

**HCMC University Of Technology**  
**Faculty Of Computer Science & Engineering**



---

## **Course: Operating System**

---

### **Assignment #1 – System Call**

---

**TA: Vũ Văn Thông**

**Student: Đặng Nhật Quân**

**Student ID: 1813694**



# Contents

0. Preparation	3
1. Adding a new systemcall	4
2. System call Implementation	6
3. Compilation and Installation Process	6
4. Make API for system call	7

## 0. Preparation:

The first thing to do in this assignment is to prepare a new kernel. Our mission is following the instruction in the file PDF, downloading, installing and compiling a new kernel. Before installing and compiling, we rename the new kernel version with student ID:

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

**ANSWER:** Because kernel-package is a package containing many kernel versions, 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.

```
osboxes@osboxes: ~$ uname -r
4.4.21.1813694
osboxes@osboxes: ~$
```

**QUESTION:** Why do 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 advantages of using a new kernel from a server such as <http://www.kernel.org>:

- Process some special requirements or handle conflicts between hardware and default kernel.
- Using many kernel options which are not supported by the default kernel.
- Optimize kernel.
- Running kernel version for developer.

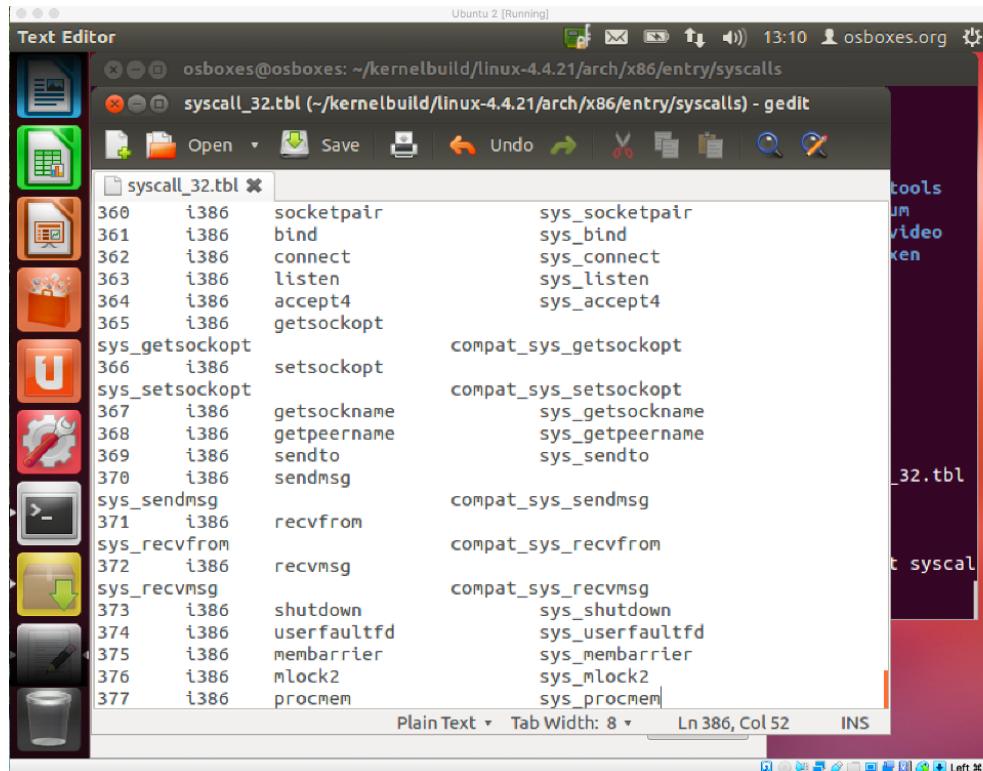
Yes, we can also compile the original kernel but we have to check to make sure the kernel version is compatible with our configuration and in case we have problems while working with the original kernel, system crash can happen. If we use the original kernel, make sure there is nothing wrong.

## 1. 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 .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](#) 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 Application Binary Interface ( ABI)
- [procmem]: this is the name of our 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` :

```

asmlinkage long sys_process_vm_writev(pid_t pid,
                                     const struct iovec __user *lvec,
                                     unsigned long liovcnt,
                                     const struct iovec __user *rvec,
                                     unsigned long riovlen,
                                     unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                        unsigned long id1, unsigned long id2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                           const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                             unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);
asmlinkage long sys_execveat(int dfd, const char __user *filename,
                            const char __user *const __user *argv,
                            const char __user *const __user *envp, int flags);
asmlinkage long sys_membarrier(int cmd, int flags);
asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);
asmlinkage long sys_procmem(int pid, struct proc_segs * info);
| #endif

