

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND
ENGINEERING



OPERATING SYSTEM

ASSIGNMENT1

SYSCALL

Giảng viên hướng dẫn: Hoàng Lê Hải Thanh

Lớp: L04

Nhóm 7: Trần Minh Trí - 1910637
Trần Quốc Vinh - 1915953
Phạm Duy Quang - 2011899
Võ Văn Hiền - 2020023

Mục lục

BẢNG PHÂN CHIA CÔNG VIỆC.....	2
1. BIÊN DỊCH KERNEL.....	2
1.1. Chuẩn bị.....	2
<i>Cài đặt kernel-package:</i>	2
<i>Trả lời:</i>	2
<i>Trả lời:</i>	3
1.2. Cấu hình.....	3
1.3. Xây dựng kernel đã cấu hình.....	3
1.4. Cài đặt kernel mới.....	4
2. LÀM GỌN KERNEL.....	5
3. SYSTEM CALL.....	6
3.1. Hiện thực system call.....	6
3.2. Testing.....	9
3.3. Wrapper.....	9
3.4. Validation.....	10
Ghi chú và tài liệu tham khảo:.....	12
<i>Tại sao lại SYSCALL_DEFINE2(...)</i>	12
<i>task_struct?</i>	12
<i>comm?</i>	12
<i>real_parent?</i>	12
<i>copy_to_user?</i>	12
<i>list_first_entry_or_null</i>	12

BẢNG PHÂN CHIA CÔNG VIỆC

STT	Tên	MSSV	Công việc
1	Trần Quốc Vinh	1915953	Viết report, trả lời câu hỏi
2	Trần Minh Trí	1910637	Hiện thực system call
3	Phạm Duy Quang	2011899	Viết report, trả lời câu hỏi
4	Võ Văn Hiền	2020023	Biên dịch kernel

1. BIÊN DỊCH KERNEL

1.1. Chuẩn bị

Thiết lập máy ảo: máy ảo được cài đặt **Ubuntu 18.04**, 4GB RAM và tối thiểu 40GB dung lượng ổ cứng.

Cài đặt core packages: sau khi cài đặt package, ta cài Ubuntu's toolchain (gcc, make).

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

Cài đặt kernel-package:

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

Hỏi: Tại sao chúng ta cần cài đặt kernel package?

Trả lời:

Sự tiện lợi: Kernel-package đã được viết để thực hiện tất cả các bước cần thiết. Điều này giúp giảm thời gian build kernel hơn thay vì phải thực hiện mọi việc.

Hỗ trợ nhiều image: Cho phép lưu trữ nhiều image của kernel mà không gặp phải phiền phức nào.

Nhiều tác vụ của cùng một phiên bản kernel: Cho phép giữ nhiều tác vụ của cùng một kernel version (ví dụ: ta có thể sử dụng phiên bản 2.0.36 hoặc 2.0.36 bản vá với các quy trình điều khiển mới nhất) mà không phải lo lắng việc làm ảnh hưởng tới các modules trong /lib/modules.

Được xây dựng một cách mặc định: Đảm nhận việc di chuyển đúng tệp vào đúng vị trí, truy xuất đúng mục tiêu thích hợp.

Sự kết nối đến các modules: Một số kernel modules được kết nối với kernel-package, vì vậy ta có thể biên dịch các modules cùng một lúc trong quá trình biên dịch hạt nhân, và các modules được biên dịch sẽ là tương thích.

Giúp theo dõi cấu hình: được lưu trữ trong tập tin /boot.

Nhiều tệp cấu hình: Cho phép chỉ định một thư mục với các tệp cấu hình khác nhau.

Cho phép bảo trì nhanh chóng: Đi kèm với các tập lệnh bảo trì cho phép người dùng thêm các lệnh khi trạng thái của kernel-package thay đổi.

Có thể thêm mô tả cho phiên bản kernel được cài đặt: Ví dụ: Ta có thể đặt tên cho phiên bản kernel của mình là .1910637.

