ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO BÀI TẬP LỚN HỆ ĐIỀU HÀNH

Lớp L06

Bài tập lớn 1

Giảng viên: **Trần Việt Toản**

Sinh viên thực hiện: Đặng Quốc Thắng 1912084

Võ Đình Thanh 1912041

TP. Hồ Chí Minh, tháng 5 năm 2021

Ở bài tập lớn này, chúng ta sẽ hiện thực một lệnh hệ thống (system call) mới vào nhân hệ điều hành.

I – Biên dịch nhân linux

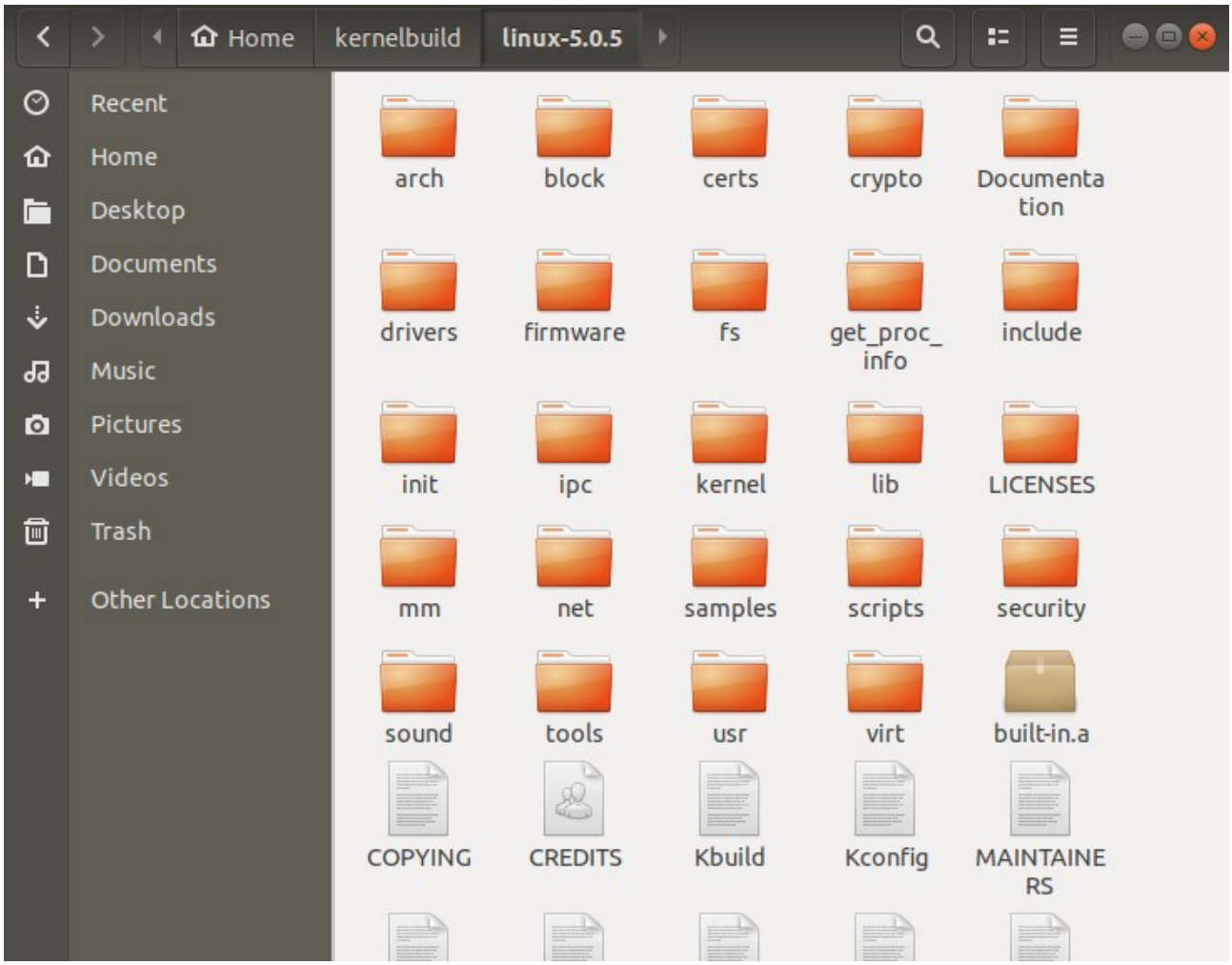
Để có thể biên dịch nhân linux, trước tiên chúng ta cần cài đặt kernel-package và các package cần thiết khác.

