

به نام خدا

پروژه اول آزمایشگاه سیستم عامل

گروه ۳: علی هدائی (۸۱۰۱۹۹۵۱۳)، پویا صادقی (۸۱۰۱۹۹۴۴۷)، علی عطااللهی (۸۱۰۱۹۹۴۶۱)

۱- معماری سیستم عامل xv6 چیست؟ چه دلایلی در دفاع از نظر خود دارید؟

سیستم عامل xv6 بر اساس unix version 6 (v6) ساخته دنیس ریچی و کن تامپسون ساخته شده است. پس از ساختار و استایل v6 بهره میبرد و اما با ANCI C برای پردازنده های چند هسته ای مبتنی بر x86 پیاده سازی شده است.

در فایل x86.h ذکر شده است که دستور های نوشته شده از ساختار و دستورهای مربوط به پردازنده های مبتنی بر x86 استفاده شده است. در فایل asm.h هم ذکر شده است که از ماکرو های اسمبلر برای ساختن بخش های x86 استفاده شده است. همچنین در traps.h هم ذکر شده که از trap های x86 استفاده شده است. همچنین در mmu.h هم نوشته شده که از mmu مربوط به x86 استفاده شده است.

۲- یک پردازنده در سیستم عامل xv6 از چه بخش هایی تشکیل شده است؟ این سیستم عامل به صورت کلی چگونه پردازنده را به پردازنده های مختلف اختصاص میدهد؟

هر پردازنده (process) در xv6 از user space memory که شامل stack و data و instruction و همچنین وضعیت چیش پردازش (pre-process state) که مربوط به هسته (kernel) است، تشکیل شده است. xv6 میتونه پردازنده های time-share انجام بده. آن بین cpu های موجود برای مجموعه ای از پردازش ها منتظر اجرا هستن سوییچ میکند. زمانی که پردازنده ای اجرا نمیشود، xv6 رجیستر های cpu آن را ذخیره میکند تا برای پردازش دفعه بعدی استفاده کند. کرنل به هر پردازنده یک آیدی یا همان pid نسبت میدهد.

۴- فراخوانی های سیستمی fork و exec چه عملی انجام میدهند؟ از نظر طراحی ادغام نکردن این دو چه مزیتی دارد؟

fork: پردازشی که ممکن است به پردازش دیگه تولید کند از فراخوانی سیستمی fork استفاده میکند. Fork به پردازش جدیدی به نام پردازش child میسازد، دقیق با محتوای حافظه که آن پردازش را فراخوانی کرده است، که پردازش parent نامیده میشود. Fork هم در child و هم در parent مقداری برمیگرداند. در parent، pid ی child را برمیگرداند و در child، ۰ را برمیگرداند.

Exec: فراخوانی سیستمی exec، مموری پردازشی که فراخوانی شده است را با یک مموری ایمیچی که در یک فایل در سیستم ذخیره شده است جایگزین میکند. آن فایل باید فورمت مشخصی داشته باشد که مشخص شده باشد که در کدام قسمت فایل دستورات ذخیره شده، کدام قسمت از آن داده است و از کدام دستور باید شروع کند. xv6 از فورمت ELF

استفاده میکند. زمانی که exec با موفقیت به پایان میرسد، به برنامه ای که آن را فراخوانده است باز نمیگردد بلکه دستورات ای که از فایل لود شده اند، از نقطه مشخص شده شروع به اجرا میشوند. exec دو آرگومان میگیرد یکی اسم فایلی که قرار است اجرا شود و دیگری آرایه ای استرینگ از آرگومان ها.

جدا کردن فراخوانی ها برای ساختن یک پردازش و لود کردن پردازش استفاده های هوشمندانه ای در IO redirection دارد. برای جلوگیری از پردازش های تکراری و سریع جایگزین کردن آن، کرنل ها با استفاده از تکنیک های مموری مجازی، پیاده سازی fork را برای این زمینه بهبود میبخشند.

