الهی به امید تو
آزمایشگاه سیستم عامل - پروژه ۱

غزل نیسی مینایی ۸۱۰۱۹۶۶۸۳
سارا توکلی۸۱۰۱۹۶۶۸۴
زهرا موسوی موحد ۸۱۰۱۹۶۶۲۹

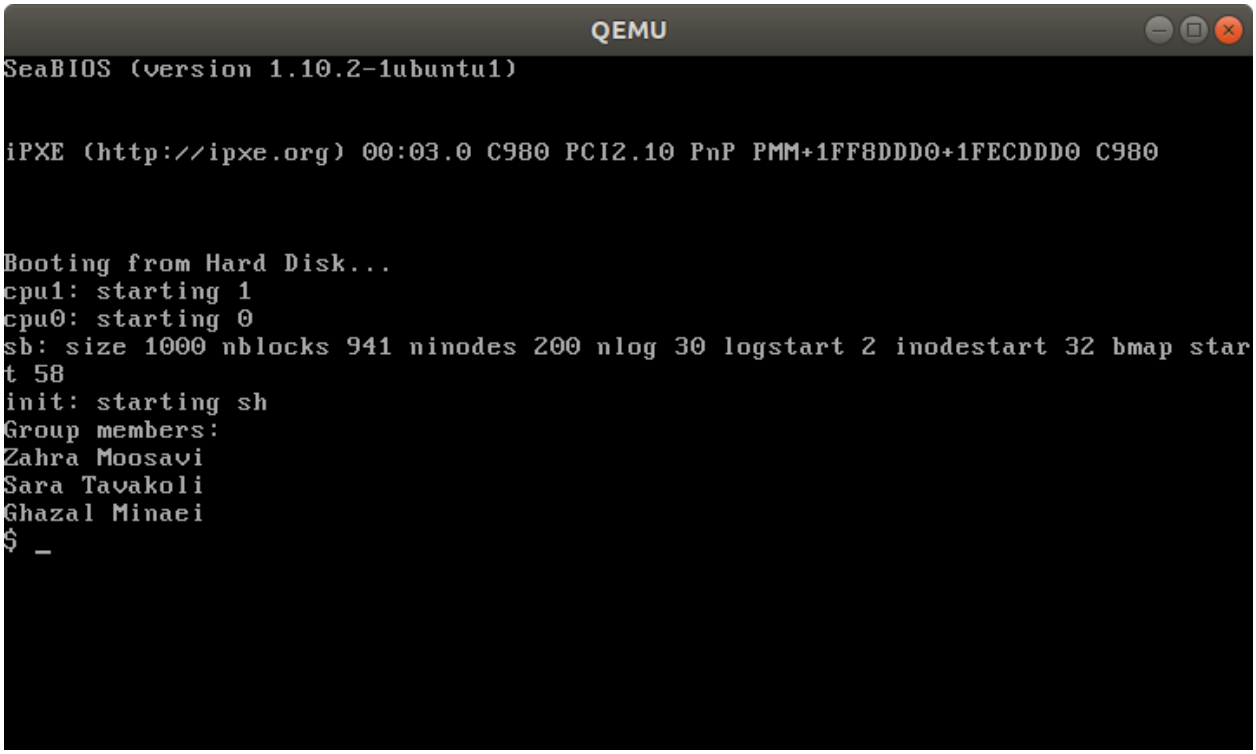
بخش اول:

آشنایی با سیستم عامل xv6

- سوال ۱: کرنل آن به صورت monolithic (یکپارچه) پیاده سازی شده است. در این سیستم عامل تمام سرویس ها در فضای هسته پیاده سازی شده است. بنابراین بسیار سبک است و این نوع طراحی باعث بالا رفتن سرعت و بازدهی سیستم میشود. Xv6 روی پردازنده ی Intel 80386 اجرا میشود.
- سوال۲: در سیستم عامل xv6، هر پردازه شامل یک حافظه در فضای کاربر (user-space) است (که شامل دستورالعمل ها (instructions)، داده (data) و پشته (stack) است) و وضعیت هر پردازه است که فقط توسط هسته قابل دسترسی است. وقتی یک پردازه میخواهد از خدمات هسته استفاده کند، با استفاده از system call در interface این کار را انجام میدهد. system call وارد هسته می شود و خدمات توسط هسته انجام میشود. بنابراین وقتی که cpu در حال انجام کاری نباشد و پردازه ای در حال اجرا نباشد، cpu به پردازش پردازه دیگری می پردازد.
- سوال۳: با فراخوانی سیستمی fork، یک پردازه ی فرزند ایجاد می شود (که حافظه ی آن کاملا شبیه حافظه پردازه فراخواننده است) و میتوان ادامه ی اجرای بخشی از دستورات را به آن داد. اما در فراخوانی سیستمی exec (حافظه پردازه فراخواننده را با فرزندش جایگزین میکند)، تعدادی آرگومان ورودی و یک مسیر فایل به عنوان ورودی میگیرد و پردازه ای که در حال اجرا است را متوقف میکند و کد موجود در مسیر فایل ورودی را، با آرگومان های ورودی اجرا میکند. در fork، قسمتی از دستورات به فرزند داده میشود اما با وجود exec، پردازه فرزند کار متفاوتی نسبت به فراخواننده اش انجام میدهد. به طور مثال هنگام ورود دستور، به کمک fork یک پردازش جدید ایجاد میکنیم و با exec این فرآیند جدید را با برنامه ای بر روی سیستم جایگزین میکنیم.
- سوال۴: file descriptor عددی صحیح و نامنفی است (که نماینده ی یک شئ تحت مدیریت هسته است) برای مجرد سازی فایل، که پردازه ها می توانند از آن بخوانند یا در آن بنویسند. File descriptor با interface ای که ارائه میدهد، تفاوت بین file و pipe و device را abstract میکند و به همین دلیل سیستم عامل با آن ها به عنوان جریانی از بایت ها برخورد میکند. Pipe در واقع یک جفت از file descriptor هاست که از یکی از آن ها برای خواندن و از دیگری برای نوشتن استفاده میشود. بدین صورت که نوشتن داده در انتهای لوله، آن داده را برای خواندن از سر دیگر لوله آماده میکند. پس pipe ارتباط بین پردازه ها را فراهم میکند و از آن ها برای اتصال دو پردازه استفاده میکنیم. (با نوشتن داده در یک پردازه، داده ی آماده در پردازه ی دیگر قابل خواندن است.)

اجرای و اشکال زدایی

اضافه کردن یک متن به Boot Message



بخش دوم:

- ۲:

#basic headers:

تعاریف و مقدارهای اولیه مثل انواع تایپ ها، پارامترهایی مثل حداکثر تعداد پردازنده ها، سایز استک، آدرس های مربوط به مموری، اعلان توابع و ساختارهای اصلی، روتین هایی که به زبان c اجازه انجام دستورات x86 را میدهد و سایر تعاریف و اعلان های پایه ای.

#entering xv6

فایل های آغازین سیستم عامل هستند که هسته کار خود را از آنجا آغاز میکند. مقدار دهی به رجیسترهای segment تغییر از real mode به protected mode و وارد شدن به تابع main از جمله کارهایی است که در این بخش صورت میگیرد.

#locks

تعریف lock و استفاده از آن ها برای مدیریت، ایجاد کردن و آزاد کردن قفل ها و هر فرایندی که به آن مربوط میشود.

#processes

مدیریت process ها از جمله ذخیره مقادیر رجیسترها در زمان context switch، مدیریت آدرس های مجازی در این بخش صورت میگیرد.

#system calls

تعریف trap ها، تعریف انواع interrupt ها، رسیدگی به trap ها و interrupt ها و تعریف و مدیریت انواع system call در این بخش صورت میگیرد.

