دانشگاه صنعتی اصفهان
دانشکده مهندسی برق و کامپیوتر

پروژه اول درس سیستم‌های عامل ۱

نیمسال تحصیلی پاییز ۱۴۰۰

دکتر محمّدرضا حیدرپور - دکتر زینب زالی

دستیاران آموزشی:
محمّد روغنی - مجید فرهادی - عرفان مظاهری - دانیال مهرآیین - محمّد نعیمی

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 می‌شود!
به همین دلیل معمولا توسعه دهندگان کرنل از روش دوم استفاده می‌کنند. برای آشنایی بیشتر با شبیه سازهای سیستم می‌توانید 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 سیستم‌عامل بسیار سبک 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 شده‌است. 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* می‌نماید و در نهایت آنرا در محیط شبیه‌سازی *qemu* اجرا می‌کند. پس از صحبت‌های زیادی که *make* برای انجام این کارها در ترمینال چاپ می‌کند 🙂 بالاخره به وضعیتی می‌رسد که در شکل زیر نمایش داده شده‌است:

```
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 می‌کنید، *qemu* با کرنل *xv6* در همان محیط *shell* شروع به اجرا شدن می‌کند. (توجه کنید که به جای دستور آخر می‌توانید از دستور `make qemu` استفاده‌کنید. در این حالت برای اجرای *qemu* یک پنجره مجزا باز می‌شود.) اکنون گویا شما در *shell* سیستم‌عامل *xv6* قرار دارید و مثلا با اجرای 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 می‌توانید 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 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 می‌توانید 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 می‌بایست 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 فولدر *xv6-public* می‌توانید نام فایل ها و شماره خطی که در آنجا *getpid* نوشته شده را بیابید:

```
grep -nri getpid
```

توجه کنید که قبل از اجرای دستور فوق، دستور زیر را اجرا کنید تا تنها فایل‌های *source code* در فولدر پروژه باقی بمانند.

```
make clean
```

- مشاهده این ویدیو و این صفحه که توسط پروفسور *Arpaci-Dusseau* (مولف کتاب *Operating Systems: Three Easy Pieces*) تهیه شده‌است.

- مطالعه دقیق کرنل: همانطور که قبلا اشاره شد، پروژه *xv6* با هدف آموزشی ایجاد شده است. به همین دلیل این کرنل به خوبی توضیح داده شده‌است. این کتاب که توسط مولفان *xv6* نوشته‌شده به توضیح این سیستم عامل پرداخته و برای اینکه بتواند به صورت دقیق صحبت کند در حین توضیحاتی که می‌دهد محل دقیق کد مورد بحث را با شماره خط مربوطه آن در فایل *xv6.pdf* مشخص می‌کند.

۴ خواسته‌های پروژه

الف) (۲۰ 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) (۱۰۰ نمره) در این پروژه قرار است برنامه *ps* را به همراه یک فراخوانی سیستمی ساده برای این برنامه، به سیستم عامل *xv6* اضافه کنید.

لینوکس برای مشاهده اطلاعات مربوط به فرآیندهای سیستم ابزاری به نام *ps* در اختیار ما قرار می‌دهد، که مخفف عبارت *"Process Status"* است. از دستور *ps* برای دریافت لیست فرآیندهای در حال اجرا استفاده می‌شود که *PID* های آنها و با توجه به گزینه‌های مختلف، برخی اطلاعات دیگر را نمایش می‌دهد. در مثال زیر این برنامه برای هر فرآیند، *PID* فرآیند والد و بعضی اطلاعات دیگر را نمایش می‌دهد.

```
mohammad@mohammad-pc:~$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   47689   38001  0  80   0 -  5691 do_wai pts/2    00:00:00 bash
4 R  1000   47728   47689  0  80   0 -  5971 -      pts/2    00:00:00 ps
```

در فراخوانی سیستمی که شما طراحی می‌کنید، باید مانند شکل زیر با دسترسی به جدول فرآیندها، که شامل *PCB* فرآیندها می‌باشد، نام هر فرایند را به همراه *state* فرآیند، *PID* ، *PID* والد و *PID* های همزادهایش استخراج و در ترمینال چاپ نماید. این فراخوانی سیستمی را `pssyscall()` می‌نامیم. درواقع برنامه *ps* که شما طراحی می‌کنید، تنها فراخوانی سیستمی *pssyscall* را فراخوانی می‌کند و پایان می‌یابد.

```
SeaBIOS (version 1.14.0-2)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4C0+1FECB4C0 CA00



Booting from Hard Disk..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
$ ps
 name    pid     state          parent         sibling
 init    1       SLEEPING       623            NS
 sh      2       SLEEPING       1              NS
 ps      3       RUNNING        2              NS
$ QEMU: Terminated
```

این فراخوانی سیستمی را به فرمت زیر پیاده‌سازی کنید و در آن باید به جدول فرآیندها دسترسی پیدا کنید.