Tạo thư mục biên dịch kernel: Tạo một thư mục kernelbuild trong Home. Sau đó, tải kernel source về và giải nén đối với **phiên bản Linux-5.0.5**.

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

Hỏi: Tại sao chúng ta phải sử dụng những kernel source khác từ những server như <http://cdn.kernel.org>. Chúng ta có thể biên dịch kernel source gốc (kernel của OS hiện hành) trực tiếp được không?

Trả lời:

Chúng ta thường sử dụng các phiên bản kernel khác nhau từ một nguồn bởi vì:

Đảm bảo chúng ta có thể sử dụng cùng một phiên bản được hiển thị trong hướng dẫn, để tránh bất kỳ sự cố tương thích nào với các phiên bản kernel mới hơn / cũ hơn.

Chúng ta thường mong muốn không có quá nhiều sự thay đổi đối với phiên bản hạt nhân máy của mình, vì vậy, nếu có gì đó thay đổi xảy ra, nó có thể làm hỏng hệ thống của máy tính.

Tối ưu hóa hạt nhân bằng cách loại bỏ các trình điều khiển vô ích để tăng tốc độ khởi động

Chúng ta có thể biên dịch trực tiếp một kernel nhưng nó lại mất nhiều thời gian và dễ gây ra lỗi.

1.2. Cấu hình

Cấu hình của kernel nằm trong file .config, ta có thể cá nhân hóa kernel bằng cách thay đổi các tùy chỉnh mặc định.

Để tùy chỉnh cấu hình, ta copy file cấu hình của kernel hiện tại sang thư mục Linux-5.0.5:

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

Để tùy chỉnh file cấu hình, ta cần cài đặt các package:

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

Chạy lệnh make **menuconfig** hoặc **make nconfig** để mở kernel configuration:

```
$make nconfig
```

Tiếp theo, ta thêm vào phần mở rộng của phiên bản kernel mã số sinh viên (như yêu cầu đề bài) bằng cách vào mục **General setup version** → **(-ARCH) Local version – append to kernel release** và gõ vào .MSSV. Lưu cài đặt và thoát ra.

1.3. Xây dựng kernel đã cấu hình

Đầu tiên, ta phải biên dịch kernel và tạo máy ảo **vmlinuz**. Sẽ mất một khoảng thời gian khá dài để hoàn thành việc này. Trong terminal ta di chuyển đến thư mục Linux-5.0.5 và thực hiện make:

```
$cd linux-5.0.5
$make
```

Để giảm thời gian make, ta có thể thay thế lệnh make bằng:

```
$make -j 4
```

```
minhtri1910637@minhtri1910637-VirtualBox:~/kernelbuild/linux-5.0.5$ make -j 4
Makefile:594: include/config/auto.conf: No such file or directory
HOSTCC scripts/kconfig/conf.o
HOSTLD scripts/kconfig/conf
scripts/kconfig/conf --synconfig Kconfig
SYSHDR arch/x86/include/generated/asm/unistd_32_ia32.h
SYSTBL arch/x86/include/generated/asm/syscalls_32.h
WRAP arch/x86/include/generated/uapi/asm/bpf_perf_event.h
WRAP arch/x86/include/generated/uapi/asm/poll.h
SYSHDR arch/x86/include/generated/asm/unistd_64_x32.h
```

Xây dựng loadable kernel modules:

```
$make modules
```

Tương tự như make, ta cũng có thể giảm thời gian make modules bằng lệnh thay thế:

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

```
minhtri1910637@minhtri1910637-VirtualBox:~/kernelbuild/linux-5.0.5$ make -j 4 modules
DESCEND objtool
CALL scripts/checksyscalls.sh
Building modules, stage 2.
MODPOST 57 modules
```

Hỏi: Ý nghĩa của việc thực hiện make và make modules là gì? Cái gì được tạo ra và để làm gì?

Trả lời:

Make: được sử dụng để xây dựng và duy trì một nhóm các chương trình và các files từ mã nguồn. Trong bài này, make được sử dụng để biên dịch kernel và tạo ra vmlinuz.