Kernel-package là một gói được phát triển nhằm mục đích tự động hóa các bước cần thiết để biên dịch và cài đặt một kernel tùy chỉnh. Nó tự động, tiện lợi, cho phép giữ nhiều phiên bản kernel và nhiều tiện ích khác.

```
mayao@mayao-VirtualBox:~$ sudo apt-get install kenel-package
```

Vì quá trình biên dịch và cài đặt kernel là công việc rủi ro và có thể khiến cho kernel của máy bị hỏng nên ta sẽ sử dụng một kernel khác từ server để thực hiện bài tập lớn này. Ở đây chúng ta sử dụng phiên bản 5.0.5 từ http://www.kernel.org .

Tiến hành download và giải nén:



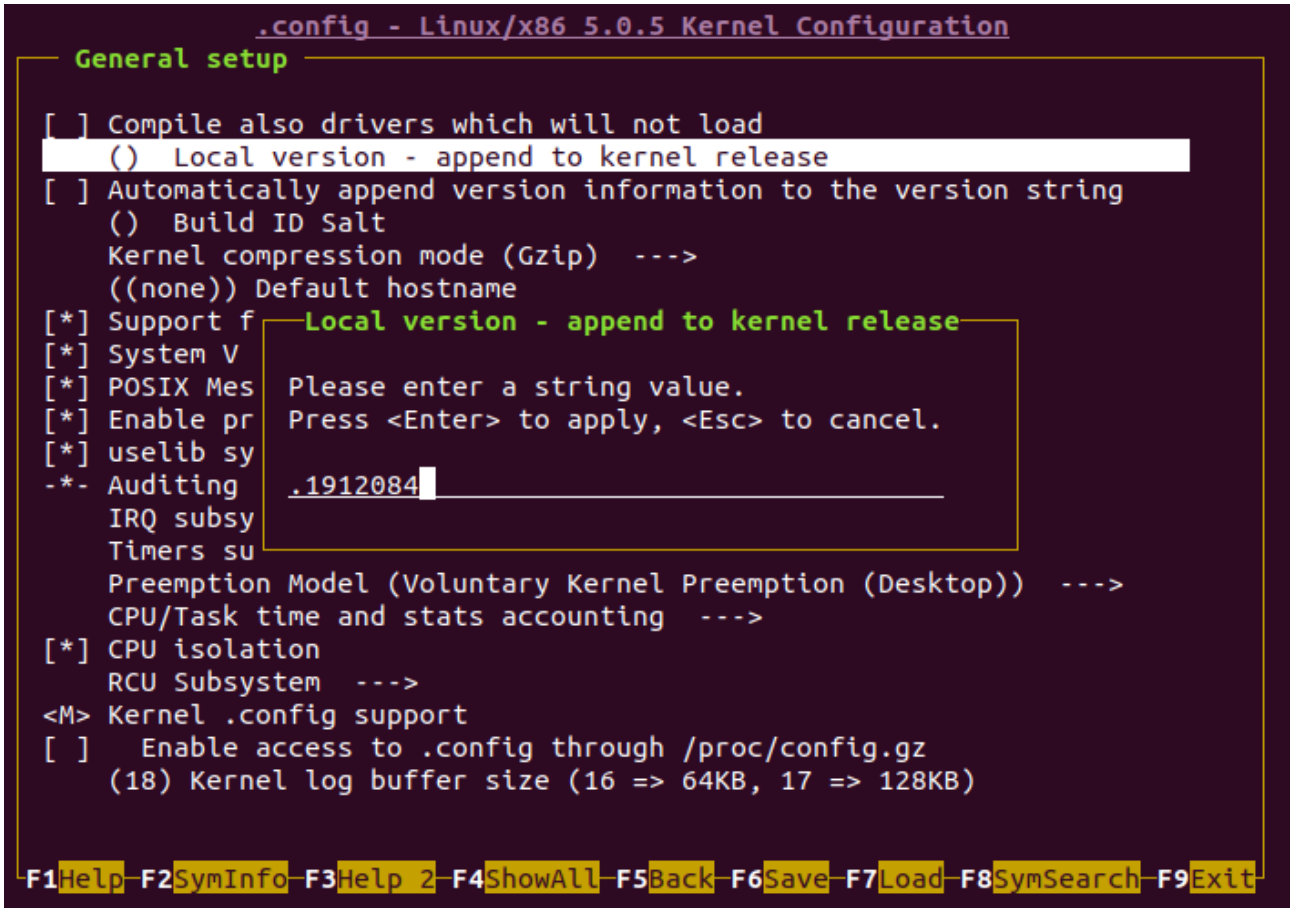
Để tùy chỉnh hạt nhân, chúng ta có thể mượn tệp cấu hình của một hạt nhân đang có trong máy, sao chép nó sang thư mục nguồn:

```
mayao@mayao-VirtualBox:~$ cp /boot/config-$(uname -r) ~/kernelbuild/linux-5.0.5
/.config
```