```
int pssyscall (void)
```

با مطالعه ساختمان داده پیاده‌سازی شده برای *PCB*، می‌توانید اطلاعات موردنظر را استخراج نمایید. همچنین مطالعه بدنه پیاده‌سازی شده برای تابع `procdump` در فایل *proc.c* می‌تواند شما را به خوبی راهنمایی کند. به دلیل اینکه مباحث همزمانی در این پروژه موردنظر نمی‌باشد، در *Makefile*، مقدار *CPUS* را مانند شکل زیر، برابر با *1* قراردهید تا شبیه‌ساز *qemu*، محیط را با یک *CPU* شبیه‌سازی کند. [۱]

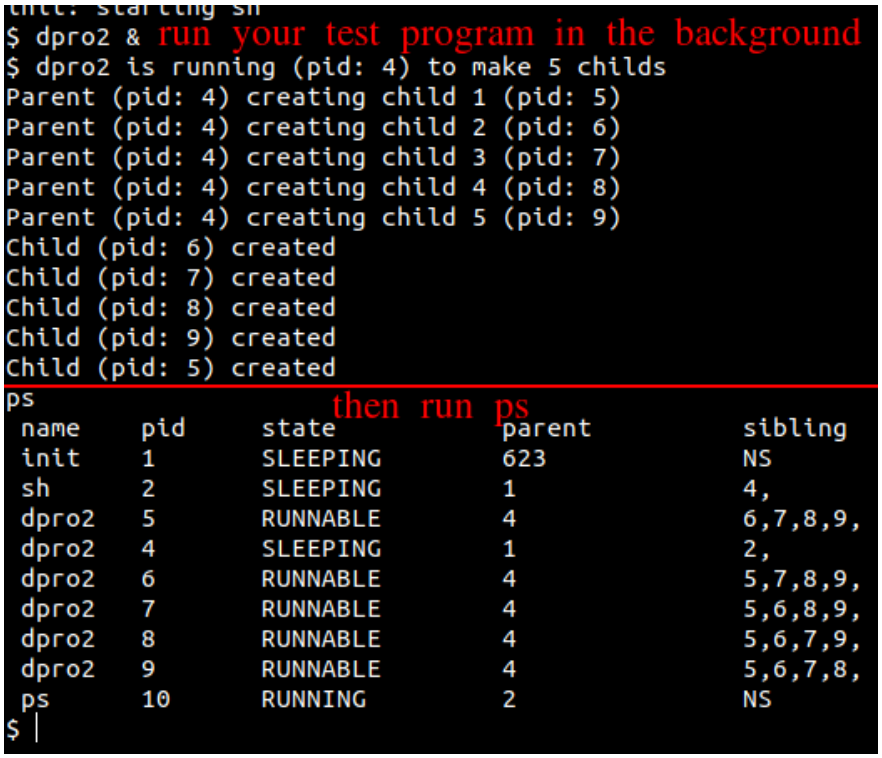
```
ifndef CPUS
CPUS := 1
endif
```

راهنمایی: 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.

ج) (۴۰ نمره) برنامه *ps* را پیاده‌سازی نمایید (به زبان *C*) که از فراخوانی سیستمی که پیاده‌سازی نموده‌اید، استفاده کند. نام برنامه را *ps* بگذارید. این برنامه باید پس از اجرای *xv6* در داخل *qemu* به صورت `./ps` یا `ps` قابل اجرا توسط *shell* باشد و خروجی مناسب را در ترمینال چاپ کند.
راهنمایی: مجدد می‌توانید از یکی از برنامه های کاربردی از قبل نوشته شده مانند `cat` به صورت مهندسی معکوس استفاده کنید. در اینجا، مانند دیگر برنامه‌های کاربردی، برنامه خود را به زبان *C* در دایرکتوری پروژه پیاده‌سازی کنید و در *Makefile* آن را برای شبیه‌ساز معرفی کنید.

د) (۳۰ نمره) یک برنامه ساده برای تست فراخوانی سیستمی خود پیاده‌سازی نمایید.
با اجرای این برنامه، ۵ عدد فرآیند فرزند تولید می‌کند و فرآیند والد منتظر می‌ماند تا با پایان اجرای همه فرزندان، پایان یابد. هرکدام از فرزندان باید در یک حلقه طولانی مدت اجرا شوند و محاسباتی را انجام دهند تا هنگام اجرای برنامه *ps*، اجرای آن‌ها ادامه داشته‌باشد. برای اجرای برنامه تست از *&* در انتهای دستور اجرای برنامه استفاده کنید تا برنامه در پس‌زمینه (*Background*) اجرا شود و شل *xv6* برای اجرای برنامه *ps* در اختیار ما قرار بگیرد. سپس برنامه *ps* را اجرا نمایید. به مثال صفحه بعد توجه کنید.

[۱] همچنین می‌توانید قبل و بعد از دسترسی به جدول فرآیندها، با استفاده از دستورات `cli()` و `sti()` وقفه‌ها را غیرفعال و سپس فعال نمایید.



نحوه تحویل (۱۰ نمره) :

برای این تمرین می‌بایست یک فولدر 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)

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

```
make clean
```

سپس فولدر خود را با دستور زیر بایگانی و فشرده‌سازی 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