

**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**

*Ho Chi Minh, 5/2020*



# Contents

<b>1. Compiling the new kernel</b>	<b>3</b>
<b>2. System call</b>	
a. Adding new system call	5
b. System call Implementation	9
c. Complitation and Installation process	10
<b>3. Testing</b>	<b>10</b>
<b>4. Validation</b>	<b>11</b>
<b>5. Wapper</b>	<b>13</b>
<b>6. Documentary</b>	<b>14</b>



## 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 directory and download linux-4.4.21.tar.xz into it and decompression:

```
$ mkdir ~/kernelbuild                                //Create directory  
$ cd ~/kernelbuild                                  //Go inside directory  
$ wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.21.tar.xz //Download 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 advantages of using a new kernel from 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
```

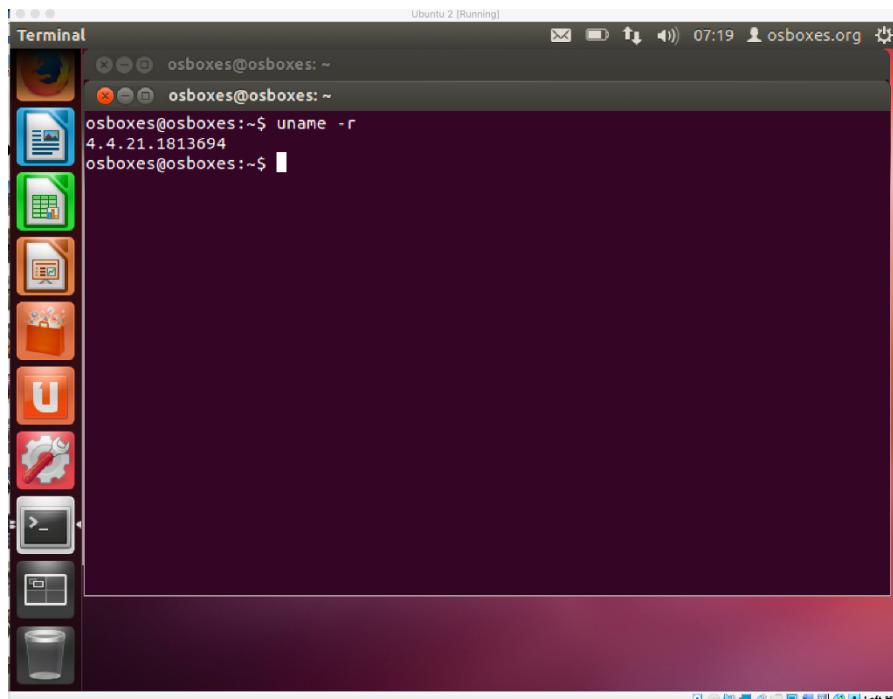
Then install the kernel

```
$ 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 proc_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 :

Number	CPU Type	System Call Name	Kernel Function Name
360	i386	socketpair	sys_socketpair
361	i386	bind	sys_bind
362	i386	connect	sys_connect
363	i386	listen	sys_listen
364	i386	accept4	sys_accept4
365	i386	getsockopt	compat_sys_getsockopt
		sys_getsockopt	
366	i386	setsockopt	compat_sys_setsockopt
		sys_setsockopt	
367	i386	getsockname	sys_getsockname
368	i386	getpeername	sys_getpeername
369	i386	sendto	sys_sendto
370	i386	sendmsg	compat_sys_sendmsg
		sys_sendmsg	
371	i386	recvfrom	compat_sys_recvfrom
		sys_recvfrom	
372	i386	recvmsg	compat_sys_recvmsg
		sys_recvmsg	
373	i386	shutdown	sys_shutdown
374	i386	userfaultfd	sys_userfaultfd
375	i386	membarrier	sys_membarrier
376	i386	mlock2	sys_mlock2
377	i386	procmem	sys_procmem

**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 do the same thing with syscall\_64.tbl :

