BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

Course: Operating Systems

Assignment #1 - System Call

Lớp: L06 GVHD: Phạm Trung Kiên

Sinh viên: Nguyễn Linh Đăng Minh MSSV:1712177

I) THÔNG TIN SINH VIÊN

Lớp: L06 – Hệ điều hành (TN) – Tiết 9-10 – phòng H6-702

Tên: Nguyễn Linh Đăng Minh

MSSV: 1712177

KHOA: Khoa học và kĩ thuật máy tính

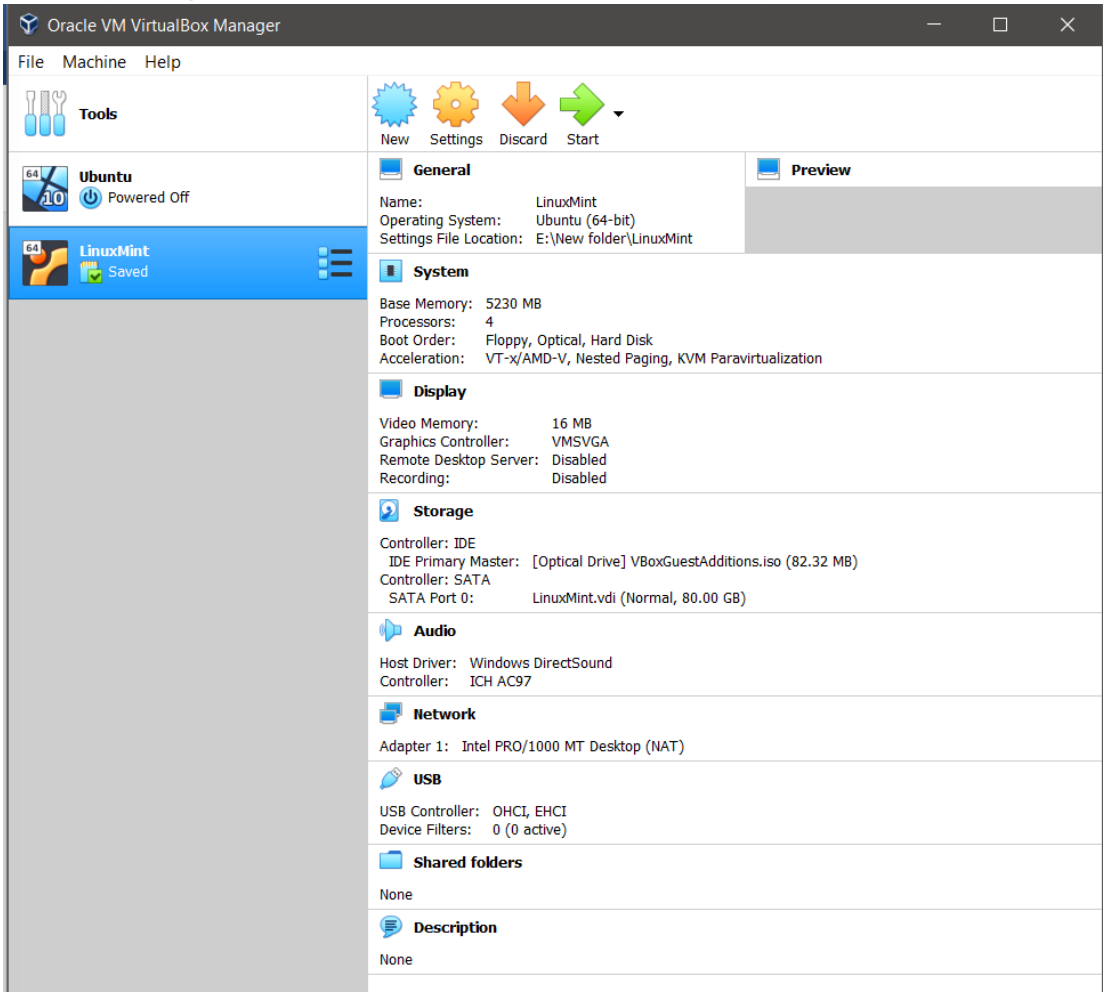
PHÂN NGÀNH: Khoa học máy tính (MT17KH2)

