آزمایش ۵ - آشنایی با فراخوانی های سیستمی

۵.۱ مقدمه

در این جلسه از آزمایشگاه با برخی از مهم‌ترین فراخوانی‌های سیستمی در سیستم‌عامل لینوکس آشنا خواهیم شد و به کمک آنها چند برنامه خواهیم نوشت. همچنین روش اضافه کردن فراخوانی‌های سیستمی به هسته ی لینوکس را خواهیم آموخت.

۵.۱.۱ اهداف

انتظار می‌رود در پایان این جلسه دانشجویان مطالب زیر را فرا گرفته باشند:

آشنایی با مفهوم فراخوانی سیستمی.
نحوه ی اجرای فراخوانی‌های سیستمی.
فراخوانی‌های سیستمی مهم و پرکاربرد در سیستم‌عامل لینوکس.
نحوه ی ایجاد فراخوانی‌های سیستمی جدید.

۵.۱.۲ پیش‌نیازها

انتظار می‌رود که دانشجویان با موارد زیر از پیش آشنا باشند:

برنامه‌نویسی به زبانc/c++

۵.۲ فراخوانی سیستمی چیست؟

فراخوانی‌های سیستمی یا system call ها توابعی هستند که در هسته ی سیستم‌عامل پیاده‌سازی شده اند. هنگامی که یک برنامه یک فراخوانی سیستمی انجام می‌دهد، کنترل اجرا از آن برنامه به هسته منتقل می‌شود تا عملیات درخواست شده صورت پذیرد. فراخوانی‌های سیستمی برای کارهای مختلفی مانند دسترسی به منابع، تخصیص آنها، خاموش کردن یا راه‌اندازی مجدد سیستم‌عامل و ... مورد استفاده قرار می گیرند. برخی از این فراخوانی های سیستمی تنها در پروسه‌هایی قابل استفاده اند که توسط super-user اجرا شده باشند.

هر فراخوانی سیستمی با یک شماره‌ی ثابت شناخته می‌شود. این شماره، پیش از کامپایل شدن هسته باید مشخص گردد. به همین دلیل در سیستم‌عامل لینوکس افزودن فراخوانی‌های سیستمی تنها با کامپایل و نصب مجدد هسته امکان پذیر است.

برای اطلاعات بیشتر در مورد فراخوانی‌های سیستمی در لینوکس به فایل ضمیمه مراجعه کنید.

۵.۳ شرح آزمایش

۵.۳.۱ مشاهده ی فراخوانی‌های سیستمی تعریف شده

وارد سیستم‌عامل مجازی ایجاد شده در جلسات قبل شوید.

سیستم‌عامل لینوکس در حال حاضر شامل بیش از ۳۰۰ فراخوانی سیستمی است. فایل زیر را به کمک یک ویرایشگر باز کنید؛ در این فایل می توانید لیست فراخوانی‌های سیستمی به همراه شماره ی آنها را بیابید.

/usr/include/i386-linux-gnu/asm/unistd_32.h

۵.۳.۲ اجرای یک فراخوانی سیستمی

در پوشه‌ی خانه‌ی خود فایلی با نام testsyscall.cpp ایجاد کنید.

کد زیر با استفاده از فراخوانی سیستمی mkdir یک پوشه‌ی جدید ایجاد می‌کند. آن را در فایلی که در مرحله‌ی قبل ایجاد کرده‌اید وارد کنید:

برنامه‌ی نمونه برای ایجاد یک پوشه:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>
int main () {
   long result;
   result = syscall(__NR_mkdir , "testdir", 0777);
   printf("The result is %ld.\n", result);
```