```
Ubuntu 2 [Running]
You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture the keyboard every time the VM window is activated and make it unavailable to other applications
File Open Save Undo Redo Cut Copy Paste Find Replace Insert Plain Text Tab Width: 8 Ln 375, Col 11 INS Left 36
syscall_64.tbl ×
520 x32 execve stub_x32_execve
521 x32 ptrace compat_sys_ptrace
522 x32 rt_sigpending compat_sys_rt_sigpending
523 x32 rt_sigtimedwait compat_sys_rt_sigtimedwait
524 x32 rt_tgsigqueueinfo compat_sys_rt_tgsigqueueinfo
525 x32 sigaltstack compat_sys_sigaltstack
526 x32 timer_create compat_sys_timer_create
527 x32 mq_notify compat_sys_mq_notify
528 x32 kexec_load compat_sys_kexec_load
529 x32 waitid compat_sys_waitid
530 x32 set_robust_list compat_sys_set_robust_list
531 x32 get_robust_list compat_sys_get_robust_list
532 x32 vmsplice compat_sys_vmsplice
533 x32 move_pages compat_sys_move_pages
534 x32 preadv compat_sys_preadv64
535 x32 pwritev compat_sys_pwritev64
536 x32 rt_tgsigqueueinfo compat_sys_rt_tgsigqueueinfo
537 x32 recvmsg compat_sys_recvmsg
538 x32 sendmsg compat_sys_sendmsg
539 x32 process_vm_readv compat_sys_process_vm_readv
540 x32 process_vm_writev compat_sys_process_vm_writev
541 x32 setsockopt compat_sys_setsockopt
542 x32 getsockopt compat_sys_getsockopt
543 x32 io_setup compat_sys_io_setup
544 x32 io_submit compat_sys_io_submit
545 x32 execveat stub_x32_execveat
546 64 procmem sys_procmem
```

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:

```
Ubuntu 2 [Running]
You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture the keyboard every time the VM window is activated and make it unavailable to other applications.
File Open Save Undo Redo Cut Copy Paste Find Replace
syscalls.h
asmlinkage long sys_process_vm_writev(pid_t pid,
                                      const struct iovec __user *lvec,
                                      unsigned long liovcnt,
                                      const struct iovec __user *rvec,
                                      unsigned long riovcnt,
                                      unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                        unsigned long idx1, unsigned long idx2);
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
C/C++/ObjC Header Tab Width: 8 Ln 894, Col 1 INS Left
```

**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;
#ifndef __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	Arguments
copy_to_user — Copy a block of data into user space.	<i>to</i> Destination address, in user space.
Synopsis	<i>from</i> Source address, in kernel space.
<code>unsigned long copy_to_user (void __user * to,                            const void * from,                            unsigned long n);</code>	<i>n</i> Number of bytes to copy.

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

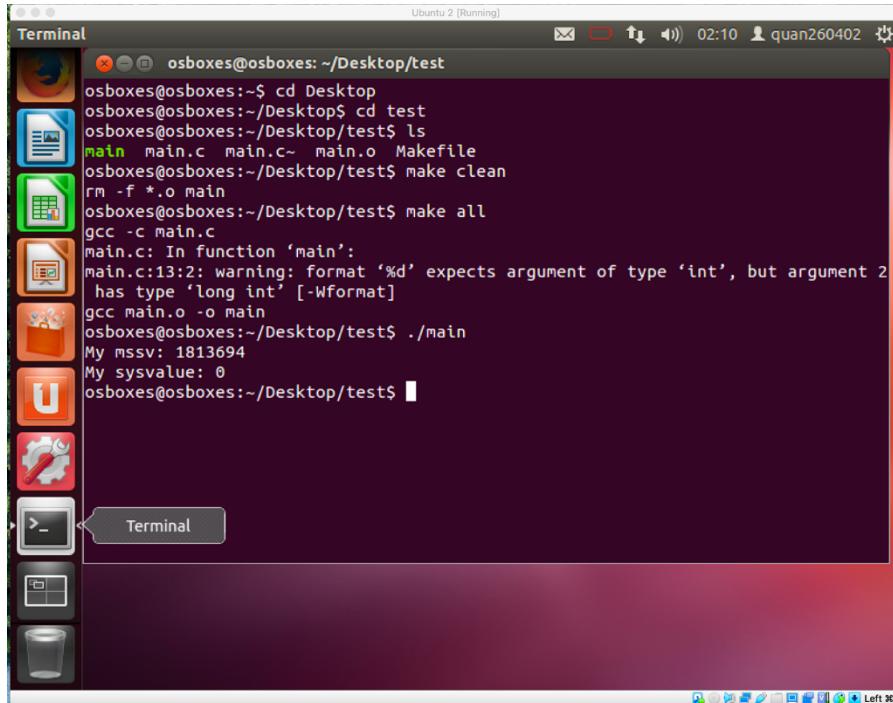
```
int flag = copy_to_user(info,&buff,sizeof(buff));  
if(flag != 0) return -1;  
else return 0;
```

## Compiling and Installation.

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

### 3. Testing

