**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**٭٭٭٭٭**

****

**BÁO CÁO ĐỒ ÁN: SYSCALL & HOOK**

**MÔN HỌC: HỆ ĐIỀU HÀNH**

LỚP : 18CNTN

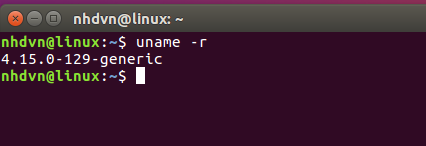
SINH VIÊN THỰC HIỆN : 18120019 – Nguyễn Hoàng Dũng

18120052 – Lê Hạnh Linh

*Thành phố Hồ Chí Minh, ngày 12 tháng 01 năm 2021*

1. **SYSCALL:**
   1. **Cài đặt Ubuntu và kiểm tra kernel version:**

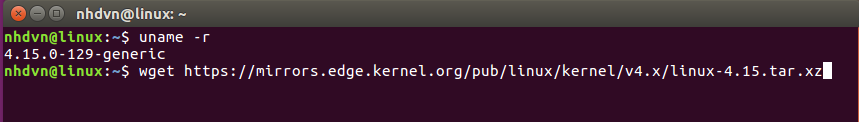
Nhóm em sử dụng Ubuntu 16.04.7 và có kernel version là 4.15.0



* 1. **Tải phiên bản kernel để làm việc với syscall:**

Vào đường dẫn sau: <https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/>

Phiên bản kernel phù hợp nhất có dạng linux-4.15.x. Nhóm em chọn 4.15 và sử dụng wget để tại về máy.



Sau đó giải nén và copy vào thư mục /usr/src

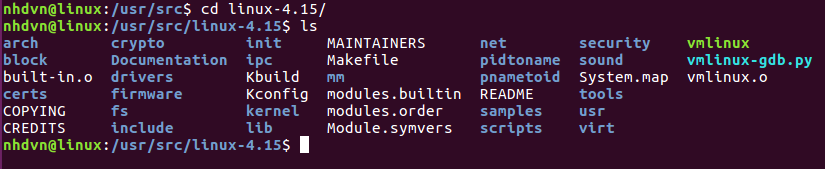
Thư mục này chưa tất cả các kernel version hiện có trong máy

* 1. **Tiến hành cài đặt 2 syscall pnametoid và pidtoname**

Việc cài đặt 2 syscall để thực hiện các nhiệm vụ tương tự như việc viết hàm trong C. Điều cần chú ý duy nhất ở đây là trước các hàm có thêm từ khóa asmlinkage thông báo rằng các parameter của hàm này sẽ không nằm trên các thanh ghi mà nằm trên CPU stack.

Trong báo cáo này chúng em sẽ trình bày tiếp các bước để biên dịch kernel chứa 2 syscall này.

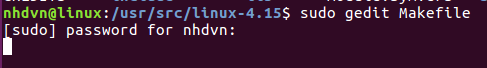
Trong thư mục kernel ta cần tạo thêm 2 folder là pidtoname và pnametoid để chứa 2 syscall tương ứng.



Nội dung của 2 folder này chính là mã nguồn C kèm theo makefile.

Bước tiếp theo cần config để khi biên dịch kernel có thể nhận ra và thêm 2 syscall này vào hệ điều hành.

Ta cần chỉnh sửa makefile toàn cục như sau:



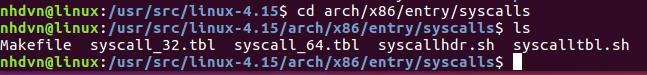
Sử dụng gedit để chính sửa dưới quyền root, ta ctrl-f và tìm từ khóa “core-y”



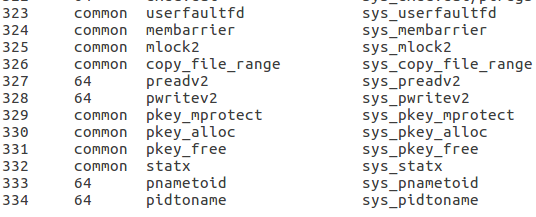
Ta cần thêm đường dẫn đến 2 syscall tại đây. Sau đó save file.

Tiếp đến là cần cấp cho 2 syscall này các số hiệu để khi interrupt, userspace sẽ biết cần phải gọi syscall number nào.