```
    return 0;
}
```

کد را کامپایل کنید و سپس اجرا نمایید.

- ? نتیجه‌ی اجرای آن را شرح دهید.
- ? در مثال بالا، نقش __NR_mkdir چیست؟
- ? در مورد نحوه‌ی استفاده از دستور syscall و ورودی‌ها و خروجی‌های آن توضیح دهید.

۵.۳.۳ اجرای ساده‌تر فراخوانی سیستمی

برای کاربرد ساده‌تر فراخوانی‌های سیستم بدون نیاز به شماره‌ی آن‌ها، می‌توان از توابعی استفاده کرد که از پیش به عنوان wrapper برای آن‌ها نوشته شده‌اند. برای مثال برای فراخوانی سیستمی بخش قبلی می‌توان از تابع mkdir()که در sys/stat.h قرار دارد استفاده کرد. به دلیل خوانایی بالاتر و سادگی کاربرد، معمولا ترجیح بر استفاده از این توابع به جای استفاده ی مستقیم از دستور syscall است.

- ? کد بخش قبل را به کمک تابع mkdir() بازنویسی کنید و در فایل testsyscall2.cpp ذخیره نمایید.

۵.۳.٤ آشنایی با چند فراخوانی سیستمی پرکاربرد

در هر کدام از فعالیت‌های این بخش، یک فراخوانی سیستمی معرفی می شود؛ به کمک این فراخوانی سیستمی برنامه‌های خواسته‌شده را بنویسید. برای دریافت راهنمایی در مورد هر کدام از این فراخوانی‌های سیستمی می توانید از دستور man 2 [syscall_name] استفاده کنید.

- ? برای دیدن امکان دسترسی به فایل‌ها، فراخوانی سیستمی access مورد استفاده قرار می‌گیرد. برنامه‌ای بنویسید که به عنوان آرگومان ورودی یک آدرس را دریافت کند و ببیند که آیا اولاً آن آدرس وجود دارد یا خیر و ثانیاً آیا دسترسی به آن برای پروسه ی اجرا شده امکان پذیر است؟

- ? به کمک فراخوانی‌های سیستمی open, write, close برنامه‌ای بنویسید که یک فایل با اسم oslab2.txtایجاد کرده و نامتان را در آن فایل بنویسد.

? به کمک فراخوانی سیستمی sysinfo برنامه‌ای بنویسید که میزان حافظه RAM کل و همچنین حافظه ی خالی را در خروجی چاپ کند.

? به کمک فراخوانی سیستمی getrusage برنامه‌ای بنویسید که تعداد context swith های خود (داوطلبانه یا غیر داوطلبانه) را چاپ کند.

۵.۳.۵ اضافه کردن یک فراخوانی سیستمی به سیستم‌عامل

همان طور که در ابتدا بیان شد، برای اضافه کردن فراخوانی های سیستمی به هسته ی لینوکس نیازمند آن هستیم که هسته را مجدداً کامپایل و نصب کنیم. برای اضافه کردن یک فراخوانی سیستمی سه گام اصلی باید انجام شود:

۱. اضافه کردن تابع جدید،
۲. به‌روزرسانی فایل‌های سرآیند،
۳. به‌روزرسانی جدول فراخوانی‌های سیستمی.

در اینجا قصد داریم که یک فراخوانی سیستمی ساده را به سیستم‌عامل اضافه کنیم. مراحل دقیق این کار بسته به این که چه نسخه ای از هسته را انتخاب می‌کنید و قصد دارید تا آن را برای اجرا روی چه معماری‌های کامپیوتری ای کامپایل کنید، تفاوت می‌کند.

در ادامه یک دستورالعمل ساخت فراخوانی سیستمی ضمیمه شده است.

لینک‌هایی برای مطالعه ی بیشتر

در این لینک توضیحات خوبی در مورد روند کار ارائه شده. همچنین به syntax مورد نیاز برای فراخوانی های سیستمی دارای آرگومان نیز پرداخته شده. به عنوان مرجعی دیگر برای راهنمایی گام به گام از این لینک می‌توانید استفاده کنید.

پیشنهاد می‌شود که به این لینک زیر برای توضیحاتی فنی، به‌روز و قابل اعتماد در خصوص اضافه کردن system callمراجعه کنید.

ساخت یک فراخوانی سیستمی

در لینوکس می‌توانید فراخوانی سیستمی خود را بسازید و آن را به‌گونه‌ای جاسازی کنید که در تراز هسته اجرا شود. شیوه‌ی ساخت فراخوانی را اینجا خواهیم آموخت و آن را به هسته می‌شناسانیم. گام‌های ساخت یک فراخوانی سیستمی اینگونه است:

۱. نخست باید دانست که کار فراخوانی چه خواهد بود. هر فراخوانی تنها باید یک کار انجام دهد. باید دانست که آرگومانهای آن چه هستند و چه بازمی‌گرداند. همچنین، باید دانست که چه نمادهای هشداری نیاز دارد.

۲. فراخوانی باید درستی داده‌های ورودی خود را بررسی کند. زیرا فراخوانی در تراز هسته کار می‌کند و اگر کاربر بتواند داده‌های نادرست را از این راه به درون هسته بفرستد کارکرد و پایداری همه‌ی سامانه دچار چالش می‌شود. برای نمونه، فراخوانی‌های ورودی و خروجی باید درستی «نشانگر فایل[۲۴]» را بررسی کنند، همانگونه که فراخوانی‌های کار با فرایندها باید درستی «شناسه‌ی فرایند» را.

ساخت یک فراخوانی ساده

فراخوانی زیر را در فایلی به نام oslab.c می‌نویسیم.

24 File Descriptor

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

SYSCALL_DEFINE0(oslab)
{
        printk("\nThis is OSLAB's system call.\n");
        return 0;
}
```