Đi tới thư mục nguồn, chạy lệnh:

```
mayao@mayao-VirtualBox:~/kernelbuild/linux-5.0.5$ make nconfig
```

Đi tới mục *General Setup*, Thêm MSSV vào đuôi phiên bản kernel:



Tiến hành biên dịch kernel và tạo vmlinuz. vmlinuz là kernel image, không bị nén và được tải vào bộ nhớ GRUB hoặc bất kì bộ tải khởi động nào khác đang sử dụng.

```
mayao@mayao-VirtualBox:~/kernelbuild/linux-5.0.5$ make
```

Xây dựng các loadable kernel module, là các tệp đối tượng chứa mã để mở rộng kernel.

```
mayao@mayao-VirtualBox:~/kernelbuild/linux-5.0.5$ make modules
```

Tiến hành cài đặt kernel. Trước tiên ta cài đặt các modules. Vì cài đặt cần quyền hạn của super user nên ta thêm sudo phía trước:

```
mayao@mayao-VirtualBox:~/kernelbuild/linux-5.0.5$ sudo make modules_install
```

Tiếp theo cài đặt kernel:

```
mayao@mayao-VirtualBox:~/kernelbuild/linux-5.0.5$ sudo make install
```

Sau khi quá trình cài đặt hoàn tất, ta khởi động lại máy. Để kiểm tra, ta sử dụng lệnh “uname -r”:

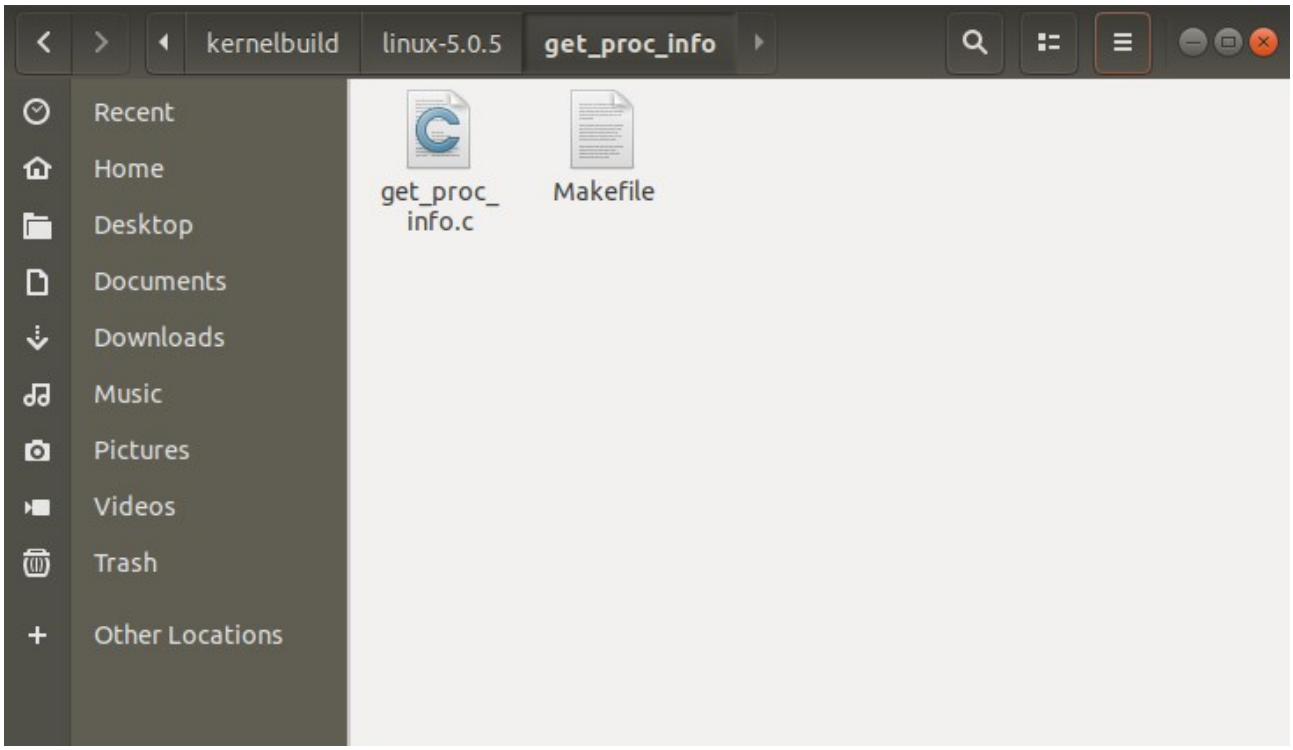
```
mayao@mayao-VirtualBox:~$ uname -r
5.0.5.1912084
mayao@mayao-VirtualBox:~$
```

Vậy là chúng ta đã biên dịch và cài đặt nhân tùy chỉnh của chúng ta thành công.

II - System Call

Ở phần này chúng ta sẽ hiện thực một lệnh hệ thống mới cho phép người dùng lấy thông tin của một tiến trình và thông tin của tiến trình cha và một tiến trình con của nó.

Trong thư mục nguồn, tạo một thư mục mới chứa file .c và Makefile của lệnh hệ thống mới:



Hiện thực system call:

Open ▾ get_proc_info.c ~/kernelbuild/linux-5.0.5/get_proc_info Save

```
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/syscalls.h>

struct proc_info {  //info about a single process
        int pid;  //pid of the process
        char name[16];  //file name of the program executed
};

struct procinfos {  //info about processes we need
        long studentID;  //for the assignment testing
        struct proc_info proc;  //process with pid or current process
        struct proc_info parent_proc;  //parent process
        struct proc_info oldest_child_proc;  //oldest child process
};

SYSCALL_DEFINE2(get_proc_info, int, pid, struct procinfos*, info) {
  // TODO: implement the system call
        info->studentID = 1912084;

        int i;

        struct task_struct* p;
        if (pid == -1) p = current;
        else {
                if (find_get_pid(pid) == NULL) return EINVAL;
                p = get_pid_task(find_get_pid(pid),PIDTYPE_PID);
        }
```

C ▾ Tab Width: 8 ▾ Ln 5, Col 1 ▾ INS

Makefile:

Open ▾ Makefile ~/kernelbuild/linux-5.0.5/get_proc_info Save

```
obj-y := get_proc_info.o
```

