ tar –xvJf linux-4.4.21.tar.xz                                //Decompress
```

**QUESTION:** Why we have to use another kernel source from the server such as http://www.kernel.org, can we compile the original kernel (the local kernel on the running OS) directly ?
**ANSWER:** There are some advanteges of using a new kernel form server such as http://www.kernel.org :

- Process some special requirement or process or handle conflict between hardware and default kernel.
- Using many kernel options which is not supported for default kernel.
- Optimize kernel.
- Running kernel version for developer.

Yes, we can also compile the original kernel but we have to check to make sure the kernel version is compatible to our configuration and in case we have problem while working with original kernel, system crash can happen. If we use original kernel, make sure there is nothing wrong.

# Configuration:

We borrow the content of configuration file of an existing kernel currently used by our virtual machine. Now we copy it from /boot/config-3.11.0-15-generic to ~/kernelbuild/.config( my original kernel is 3.11.0-15-generic ) by command line:

$ cp /boot/config-3.11.0-15-generic ~/kernelbuild/.config

After that we install libncurses5-dev package to edit configure file by command line:

$ sudo apt-get install libncurses5-dev

Then we go to inside linux-4.4.21 directory and open Kernel Configuration by command line:

$ make nconfig or $ make menuconfig

Now we select General set up  and Local version. Then enter ".1813694" ( my student ID). Finally save and exit.After that we install opendssl – a missing package during compiling. By using command line:

$ sudo apt-get install openssl libssl-dev

# Build the configured kernel

To compile the kernel and create vmlinuz, we run make  -j 4 command line and then build the loadable kernel modules by using commnad line make –j 4 modules

$ make –j 4

$ make –j 4 modules

**QUESTION :** What is the meaning of these two stages, namely "make" and "make modules"?

**ANSWER :**

- *make* compiles and links the kernel image. Then create a single file name vmlinuz.
- *make modules* compiles individual module files for each question we answered M (module) during kernel configuration.

# Installing the new kernel

First install the modules

$ sudo make –j 4 modules_install

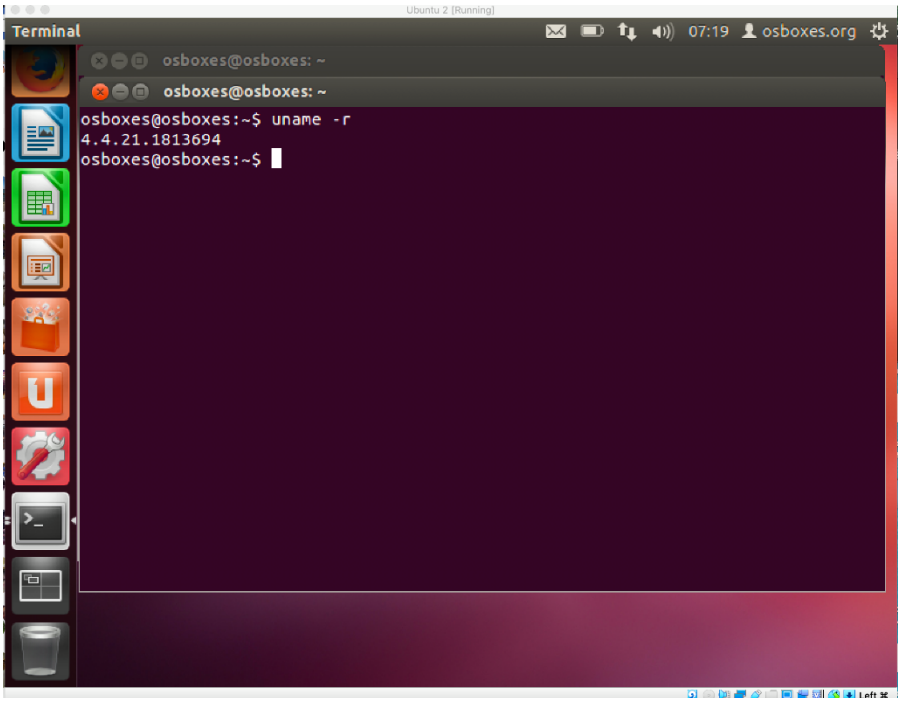
Then install the kernel

$ sudo make –j 4 install

After running, now we can check your kernel by reboot the OS and using this command line:

$ uname –r

to check your kernel version.




# 2. System call

# Adding new system call

In this assignment we will try to create a new system call whose mission is sumary the information of a process in kernel mode and then send it to a variable in user mode. We will create a new struct :