Di chuyển đến thư mục sau:

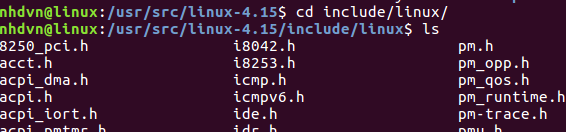


Ta sẽ chỉnh sửa syscall\_64.tbl để thêm 2 number cho syscall mới. Đây là một bảng các syscall mà khi thực hiện interrupt, kernel sẽ biết user đang muốn tìm đến kernel function nào.

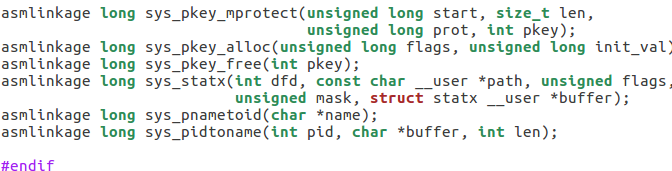


Ta sẽ thêm 2 number mới vào cuối danh sách. Cần lưu ý khai báo các number còn trống, tránh trùng với các number đã có.

Tiếp đến cần thông báo cho kernel biết nguyên mẫu của 2 hàm này sẽ như thế nào. Ta di chuyển đến thư mục sau:



Thư mục này chứa các header để kernel biên dịch. Có rất nhiều header, nhưng ta chỉ cần quan tâm syscall.h



Ta thêm 2 nguyên mẫu hàm tương ứng với 2 syscall. Các nguyên mẫu hàm cần khớp với các hàm ta đã cài đặt trong 2 thư mục /pidtoname và /pnametoid ban đầu.

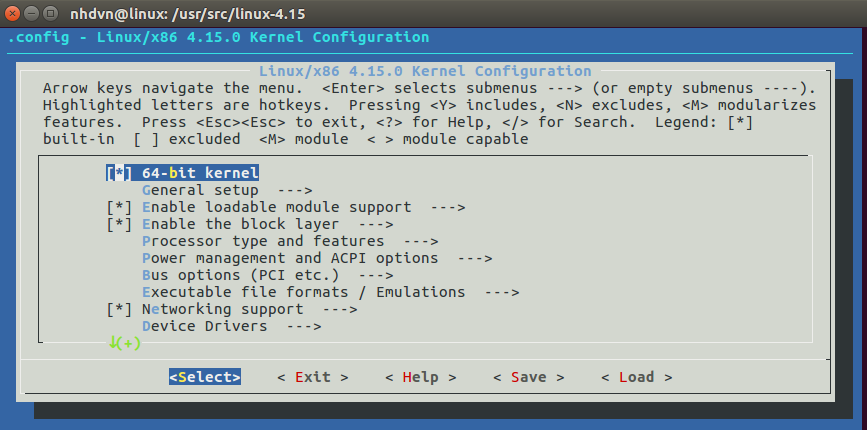
* 1. **Biên dịch Kernel:**

Để cập nhật các thay đổi mới đến kernel, ví dụ như thêm hoặc sửa các system call, ta cần biên dịch kernel. Khi biên dịch lần đầu ta cần phải khởi tạo file Kconfig bằng lệnh sudo make menuconfig.

Kconfig là một file giúp ta đảm bảo hệ điều hành đã có đầy đủ các dependency cần thiết để phục vụ cho quá trình build kernel.

Thông thường khi biên dịch sẽ có thể có nhiều option đi kèm theo. Ta có thể chỉnh một cách thủ công trong file Kconfig. Tuy nhiên điều này là không được khuyến khích vì thường Kconfig chứa rất nhiều thông tin về các cài đặt liên quan đến nhau. Nếu ta tùy ý thay đổi mà không cân chỉnh cài đặt khác đồng bộ theo thì sẽ có thể dẫn đến quá trình build thất bại hoặc crash hệ thống.

Thay vào đó công cụ menuconfig có sẵn sẽ giúp ta thực hiện quá trình này. Khi lần đầu biên dịch menuconfig sẽ giúp ta tạo file Kconfig cơ bản đầy đủ.

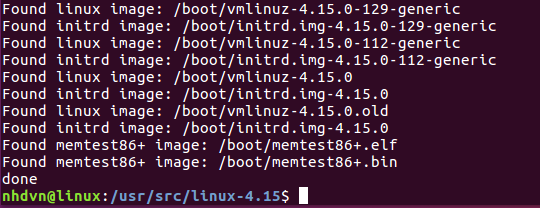


Mặc định đã có sẵn các setup cần thiết. Ta chỉ cần save và exit.

Sau đó là tiến hành make để biên dịch kernel.