۸- در Makefie متغیرهایی به نام UPROGS و ULIB تعریف شده است. کاربرد آنها چیست؟

UPROGS : شامل لیستی از user program هایی که باید بیلد شوند و یوزر میتواند از آنها استفاده کند را نگه میدارد. به همین دلیل اگر بخواهیم user program ای بسازیم و اضافه کنیم باید در این قسمت نامش را ذکر کنیم.

برای مثال برنامه یا دستور هایی که کاربر میتواند اجرا کند شامل
_cat\ _echo\ _forktest\ _grep\ _init\ _kill\ _ln\ _ls\ _mkdir\ _rm\ _sh\ _stressfs\
_usertests\ _wc\ _zombie\

که میبینیم نامشان همگی در UPROG لیست شده اند

ULIB : برای اینکه کاربران بتوانند برنامه هایی توسعه دهند یکسری دستورات و لایبرری های برای آنها ساخته شده است که این متغیرمسئول نگه داشتن آنهاست مثل فایل آجکت ulib.o که دستور های strcpy و memset و ... را دارد. فایل آجکت usys.o که سیستم کال ها را شامل میشود، umalloc.o که شمال دستور های dynamic memory allocation هاست و printf.o .

۱۱- برنامه های کامپایل شده در قالب فایل های دودویی نگه داری می شوند. فایل مربوط به بوت نیز دودویی است. نوع این فایل دودویی چیست؟ تفاوت این نوع فایل دودویی با دیگر فایل های دودویی کد xv6 چیست؟ چرا از این نوع فایل دودویی استفاده شده است؟ این فایل را به زبان قابل فهم انسان (اسمبلی) تبدیل نمایید.

همه آجکت فایل ها ELF اند. (هدر ELF دارند). ولی فایل bootblock این هدر را ندارد و در واقع یک فایل raw binary و هدری ندارد. و این فایل در سکتور اول مموری لود میشود (سکتور اول مربوط به بوت است و کرنل از سکتور دوم به بعد است). از آنجا که وقتی سیستم بالا می آید cpu از سکتور اول که مربوط به بوت است شروع به اجرای دستورات میکند و قاعدتا چون هنوز سیستم عامل اجرا نشده است نمیتواند فایل ELF را متوجه شود (این سیستم عامل است که ELF را میشناسد) به همین دلیل فقط قسمت تکست bootblock.o که مربوط به instruction ها است را جدا میشود و در حافظه قرار میگیرد تا cpu بتواند آنرا اجرا کند. همچنین این کد به معماری cpu وابسته است. به همین دلیل باید برای human readable کردن آن باید معماری آنرا که i386 است و ۱۶ بیت است ذکر کنیم.

```

amd64      Display instruction in AMD64 ISA
intel64     Display instruction in Intel64 ISA
ali@ali-ASUS-TUF-Gaming-A17-FA706II-FX706II:~/Desktop/OS/project1/os-project1/codes/xv6$ objdump -D -b binary -m386 -Maddr16,data16 bootblock

bootblock:      file format binary

Disassembly of section .data:

00000000 <.data>:
 0:  fa                cli
 1:  31 c0             xor    %ax,%ax
 3:  8e d8             mov    %ax,%ds
 5:  8e c0             mov    %ax,%es
 7:  8e d0             mov    %ax,%ss
 9:  e4 04             in     $0x64,%al
 b:  a8 02             test   $0x2,%al
 d:  75 fa             jne    0x9
 f:  b0 d1             mov    $0xd1,%al
11:  e6 04             out    %al,$0x64
13:  e4 04             in     $0x64,%al
15:  a8 02             test   $0x2,%al
17:  75 fa             jne    0x13
19:  b0 df             mov    $0xdf,%al
1b:  e6 60             out    %al,$0x60
1d:  0f 01 16 78 7c    lgdtw  0x7c78
22:  0f 20 c0          mov    %cr0,%eax
25:  66 83 c8 01       or     $0x1,%eax
29:  0f 22 c0          mov    %eax,%cr0
2c:  ea 31 7c 00 00    ljmp   $0x8,$0x7c31
31:  66 b8 10 00 8e d8  mov    $0xd88e0010,%eax
37:  8e c0             mov    %ax,%es
39:  8e d0             mov    %ax,%ss
3b:  66 b8 00 00 8e e0  mov    $0xe08e0000,%eax
41:  8e e8             mov    %ax,%gs
43:  bc 00 7c          mov    $0x7c00,%sp
46:  00 00             add    %al,(%bx,%si)
48:  e8 fc 00          call   0x147
4b:  00 00             add    %al,(%bx,%si)
4d:  66 b8 00 8a 66 89  mov    $0x89668a00,%eax
53:  c2 66 ef          ret     $0xef66
56:  66 b8 e0 8a 66 ef  mov    $0xef66ae0,%eax
5c:  eb fe             jmp     0x5c