```
struct proc_segs {

        unsigned long mssv;
        unsigned long start_code;
        unsigned long end_code;
        unsigned long start_data;
        unsigned long end_data;
        unsigned long start_heap;
        unsigned long end_heap;
        unsigned long start_stack;
};
```

And prototype of system call is :

```
        long procmem(int pid, struct prog_segs * info);
```

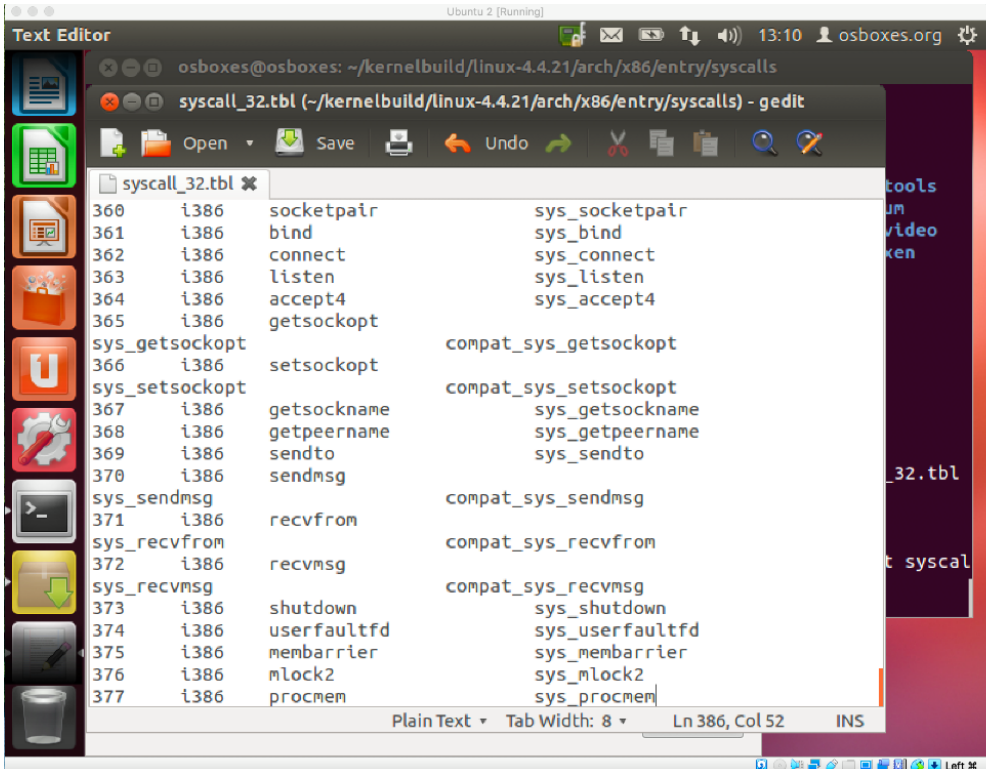
The first thing to do is adding the new system call to the kernel. Go to:

~/kernelbuild/linux-4.4.21/arch/x86/entry/syscalls.

Then using command line:

$ gedit syscall_32.tbl

to edit syscall_32.tbl file and add system at the end of the file :

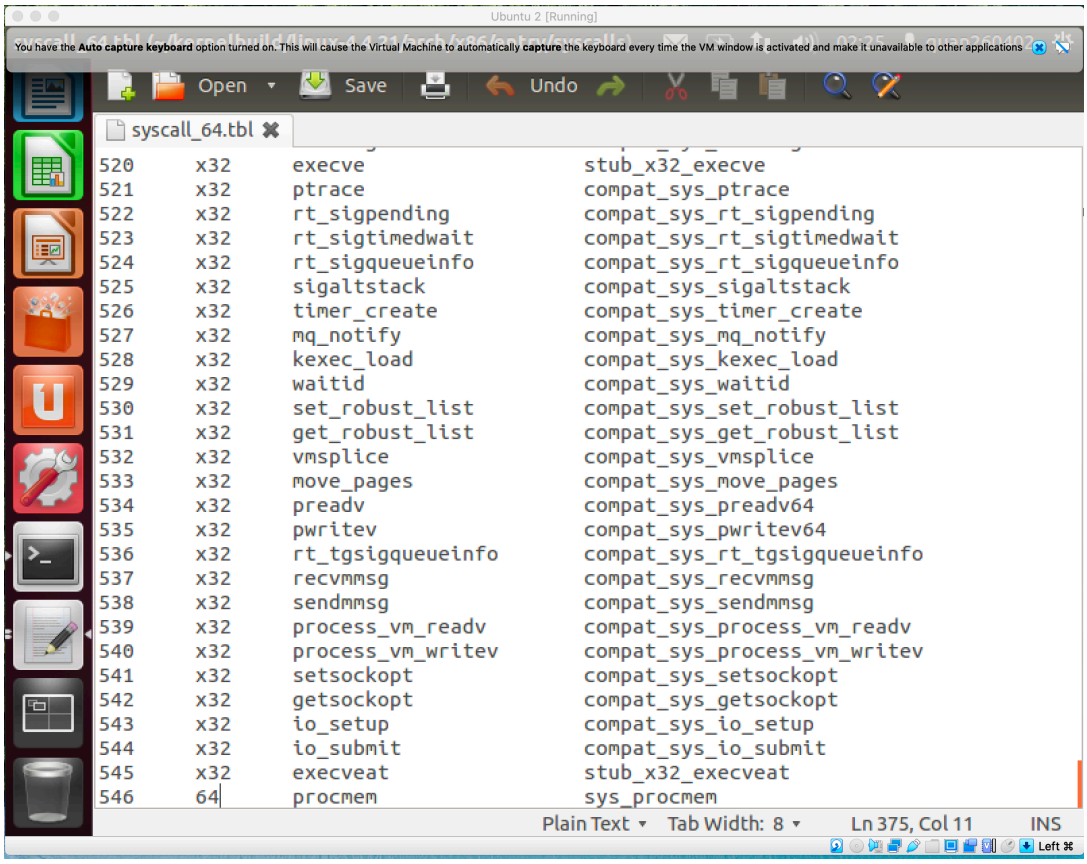


**QUESTION:** What is the meaning of the other parts, i.e. i386, procmem, sys_procmem ?

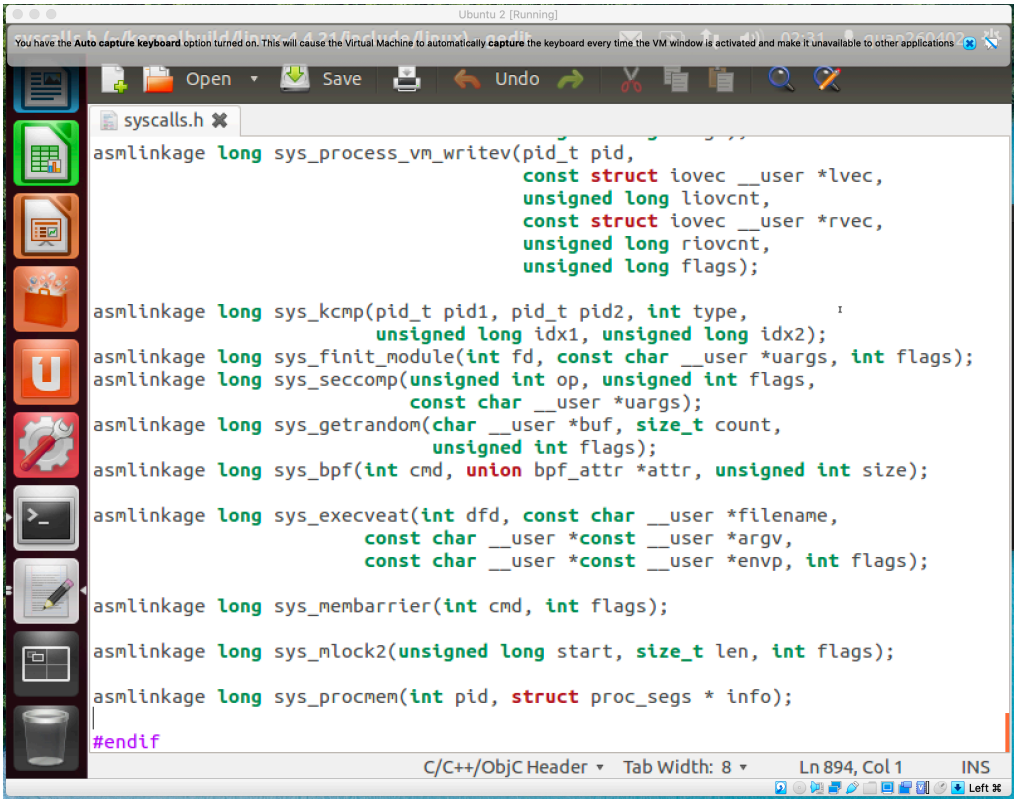
**ANSWER :**

- [number] : all syscall defined by a number to call the system call.
- [i386] : is an Aplication Binary Interface ( ABI)
- [procmem]: this is the name of out system call.
- [sys_procmem]: the entry point , the name of function called to process the system call. The rule to name is " sys_[name] ". For example, " sys_procmem ".

Then save it and to the same thing with syscall_64.tbl :



After that, explicitly define this system call. To do so, we must add necessary information to kernel's header files. Open the file include/linux/syscalls.h and add the following line to the end of this file:




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

**ANSWER :**

struct proc_segs; // declare a struct of arguement which using in function
asmlinkage long sys_procmem(int pid, struct proc_segs * info);

A system call return long variable. The first arguement pid is the process ID which the user want to get the information .The second arguement info "asmlinkage" is a tag which is defined to notice to gcc compilers that this system call is not looking forward to find out any arguements in registers in common but in stack of CPU.All systemcall which is defined by tag 'asmlinkage' tag find their arguement in stack.

# System call implement

Now we implement our system call. We will create a new source file named sys_procmem.c in arch/x86/kernel directory:

The first thing to do in this file is declare new struct. And then implement system call.To get information of process, Linux data structure provide us a task_struct which describes a process or task in the system.

**task_struct**

Each `task_struct` data structure describes a process or task in the system.