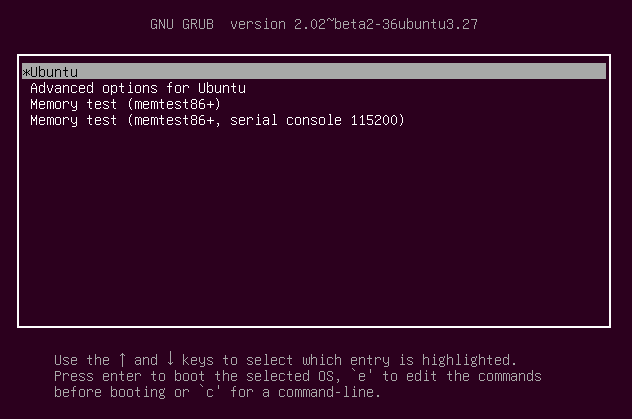


Vì chúng ta phải biên dịch lại toàn bộ kernel, kể cả những file có sẵn để có thể sử dụng được 2 syscall mới, nên sẽ tốn rất nhiều thời gian biên dịch. Cho nên nếu có thể, cần sử dụng nhiều core để hỗ trợ quá trình biên dịch nhanh hơn.

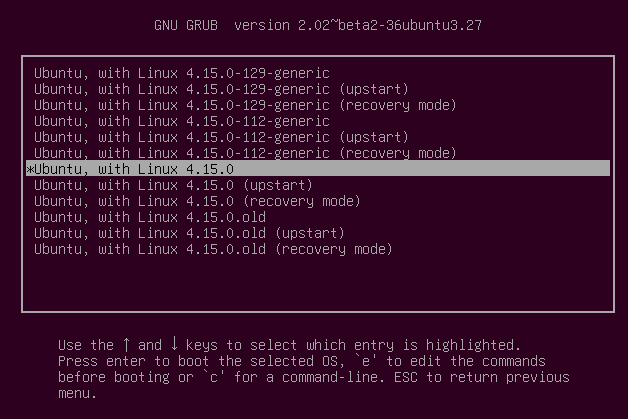


Khi đã biên dịch xong ta sẽ tiến hành test 2 syscall mới. Để test được ta cần switch tạm thời giữa kernel đang sẵn có, và kernel 4.15 mới được build. 2 kernel này đều nằm trong thư mục /usr/src như đã đề cập trước.

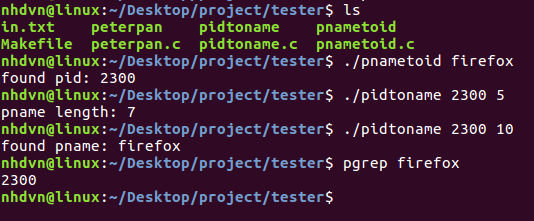
Ta sẽ reboot hệ thống và nhấn ESC để vào advanced menu option.



Cần chọn đến đúng phiên bản kernel là 4.15, chính là bản mà ta đã biên dịch.



* 1. **Test syscall**



2 file test sẽ sử dụng cơ chế command line argument. Trong đó, pnametoid sẽ nhận thêm 1 chuỗi chứa tên tiến trình ta muốn xem id, còn pidtoname sẽ nhận thêm 2 biến lần lượt là id và độ lớn buffer dành cho tên trả về. Nếu độ lớn buffer không đủ pidtoname sẽ trả về giá trị buffer tối thiếu để ta chỉnh sửa.

1. **HOOK**

Trong mục này ta sẽ xây dựng module kernel có chức năng hook khi 1 system call được gọi. Việc xây dựng kernel module đã được thực hiện ở đồ án 2 nên chúng em sẽ hạn chế nhắc lại và giải thích thêm về nguyên lý của hook.

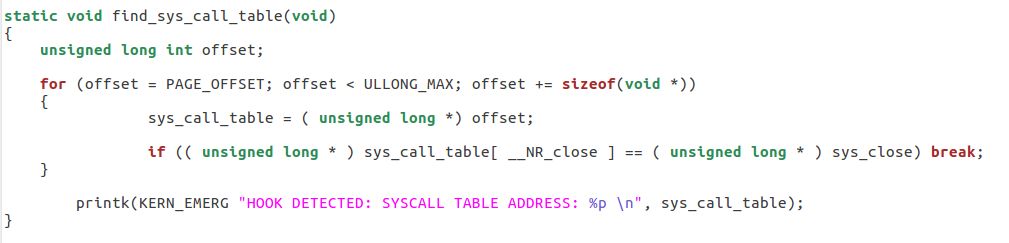
Khi 1 system call được gọi, ở mức độ mã máy, user space sẽ thực hiện interrupt 0x80 đến kernel. 0x80 là số hiệu của kernel interrupt handler, có chức năng xử lý các lời gọi đến kernel function. Khi ta muốn dùng hàm write ở user space, ta đang gọi đến 1 hàm write tương ứng trong kernel space, hàm write cũng như các hàm khác trong kernel space, đều có 1 số hiệu gọi là syscall number.