Make modules: được sử dụng để xây dựng và duy trì các modules của kernel. Trong bài này, make modules được sử dụng biên dịch các tệp riêng lẻ cho kernel, một mã liên kết được tạo ra liên kết các tệp đã biên dịch với kernel mới được tạo ra, cụ thể là vmlinuz.

1.4. Cài đặt kernel mới

Cài đặt các modules:

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

Cài đặt kernel mới:

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

Khi kernel mới được cài đặt xong, ta tiến hành khởi động lại máy:

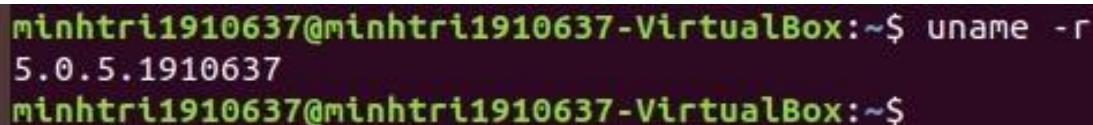
```
$sudo reboot
```

Khi máy ảo đang được khởi động, nhấn giữ phím shift phải để mở menu của **GRUB**, từ đó ta có thể chọn kernel mà ta muốn khởi động cùng với máy ảo, ở đây ta chọn kernel mà ta mới cài đặt.

Sau khi máy ảo hoàn tất quá trình khởi động, ta vào terminal, kiểm tra cài đặt bằng command:

```
$ uname -r
```

Nếu kết quả trả về có chứa MSSV (5.0.5.MSSV) là cài đặt thành công.



```
minhtri1910637@minhtri1910637-VirtualBox:~$ uname -r  
5.0.5.1910637  
minhtri1910637@minhtri1910637-VirtualBox:~$
```

2. LÀM GỌN KERNEL

Sau khi cài đặt kernel thành công, ta có kernel với cấu hình mặc định. Nó có tất cả các gói hỗ trợ cho kernel, trong đó bao gồm những gói không thực sự cần thiết. Để loại bỏ những gói này, ta thực hiện lại việc **make nconfig** và tùy biến cấu hình mong muốn. Hoặc đơn giản hơn và tránh bỏ qua những gói hỗ trợ cần cho kernel hoạt động ổn định, trước khi biên dịch lại, ta chỉ cần command **make localmodconfig**. Make localmodconfig tạo cấu hình dựa trên cấu hình hiện tại và các modules được tải trước đó, đồng thời nó cũng giúp ta loại bỏ những modules không cần thiết. Việc này cũng giúp cho thời gian biên dịch kernel giảm xuống đáng kể.

```

minhtri1910637@minhtri1910637-VirtualBox:~/kernelbuild/linux-5.0.5$ make localmodconfig
using config: '.config'
*
* Restart config...
*
*
* PCI GPIO expanders
*
AMD 8111 GPIO driver (GPIO_AMD8111) [N/m/y/?] n
BT8XX GPIO abuser (GPIO_BT8XX) [N/m/y/?] (NEW) N
OKI SEMICONDUCTOR ML7213 IOH GPIO support (GPIO_ML_IOH) [N/m/y/?] n
ACCES PCI-IDIO-16 GPIO support (GPIO_PCI_IDIO_16) [N/m/y/?] n
ACCES PCIe-IDIO-24 GPIO support (GPIO_PCIE_IDIO_24) [N/m/y/?] n
RDC R-321x GPIO support (GPIO_RDC321X) [N/m/y/?] n
*
* PCI sound devices
*
PCI sound devices (SND_PCI) [Y/n/?] y
  Analog Devices AD1889 (SND_AD1889) [N/m/?] n
  Avance Logic ALS300/ALS300+ (SND_ALS300) [N/m/?] n
  Avance Logic ALS4000 (SND_ALS4000) [N/m/?] n
  ALi M5451 PCI Audio Controller (SND_ALI5451) [N/m/?] n
  AudioScience ASIXxxx (SND_ASIXHPI) [N/m/?] n

minhtri1910637@minhtri1910637-VirtualBox:~/kernelbuild/linux-5.0.5$ sudo vim /etc/default/grub
[sudo] password for minhtri1910637:
minhtri1910637@minhtri1910637-VirtualBox:~/kernelbuild/linux-5.0.5$ sudo update-grub
Sourcing file `/etc/default/grub'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.0.5.1910637
Found initrd image: /boot/initrd.img-5.0.5.1910637
Found linux image: /boot/vmlinuz-4.15.0-159-generic
Found initrd image: /boot/initrd.img-4.15.0-159-generic
Found linux image: /boot/vmlinuz-4.15.0-20-generic
Found initrd image: /boot/initrd.img-4.15.0-20-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
minhtri1910637@minhtri1910637-VirtualBox:~/kernelbuild/linux-5.0.5$