II) MỤC LỤC

III) Updated OS-Ubuntu Server-New Kernel 5.0.5-StudentID

1) Preparation

-Cài đặt máy ảo: Dùng Virtual Box 6.0 và cài đặt LinuxMint



- ***Install the core packages:*** *Get Ubuntu's toolchain (gcc, make, and so forth) by installing the build-essential metapackage:*

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

- ***Install kernel-package:***

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

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

ANSWER: *Kernel-package cung cấp những công cụ giúp và hỗ trợ cho việc compile, install và configured kernel một cách tự động và dễ dàng hơn so với compile và install kernel bằng cách truyền thống*

- ***Download the kernel source:***

```
$ cd ~/kernelbuild
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.0.5.tar.xz
```

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: *Có thể compile original kernel (bản kernel trong OS đang dùng) một cách trực tiếp, nhưng sẽ gây ra những rủi ro và rất khó để để sửa lỗi cho hệ thống. Do đó, chúng ta nên sử dụng kernel độc lập với kernel của hệ điều hành để tránh và giảm rủi ro, tăng độ an toàn, độc lập với kernel hệ điều hành (có thể back-up lại trong trường hợp lỗi), và dễ dàng sửa lỗi.*

- ***Unpack the kernel source:***

  Within the build directory, unpack the kernel tarball:

```
$ tar -xvJf linux-5.0.5.tar.xz
```

2) Configuration

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

Important:

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

Then, run $ make menuconfig or $ make nconfig to open Kernel Configuration.

```
$ make nconfig
```

To change kernel version, go to General setup option, Access to the line “(-ARCH) Local version – append to kernel release”. Then enter a dot “.” followed by your MSSV:

```
.1712177
```

- Note: During compiling, you can encounter the error caused by missing openssl packages. You need to install these packages by running the following command:

```
$ sudo apt-get install openssl libssl-dev
```

3) Trim the kernel

- Thực hiện Trim Kernel để cho việc compile kernel trở nên nhanh chóng và giảm dung lượng bộ nhớ.
- Cách thực hiện: Vào trong thư mục kernelbuild , Dùng lệnh

```
make localmodconfig
```

nhấn chọn N cho những option không cần thiết

- Lệnh make **localmodconfig**: có chức năng lấy thông tin những process của hệ điều hành đang thực thi hiện tại, rồi ghi vào file config trong thư mục kernelbuild nên giảm bớt những phần không cần thiết trong config và chỉ compile những phần nào cần thiết cho kernel.

4) Build the configured kernel

- First run “make” to compile the kernel and create vmlinuz. It takes a long time to “$ make”, we can run this stage in parallel by using tag “-j np”, where np is the number of processes you run this command.

```
$ make
or
$ make -j 4
```

- vmlinuz is “the kernel”. Specifically, it is the kernel image that will be uncompressed and loaded into memory by GRUB or whatever other boot loader you use. Then build the loadable kernel modules Similarly, you can run this command in parallel.

```
$ make modules
or
$ make -j 4 modules
```

QUESTION: What is the meaning of these two stages, namely “make” and “make modules”? What are created and what for?

ANSWER:

- *make : compiles ra 1 file tên là vmlinuz (binary file) vào thư mục kernel*
- *make modules: compiles từng file riêng khi đã lựa chọn* `M` *trong lúc config (ở bước 2)*
  (Đối với những phần chọn `Y` *trong config thì sẽ nằm trong vmlinuz, và những phần chon* `N` *thì sẽ được bỏ qua)*

5) Installing the new kernel

First install the modules:

```
$ sudo make modules_install
or
$ sudo make -j 4 modules_install
```

Then install the kernel itself:

```
$ sudo make install
or
$ sudo make -j 4 install
```