```
/* open file information */
  struct files_struct  *files;
/* memory management info */
  struct mm_struct     *mm;
/* signal handlers */
  struct signal_struct *sig;
#ifdef __SMP__
```

We can see, task_struct have a component is struct mm_struct mm ( mm means memory ).

**mm_struct**

The `mm_struct` data structure is used to describe the virtual memory of a task or process.

```
struct mm_struct {
  int count;
  pgd_t * pgd;
  unsigned long context;
  unsigned long start_code, end_code, start_data, end_data;
  unsigned long start_brk, brk, start_stack, start_mmap;
  unsigned long arg_start, arg_end, env_start, env_end;
  unsigned long rss, total_vm, locked_vm;
  unsigned long def_flags;
  struct vm_area_struct * mmap;
  struct vm_area_struct * mmap_avl;
  struct semaphore mmap_sem;
};
```

Let take a detail look about struct mm_struct. They have all component have the same meaning with the component of struct proc_segs of us. So we will declare a variable which is belong to **struct task_struct.**

Then we will use for_each_process for searching process have the same pid. We have some condition to make sure the pid is valid:

```
for_each_process(proc){
        if( proc->pid == pid){
                if(proc->mm != NULL){
                        //Do the code
                }
        }
}
```

Inside of the code, we declare a buffer variable belong to **struct proc_segs.** which will contains the value of memory of process's information. And then we use copy_to_user to copy *buff* (in kernel mode) to *info* (in user mode).

### Name

copy_to_user — Copy a block of data into user space.

### Synopsis

```
unsigned long copy_to_user (void __user *  to,
                            const void *    from,
                            unsigned long  n);
```

### Arguments

*to*

Destination address, in user space.

*from*

Source address, in kernel space.

*n*

Number of bytes to copy.

We will add an condition to make this copy process will do right thing or not.