Khi ta thực hiện interrupt, thanh ghi EAX sẽ chứa syscall number của kernel function mà ta muốn gọi. Các số hiệu này được chứa trong 1 file table, chính là file syscall\_64.tbl mà ở mục trên nhóm em có nhắc đến.

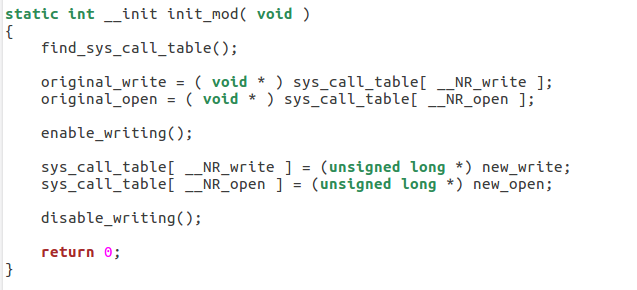
Như vậy, số hiệu sẽ tương ứng với 1 hàm trong kernel. Hàm trên userspace lại tương ứng với số hiệu đó. Sẽ ra sao nếu ta thay đổi hàm write trong table này mà vẫn giữ nguyên số hiệu. Khi đó tất cả các tiến trình trên userspace thay vì gọi đến hàm write mặc địch của kernel sẽ gọi đến hàm write mà ta đã thay thế.

Đây chính là nguyên lý của hook.

Như vậy điều đầu tiên ta cần làm chính là phải tìm được table syscall và thay thế 2 hàm open và write.



Sau đó ta cần cấp quyền chỉnh sửa để thay đổi địa chỉ hàm open và write trong table bằng địa chỉ hàm new\_write và new\_open mà ta đã custom trong module.

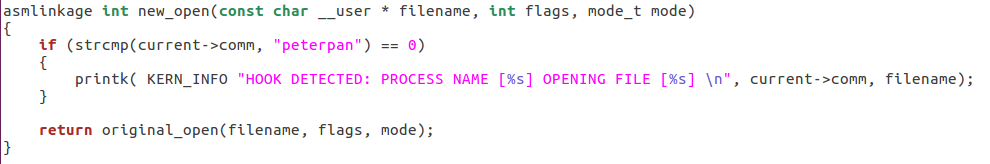


Sau đó tiến hành make để biên dịch module và thực hiện insmod



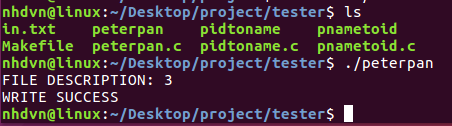
Hàm tìm kiếm đã phát huy tác dụng và tìm thấy Syscall Table tại địa chỉ trên.

Tuy nhiên, mặc định trong hệ điều hành đã có rất nhiều các tiến trình khác luôn thực hiện việc open và write nên nếu không có điều kiện chặn, các hàm open và write đã custom của chúng em sẽ liên tục được gọi và thông báo ra màn hình.

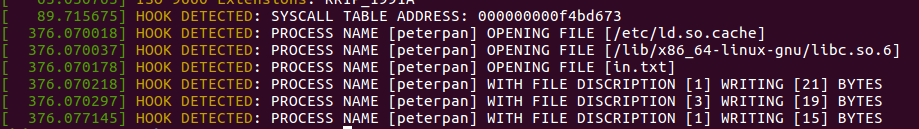


Thay vào đó chúng em thêm 1 điều kiện chỉ các thao tác mở và ghi file tại tiến trình nhất định trên userspace mới cho phép hook để demo. Trong trường hợp này là file test peterpan (1 cái tên lạ để không nhầm với các tiến trình sẵn có).

Bây giờ nhóm em sẽ thử chạy file peterpan để kiểm chứng.



Peterpan đã mở và ghi trên 1 file có chỉ số file descriptor là 3. Giờ nhóm em sẽ kiểm tra trong kernel, hooker đã hook được hay chưa.



Có thể thấy ngoài việc đụng đến file in.txt, peterpan còn mở và ghi trên các file khác. Vì tất cả đều nằm trong context của peterpan nên hooker đều bắt được hết và ghi nhận lại.

**END**