در این تکه‌برنامه از تابع printk بهره برده‌ایم. این تابع رشته‌ی ورودی خود را در فایل log (رویدادهای) هسته می‌نویسد. همچنین، می‌بینید که نام فراخوانی آرگومان ورودی SYSCALL_DEFINE0. عدد صفر نشان‌دهنده‌ی این است که آرگومان ورودی ندارد.

هشدار: در هسته ما به دستور printf دسترسی نداریم! از اینرو تنها می‌توانیم در پرونده‌ی رویدادهای هسته با کمک فرمان printk بنویسیم.

جایگذاری فراخوانی پایین‌رده در سیستم عامل

۱. در پوشه‌ای که از نافشرده‌سازی هسته‌ی بارگیری‌شده در آزمایش پیشین (کامپایل هسته) به دست آمد، پوشه‌ای تازه برای فراخوانی‌های سیستمی خودمان می‌سازیم (ما نام آن را mysyscalls) می‌نامیم.

```
$ cd linux-5.9.6/
$ mkdir mysyscalls
```

۲. سپس، فایل فراخوانی oslab.c را درون این پوشه می‌گذاریم.

۳. روند کار اینگونه است که فایل oslab.c در «هنگام کامپایل و ساخت هسته» کامپایل شود و برنامه‌ی هسته به آن دسترسی داشته باشد. برای این که در هنگام کامپایل، بدانیم که این فایل باید چگونه ساخته (make) شود، برای آن یک Makefile می‌سازیم.

```
$ cd mysyscalls
$ nano Makefile
```

در این فایل چنین چیزی می‌نویسیم.

```
  GNU nano 4.8                    Makefile
obj-y := oslab.o
```

نگاره ۴: پرونده‌ی Makefile برای فراخوانی سیستم oslab.c

۴. اکنون باید در فایل Makefile اصلی هسته، که هسته برپایه‌ی آن ساخته می‌شود، به سازنده‌ی هسته نشان دهیم که فراخوانی سیستم تازه‌ی ما و فایل سازنده‌ی (Makefile) آن کجاست. پس، فایل Makefile هسته را باز می‌کنیم. رشته‌ی «core-y» را جستجو می‌کنیم و نشانی پوشه‌ی mysyscalls را به فهرست پوشه‌های روبروی آن می‌افزاییم. در دنباله‌ی دستورهای گام ۳ این دستورها را می‌توانیم به کار ببریم:

```
$ cd ..
$ nano Makefile
```

پس از جستجوی «core-y» در خطی مانند زیر نشانی پوشه را می‌افزاییم (در ویرایشگر nano می‌توانید رشته‌ای را با کلید ترکیبی ctrl+w جستجو کنید.) فایل را ذخیره می‌کنیم و می‌بندیم.

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

vmlinux-dirs    := $(patsubst %/,%,$(filter %/, \
```

۵. افزودن فراخوانی به جدول فراخوانی‌های معماری؛ این جدول درون یک فایل است. در این فایل، در هر خط یک فراخوانی به هسته شناسانده شده و یک شماره به آن داده شده است. شماره‌گذاری از «۰» است. برای هر معماری که از فراخوانی پشتیبانی می‌کند، این جایگذاری باید انجام شود. برای نمونه، در سیستم ما که x86_64 است ما فراخوانی را به جدول فراخوانی‌ها برای معماری ۶۴ بیت می‌افزاییم. پس از ذخیره و بستن فایل Makefile در گام پیشین می‌توانید دستورهای زیر را انجام دهید.

```
$ nano arch/x86/entry/syscalls/syscall_64.tbl
```

در فایلی که باز می‌شود، پس از فراخوانی‌های سیستمی ویژه‌ی ۶۴ بیت و پیش از فراخوانی‌های ویژه‌ی ۳۲ بیت باید فراخوانی تازه را به فهرست بیافزاییم. چیزی مانند نگاره ۵.