```

۱۲ _ علت استفاده از دستور objcopy در حین اجرای عملیات make چیست؟

همانطور که در بخش قبل گفتیم برای اینکه قسمت تکست فایل bootblock.o را که همان instruction ها هستند را بتواند کپی کند و فایل raw binary عه bootblock را بسازد از objcopy استفاده شده است.

۱۴ _ یک ثبات عام منظوره ، یک ثبات قطعه، یک ثبات وضعیت و یک ثبات کنترلی در معماری را نام برده و وظیفه هر یکرا به طور مختصر توضیح دهید.

General purpose registers : EAX, EBX, ECX, EDX, ESI, and EDI رجیسترهایی هستند برای نگه داشتن متغیر ها حین محاسبه کردن. و همچنین رجیستر EIP همان PC است.(instruction pointer)

segment registers : اشاره گر به قطعه های مختلف مثل استک ، داده و کد در آنها نگه داری میشود مثل SS که پوینتر به استک یا CS که پوینتر به کد را نگه میدارند.

Status Registers : اطلاعات وضعیت کنونی پردازنده را نگه میدارند. مثل EFLAGS که اطلاعاتی درباره فلگ های zero و overflow و... را نگه میدارد.

Control registers : مسئول تغییر و یا کنترل پردازنده را بر عهده دارند مثل CR0 که وظیفه های مثل فعال کردن protected mode و یا سوییچ کردن بین تسک ها را دارد.

۱۸ _ کد معادل entry.s را درهسته لینوکس بیابید.

<https://github.com/torvalds/linux/blob/master/arch/arm64/kernel/entry.S>

۱۹ _ چرا این آدرس فیزیکی است؟

چون اگر آن را به صورت مجازی در نظر می‌گرفتیم باز باید یک بخش فیزیکی در نظر می‌گرفتیم که آدرس این مکان مجازی را به فیزیکی تبدیل کند که عملاً کار بیهوده ای است.

۲۲_ علاوه بر صفحه بندی در حد ابتدایی از قطعه بندی به منظور حفاظت هسته استفاده خواهد شد. این عملیات توسط sgininit انجام می‌گردد. همان طور که ذکر شد، ترجمه قطعه تأثیری بر ترجمه آدرس منطقی نمی‌گذارد. زیرا تمامی قطعه ها اعم از کد و داده روی یکدیگر می‌افتند. با این حال برای کد و داده های سطح کاربر پرچم SEG_USER تنظیم شده است. چرا؟

زیرا بتوان تفاوتی بین پردازش های سطح کاربر و پردازش های سطح کرنل ایجاد نمود. از آنجایی که محتوای هر دو این پردازنده ها در یک فضای فیزیکی قرار گرفته اند، با این کار می‌توان تشخیص داد که آن داده ها یا کد ها، داده ها و کد های سطح کاربر می‌باشند و اجازه دسترسی به هسته را ندارند.

۲۳_ جهت نگه داری اطلاعات مدیریتی برنامه های سطح کاربر ساختاری تحت عنوان struct proc ارائه شده است. اجزای آن را توضیح داده و ساختار معدل آن در سیستم عامل لینوکس را بیابید.