Makefile ▾ Tab Width: 8 ▾ Ln 1, Col 25 ▾ INS

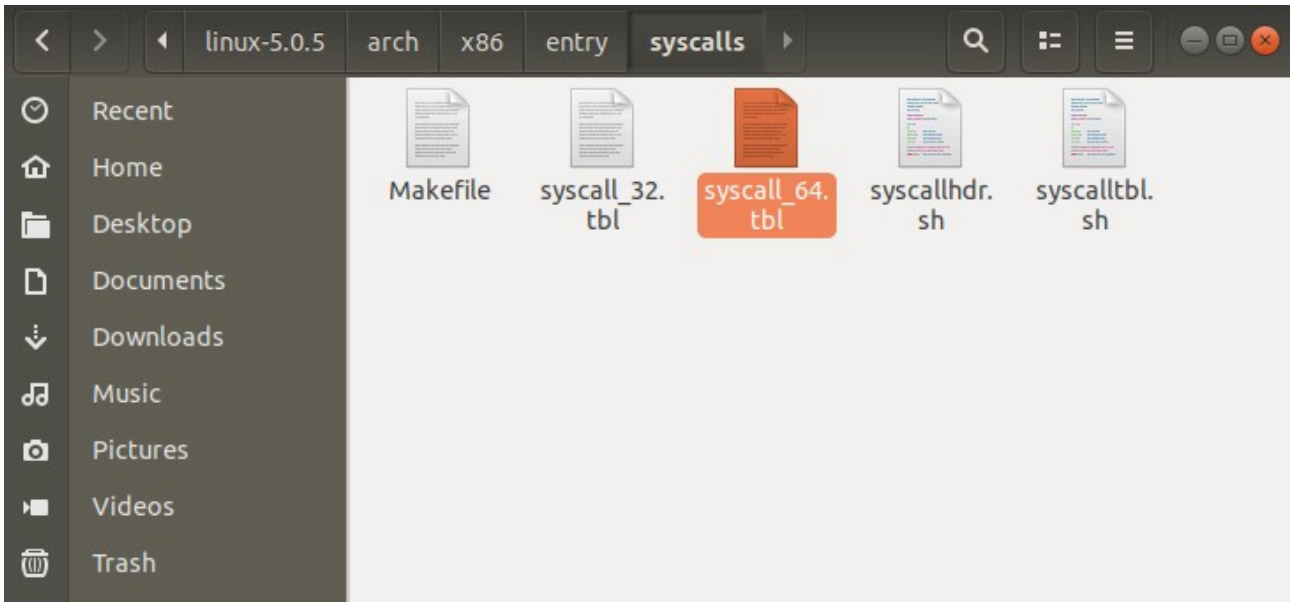
Thêm thư mục vừa tạo vào Makefile của kernel:

Open ▾ Makefile ~/kernelbuild/linux-5.0.5 Save

```
ifdef CONFIG_STACK_VALIDATION
  has_libelf := $(call try-run,\
                echo "int main() {}" | $(HOSTCC) -xc -o /dev/null -lelf -,1,0)
  ifeq ($(has_libelf),1)
    objtool_target := tools/objtool FORCE
  else
    SKIP_STACK_VALIDATION := 1
    export SKIP_STACK_VALIDATION
  endif
endif

PHONY += prepare0

ifeq ($(KBUILD_EXTMOD),)
core-y          += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/
get_proc_info/

vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                     $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
                     $(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, \
                     $(init-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))

init-y          := $(patsubst %/, %/built-in.a, $(init-y))
core-y          := $(patsubst %/, %/built-in.a, $(core-y))
drivers-y       := $(patsubst %/, %/built-in.a, $(drivers-y))
net-y           := $(patsubst %/, %/built-in.a, $(net-y))
```

Makefile ▾ Tab Width: 8 ▾ Ln 961, Col 87 ▾ INS

Để thêm lệnh hệ thống mới vào bảng lệnh hệ thống, ta đi tới arch/x86/entry/syscalls/ :

