باسمه تعالی

دانشگاه صنعتی اصفهان
سیستم های عامل ـ پروژه اول
موعد تحویل: جمعه ۹ آبان ۹۹

شرح پروژه

در این پروژه شما با نحوه عملکرد و توسعه یک فراخوانی سیستمی (SysCall) آشنا خواهید شد. روند انجام یک SysCall به سادگی فراخوانی یک پروسه ساده نخواهد بود! این کار نیاز به پیمودن یک مسیر ویژه دارد و از طریق یک پروتکل دقیق از جانب سیستم عامل و با هماهنگی سخت افزار صورت می‌گیرد تا حالت‌های برنامه در هر بار ورود و خروج به درستی ذخیره و بازیابی شوند. بدین جهت قصد داریم، به منظور آشنایی بیشتر با این روند، نحوه پیاده‌سازی SysCall را در سیستم عامل xv6 مورد بررسی قرار دهیم. بنابراین برای انجام این پروژه، به عنوان اولین گام، شما نیاز خواهید داشت تا شبیه ساز qemu و سیستم عامل xv6 را نصب و راه‌اندازی نمایید.

۱ نصب qemu

برای تغییر در کرنل دو راه وجود دارد. راه اول آن است که کرنل را پس از تغییر به عنوان کرنل جدید بر روی سیستم واقعی خود نصب کنید و سیستم را reboot نمایید تا با کرنل جدید بالا بیاید. راه دوم استفاده از شبیه ساز سیستم مانند qemu است.
راه اول برای debugging راه زمانبر و طاقت فرسایی است چون برای هر دفعه تغییر جدید باید سیستم خود را reboot کنید. در ضمن در صورت وجود bug کل سیستم شما قفل می شود!
به همین دلیل معمولا توسعه دهندگان کرنل از روش دوم استفاده می کنند. برای آشنایی بیشتر با شبیه سازهای سیستم می توانید به این لینک مراجعه کنید. برای نصب qemu دستور زیر را در shell اجرا کنید:

```
sudo apt-get install qemu-kvm
```

۲ نصب xv6 و راه اندازی آن بر روی qemu

xv6 یک سیستم عامل بسیار سبکی است که به منظور امور تحقیقاتی و آموزشی مورد استفاده قرار می‌گیرد. هسته xv6 بر مبنای یک نسخه اولیه از یونیکس در دانشگاه MIT توسعه داده شده است. برای نصب xv6 (که بسیار سریع و راحت خواهد بود!) کافی است مراحلی که در زیر آمده را به ترتیب اجرا نمایید.
نصب git و clone کردن source code مربوط به xv6:

```
sudo apt-get install git
git clone https://github.com/mit-pdos/xv6-public.git
```

کامپایل کردن xv6 و راه اندازی آن بر روی qemu:

```
cd xv6-public
make qemu-nox
```

پس از اجرای دستور فوق، طبق نسخه ای که در Makefile پیچیده شده ، برنامه `make` شروع به ساختن کرنل xv6 می نماید و در نهایت آنرا در محیط شبیه سازی qemu اجرا می کند. پس از صحبت های زیادی که make برای انجام این کار ها در ترمینال چاپ می کند 🙂 بالاخره به وضعیتی می رسد که در شکل زیر نمایش داده شده است:

```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
$
```

همانطور که ملاحظه می کنید، qemu با کرنل xv6 در همان محیط shell شروع به اجرا شدن می کند [1] . اکنون گویا شما در shell سیستم عامل xv6 قرار دارید و با اجرای مثلا دستور `ls` خروجی زیر را خواهید دید:

```
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2327
cat            2 3 13404
echo           2 4 12476
forktest       2 5 8192
grep           2 6 15224
init           2 7 13064
kill           2 8 12528
ln             2 9 12424
ls             2 10 14648
mkdir          2 11 12548
rm             2 12 12524
sh             2 13 23168
stressfs       2 14 13204
usertests      2 15 56076
wc             2 16 14056
zombie         2 17 12256
console        3 18 0
$
```

برای خارج شدن از qemu و بازگشت به shell سیستم خودتان می توانید کلیدهای ترکیبی زیر را استفاده کنید:

```
C-a x
```

که در آن منظور از `C-a` فشردن همزمان کلیدهای Ctrl و a صفحه کلید است.