Sz : سایز مموری مربوط به پردازش (به بایت)

Pgdir : پوینتر به page table است.

Kstack : پایین استک کرنل برای این پردازش را مشخص می‌کند.

State : وضعیت پردازش را مشخص می‌کند.

Pid : آیدی اختصاص داده شده به این پردازش را مشخص می‌کند.

Parent : والد یا پدر با در واقع سازنده این پردازش را مشخص می‌کند.

Tf : فریم trap برای system call فعلی

Context : context switching را نگه می‌دارد.

Chan : اگر صفر نباشد به معنی خوابیدن پردازش است.

Killed : اگر صفر نباشد به معنی kill شدن پردازش است.

Ofile : فایل های باز شده توسط این پردازش را مشخص می‌کند.

Cwd : پوشه فعلی را مشخص می‌کند.

Name : نام این پردازش را مشخص می‌کند.

معدل آن در لینوکس در کد زیر است(task_struct):

<https://github.com/torvalds/linux/blob/master/include/linux/sched.h>

۲۷_ کدام بخش از آماده سازی سیستم، بین تمامی هسته های پردازنده مشترک و کدام بخش اختصاصی است؟
(از هر کدام یک مورد را با ذکر دلیل توضیح دهید). زمان بند روی کدام هسته اجرا می شود؟

همانطور که در کد main.c مشخص است یکسری دستورات مثل allocate کردن physcal page و ساختن trap vector ها ، ساختن لینک لیستی از بافر ها و ... به عهده Bootstrap processor است. ولی کارهایی مثل مپ کردن آدرس منطقی به آدرس مجازی که توسط تابع seginit انجام میشود بین آنها مشترک است.

```
int
main(void)
{
    kinit1(end, P2V(4*1024*1024)); // phys page allocator
    kvmalloc(); // kernel page table
    mpinit(); // detect other processors
    lapicinit(); // interrupt controller
    seginit(); // segment descriptors
    picinit(); // disable pic
    ioapicinit(); // another interrupt controller
    consoleinit(); // console hardware
    uartinit(); // serial port
    pinit(); // process table
    tvinit(); // trap vectors
    binit(); // buffer cache
    fileinit(); // file table
    ideinit(); // disk
    startothers(); // start other processors
    kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()
    userinit(); // first user process
    mpmain(); // finish this processor's setup
}

// Other CPUs jump here from entryother.S.
static void
mpenter(void)
{
    switchkvm();
    seginit();
    lapicinit();
    mpmain();
}
```

اشکال زدایی

سوال ۱- برای مشاهده Breakpoint ها از چه دستوری استفاده میشود؟

از دستور maintenance info breakpoints استفاده میشود. (همچنین دستور info break)

سوال ۲- برای حذف یک BreakPoint از چه دستوری و چگونه استفاده میشود؟

میتوان با دستور clear <filename>:<linenumber>، بریک پوینت قرار داده شده در خط linenumber از فایل filename را حذف کرد. همچنین دستورات مشاهده BreakPoint ها، به هرکدام یک شماره نسبت میدهند که به فرمت del

<number> میتوان BreakPoint مورد نظر را حذف کرد.