```
424     common  pidfd_send_signal       sys_pidfd_send_signal
425     common  io_uring_setup          sys_io_uring_setup
426     common  io_uring_enter          sys_io_uring_enter
427     common  io_uring_register       sys_io_uring_register
428     common  open_tree               sys_open_tree
429     common  move_mount              sys_move_mount
430     common  fsopen                  sys_fsopen
431     common  fsconfig                sys_fsconfig
432     common  fsmount                 sys_fsmount
433     common  fspick                  sys_fspick
434     common  pidfd_open              sys_pidfd_open
435     common  clone3                  sys_clone3
436     common  close_range             sys_close_range
437     common  openat2                 sys_openat2
438     common  pidfd_getfd             sys_pidfd_getfd
439     common  faccessat2              sys_faccessat2
440     common  oslab                   sys_oslab
#
# x32-specific system call numbers start at 512 to avoid cache
```

نگاره ۵: فایل syscall_64.tbl و افزودن نام و شماره‌ی فراخوانی تازه به فهرست فراخوانی‌ها.

شماره‌ی فراخوانی تازه‌ی ما ۴۴۰ شده است. در ویرایش‌های دیگر هسته این شناسه می‌تواند چیز دیگری باشد.

۶. اکنون باید دستینه‌ی فراخوانی تازه را در «سربرگ فراخوانی‌های لینوکس» بگذاریم. این همان سربرگی است که در زبان سی بالای برنامه می‌افزاییم تا این فراخوانی در دسترس باشد. در دنباله‌ی دستورهای گام پیشین دستور زیر را می‌نویسیم.

```
$ nano include/linux/syscalls.h
```

در پایان این پرونده، و پیش از دستور #endif دستینه‌ی دستورکار خود را می‌افزاییم (نگاره ۶).

```
int __sys_getsockopt(int fd, int level, int optname, char __user *optval,
                int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                int optlen);

asmlinkage long sys_oslab(void);

#endif
```

نگاره ۶: فایل include/linux/syscalls.h و افزودن دستینه‌ی فراخوانی سیستمی به پایان آن.

در اینجا جایگذاری فراخوانی در میان پرونده‌های هسته پایان یافت. اکنون باید این هسته را مانند آنچه در «**دستورکار پیشین**» انجام شد «**می‌سازیم**».

این هسته را بالا بیاورید. فرآیندهای تراز کاربر می‌توانند از فراخوانی ()oslab بهره ببرند. این روند ساخت یک فراخوانی بود. ما همچنین می‌توانیم فراخوانی‌های کنونی هسته را نیز دستکاری کنیم. برای این کار تنها برنامه و کدها را دستکاری کنیم و نیازی به جایگذاری فایل‌ها نیست.

دسترسی به فراخوانی در تراز کاربر

پس از بالا آمدن هسته‌ی تازه، برنامه‌ی ساده‌ای می‌نویسیم تا از فراخوانی سیستمی oslab که در هسته گنجاندیم در آن بهره ببریم. ما برنامه‌ی ساده‌ی زیر (test_added_syscall.c) را به کار بردیم. به سربرگ‌هایی که در این برنامه خوانده‌ایم نگاه کنید. نام‌های آن‌ها آشنا نیست؟ همچنین در اینجا فراخوانی را با شماره‌ی آن، «۴۴۰»، فراخوانده‌ایم.

```
 GNU nano 4.8                 test_added_syscall.c                 Modifie
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

int main(){
        long int syscall_return_value = syscall(440);
        printf("The output of oslab syscall: %ld\n", syscall_return_value);
        return 0;
}
```

کامپایل و اجرای این برنامه خروجی زیر را می‌دهد.

```
The output of oslab syscall: 0
```

می‌بینید که مقدار بازگشتی فراخوانی، که «۰» است، نشان داده می‌شود. ولی، فراخوانی ما کار دیگری انجام می‌داد و آن هم نوشتن رشته‌ای در فایل log هسته بود (با دستور printk). برای اینکه ببینیم در فایل لاگ هسته چه رخ داده است باید دستور زیر را اجرا کنیم.

```
$ dmesg
```

که فایل را نمایش می‌دهد. می‌بینید که در پایان این فایل آنچه فراخوانی ما، با دو بار اجرا، نوشته است افزوده شده است.