Check out your work: After installing the new kernel by steps described above. Reboot the virtual machine by typing

```
sudo reboot
```

After logging into the computer again, run the following command.

```
uname -r
```

IV) SYSTEM CALL

1) The role of system call

The main part of this assignment is to implement a new system call that lets the user determine the information about the parent and the oldest child process. The information about the process's information is represented through the following struct:

```
struct procinfos { //info about processes we need
long studentID;
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
};
```

Where the proc info is defined as follows:

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

2) Prototype

The prototype of our system call is described as below:

```
long get_proc_info(pid_t pid, struct proinfos* info);
```

3) Implementation

```
$ pwd
~/kernelbuild
$ mkdir get_proc_info
$ cd get_proc_info
$ touch sys_get_proc_info.c
```

Add the following lines to sys_get_proc_info.c

```
#include <linux/kernel.h>
#include <linux/sched.h>          // task_struct
#include <linux/errno.h>          // EINVAL
#include <linux/string.h>         // strcpy
#include <linux/syscalls.h>       // macro SYSCALL_DEFINEx
#include <linux/uaccess.h>        // copy_to_user
struct proc_info { //info about a single process
pid_t 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(sys_get_proc_info, pid_t, pid, struct procinfos*, info)
{
   struct task_struct  *proc = NULL;
   struct task_struct  *parent_proc = NULL;
   struct task_struct  *oldest_child_proc = NULL;
   void *isChild = NULL;

   /*      Allocating Memory in the Kernel      */
   struct procinfos *kinfo = (struct procinfos*) kmalloc(
      sizeof(struct procinfos), GFP_KERNEL
   );

   if(kinfo == NULL)      // Check error >-<
      return ENOMEM;      // cannot allocate memory
   /*-------------------------------------------*/

   /*  Find current process or process with pid  */
   if(pid == -1) proc = current;
   else proc = find_task_by_vpid(pid);

   if(proc == NULL)       // Check error >-<
      return ESRCH;       // No such process

   kinfo->proc.pid = proc->pid;
   strcpy(kinfo->proc.name, proc->comm);
   /*-------------------------------------------*/

   /*    Find parent process of first process    */
   parent_proc = proc->parent;

   if(parent_proc == NULL) // Check error >-<
      return ESRCH;       // No such process

   kinfo->parent_proc.pid = parent_proc->pid;
   strcpy(kinfo->parent_proc.name, parent_proc->comm);
```

```
    /*-------------------------------------------*/

   /* Find oldest child process of first process */
   isChild = list_first_entry_or_null (
      &proc->children, struct task_struct, sibling);

   if(isChild == NULL) {
      kinfo->oldest_child_proc.pid = (pid_t) -1;
      strcpy(kinfo->oldest_child_proc.name, "No Name");
   }
   else {
      oldest_child_proc  = list_last_entry(
         &proc->children, struct task_struct, sibling);

      if(oldest_child_proc == NULL) // Check error >-<
      return ESRCH;              // No such process

      kinfo->oldest_child_proc.pid = oldest_child_proc->pid;
      strcpy(kinfo->oldest_child_proc.name, oldest_child_proc->comm);
   }
   /*-------------------------------------------*/
   kinfo->studentID = 1712177; // My student ID

   /* Copy data from kernel space to user space  */
   copy_to_user(info, kinfo, sizeof(struct procinfos));
   kfree(kinfo); // Deallocate kernel memory
   /*-------------------------------------------*/

   return 0;
}
```

Create a Makefile for the source file:

```
$ pwd

~/kernelbuild/get_proc_info

$ touch Makefile

$ echo "obj-y := sys_get_proc_info.o"
```

Add get_proc_info/ to the kernel Makefile

```
$ pwd
~/kernelbuild/get_proc_info
$ cd ..
$ pwd
 ~/kernelbuild/
$ vim Makefile
```

Find the following line:

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

Add get_proc_info/ to the end of this line.