```

3. SYSTEM CALL

3.1. Hiện thực system call

Trong thư mục kernelbuild, tạo một thư mục có tên *get_proc_info*, sau đó tạo một file *sys_get_proc_info.c* bên trong thư mục *get_proc_info*:

```

$cd ~/kernelbuild/linux-5.0.5
$mkdir get_proc_info
$cd get_proc_info
$touch sys_get_proc_info.c

```

Viết code vào file *sys_get_proc_info.c*:

```

#include <linux/kernel.h>
#include <linux/string.h>
#include <linux/syscalls.h>
#include <asm/uaccess.h>
#include <linux/sched.h>
#include <linux/sched/signal.h>
#include <asm/current.h>
#include <linux/module.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;
};

SYSCALL_DEFINE2(get_proc_info,pid_t, pid, struct procinfos*, info){
    struct procinfos proc_infos;
    struct task_struct *process = NULL, *child_process = NULL;
    proc_infos.studentID = 1910637;
    if(pid == -1){
        pid = current->pid;
    }
    for_each_process(process){
        if(process->pid == pid){
            proc_infos.proc.pid = process->pid;
            strcpy(proc_infos.proc.name, process->comm);

            //parent process here
            if(process->real_parent != NULL){
                proc_infos.parent_proc.pid = process->real_parent->pid;
                strcpy(proc_infos.parent_proc.name, process->real_parent->comm);
            }else{
                proc_infos.parent_proc.pid = 0;
                strcpy(proc_infos.parent_proc.name, "No name here");
            }

            //Child process here
            child_process = list_first_entry_or_null(&process->children,struct task_struct,sibling);
            if(child_process != NULL){
                proc_infos.oldest_child_proc.pid = child_process->pid;
                strcpy(proc_infos.oldest_child_proc.name,child_process->comm);
            }else{
                proc_infos.oldest_child_proc.pid = 0;
                strcpy(proc_infos.oldest_child_proc.name, "No name here");
            }
            copy_to_user(info,&proc_infos,sizeof(struct procinfos));
            return 0;
        }
    }
    return EINVAL;
}

```


Tạo *makefile*:

```
$pwd
~/kernel/get_proc_info
$touch Makefile
$echo "obj-y := sys_get_proc_info.o" >> Makefile
```

Vào kernel *Makefile* tìm dòng:

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

Bổ sung thành:

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

Bổ sung một system call vào system call table:

```
$pwd
~/kernelbuild/linux-5.0.5
$cd arch/x86/entry/syscalls/
$echo "548 64 get_proc_info __x64_sys_get_proc_info" >> syscall_64.tbl
```

Hỏi: Ý nghĩa của từng thông tin được đưa vào trong system call table (548 64 get_proc_info sys_get_proc_info)?

