System Call

Trong kiến trúc Linux, không gian bộ nhớ được chia thành hai phần là user space và kernel space. Theo đó, cũng tồn tại hai chế độ (mode) là user mode và kernel mode. Các chỉ lệnh được gọi từ chương trình như đóng mở file (fopen, fclose), hoặc in một thông tin (printf) chỉ có thể thực thi và truy cập vùng nhớ ở tầng user mà không được truy cập vùng nhớ của kernel.

Cơ chế phân tách user space với kernel space và không cho phép người dùng tự ý truy cập tài nguyên của kernel giúp quản lý và bảo vệ kernel cũng như toàn bộ hệ thống. Thật vậy, khi bạn truy cập vùng nhớ trái phép trên user space thì chỉ ứng dụng của bạn crash, còn khi bạn truy cập trái phép vùng nhớ của kernel thì toàn bộ thiết bị của bạn sẽ bị crash.

Vấn đề đặt ra là làm cách nào để user gọi xuống kernel hay thao tác điều khiển các device driver? Để đáp ứng yêu cầu này, kernel cung cấp cho user space các API (còn gọi là các dịch vụ) là system call.

System call là một cửa ngõ vào kernel, cho phép tiến trình trên tầng user yêu cầu kernel thực thi một vài tác vụ cho mình. Những dịch vụ này có thể là tạo một tiến trình mới (fork), thực thi I/O (read, write), hoặc tạo ra một pipe cho giao tiếp liên tiến trình (IPC).

Có một số điều cần chú ý về system call như sau:

- Khi một tiến trình gọi một system call, CPU sẽ chuyển từ chế độ user mode sang kernel mode, điều này cho phép CPU truy cập các vùng nhớ và thực hiện các chỉ lệnh của kernel.
- Mỗi system call được kernel định danh bằng một số duy nhất. Tiến trình trên tầng user không biết đến các số này, thay vào đó, nó gọi một system call bằng tên hàm (ví dụ như open(), read()...).
- Mỗi system call có thể có một số tham số truyền để cung cấp thông tin từ user truyền xuống kernel và ngược lại.

Quá trình thực thi system call

Đứng trên góc nhìn của lập trình viên, việc gọi một system call trông có vẻ như là chỉ gọi một hàm C bình thường. Tuy nhiên, đằng sau việc đó là rất nhiều bước được thực hiện từ user space xuống kernel space.

Cụ thể, chúng ta thử xét việc gọi một hàm thư viện được dùng rất thường xuyên sau đây:

Hàm gọi:

```
#include<stdio.h>

FILE *fopen(const char *filename, const char *mode)
```

Hàm fopen() là một hàm thư viện (wrapper function) được dùng để thực thi việc chuyển xuống kernel mode và yêu cầu kernel mở một file dưới kernel có đường dẫn là “filename” với chế độ "mode", chi tiết về hàm này các bạn xem tại đây. Hàm fopen() được triển khai bằng cách gọi system call open(), cụ thể các bước như sau:

1. Hàm wrapper copy các đối số (trong trường hợp này là "filename" và "mode") vào các thanh ghi, nơi mà các lệnh của luồng thực thi system call sẽ đọc và sử dụng được.

2. Hàm wrapper sao chép số system call vào một thanh ghi của CPU (%eax). Ví dụ system call number của open() là 5, hàm này sẽ sao chép giá trị 5 vào thanh ghi %eax.

3. Hàm wrapper thực hiện một chỉ lệnh máy gọi là trap machine instruction để chuyển chế độ CPU từ user mode sang kernel mode. Chỉ lệnh này có thể là một ngắt mềm (software interrupt) với số ngắt (interrupt number) là 0x80 (int 0x80) hoặc chỉ lệnh SYSENTER (trong các kiến trúc Intel gần đây) hoặc chỉ lệnh SYSCALL (trong AMD)

4. Kernel gọi đến luồng system_call (nằm trong file arch/x86/entry_32.S), tại đây nó sẽ làm các công việc: copy giá trị các đối số trong các thanh ghi mà đã copy vào trong bước 1 vào kernel stack; kiểm tra tính hợp lệ của các đối số; gọi đến system call service routine thích hợp bằng cách tra cứu số system call được sao chép ở bước 2 trong bảng system call routine (sys_call_table); gửi kết quả trả về lên cho hàm wrapper và cuối cùng là chuyển chế độ của CPU từ kernel mode sang user mode.