۳ فراخوانی سیستمی

هر فراخوانی سیستمی یک انتقال محافظت شده کنترل از یک برنامه کاربر (که در حالت user اجرا می‌شود) به سیستم عامل (که در حالت kernel اجرا می‌شود) است. در واقع این همان شیوه مرسوم "اجرای مستقیم

[1] توجه کنید که به جای دستور آخر می توانید از دستور `make qemu` استفاده کنید. در این حالت برای اجرای qemu یک پنجره مجزا باز می شود.

محدود شده (LDE) " است که این امکان را به kernel می‌دهد تا ضمن حفظ کنترل خودش بر ماشین به برنامه‌های کاربر اجازه دهد تا به صورت موثر و بدون دخالت مداوم آن اجرا شوند. به این ترتیب نیاز خواهد بود تا هر زمان که یک فراخوانی سیستمی صدا زده می‌شود اتفاقات متعددی تحت کنترل سیستم عامل رخ دهد. بنابراین برای انجام این پروژه شما نیاز خواهید داشت تا اطلاع کافی از روند اجرای یک فراخوانی سیستمی در xv6 بدست آورید. برای آشنایی با نحوه پیاده سازی یک فراخوانی سیستمی موارد زیر را انجام دهید:

- روش مهندسی معکوس: شما می توانید یک فراخوانی سیستمی که از قبل در xv6 نوشته شده است را در کل پروژه جستجو کنید و نهایتا با الهام گیری از این بررسی متوجه شوید که برای اضافه شدن یک فراخوانی سیستم جدید می بایست چه کدهای را در چه فایلهایی اضافه کنید. برای مثال در xv6 یکی از فراخوان های سیستمی موجود `getpid` است. بنابراین با اجرای دستور زیر در فلدر xv6-public می توانید نام فایل ها و شماره خطی که در آنجا getpid نوشته شده را بیابید:

```
grep -nri getpid
```

- مشاهده این ویدیو و این صفحه که توسط پروفسور آرپادوسی (مولف کتاب اصلی درس) تهیه شده است.
- مطالعه دقیق کرنل: [۲] همانطور که قبلا اشاره شد، پروژه xv6 با هدف آموزشی ایجاد شده است. به همین دلیل این کرنل به خوبی توضیح داده شده است. برای این کار منابع زیر در دسترس هستند:
  ۱ ـ ابتدا در حالی که در فلدر xv6 هستید دستور زیر را در shell اجرا کنید:

```
make xv6.pdf
```

در این صورت یک فایل pdf تولید می شود که در آن تمام خط های کد در فایل های مختلف source code به صورت دو ستونه و شماره گذاری شده قرار داده شده است.
۲ ـ این کتاب که توسط مولفان xv6 نوشته شده به توضیح این سیستم عامل پرداخته و برای اینکه بتواند به صورت دقیق صحبت کند در حین توضیحاتی که می دهد محل دقیق کد مورد بحث را با شماره خط مربوطه آن در فایل xv6.pdf مشخص می کند.

۴ خواسته‌های پروژه

الف) (۱۰۰ نمره) روند اتفاقاتی که در اجرای یک SysCall (به صورت خاص در xv6) رخ می‌دهد را پیگیری و بیان نمایید. (می‌توانید این روند را به صورت یک دیاگرام، فلوچارت یا ... رسم نمایید)

ب) (۵۰۰ نمره) یک فراخوانی سیستمی ساده به سیستم عامل xv6 اضافه کنید که تعداد فراخوانی‌های سیستمی `read()` را که از زمان boot شدن تا زمان کنونی انجام شده است را برگرداند. این فراخوانی سیستمی را `getreadcount()` می‌نامیم.

فرمت فراخوانی سیستمی

فراخوانی سیستمی که شما طراحی می‌کنید، باید به فرمت زیر باشد:

```
int getreadcount(void)
```

[۲] به صورت اولیه اگر روشهای قبل را به خوبی انجام دهید نیازی به انجام این روش برای این پروژه نمی باشد. اما برای آشنایی دقیق تر با کرنل xv6 انجام این روش توصیه می شود.