سوال ۳- دستور bt را اجرا کنید. خروجی آن چه چیزی را نشان میدهد؟

این دستور، همان backtrace و یک لیست از زنجیره توابع فراخوانی شده تا رسیدن به آن Breakpoint را نشان میدهد. این کار را به کمک استک فراخوانی توابع انجام میدهد و باعث میشود بدانیم که تابع چگونه به جایگاه کنونی (محل Breakpoint) که میتواند یک تابع خاص یا یک آدرس از حافظه ویا خطی در سورس کد باشد) رسیده است.

```
pouya@pouya-VivoBook: ~/opeeating system/lab/os-project...
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel...
+ target remote localhost:26000
The target architecture is set to "i8086".
[f000:ffff] 0xffff0: jmp $0x3630,$0xf000e05b
0x0000ffff in ?? ()
+ symbol-file kernel
(gdb) b sleep
Breakpoint 1 at 0x80103f31: sleep. (2 locations)
(gdb) c
Continuing.
The target architecture is set to "i386".
=> 0x80104040 <sleep>: push %ebp

Thread 1 hit Breakpoint 1, sleep (chan=0x8010a554 <bcache+52>, lk=0x80111600 <idelock>) at proc.c:61
61      pushcli();
(gdb) bt
#0  sleep (chan=0x8010a554 <bcache+52>, lk=0x80111600 <idelock>) at proc.c:61
#1  0x8010236e in iderw (b=0x8010a554 <bcache+52>) at ide.c:163
#2  0x80100191 in bread (dev=1, blockno=1) at bio.c:103
#3  0x801015ac in readsb (sb=0x801115b4 <sb>, dev=1) at fs.c:36
#4  init (dev=1) at fs.c:181
#5  0x801038b4 in forkret () at proc.c:408
#6  forkret () at proc.c:397
#7  0x80105852 in alltraps () at trapasm.S:21
(gdb)
```

در این تصویر، #۰ مربوط فریم فعلی و محل bp میباشد. در #۱، #۰ صدا زده شده و الی آخر.

سوال ۴- دو تفاوت دستورهای x و print را توضیح دهید. چگونه میتوان یک محتوای خاص را چاپ کرد؟

دستور print، یک عبارت (EXP) میگیرد که مقدار (value) آنرا با درنظر گرفتن آپشن های مشخص شده و به فرمت مشخص شده نمایش میدهد. مبنای کارکرد دستور x، آدرس میباشد و محتوای (content) آنرا نمایش میدهد. همچنین بخش FMT در این دستور (برخلاف print) الزامی میباشد و بصورت تعداد تکرار همراه با حرف فرمت (مثل o, x, d, u, t, مثل a, f, i, c, s, z و حرف اندازه (مثل b, h, w, g) میباشد. یک نمونه از استفاده از دستور print:

```
pouya@pouya-VivoBook: ~/opeeating system/lab...
(gdb) print input.e
$4 = 48
(gdb) print input.w
$5 = 48
```

با کمک دستور <reg_name> info register میتوان محتوای یک رجیستر خاص را چاپ کرد:

```
pouya@pouya-VivoBook: ~/o...
(gdb) info register edi
edi                0x1                1
(gdb) info register esi
esi                0x1                1
(gdb)
```

خروجی دستور list:

```
pouya@pouya-VivoBook: ~/opeeating system/lab/os-project...
Continuing.
The target architecture is set to "i386".
=> 0x80104040 <sleep>: push %ebp

Thread 1 hit Breakpoint 1, sleep (chan=0x8010a554 <bcache+52>, lk=0x80111600 <idelock>) at proc.c:61
61      pushcli();
(gdb) bt
#0  sleep (chan=0x8010a554 <bcache+52>, lk=0x80111600 <idelock>) at proc.c:61
#1  0x8010236e in iderw (b=0x8010a554 <bcache+52>) at ide.c:163
#2  0x80100191 in bread (dev=1, blockno=1) at bio.c:103
#3  0x801015ac in readsb (sb=0x801115b4 <sb>, dev=1) at fs.c:36
#4  iinit (dev=1) at fs.c:181
#5  0x801038b4 in forkret () at proc.c:408
#6  forkret () at proc.c:397
#7  0x80105852 in alltraps () at trapasm.S:21
(gdb) list
56      // while reading proc from the cpu structure
57      struct proc*
58      myproc(void) {
59          struct cpu *c;
60          struct proc *p;
61          pushcli();
62          c = mycpu();
63          p = c->proc;
64          popcli();
65          return p;
(gdb)
```

سوال ۵- برای نمایش وضعیت ثباتها از چه دستوری استفاده میشود؟ متغیرها محلی چگونه توضیح دهید که در معماری x86 رجیسترهای edi و esi نشانگر چه چیزی هستند؟

با دستور info register <reg_name> میتوان وضعیت یک ثبات خاص را مشاهده کرد.