```
int flag = copy_to_user(info,&buff,sizeof(buff);
if(flag != 0) return -1;
else return 0;
```

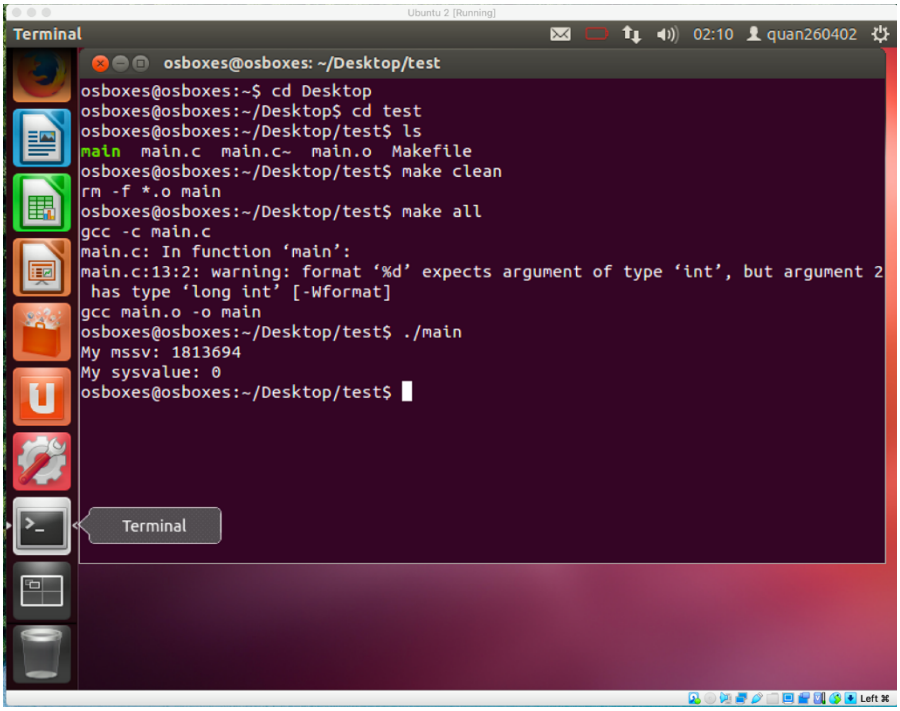
# Compiling and Installation.

Then we recompile and reinstall just like step 2 and create a small C program to check it work

# 3. Testing

```c
#include <sys/syscall.h>
#include <stdio.h>
#define SIZE    10

int main() {
     long sysvalue;
     unsigned long info[SIZE];
     sysvalue = syscall([number_32], 1, info);
     printf("My MSSV: %ul\n", info[0]);
}
```




**QUESTION :** Why this program could indicate whether our system works or not?

**ANSWER :** This program can make sure that our system works or not because in this program, we call syscall([number_32],1,info) , it will call the system call in syscall_32.tbl or syscall_64.tbl and return value into sysvalue.
If sysvalue == 0 mean it worked else sysvalue == -1 it mean it have something wrong.
We can also check the result by info[0], if it work, it will return my Student ID.

# 4. Validation