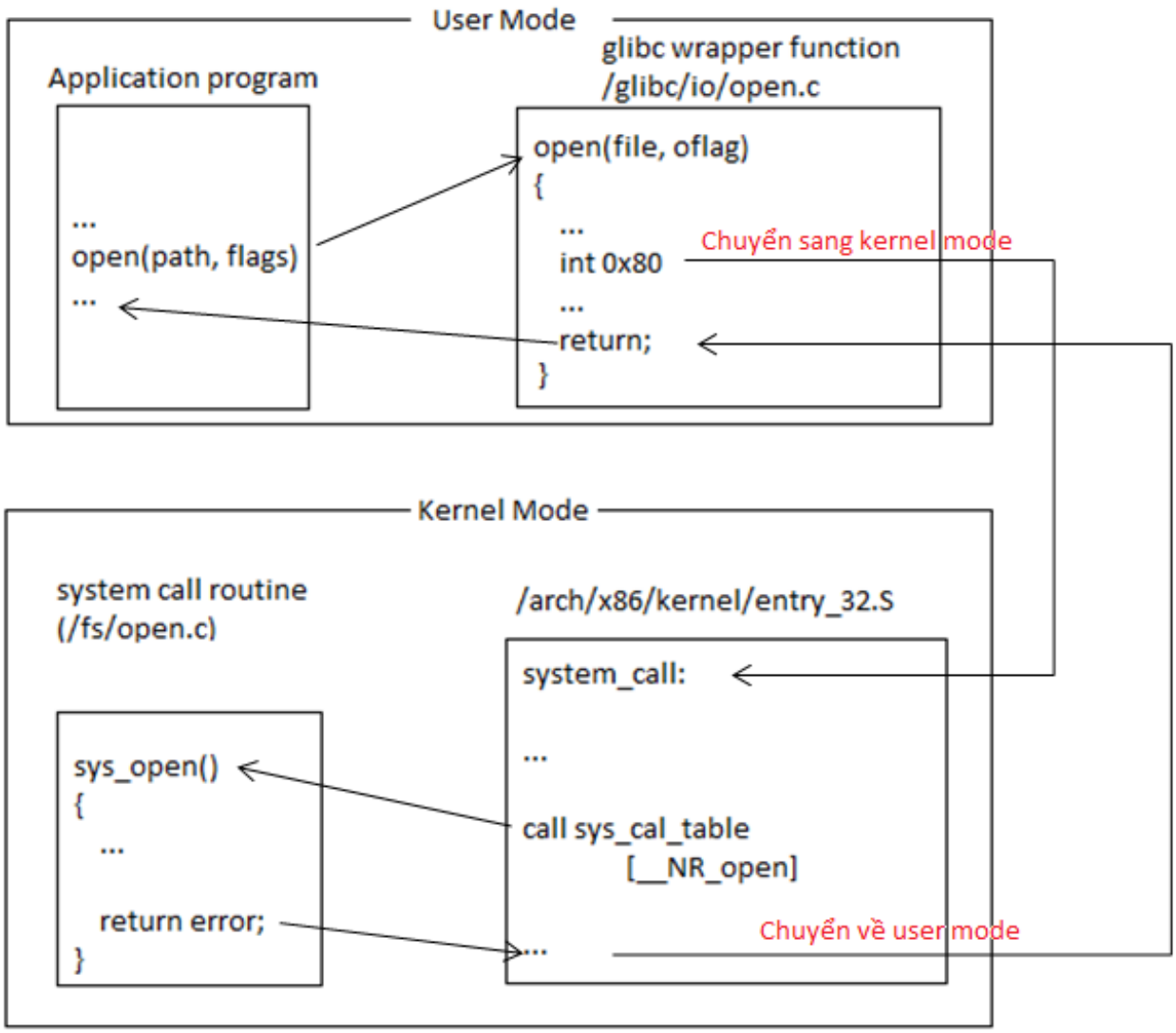
5. Hàm wrapper trả về giá trị là một số nguyên cho hàm gọi nó để thông báo lời gọi system call có thành công không. Nếu system call trả về giá trị lỗi, hàm wrapper sẽ set giá trị cho một biến toàn cục “errno” từ giá trị lỗi này.

Lưu ý rằng các bước trên chỉ giới thiệu tinh thần chung của việc tiếp nhận xử lý một system call, các kiến trúc khác nhau sẽ có các cách triển khai tinh thần chung đó khác nhau ít nhiều.

Để hiểu rõ hơn về bức tranh tổng thể của system call, chúng ta sẽ đi vào chi tiết cách system call open(). Trong Linux x86_32, open() có số system call là 5, vì vậy trong system call vector (arch/x86/entry/syscall/syscall_32.tbl), open system call sẽ là entry thứ 5, tương ứng với system call routine là sys_open.

```
# 32-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point> <compat entry point>
#
# The abi is always "i386" for this file.
#
0       i386    restart_syscall         sys_restart_syscall
1       i386    exit                    sys_exit
2       i386    fork                    sys_fork                        sys_fork
3       i386    read                    sys_read
4       i386    write                   sys_write
5       i386    open                    sys_open                        compat_sys_open
```

Từ đây, kernel sẽ gọi đến hàm sys_open() trong /fs/open.c để thực thi việc mở một file trong hệ thông file system và trả về một mô tả file fd cho user. Cụ thể, chúng ta xem hình vẽ chi tiết như hình dưới đây:



Kết luận

System call là một cơ chế quan trọng mà bất kỳ lập trình viên Linux nào cũng phải nắm được. Sử dụng system call để yêu cầu dịch vụ hoặc tài nguyên của kernel không những giúp bảo vệ được hệ thống mà còn giúp cho công việc lập trình viên trở nên dễ dàng hơn. Theo đó, bạn chỉ cần sử dụng các API thân thiện với người dùng hơn là phải quan tâm đến các thanh ghi và luồng của hệ thống.
Bạn cũng cần lưu ý rằng các kiến trúc CPU khác nhau thì cách triển khai xử lý system call cũng có thể khác nhau. Vì vậy, bạn cũng không cần cố gắng chọc ngoáy sâu xuống code của kernel về việc xử lý system call (trừ khi bạn cần tạo một system call của riêng mình), mà chỉ cần dùng các hàm wrapper của thư viện C là đủ.

Bài học sau sẽ giới thiệu về các thư viện hàm thư viện C trong lập trình Linux.