با دستور info registers میتوان وضعیت ثباتها را نشان داد:

```
pouya@pouya-VivoBook: ~/opeeating system/lab/os-project1/codes/xv6
(gdb) info registers
eax            0x0
ecx            0x1f7
edx            0x1f7
ebx            0x8010a554
esp            0x8dffff20
ebp            0x8dffff3c
esi            0x1
edi            0x1
eip            0x80104040
eflags         0x2
cs             0x8
ss             0x10
ds             0x10
fs             0x10
gs             0x0
fs_base        0x0
gs_base        0x0
kgs_base       0x0
cr0            0x80010011
cr2            0x0
cr3            0xdffe000
cr4            0x10
cr8            0x0
efer           0x0
xmm0           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int6
4 = {0x0, 0x0}, uint128 = 0x0}
xmm1           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int6
4 = {0x0, 0x0}, uint128 = 0x0}
xmm2           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int6
4 = {0x0, 0x0}, uint128 = 0x0}
xmm3           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int6
4 = {0x0, 0x0}, uint128 = 0x0}
xmm4           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int6
4 = {0x0, 0x0}, uint128 = 0x0}
xmm5           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int6
4 = {0x0, 0x0}, uint128 = 0x0}
xmm6           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int6
4 = {0x0, 0x0}, uint128 = 0x0}
xmm7           {v4_float = {0x0, 0x0, 0x0, 0x1f80}, v2_double = {0x0, 0x1f8000000000}, v16_int8 = {0x0 <repeats 16 times>, 0x80, 0x1f, 0x0, 0x0}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1f80, 0x0}, v
4_int32 = {0x0, 0x0, 0x0, 0x1f80}, v2_int64 = {0x0, 0x1f8000000000}, uint128 = 0x1f80000000000000000000000000000000}
mxcsr          0x1f80
(gdb)
```

با دستور info locals میتوان وضعیت متغیرهای محلی را نشان داد:

```
pouya@pouya-VivoBook: ~/opeeating system/lab...
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
=> 0x80103901 <mycpu+17>:      mov     0x80111784,%esi
mycpu () at proc.c:48
48      for (i = 0; i < ncpu; ++i) {
(gdb) info local
apicid = 0
i = 0
(gdb)
```


همچنین با دستور info variables نیز میتوان وضعیت متغیرهای global و static را مشاهده کرد:

```
pouya@pouya-VivoBook: ~/opeeating system/lab/os-project1/codes/xv6
All defined variables:
File /home/pouya/opeeating system/lab/os-project1/codes/xv6/kbd.h:
95: static uchar ctlmap[256];
51: static uchar normalmap[256];
34: static uchar shiftcode[256];
73: static uchar shiftmap[256];
44: static uchar togglecode[256];
File bio.c:
36: struct {
    struct spinlock lock;
    struct buf buf[30];
    struct buf head;
} bcache;
File console.c:
187: struct {
    char buf[128];
    uint r;
    uint w;
    uint e;
} input;
25: static struct {
    struct spinlock lock;
    int locking;
} cons;
129: static ushort *crt;
20: static int panicked;
File file.c:
13: struct devsw devsw[10];
17: struct {
    struct spinlock lock;
    struct file file[100];
} ftable;
File fs.c:
169: struct {
    struct spinlock lock;
    struct inode inode[50];
} icache;
28: struct superblock sb;
File ide.c:
34: static int havedisk1;
31: static struct spinlock idelock;
32: static struct buf *idequeue;
File ioapic.c:
25: volatile struct ioapic *ioapic;
File kalloc.c:
24: struct {
--Type <RET> for more, q to quit, c to continue without paging--
```

برخی از عملیاتها برای انجام، به رجیسترهای edi و esi احتیاج دارند. همانند عملیات مربوط به رشته ها یا عملیات مربوط به کپی یا مقدار دهی یک آرایه در حافظه. در اینگونه عملیاتها، esi رجیستر مکان مبدا (نگهدارنده source) و edi رجیستر مکان مقصد (نگهدارنده destination) میباشند.

