Repository Link: <https://github.com/SamanEN/Operating-System-Lab-Projects>

Latest Commit Hash: <hash>

# مقدمه

## بررسی استفاده از فراخوانی‌های سیستمی در کتابخانه‌ها (متغیر ULIB)

متغیر ULIB در Makefile متشکل از 4 آبجکت فایل به صورت زیر است:

ULIB = ulib.o usys.o printf.o umalloc.o

سورس هر کدام از فایل‌ها را بررسی می‌کنیم:

* ulib.c : این فایل شامل توابع کمکی:

strcpy, strcmp, strlen, memset, strchr, gets, stat, atoi, memmove

که در user.h دیکلر شده‌اند می‌باشد. از بین اینها، در دو تابع gets و stat از فراخوانی سیستمی استفاده شده است.

در تابع gets، از تابع سیستمی read استفاده شده است چون قرار است که یک خط از stdin را بخواند.

در تابع stat، از توابع سیستمی open, fstat, close استفاده شده است. به ترتیب، ابتدا با open فایلی باز می‌کند، سپس با fstat، metadata آن فایل (فیلدهای struct fstat مانند سایز فایل) را می‌گیرد. در نهایت با استفاده از close فایل را می‌بندد.

* usys.S : اینجا usys.o با استفاده از کد اسمبلی تولید می‌شود. در ابتدای این فایل یک ماکرو داریم:

#define SYSCALL(name) \  
 .globl name*; \*  
 name: \  
 movl $SYS\_ ## name, %eax*; \*  
 int $T\_SYSCALL*; \*  
 ret

سپس به ازای هر سیستم کال، این ماکرو با name آن استفاده می‌شود. مثلا برای SYSCALL(read) داریم:

.globl read*;*  
read:  
 movl $SYS\_read, %eax*; # SYS\_read (syscall.h) == 5*  
 int $T\_SYSCALL*; # T\_SYSCALL (traps.h) == 64*  
 ret

که همان اینستراکشن‌های لازم برای یک سیستم کال است.

برای فراخوانی این label ها در سطح سی، توابعی در user.h دیکلر شده‌اند.

ابتدا عدد سیستم کال به رجیستر EAX ریخته می‌شود و سپس INT 64 زده می‌شود. پس یک interrupt رخ داده و تابع trap صدا می‌شود و از آنجا که یک system call است به تابع syscall می‌رود. در آنجا مقدار EAX خوانده شده و می‌فهمد کدام سیستم کال را باید اجرا کند که اینجا تابع sys\_read است.

* printf.c : در این فایل تابع printf تعریف شده است که دیکلر آن در user.h است.

در این فایل دو تابع کمکی static به نام‌های putc و printint هم وجود دارد که printf و printint در نهایت putc را صدا می‌زنند.

تابع putc یک کاراکتر را با استفاده از تابع سیستمی write پرینت می‌کند.

* umalloc.c : در این فایل تابع malloc تعریف شده است که دیکلر آن در user.h است.

این تابع برای تخصیص حافظه استفاده می‌شود و در نهایت با استفاده از سیستم کال sbrk فضای پردازه را افزایش می‌دهد.

## انواع روش دسترسی سطح کاربر به هسته در لینوکس

دسترسی به سطح هسته با رخ دادن یک interrupt انجام می‌پذیرد. Interruptها می‌توانند نرم‌افزاری و یا سخت‌افزاری باشند که به interruptهای نرم‌افزاری trap نیز گفته می‌شود.

* Interruptهای سخت‌افزاری: اینگونه interruptها از طریق سخت‌افزارها (معمولا I/O) رخ می‌دهند و به صورت asynchronous هم اجرا می‌شوند. برای مثال زمانی که در کیبورد کلیدی را فشار می‌دهیم، موس را حرکت می‌دهیم و یا یک packet از طریق شبکه به ما می‌رسد، یک interrupt رخ می‌دهد.
* Interruptهای نرم‌افزاری (trap): این interrupt‌ها توسط یک برنامه و به صورت synchronous رخ می‌دهند. trapها انواع مختلفی دارند:
* System Call: فراخوانی‌های سیستمی که پیش‌تر در مورد آن‌ها صحبت شده است.
* Exception: استثناها نظیر تقسیم بر 0 و یا دسترسی بدون مجوز به حافظه.
* Signal: سیگنال‌ها در لینکوس انواع مختلفی دارند که پرکاربردترین آن‌ها عبارتند از SIGINT و SIGKILL و SIGTERM.

در لینکوس تعدادی Pseudo-File-System نظیر /proc، /dev و /sys وجود دارد که که یک اینترفیس برای ساختار داده‌های هسته در اختیار ما قرار می‌دهد. در نتیجه، استفاده از این فایل سیستم‌ها نیز، نیازمند دسترسی به هسته است.

# سازوکار اجرای فراخوانی سیستمی

# بخش سخت افزاری و اسمبلی

## آیا همه تله‌ها را می‌شود با سطح دسترسی DPL\_USER فعال نمود؟

## چرا در صورت تغییر سطح دسترسی، ss و esp روی پشته push می‌شوند؟

# بخش سطح بالا و کنترل‌کننده زبان سی تله

## توضیح توابع دسترسی به پارامترهای فراخوانی سیستمی

# بررسی گام‌های اجرای فراخوانی سیستمی در سطح کرنل توسط gdb

# find\_largest\_prime\_factor

# پیاده‌سازی فراخوانی‌های سیستمی

## فراخوانی سیستمی تغییر سایز فایل

## فراخوانی سیستمی لیست پردازه‌های فراخواننده

## فراخوانی سیستمی گرفتن پردازه پدر