```
#ifndef _PROC_MEM_H_
#define _PROC_MEM_H_
#include <unistd.h>

struct proc_segs {
     unsigned long mssv;
     unsigned long start_code;
     unsigned long end_code;
     unsigned long start_data;
     unsigned long end_data;
     unsigned long start_heap;
     unsigned long end_heap;
     unsigned long start_stack;
};
long procmem(pid_t pid, struct prog_segs * info);
#endif // _PROC_MEM_H_
```

After that, we create a procmem.h and copy it to /usr/include contain abow

**QUESTION:** Why we have to re-define proc segs struct while we have already defined it inside the kernel?

**ANSWER:** The last we define proc_segs when it still in kernel mode, now we are in user mode so to do it conveniently, we define it one again.

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

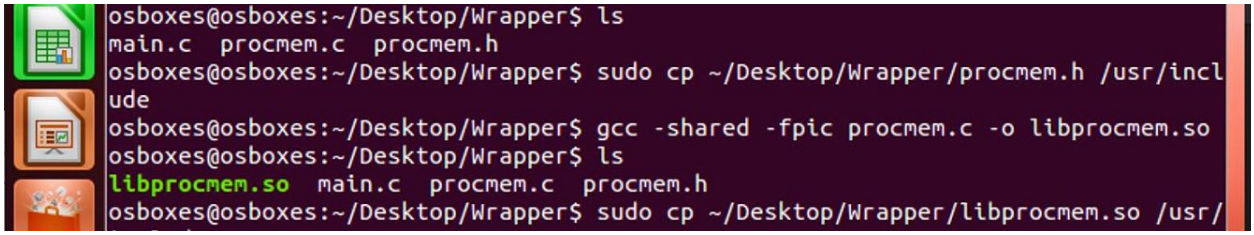
**ANSWER: /usr** is a child directory of root so to work with this data, we have to allocate permission for every single command line. We can see if we do not add "sudo" before command line, it will return a noctice that permission is denied. Clearly, sudo mean "super user do", we use "sudo" to work with administrative applications .

# 5. Wrapper

Next step, we create a file named procmem.c to hold source code file for wrapper.
In this file, we just declare a long sysvalue to contain the value of system call and return sysvalue.

We then compile our source code as a shared object to allow user to integrate our system call to their applications. To do so, run the following command:

$ gcc -shared -fpic procmem.c -o libprocmem.so



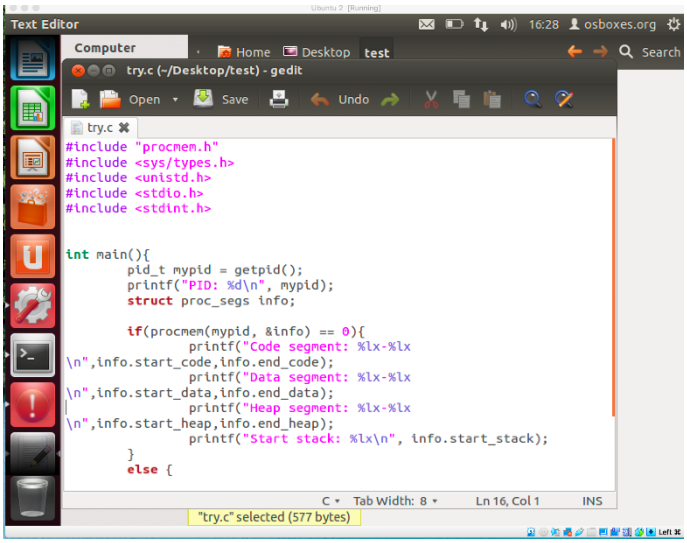

Then copy libprocmem.so to /usr/lib.

**QUESTION**: Why we must put -share and -fpic option into gcc command?

**ANSWER:** -shared: use to create a public library for conveniently using.

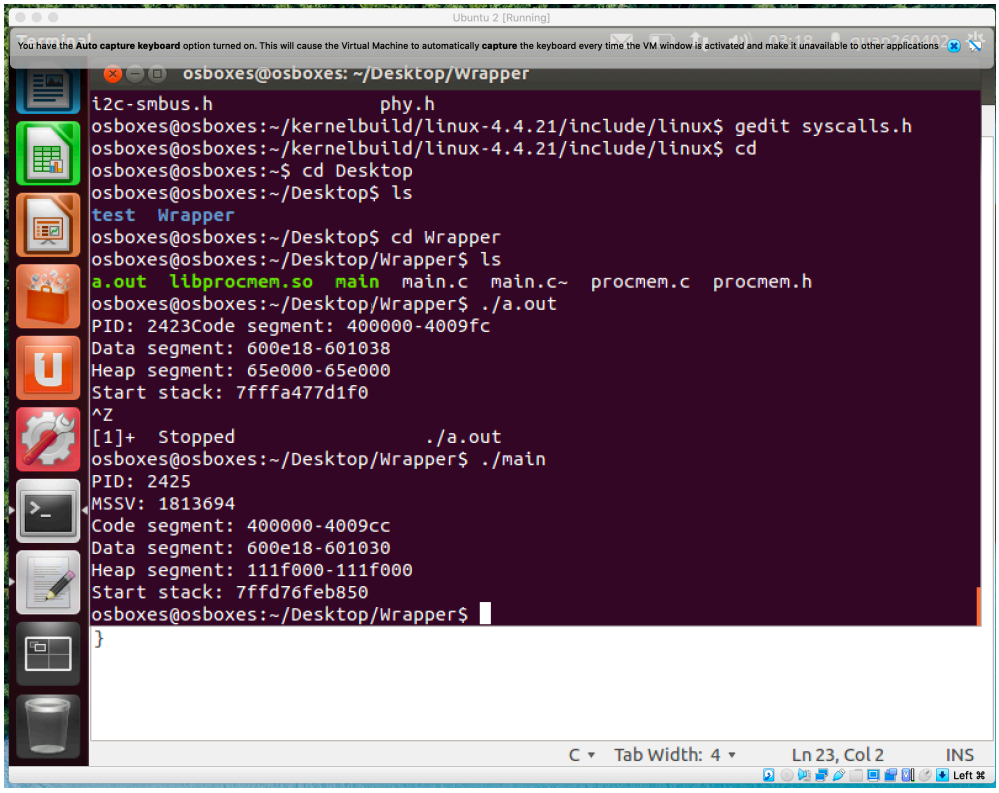
-fpic: create the independence of shared-library.

Final step, we write  a program as below and compile it with option *-lprocmem*:




Compile with -lprocmem option

$ gcc -o main main.c lprocmem



# 6. Documentary

https://www.kernel.org/doc/html/latest/vm/active_mm.html

https://www.tldp.org/LDP/tlk/ds/ds.html

https://gist.github.com/anryko/c8c8788ccf7d553a140a03aba22cab88

https://www.fsl.cs.sunysb.edu/kernel-api/re256.html

- Thank you for reading -