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

Add the new system call to the system call table:

```
$ pwd
~/kernelbuild/
$ cd arch/x86/entry/syscalls/
$ echo "548 64 get_proc_info sys_get_proc_info" >> syscall_64.tbl
```

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

ANSWER: Ý nghĩa của các field:

- 548: mã định danh của syscall mới trong danh sách các syscall của kernel.
- 64 : phiên bản 64-bit linux.
- get_proc_info: tên của system call mới.
- sys_get_proc_info: điểm nhập – tên gọi một hàm xử lý syscall, đặt theo cấu trúc sys_+*tên syscall*.

Add new system call to the system call header file:

```
$ ~/kernelbuild/include/linux/
```

Open syscalls.h and add the following line before #endif statement.

```
struct proc_info;

struct procinfos;

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

QUESTION: What is the meaning of each line above?

Answer: Khai báo các struct mới (*proc_info* và *procinfos*) được dùng trong code ở file hiện thực syscall mới get_proc_info.c và chữ ký cho hàm mới (*asmlinkage long sys_get_proc_info(pid_t pid, struct procinfos* info)*).

Finally recompile the kernel and restart the system to apply the new kernel:

```
$ make -j 8

$ make modules -j 8

$ sudo make modules_install

$ sudo make install

$ sudo reboot
```

4) Testing

```
#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?

ANSWER: Chương trình này đã gọi system call chúng ta mới thêm vào thông qua mã định danh của nó trong kernel và truyền địa chỉ biến info[] để giữ kết quả trả về, nếu system call chúng ta hoạt động đúng thì nó sẽ in ra màn hình mã số sinh viên (thành phần đầu tiên của struct procinfos) khi gọi info[0] (type casting trong C)

5) Wrapper

get proc info.h

```
#ifndef _GET_PROC_INFO_H_
#define _GET_PROC_INFO_H_
#include <unistd.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 get_proc_info(pid_t pid, struct procinfos* info);
#endif // _GET_PROC_INFO_H_
```

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

ANSWER: Vì kernel space và user space là riêng biệt, user space không thể truy cập vào kernel space để lấy thông tin các struct này nên phải khai báo lại.

get proc info.c

```
#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){
   long sys_return_value;
   sys_return_value = syscall(335, pid , info);
   return 0;
}
```

6) Validation

Run following command to copy our header file to header directory of our system:

```
$ sudo cp <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?

ANSWER: Vì thư mục /usr/include thuộc quyền sở hữu của root nên muốn copy header file vào thư mục này thì cần quyền root và phải nhập đúng mật khẩu để có được quyền.

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 -share -fpic get_proc_info.c -o libget_proc_info.so
```

If the compilation ends successfully, copy the output file to /usr/lib. (Remember to add sudo before cp command).

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

ANSWER: Cần phải thêm các option này vì -shared giúp chia sẻ file object được biên dịch ra để liên kết với các file object khác tạo nên file thực thi, còn -fpic dùng để sinh code cho tương thích với option -shared vì chúng cùng tập option.

We only have the last step: check all of your work. To do so, write following program, and compile it with -lget proc info option. The result should be consistent with the process information.

```
#include <get_proc_info.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 procinfos info;

    if (get_proc_info(mypid, &info) == 0) {
// TODO: print all information in struct procinfos info
    printf("Student ID: %lu \n",info.studentID );
    printf("My Process info : \n");
   printf("PID: %d \n",info.proc.pid);
    printf("Name: %s \n",info.proc.name);
   printf("My Parent Process info : \n");
   printf("PID: %d \n",info.parent_proc.pid);
    printf("Name: %s \n",info.parent_proc.name);
   printf("My Oldest child Process info : \n");
   printf("PID: %d \n",info.oldest_child_proc.pid);
    printf("Name: %s \n",info.oldest_child_proc.name);
}else{
printf("Cannot get information from the process %d\n", mypid);
}
// If necessary, uncomment the following line to make this program run
// long enough so that we could check out its dependence
// sleep(100);
}
```