```
dDevice= 1.00
[  646.323945] usb 2-1: New USB device strings: Mfr=1, Product=3, SerialNumber=
0
[  646.323948] usb 2-1: Product: USB Tablet
[  646.323950] usb 2-1: Manufacturer: VirtualBox
[  646.545507] input: VirtualBox USB Tablet as /devices/pci0000:00/0000:00:06.0
/usb2/2-1/2-1:1.0/0003:80EE:0021.0002/input/input7
[  646.606203] hid-generic 0003:80EE:0021.0002: input,hidraw0: USB HID v1.10 Mo
use [VirtualBox USB Tablet] on usb-0000:00:06.0-1/input0
[  649.603687] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control
: RX
[ 1245.770447]
              This is OSLAB's system call.
[ 1463.555994]
              This is OSLAB's system call.
```

اگر می‌خواهید درون این فایل رویدادنگاری پاک شود، می‌توانید از گزینه‌ی clear آن اینگونه بهره ببرید:

```
$ dmesg --clear
```

دستور کار

۱. با فراخوانی fork پیش‌تر کار کرده‌اید. این فراخوانی چه آرگومان‌های ورودی دریافت می‌کند و چه بازگشتی‌هایی دارد؟ اگر پس از انجام این دستور، فرایند فرزند ساخته نشد، این فراخوانی چه چیزی برمی‌گرداند؟

- در سیستم عامل ویندوز چه فراخوانی‌ای کار fork را انجام می‌دهد؟

۲. چندتا از فراخوانی‌های سیستمی خانواده‌ی exec را بررسی کنید.

۳. یک برنامه با چنین ساختاری داریم. به جای «؟» یک شماره گذاشته می‌شود و به برنامه‌ی فراخواننده چیزی برمی‌گرداند. چند نمونه از این شماره‌ها پیدا کنید و بگویید این‌ها چه هستند؟ هر کدام نشان از چه دارند؟

```
int main(){
        return ?;
}
```

۴. آنچه که در فایل Makefile برای فراخوانی سیستمی oslab.c نوشتیم را توضیح دهید. منظور از بخش‌های گوناگون آن چیست؟

۵. فراخوانی سیستمی‌ای «**با نام خودتان**» به هسته‌ی لینوکس بیافزایید که کارش نوشتن نام شما در فایل log (رویداد) هسته باشد. در لینوکس با این هسته‌ی کامپایل‌شده، برنامه‌ی ساده‌ای بنویسید که کاربرد این فراخوانی را نشان دهد (اگر نام شما sohrab rostami است نام فراخوانی شما باید sohrabrostami باشد.)

۶. برنامه‌ی زیر چه کاری انجام می‌دهد؟ این برنامه کار چه برنامه‌ی دیگری را شبیه‌سازی می‌کند؟

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

void main(){
        char cmd[100];
        pid_t fake_shell_pid, copied_shell_pid;
        int status;
        //fake_shell_pid = fork();
        while(1){
                printf("\noslabe@scu:$ Enter a Command : ");
                scanf("%s",cmd);
                //gets(cmd);
                printf("Original Shell PID: %d\n", getpid());
                copied_shell_pid = fork();
                if(copied_shell_pid < 0)
                        printf("Failed to run the command.\n");
                else if (copied_shell_pid > 0)
                        waitpid(-1, &status, 0);
                else{
                        printf("Copied Shell PID: %d\n", getpid());
                        char *args[]={cmd, NULL};
                        execvp(args[0],args);
                }
        }
}
```

برنامه ۱

۷. فراخوانی سیستمی که برای بازراه‌اندازی (reboot) سیستم‌عامل است را به‌گونه‌ای دست‌کاری کنید که: هر بار که سیستم‌عامل را بازراه‌اندازی (reboot یا restart) می‌کنیم نوشته‌ای ویژه مانند زیر در فایل لاگ (log) لینوکس نوشته شود. می‌بینید که در فایل رویداد زیر، دو خط دارای ستاره (*) و رشته‌های میان آن‌ها با هر بار بازراه‌اندازی لینوکس ما نوشته می‌شود.

```
[   16.504066] systemd[1]: Set hostname to <oslab-VirtualBox>.
[   16.641972] ********************************
[   16.641973] REBOOT IS DONE BY OSLAB VERSION!
[   16.641974] REBOOT IS DONE BY OSLAB VERSION!
[   16.641974] REBOOT IS DONE BY OSLAB VERSION!
[   16.641975] REBOOT IS DONE BY OSLAB VERSION!
[   16.641975] ********************************
```

(**راهنمایی**: فراخوانی سیستمی reboot را در فایل linux/kernel/reboot.c از پوشه‌ی هسته‌ی بارگیری‌شده پیدا کنید. برای دیدن لاگ (رویدادها) پس از کامپایل نیز از دستور dmesg کمک بگیرید.)