برای بررسی تخصصی تر:

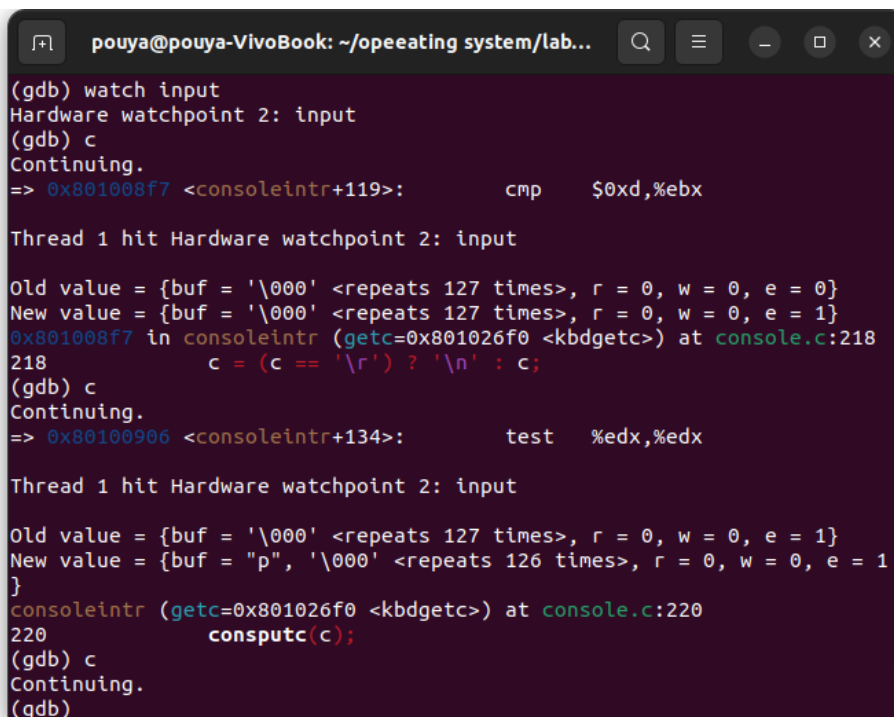
باید ذکر کرد که DI مخفف برای Destination Index و SI مخفف برای Source Index میباشند. فقط تعدادی عملیات وجود دارد که با این رجیسترها میتوان انجام داد: REP STOSB|MOVSB|SCASB که عملیات ذخیره سازی، بارگذاری و اسکن مکرر را انجام میدهند. برای خلاصه کردن، به این مثال اکتفا میکنیم که MOVSB میتواند برای انتقال داده ها از یک بافر به بافر دیگر بکار برود (کلا جابه جایی بایت ها)

سوال ۶- به کمک استفاده از GDB، درباره ساختار input struct موارد زیر را توضیح دهید:

- توضیح کلی این struct و متغیرهای درونی آن و نقش آنها

- نحوه و زمان تغییر مقدار متغیرهای درونی (برای مثال، input.e در چه حالتی تغییر میکند و چه مقداری میگیرد)

این استراکچر که در فایل console.c تعریف شده است، وظیفه دریافت و نگهداری ورودی و احتمالا وظایف مرتبط با آنرا برعهده دارد. این ساختار حاوی متغیر buf از نوع آرایه ای از کاراکترها را شامل میشود که وظیفه نگهداری از ورودی را دارد، همچنین سه متغیر r, w, e که قرار است کارکرد آنها را متوجه شویم. برای اینکار، واچ پوینت‌مان را برروی Input قرار میدهیم. حال منسول را باز کرده و حرف p را وارد میکنیم:



```
(gdb) watch input
Hardware watchpoint 2: input
(gdb) c
Continuing.
=> 0x801008f7 <consoleintr+119>:      cmp     $0xd,%ebx

Thread 1 hit Hardware watchpoint 2: input

Old value = {buf = '\000' <repeats 127 times>, r = 0, w = 0, e = 0}
New value = {buf = '\000' <repeats 127 times>, r = 0, w = 0, e = 1}
0x801008f7 in consoleintr (getc=0x801026f0 <kbdgetc>) at console.c:218
218      c = (c == '\r') ? '\n' : c;
(gdb) c
Continuing.
=> 0x80100906 <consoleintr+134>:      test    %edx,%edx

Thread 1 hit Hardware watchpoint 2: input

Old value = {buf = '\000' <repeats 127 times>, r = 0, w = 0, e = 1}
New value = {buf = "p", '\000' <repeats 126 times>, r = 0, w = 0, e = 1}
consoleintr (getc=0x801026f0 <kbdgetc>) at console.c:220
220      consputc(c);
(gdb) c
Continuing.
(gdb)
```