#file system

مدیریت هر چیزی که به عنوان file شناخته میشود(دایرکتوری، فایل و دیوایس) ، تعریف انواع و ساختار ها، خواندن و نوشتن در آنها، چک کردن آرگومان هایی که از سمت برنامه های سطح کاربر پاس داده شده است.

#pipes

تعریف، مدیریت و استفاده از pipe ها برای مرتبط کردن فرایندها در دو سمت pipe با یکدیگر.

#string operations

توابع پایه ای برای استفاده راحت تر از string ها.

#low-level hardware

تعاریف و ساختارهای مربوط به سخت افزارهای پایه ای، پشتیبانی از چند هسته ای بودن، مدیریت interrupt های I/O، تعریف کیبورد و کنسول.

#user-level

آماده کردن یک interface برای برنامه های سطح کاربر و کاربر که بتواند با kernel ارتباط برقرار کند. (از طریق shell)

#bootloader

انجام کارهای اولیه برای boot و اجرای bootmain و بالا آوردن kernel image از اولین بخش حافظه.

#link

یک اسکریپت linker

- ۳:

در اینجا با لینک شدن object file ها فایل kernel ساخته میشود:

```
ld -m    elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o
ioapic.o kalloc.o kbd.o lapic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o
spinlock.o string.o swtch.o syscall.o sysfile.o sysproc.o trapasm.o trap.o uart.o vectors.o
vm.o  -b binary initcode entryother
```

objdump -S kernel > kernel.asm

objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym

dd if=/dev/zero of=xv6.img count=10000

dd if=bootblock of=xv6.img conv=notrunc

dd if=kernel of=xv6.img seek=1 conv=notrunc

در خط آخر فایل نهایی یعنی xv6.img با کپی شدن فایل if به فایل of تشکیل میشود.

- ۶:

ld -m elf_i386 -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o

دستور ld وظیفه لینک کردن object file ها رو برعهده دارد. با توجه به option های به کار رفته در ادامه دستور، محتوای لینک شده این سه object file یعنی **bootblock.o bootasm.o bootmain.o** را در حافظه **0x7C00** قرار میدهد.

- ۸: bootstrap به دنبال یک فایل اسمبلی برای ادامه کار و بوت شدن سیستم عامل میگردد. هم چنین انجام کارها و مقدمات سخت افزاری باید به زبان اسمبلی صورت گیرد.
- ۹:

ثبات عام منظوره: رجیستر AX :

Accumulator register:

برای عملیات ریاضی استفاده می شود.

ثبات قطعه: رجیستر CS :

Code segment:

اشاره گر به کد.

ثبات وضعیت: رجیستر ZF:

Zero flag:

اگر نتیجه عملیات صفر باشد ست میشود.

ثبات کنترلی: رجیستر PE :

Protected Mode Enables:

اگر سیستم در حالت protected باشد 1 میشود و اگر در حالت real mode باشد 0 میشود.

- ۱۱:

در این مد ۸ رجیستر ۱۶ بیتی چند مقصوده و ۴ segment رجیستر به نام های %cs برای instruction fetches و %ds برای نوشتن و خواندن داده ها و %ss برای نوشتن و خواندن از استک و %es داریم. و برای آدرس دهی به مموری از آن ها استفاده میکنیم . به این صورت که ۱۶ بیت segment را چهار بیت به چپ شیفت میدهیم(۱۶ برابر میکنیم) و بعد با offset جمع می کنیم تا آدرس ۲۰ بیتی به دست بیاید . در این مد هر سه آدرس منطقی و مجازی و فیزیکی با هم برابرند.

- ۱۳:

آخرین مرحله ی boot loader ورود به هسته است که دستور مربوط به آن در entry.s نوشته شده است و معادل آن در linux فایل head64.c است.