در واقع فراخوانی سیستمی که شما طراحی کرده‌اید، مقدار یک شمارنده را (می‌تواند برای مثال readcount یا چیزی شبیه آن باشد) که هر بار یک پروسه، فراخوانی `read()` را انجام می‌دهد افزایش پیدا می‌کند، بر می‌گرداند.
راهنمایی ۱: همانطور که در قسمت قبل توضیح داده شد، یک روش موثر برای دست بردن در یک کد بزرگ این است که شما کار مشابه آن چیزی را که می‌خواهید انجام دهید در آن کد پیدا و با دقت از آن تقلید کنید!! در اینجا (در xv6) هم شما می‌توانید فراخوانی‌های دیگری، مثلا `getpid()` یا هر فراخوانی ساده دیگری را بیابید و از کدها و نحوه توسعه آن پیروی کنید. همه موارد مربوط به آن را به نحوی که ضروری می‌دانید کپی کنید و مواردی را که نیاز است را به نحو مناسب تغییر دهید تا عملکرد مورد نظر شما پیاده‌سازی شود.
راهنمایی ۲: در قسمت پیوست روشی برای ارزیابی صحت عملکرد فراخوانی سیستمی جدید که نوشته اید معرفی شده است.

ج) (۲۰۰ نمره) یک برنامه (به زبان C) بنویسید که از فراخوانی سیستمی که نوشته اید استفاده کند. نام برنامه را rdc بگذارید. این برنامه باید به صورت `./rdc` قابل اجرا توسط shell باشد (پس از اجرای xv6 در داخل qemu) و خروجی زیر را در ترمینال چاپ کند:

Hi, the number of read syscall is ? so far!

که به جای ? باید خروجی فراخوانی سیستمی `getreadcount` قرار بگیرد.
راهنمایی: مجدد می توانید از یکی از برنامه های کاربردی از قبل نوشته شده مانند `cat` به صورت مهندسی معکوس استفاده کنید.

شیوه تحویل

(۱۰۰ نمره) برای این تمرین می بایست یک فلدر به نام studentid_prj1 بسازید (به جای studentid باید شماره دانشجویی خود را قرار دهید) که شامل موارد زیر باشد:

۱. یک فایل pdf: شامل پاسخ به سوال الف (ترجیحا به زبان فارسی) که می بایست با استفاده از LaTeX ایجاد شده باشد.

۲. یک فلدر که همان xv6-public است که شما فراخوانی سیستمی جدید `getreadcount` و برنامه کاربردی `rdc` را به آن اضافه کرده اید. فراموش نکنید که بعد از اتمام کار یک بار با اجرای دستور زیر، در داخل این فلدر، فایل های غیر source code را پاک نمایید تا حجم فایل ارسالی بی جهت بزرگ نشود.

```
make clean
```

سپس فلدر خود را با دستور زیر بایگانی و فشرده سازی کنید.

```
tar zcf studentid_prj1.tgz studentid_prj1
```

و تنها فایل studentid_prj1.tgz را در سامانه یکتا در قسمت مربوط به پروژه اول بارگذاری کنید.

موفق باشید

پیوست: نحوه اطمینان از صحت عملکرد فراخوانی سیستمی

نحوه تست و اطمینان از صحت عملکرد فراخوانی سیستمی شما بسیار ساده و راحت خواهد بود. برای اینکار یک تست کننده طراحی شده است که اگر شما فرمت تعریف فراخوانی سیستمی را به درستی رعایت کرده باشید، آن تست کننده فرایند ارزیابی فراخوانی سیستمی شما را انجام خواهد داد. برای این کار شما ابتدا باید فایل‌های مربوط به تست کننده را بر روی سیستم خود و در پوشه مناسب دریافت کنید. این کار با انجام دستورات زیر امکان پذیر خواهد بود.

مرحله ۱:

```
git clone https://github.com/remzi-arpacidusseau/ostep-projects
cd ostep-projects/initial-xv6
```

در این فلدر، یک فلدر به نام src ایجاد کنید. فایل‌های سورس موجود در پوشه xv6-public (فایل‌های تغییر یافته بعد ازاضافه کردن فراخوانی سیستمی جدید که به آن اضافه کردید) را به این پوشه منتقل کنید.

مرحله ۲:

```
cd ostep-projects/initial-xv6
./test-getreadcount.sh
```

در این صورت می بایست پیام های انجام شدن تست ها با موفقیت را دریافت کنید.

مواجه با خطا

ممکن است در حین اجرای دستورات، خطاهایی دریافت کنید که ناشی از کمبود بعضی پیش‌نیازها در سیستم باشد. دو مورد از مهم‌ترین آن‌ها به صورت زیر قابل حل است:

```
sudo apt-get install gawk
sudo apt-get install expect
```