Trả lời: Các system call trong system call table có định dạng <number><abi><name><entry point>. Trong đó:

548(<number>) chỉ số thứ tự,

64 (<abi>) chỉ hệ thống 64-bit (tương tự với hệ thống 32-bit hoặc common),

get_proc_info (<name>) là tên của system call,

__x64_sys_get_proc_info tương ứng với <entry point>

Tiếp theo, ta bổ sung một sysem call mới vào trong file system call header:

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

Mở file syscalls.h và bổ sung các dòng sau vào trước *#endif*:

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

Hỏi: Ý nghĩa của các dòng trên là gì?

Trả lời: Các dòng trên được bổ sung vào file header (syscalls.h) nhằm mục đích khai báo các struct *proc_info*, *procinfos* và hàm *sys_get_proc_info(pid_t pid, struct procinfos *info)*.

Cuối cùng, ta biên dịch lại kernel và reboot lại máy ảo để hoàn tất quá trình thêm system call và kernel:

```
$make -j 8
$make modules -j 8
$sudo make modules_install
$sudo make install
$sudo reboot
```

3.2. Testing

Sau khi reboot kernel mới, ta tạo một chương trình C nhỏ để kiểm tra system call đã được tích hợp vào kernel hay chưa:

```
#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;
}
```

```
minhtri1910637@minhtri1910637-VirtualBox:~$ uname -r
5.0.5.1910637
minhtri1910637@minhtri1910637-VirtualBox:~$ gcc test.c -o test
minhtri1910637@minhtri1910637-VirtualBox:~$ ./test
My student ID: 1910637
minhtri1910637@minhtri1910637-VirtualBox:~$
```

Hỏi: Tại sao chương trình trên có thể cho biết hệ thống có hoạt động hay không?

Trả lời: Mục đích của chương trình là in ra MSSV được thêm vào file `sys_get_proc_info.c`. Nếu con số được in ra không phải là MSSV trên, system call đã thất bại.

3.3. Wrapper

Mặc dù system call `get_proc_info` đã hoạt động như mong đợi, ta vẫn phải cải tiến sao cho thuận tiện đối với việc lập trình bằng cách triển khai C wrapper. Nhằm tránh biên dịch nhiều lần, ta tạo một thư mục khác để lưu mã nguồn wrapper. Đầu tiên, ta tạo một file header của chương trình wrapper và các cấu trúc `procinfos`, `proc_info`. Ta đặt tên header là `get_proc_info.h` có chứa nội dung sau:

```
#ifndef _GET_PROC_INFO_H_
#define _GET_PROC_INFO_H_
```

```
#include <unistd.h>

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

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

long sys_get_proc_info(pid_t pid, struct procinfo* info);
#endif // _GET_PROC_INFO_H
```

Hỏi: Tại sao chúng ta cần phải định nghĩa lại các cấu trúc `procinfo` và `proc_info` trong khi đã định nghĩa chúng trong kernel?

Trả lời:

Sau đó chúng ta tạo 1 file `get_proc_info.c` để giữ mã nguồn cho wrapper:

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

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

3.4. Validation

Copy file header `get_proc_info.h` vào `/user/include`:

```
$sudo cp<path to get_proc_info.h> /user/include
```

Hỏi: Tại sao root lại đặc quyền (Ví dụ phải thêm `sudo` trước `cp` để copy file header sang `/user/include`)?

Trả lời: Vì thư mục `/usr` thuộc quyền sở hữu của `root` nên khi copy phải được cấp quyền bởi `root`.

Biên dịch source code như một đối tượng chia sẻ cho phép người dùng truy cập system call của ta thông qua ứng dụng của họ:

```
$gcc -shared -fpic get_proc_info.c -o libget_proc_info.so
```

Sau khi biên dịch thành công, copy file `libget_proc_info.so` sang `/user/lib`:

```
$sudo cp libget_proc_info.so /user/lib
```

Hỏi: Tại sao phải thêm `-shared` và `-fpic` trong `gcc` command?