```

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:

	x32		
520	x32	execve	stub_x32_execve
521	x32	ptrace	compat_sys_ptrace
522	x32	rt_sigtPending	compat_sys_rt_sigtPending
523	x32	rt_sigtimedwait	compat_sys_rt_sigtimedwait
524	x32	rt_sigsqueueinfo	compat_sys_rt_sigsqueueinfo
526	x32	timer_create	compat_sys_timer_create
525	x32	sigaltstack	compat_sys_sigaltstack
526	x32	mq_notify	compat_sys_mq_notify
527	x32	kecex_load	compat_sys_kecex_load
528	x32	waitid	compat_sys_waitid
529	x32	set_robust_list	compat_sys_set_robust_list
530	x32	get_robust_list	compat_sys_get_robust_list
531	x32	vsplice	compat_sys_vsplice
532	x32	move_pages	compat_sys_move_pages
533	x32	preadv	compat_sys_preadv64
534	x32	pwritev	compat_sys_pwritev64
535	x32	rt_tgsigsqueueinfo	compat_sys_rt_tgsigsqueueinfo
536	x32	recvmsg	compat_sys_recvmsg
537	x32	sendmsg	compat_sys_sendmsg
538	x32	process_vm_readv	compat_sys_process_vm_readv
539	x32	process_vm_writev	compat_sys_process_vm_writev
540	x32	setssockopt	compat_sys_setssockopt
541	x32	getsockopt	compat_sys_getsockopt
542	x32	io_setup	compat_sys_io_setup
543	x32	io_submit	compat_sys_io_submit
544	x32	execveat	stub_x32_execveat
545	x32	procmem	sys_procmem
546	64		

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

**ANSWER :**

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

A system call return long variable. The first argument pid is the process ID which the user want to get the information .The second argument 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 arguments in registers in common but in stack of CPU.All systemcall which is defined by tag ‘asmlinkage’ tag find their argument in stack.

## 2. 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.

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

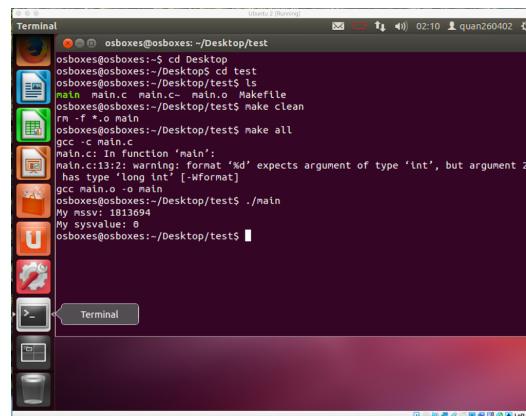
Inside of the code, we declare a buffer variable belong to `struct proc_segs`. which will contains the value of memory of process's information. And then we use `copy_to_user` to copy `buff` (in kernel mode) to `info` (in user mode).

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;
```

## 3. Compilation and Installation Process

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





**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. Making API for System Call

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

**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.

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
```

```
osboxes@osboxes:~/Desktop/Wrapper$ ls
main.c procmem.c procmem.h
osboxes@osboxes:~/Desktop/Wrapper$ sudo cp ~/Desktop/Wrapper/procmem.h /usr/include
osboxes@osboxes:~/Desktop/Wrapper$ gcc -shared -fpic procmem.c -o libprocmem.so
osboxes@osboxes:~/Desktop/Wrapper$ ls
libprocmem.so main.c procmem.c procmem.h
osboxes@osboxes:~/Desktop/Wrapper$ sudo cp ~/Desktop/Wrapper/libprocmem.so /usr/
```

Then copy `libprocmem.so` to `/usr/lib`.

**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 notice that permission is denied. Clearly, `sudo` mean “super user do”, we use “`sudo`” to work with administrative applications .



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

Compile with `-lprocmem` option

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

The screenshot shows a terminal window titled "Ubuntu 2 (Running)" with the command history and output:

```
i2c-smbus.h phy.h
osboxes@osboxes:~/kernelbuild/linux-4.4.21/include/linux$ gedit syscalls.h
osboxes@osboxes:~/kernelbuild/linux-4.4.21/include/linux$ cd Desktop
osboxes@osboxes:~/Desktop$ ls
test Wrapper
osboxes@osboxes:~/Desktop$ cd Wrapper
osboxes@osboxes:~/Desktop/Wrapper$ ls
a.out libprocmem.so main main.c main.c~ procmem.c procmem.h
osboxes@osboxes:~/Desktop/Wrapper$ ./a.out
PID: 2423
Code segment: 400000-4009fc
Data segment: 600e18-601030
Heap segment: 65e000-65e000
Start stack: 7fffa477d1f0
M2
[1]+ Stopped                  ./a.out
osboxes@osboxes:~/Desktop/Wrapper$ ./main
PID: 2425
MSSV: 1813694
Code segment: 400000-4009cc
Data segment: 600e18-601030
Heap segment: 111f000-111f000
Start stack: 7ffd76feb850
osboxes@osboxes:~/Desktop/Wrapper$ }
```

- Thank you for reading -