- ۱۴:

زیرا این آدرس آدرس جدولی است که از آن برای نگاشت آدرس ها به آدرس فیزیکی استفاده میشود. پس تا پیش از این جدول چیزی وجود ندارد که این نگاشت ها را انجام دهد. پس آدرسی که استفاده میشود باید خودش فیزیکی باشد.

- ۱۵:

تابع معادل آن در سیستم عامل لینوکس start_kernel است.

توابعی که در main صدا زده میشوند به ترتیب زیرند :

- kinit1(end, P2V(4*1024*1024));

برای آزاد کردن صفحه های فیزیکی حافظه به کار میرود.

- kvmalloc();

برای ساختن یک صفحه ی جدید به کار میرود و آدرس آن را برمیگرداند.

- mpinit();

این تابع سایر پردازنده ها را تشخیص میدهد.

- lapicinit();

این تابع interrupt ها را کنترل میکند.

- seginit();

این تابع segmentation table را سازمان دهی می کند.

- picinit();

این تابع programmable interrupt table را disable می کند.

- ioapicinit();

این تابع interrupt ها را کنترل می کند.

- consoleinit();

این تابع از نظر سخت افزاری کنسول را مدیریت می کند.

- uartinit();

این تابع serial port سخت افزار اینتل را مدیریت می کند.

- pinit();

این تابع process هارا مدیریت می کند.

- tvinit();

این تابع vector ی از trap هارا می سازد.

- binit();

برای آزاد کردن صفحه های فیزیکی حافظه به کار میرود.

- fileinit();

این تابع جدول مربوط به مدیریت فایل ها را می سازد.

- ideinit();

این تابع برای مقداردهی و شناسایی disk ها به کار می رود.

- startothers();

این تابع کار بقیه ی پردازنده هارا شروع می کند.

- kinit2(P2V(4*1024*1024), P2V(PHYSTOP));

این تابع برای آماده سازی صفحه ها برای دیگر پردازنده ها به کار می رود.

- userinit();

این تابع اولین process را ایجاد می کند.

- mpmain();

این تابع برای پایان اولین processor به کار می رود.

- ۱۸:

ساختار معادل آن در سیستم عامل لینوکس struct task_struct است.

بخش های مختلف آن عبارتند از :

- uint sz;

حافظه ی مورد نیاز برای process را به واحد بایت بیان میکند.

- pde_t* pgdir;

آدرس جدول مربوط به هر برنامه سطح کاربر را نگه داری میکند.

- **char** *kstack;

اشاره گری است به پایین ترین قسمت استک کرنل برای این process.

- **enum** procstate state;

وضعیت process را در هر لحظه بیان میکند .

- **int** pid;

برای هر process این عدد را به عنوان id نگهداری میکند.

- **struct** proc *parent;

اشاره گری است به process ی که پدر این process است.

- **struct** trapframe *tf;

برای دسترسی سیستمی چارچوب frame را نگه می دارد.

- **struct** context *context;

این ساختار context را در مواقعی که باید context switching انجام دهیم نگه می دارد.

- **void** *chan;

در صورت یک بودن نشان دهنده ی خواب بودن process است.

- **int** killed;

در صورت یک بودن نشان دهنده ی مردن process است.

- **struct** file *ofile[NOFILE];

این ساختار فایل هایی که برای process باز شده اند را نگه می دارد.

- **struct** inode *cwd;

دایرکتوری فعلی را نگه می دارد.

- **char** name[16];

اسم process را نگه داری میکند و در مواقع دیباگ مورد استفاده قرار می گیرد.