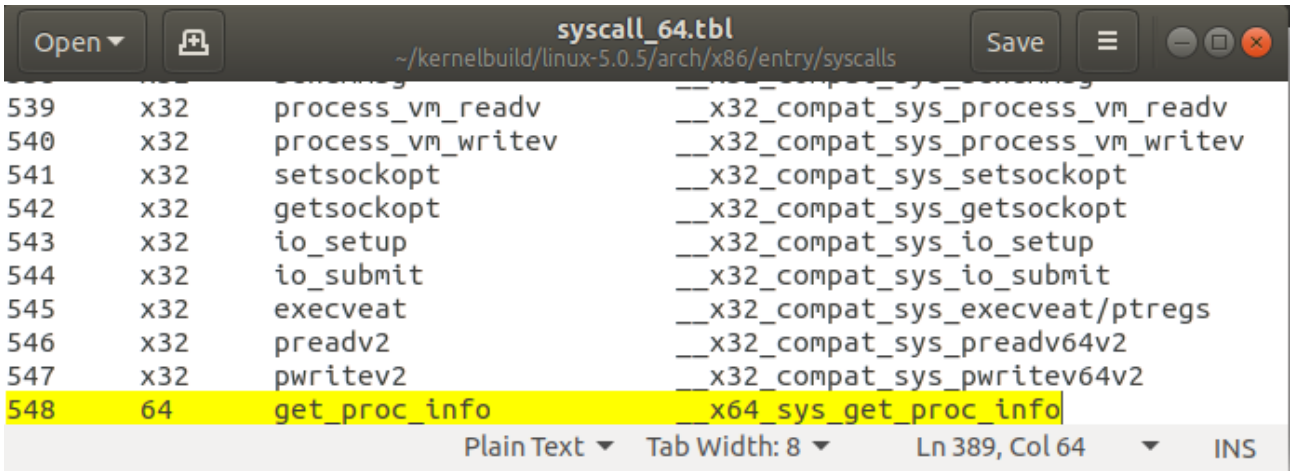


Dữ liệu được lưu trong file syscall_64.tbl với định dạng:

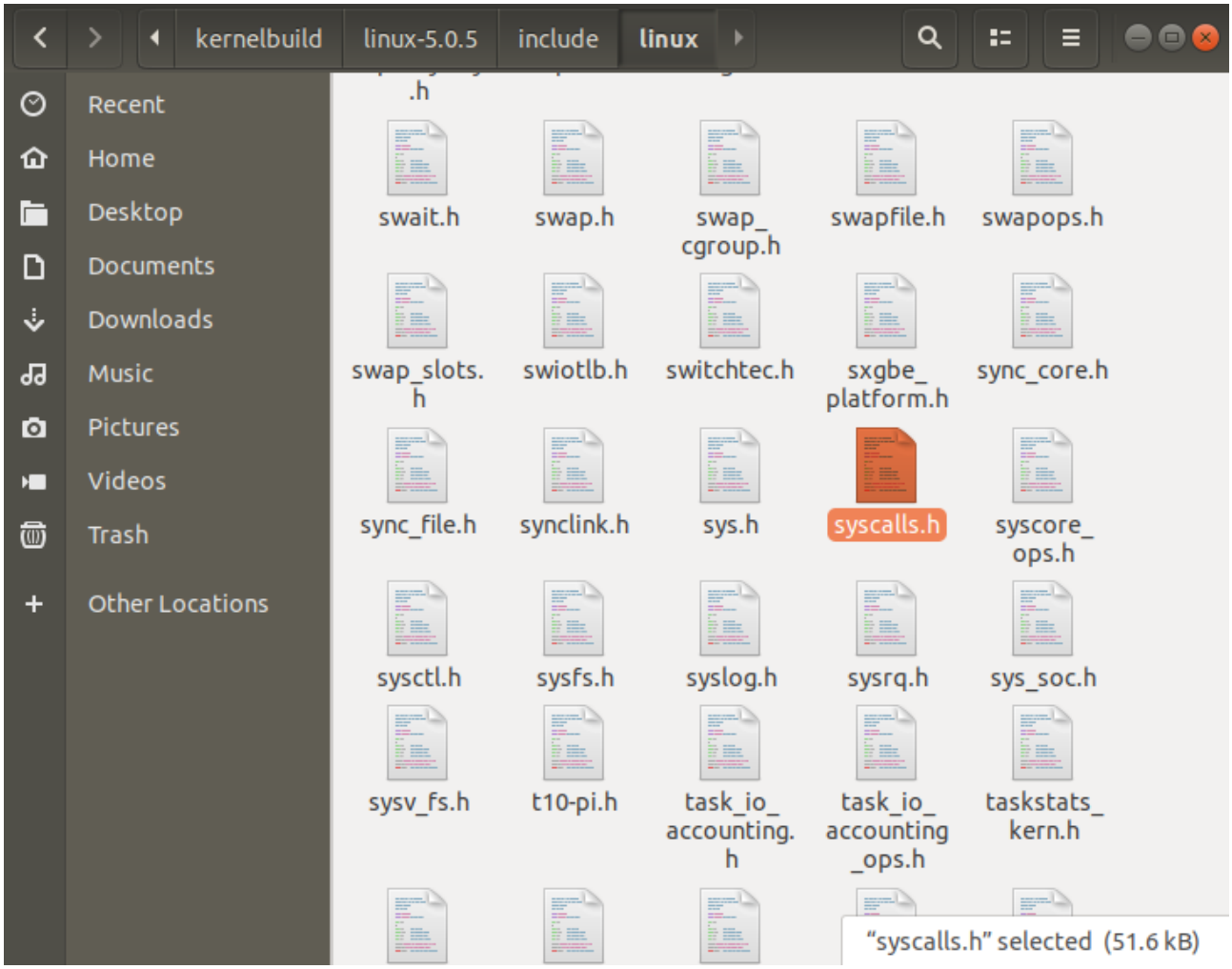
<number> <abi> <name> <entry point>

- <number> là số thứ tự của lệnh hệ thống
- <abi> có thể là *common*, *64*, hoặc *x32*
- <name> và <entry point> xác định tệp và lệnh hệ thống

Thêm lệnh mới vào cuối tệp:



Đi tới tệp tiêu đề ở include/linux/ :



Thêm hai struct và lệnh hệ thống mới vào gần cuối tệp:

Open | syscalls.h ~/kernelbuild/linux-5.0.5/include/linux | Save

```
        if (force_o_largefile())
                flags |= O_LARGEFILE;
        return do_sys_open(AT_FDCWD, filename, flags, mode);
}

extern long do_sys_truncate(const char __user *pathname, loff_t length);

static inline long ksys_truncate(const char __user *pathname, loff_t length)
{
        return do_sys_truncate(pathname, length);
}

static inline unsigned int ksys_personality(unsigned int personality)
{
        unsigned int old = current->personality;

        if (personality != 0xffffffff)
                set_personality(personality);

        return old;
}

struct proc_info;
struct procinfos;

asmlinkage long sys_get_proc_info(int, struct procinfos*);

#endif
```

C/ObjC Header | Tab Width: 8 | Ln 1323, Col 7 | INS

Cuối cùng, biên dịch lại nhân và khởi động lại hệ thống để áp dụng nhân mới. Để kiểm tra lệnh hệ thống đã được tích hợp vào nhân chưa, ta tạo chương trình C nhỏ sau:

Open | a.c ~/ | Save

```
#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;
}
```

C | Tab Width: 8 | Ln 13, Col 2 | INS

Biên dịch và chạy chương trình trên. Vì tham số thứ 2 truyền vào lệnh hệ thống là một con trỏ tới struct có biến đầu tiên kiểu long chứa MSSV nên info[0] sẽ chứa MSSV:

```
mayao@mayao-VirtualBox:~$ gcc a.c
mayao@mayao-VirtualBox:~$ ./a.out
My student ID: 1912084
mayao@mayao-VirtualBox:~$
```

Lệnh hệ thống mới đã hoạt động bình thường, tuy nhiên việc gọi thông qua số của nó khá bất tiện, vì vậy ta cần tạo trình bao bọc cho nó. Việc này có thể thực hiện bên ngoài nhân, vì vậy để tránh biên dịch lại nhân, ta rời khỏi thư mục nguồn và tạo thư mục khác để lưu trữ mã nguồn cho trình bao bọc. Trong thư mục vừa tạo, tạo tệp tiêu đề:

Open | get_proc_info.h ~/Wrapper | Save

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

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

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

long get_proc_info(pid_t pid, struct procinfos* info);
#endif
```

C/ObjC Header | Tab Width: 8 | Ln 11, Col 26 | INS

Trước đó ta đã định nghĩa hai struct ở trong nhân. Còn hiện tại ta đang thực hiện trình bao bọc ở ngoài nhân nên ta cần định nghĩa lại hai struct này.

Tạo tệp mã nguồn cho trình bao bọc:

Open | get_proc_info.c ~/Wrapper | Save

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

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

C | Tab Width: 8 | Ln 7, Col 42 | INS

Tiếp theo, chúng ta cần hiển thị tệp tiêu đề với GCC bằng cách sao chép tệp tiêu đề vào thư mục tiêu đề của hệ thống. Vì sửa đổi thư mục hệ thống cần quyền hạn super user nên ta cần thêm sudo phía trước lệnh:

```
mayao@mayao-VirtualBox:~$ sudo cp ~/Wrapper/get_proc_info.h /usr/include
```

Biên dịch mã nguồn như một đối tượng chia sẻ với hai tùy chọn sau:

-shared: dùng để chỉ thị thư viện chia sẻ

-fpic: PIC viết tắt của position-independent code, giúp mã có thể chia sẻ qua nhiều tiến trình

```
mayao@mayao-VirtualBox:~/Wrapper$ gcc -shared -fpic get_proc_info.c -o libget_p
roc_info.so
```

Cuối cùng, chúng ta kiểm tra lại toàn bộ công việc. Tạo chương trình sau:

Mở | a.c ~/Bản tải về | Lưu

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

int main(int argc, char* argv[]) {
        if (argc != 2) exit(-1);
        pid_t mypid = atoi(argv[1]);
        printf("PID: %d\n", mypid);
        struct procinfos info;

        if (get_proc_info(mypid, &info) == 0) {
                printf("Student ID: %ld\n", info.studentID);
                printf("Proc: %s (%i)\n", info.proc.name, info.proc.pid);
                printf("Parent: %s (%i)\n",info.parent_proc.name,
info.parent_proc.pid);
                if (info.oldest_child_proc.pid == -1) printf("Process has no
child\n");
                else printf("Oldest Child: %s (%i)\n",
info.oldest_child_proc.name,  info.oldest_child_proc.pid);
        }
        else {
                printf("Cannot get information from the process %d\n", mypid);
        }
}
```

C | Bề rộng Tab: 8 | Dòng 11, Cột 36 | CHÈN

Biên dịch chương trình với tùy chọn -lget_proc_info:

```
mayao@mayao-VirtualBox:~$ gcc a.c -lget_proc_info
```

Chạy thử chương trình trên:

```
mayao@mayao-VirtualBox:~$ ./a.out -1
PID: -1
Student ID: 1912084
Proc: a.out (1966)
Parent: bash (1566)
Process has no child
mayao@mayao-VirtualBox:~$
```

Vì chúng ta nhập vào tham số -1, chương trình hiện tại tức chương trình dùng để kiểm tra của chúng ta không có tiến trình con nên sẽ không có kết quả của Oldest Child.

Thử với một số tham số vào khác:

```
mayao@mayao-VirtualBox:~$ ./a.out 860
PID: 860
Student ID: 1912084
Proc: gdm-session-wor (860)
Parent: gdm3 (795)
Oldest Child: gdm-x-session (949)
mayao@mayao-VirtualBox:~$
```

```
mayao@mayao-VirtualBox:~$ ./a.out 923
PID: 923
Student ID: 1912084
Proc: systemd (923)
Parent: systemd (1)
Oldest Child: (sd-pam) (928)
mayao@mayao-VirtualBox:~$
```

```
mayao@mayao-VirtualBox:~$ ./a.out 1110
PID: 1110
Student ID: 1912084
Proc: gdbus (1110)
Parent: systemd (923)
Process has no child
mayao@mayao-VirtualBox:~$
```

```
mayao@mayao-VirtualBox:~$ ./a.out 1
PID: 1
Student ID: 1912084
Proc: systemd (1)
Parent: swapper/0 (0)
Oldest Child: systemd-journal (288)
mayao@mayao-VirtualBox:~$
```

```
mayao@mayao-VirtualBox:~$ ./a.out 500
PID: 500
Cannot get information from the process 500
mayao@mayao-VirtualBox:~$
```

Đến đây, chúng ta đã hoàn thành thêm một system call mới vào kernel!