```
#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 proc_segs * info);
#endif // _PROC_MEM_H_
```

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



**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 notice 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
```

```
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.

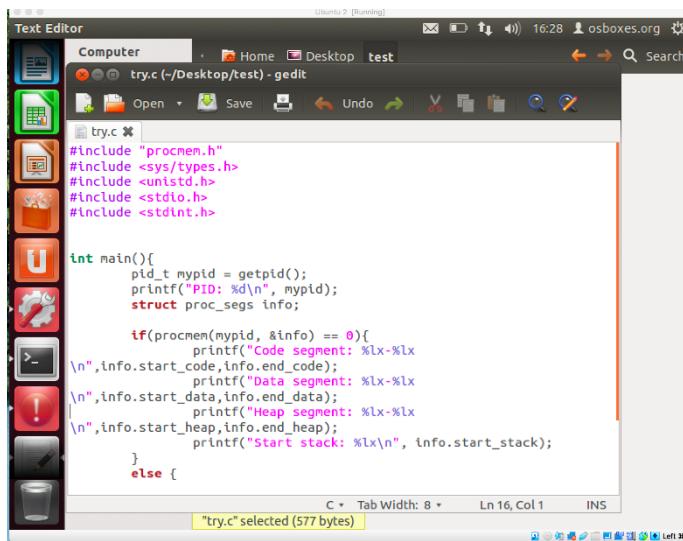
**QUESTION:** Why we must put -share and -fpic option into gcc command?

**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`:



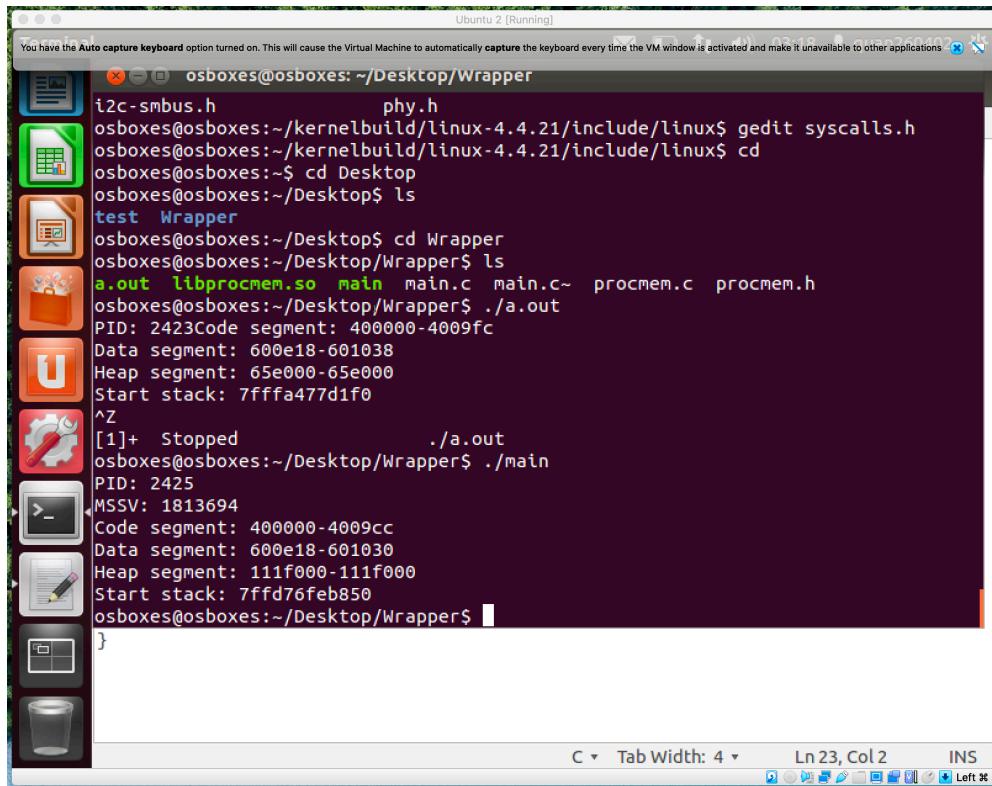
```
#include "procmem.h"
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdint.h>

int main(){
    pid_t mypid = getpid();
    printf("PID: %d\n", mypid);
    struct proc_segs info;

    if(procmem(mypid, &info) == 0){
        printf("Code segment: %lx-%lx
\n",info.start_code,info.end_code);
        printf("Data segment: %lx-%lx
\n",info.start_data,info.end_data);
        printf("Heap segment: %lx-%lx
\n",info.start_heap,info.end_heap);
        printf("Start stack: %lx\n", info.start_stack);
    }
    else {
}
```

Compile with `-lprocmem` option

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



```
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
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: 4000000-4009fc
Data segment: 600e18-601038
Heap segment: 65e000-65e000
Start stack: 7ffffa477d1f0
^Z
[1]+  Stopped                  ./a.out
osboxes@osboxes:~/Desktop/Wrapper$ ./main
PID: 2425
MSSV: 1813694
Code segment: 4000000-4009cc
Data segment: 600e18-601030
Heap segment: 111f000-111f000
Start stack: 7ffd76feb850
osboxes@osboxes:~/Desktop/Wrapper$ ]
```



## 6. Documentary

[https://www.kernel.org/doc/html/latest/vm/active\\_mm.html](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 -