بخش سوم:

اجرای اولیه اشکال زدا

:۱

```
ghazal@ghazal-X510UQR: ~/Documents/5th Semester/OS/LAB1-FINAL/xv6-public-master(1)...
File  Edit  View  Search  Terminal  Help
line to your configuration file "/home/ghazal/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ghazal/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
---Type <return> to continue, or q <return> to quit---
        info "(gdb)Auto-loading safe path"
(gdb) target remote tcp::26000
Remote debugging using tcp::26000
0x0000fff0 in ?? ()
(gdb) b usys.S:15
Breakpoint 1 at 0x37a: file usys.S, line 15.
(gdb) c
Continuing.
[Switching to Thread 2]

Thread 2 hit Breakpoint 1, read () at usys.S:15
15      SYSCALL(read)
(gdb) bt
#0  read () at usys.S:15
#1  0x000000ca in cat (fd=3) at cat.c:12
#2  0x00000054 in main (argc=<optimized out>, argv=<optimized out>) at cat.c:39
(gdb)
```

Backtrace خلاصه ای از این است که چگونه برنامه به این نقطه رسیده است.
در هر خط از خروجی این دستور شماره frame، نام تابع، مقدار PC، اسم فایل منبع، شماره خط و آرگومان های ورودی تابع را نشان میدهد.

۲:سطح هسته:

```
ghazal@ghazal-X510UQR: ~/Documents/5th Semester/OS/xv6/xv6-public-master
File  Edit  View  Search  Terminal  Help
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ghazal/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
---Type <return> to continue, or q <return> to quit---
        info "(gdb)Auto-loading safe path"
(gdb) target remote tcp::26000
Remote debugging using tcp::26000
0x0000fff0 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
0x80104b43 in sys_read () at sysfile.c:71
71      {
(gdb) bt
#0  0x80104b43 in sys_read () at sysfile.c:71
#1  0x801048a9 in syscall () at syscall.c:139
#2  0x80105805 in trap (tf=0x8dffefb4) at trap.c:43
#3  0x8010561f in alltraps () at trapasm.S:20
#4  0x8dffefb4 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb)
```

سطح کاربر:

```
ghazal@ghazal-X510UQR: ~/Documents/5th Semester/OS/xv6/xv6-public-master
File  Edit  View  Search  Terminal  Help
init" auto-loading has been declined by your `auto-load safe-path' set to "$debu
gdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /home/ghazal/Documents/5th Semester/OS/xv6/xv6-p
ublic-master/.gdbinit
line to your configuration file "/home/ghazal/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ghazal/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
---Type <return> to continue, or q <return> to quit---
        info "(gdb)Auto-loading safe path"
(gdb) target remote tcp::26000
Remote debugging using tcp::26000
0x0000fff0 in ?? ()
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
0x801043b0 in ?? ()
(gdb) bt
#0  0x801043b0 in ?? ()
(gdb)
```

:۳
با

/x

و برای اینکه همیشه در همین مبنا باشد:

Set output-radix 16

آشنایی با قابلیت های سطح پایین تر

:۴
برای n از ni
برای s از si

:۵

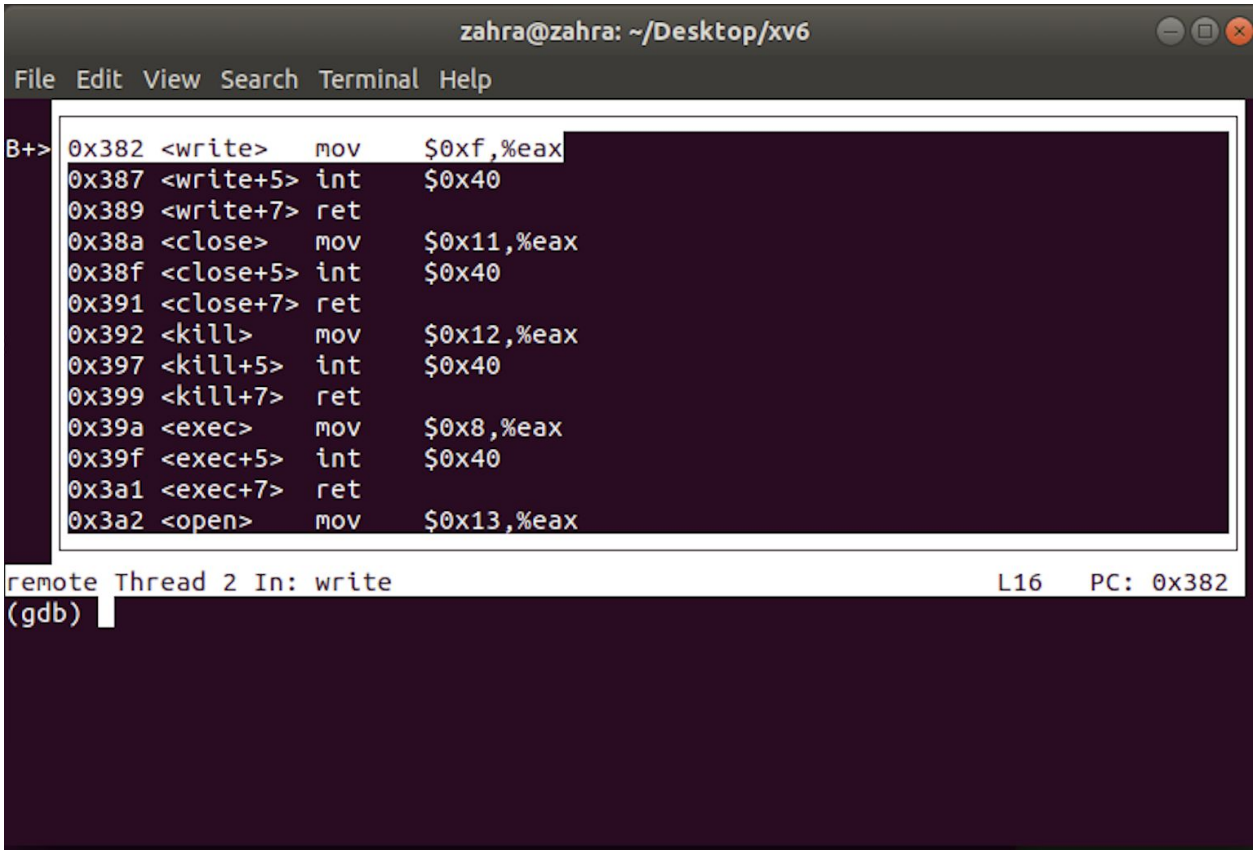
Backtrace خلاصه ای از این است که چگونه برنامه به این نقطه رسیده است.
در هر خط از خروجی این دستور شماره frame، نام تابع، مقدار PC، اسم فایل منبع، شماره خط و آرگومان های ورودی تابع را نشان میدهد.

```
ghazal@ghazal-X510UQR: ~/Documents/5th Semester/OS/xv6/xv6-public-master
File  Edit  View  Search  Terminal  Help
line to your configuration file "/home/ghazal/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
---Type <return> to continue, or q <return> to quit---
        info "(gdb)Auto-loading safe path"
(gdb) target remote tcp::26000
Remote debugging using tcp::26000
mycpu () at proc.c:48
48        for (i = 0; i < ncpu; ++i) {
(gdb) b sysfile.c:82
Breakpoint 1 at 0x80104bb0: file sysfile.c, line 82.
(gdb) c
Continuing.

Thread 2 hit Breakpoint 1, sys_write () at sysfile.c:83
83      {
(gdb) bt
#0  sys_write () at sysfile.c:83
#1  0x801048a9 in syscall () at syscall.c:139
#2  0x80105865 in trap (tf=0x8dfbefb4) at trap.c:43
#3  0x8010567f in alltraps () at trapasm.S:20
#4  0x8dfbefb4 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb)
```

۶:

دستور layout src قسمتی از برنامه که در حال اجراست را خط به خط نشان میدهد . و layout asm همین کار را به زبان assembly انجام میدهد.



دستور int به معنای رخ دادن interrupt و دستور ret به معنای return کردن می باشد.