Trả lời:

Shared Libraries còn gọi là thư viện liên kết động. Nó là file chứa các objects được build từ source code, các file objects này chứa các define của functions, variables. Những file Shared Libraries được load vào file executable của application lúc runtime. Vì nhiều chương trình có thể sử dụng chung các thư viện này nên nó được gọi là Shared Libraries. Shared thường không phụ thuộc vào vị trí. Điều này ảnh hưởng đến việc tạo mã: để không phụ thuộc vào vị trí, ta phải load toàn cục hoặc chuyển đến các hàm bằng cách sử dụng địa chỉ tương đối. *-fpic* giúp cho mã máy được tạo ra không phụ thuộc vào một địa chỉ cụ thể để hoạt động.

Bước cùng là kiểm tra toàn bộ công việc, chúng ta sử dụng chương trình sau và biên dịch với tùy chọn *-lget_proc_info* option:

```
#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 procinfo info;
    if(get_proc_info(myPid, &info) == 0){
        printf("StudentID: %ld\n", info.studentID);
        printf("Current Process with pid: %d\n", info.proc.pid);
        printf("Current process name: %s\n", info.proc.name);
        printf("Parent process: %d\n", info.parent_proc.pid);
        printf("Parent process name: %s\n", info.parent_proc.name);
        printf("Oldest child process: %d\n", info.oldest_child_proc.pid);
        printf("Oldest child process name: %s\n", info.oldest_child_proc.name);
    }else{
        printf("Cannot get the information from the process %d\n", myPid);
    }
    return 0;
}
```

```
minhtri1910637@minhtri1910637-VirtualBox:~$ uname -r
5.0.5.1910637
minhtri1910637@minhtri1910637-VirtualBox:~$ gcc test_final.c -lget_proc_info -o
final
minhtri1910637@minhtri1910637-VirtualBox:~$ ./final
PID: 11556
StudentID: 1910637
Current Process with pid: 11556
Current process name: final
Parent process: 2152
Parent process name: bash
Oldest child process: 0
Oldest child process name: No name here
minhtri1910637@minhtri1910637-VirtualBox:~$
```

Ghi chú và tài liệu tham khảo:

***real_parent*:** Trường này sẽ trỏ tới bộ mô tả tiến trình – process descriptor của tiến trình tạo ra tiến trình P (tiến trình cha của P), hoặc trỏ tới bộ mô tả tiến trình của tiến trình 1 (tiến trình init – chúng ta sẽ tìm hiểu tiến trình này sau) trong trường hợp tiến trình cha của P không còn tồn tại.

***parent*:** Trường này trỏ tới tiến trình cha hiện tại của P (tiến trình sẽ nhận được signal khi tiến trình con kết thúc). Trường này thường là *real_parent*, nhưng có một vài trường hợp trường này trỏ đến tiến trình khác. Ví dụ: Trong trường hợp một tiến trình sử dụng *ptrace()* system call mà chúng ta đề cập tới trong chương trước để theo dõi tiến trình P, trường *parent* sẽ trỏ tới tiến trình gọi *ptrace()*.

Tại sao lại SYSCALL_DEFINE2(...)

<https://williamthegrey.wordpress.com/2014/05/18/add-your-own-system-calls-to-the-linux-kernel/>
Check *linux/syscalls.h*

get_proc_info: tên của system call
type, name....

task_struct?

<http://www.science.smith.edu/~nhowe/262/oldlabs/sched.html>

Trong hạt nhân Linux, các tiến trình được định nghĩa là cấu trúc *task_struct* trong *include / linux / Sched.h*, dòng 281. Cấu trúc này chứa mọi thông tin liên quan về một tiến trình.

comm?

Tên lệnh được lưu trữ trong *current->comm*; *comm* là tên cơ sở của tệp chương trình đang được tiến trình hiện tại thực thi.

real_parent?

<https://ypl.coffee/parent-and-real-parent-in-task-struct/>

<https://superuser.com/questions/632979/if-i-know-the-pid-number-of-a-process-how-can-i-get-its-name>

copy_to_user?

<https://developer.ibm.com/articles/l-kernel-memory-access/>

list_first_entry_or_null

<https://www.kernel.org/doc/html/docs/kernel-api/API-list-first-entry-or-null.html>