همانگونه که مشاهده میشود، متغیر e دوباره افزایش مقدار یافته، یکبار برای \$ و بار دوم برای p. این متغیر مقدار نشان میدهد که حرف ورودی بعدی، در کدام اندیس از آرایه buf باید نوشته شود؛ یعنی با وارد کردن یک کاراکتر جدید، ابتدا مقدار input->e یک واحد افزایش میابد، سپس کاراکتر ورودی در [input->buf[input->e - 1] نوشته میشود (در حقیقت، `(input.buf[input.e++] = c)`. با کمی امتحان بیشتر، این فرض را تایید میکنیم. حرف p را حذف میکنیم تا کرسر یک واحد به عقب بازگردد:

```
pouya@pouya-VivoBook: ~/opeeating system/lab...
Continuing.
(gdb) c
=> 0x80100999 <consoleintr+281>:      mov     0x8010ef58,%eax

Thread 1 hit Hardware watchpoint 2: input

Old value = {buf = "p", '\000' <repeats 126 times>, r = 0, w = 0, e = 1}
New value = {buf = "p", '\000' <repeats 126 times>, r = 0, w = 0, e = 0}
consoleintr (getc=0x801026f0 <kbdgetc>) at console.c:213
213      consputc(BACKSPACE);
(gdb) c
Continuing.
(gdb)
```

حال می‌خواهیم عملکرد w را بررسی کنیم. بنظر می‌آید ارتباطی با خواندن وجود دارد. با توجه به مشاهدات قبلی، میدانیم که به کاراکترهای نوشته شده در ترمینال در لحظه نوشتن مربوط نیست و فقط `buf[e]` از آن تاثیر می‌گیرند. متوجه می‌شویم که در دو حالت این واچ پوینت تاج می‌شود؛ حال اول مربوط به زمان فشردن کلید `enter` می‌باشد و حالت دوم هم زمانی که تعداد زیادی دکمه روی کیبورد زدیم (تعداد زیادی کاراکتر ورودی جدید) می‌باشد (حالت سومی هم به صورت فشردن `ctrl + D` وجود دارد). هم چنین از گزارش هایی که `gdb` به ما نشان می‌دهد (هم چنین افزودن واچ پوینت برای `r`)، متوجه می‌شویم که در این زمانها (که واچ پوینت `w`) تاج شده، مقدار آن برابر با مقدار `input.e` قرار می‌گیرد و سپس متغیر `input.r` واحد به واحد از مقدار اولیه اش (که برابر با مقدار قبلی `input.w` می‌باشد) افزایش می‌یابد تا به مقدار جدید `input.w` برسد و هر سه متغیر مقدار برابر بگیرند:

```
pouya@pouya-VivoBook: ~/opeeating system/lab...
(gdb) c
Continuing.
=> 0x80100a3f <consoleintr+447>:      push    $0x8010ef00

Thread 1 hit Hardware watchpoint 4: input.w

Old value = 31
New value = 37
consoleintr (getc=0x801026f0 <kbdgetc>) at console.c:223
223      wakeup(&input.r);
(gdb) c
Continuing.
=> 0x80100a3f <consoleintr+447>:      push    $0x8010ef00

Thread 1 hit Hardware watchpoint 4: input.w

Old value = 37
New value = 43
consoleintr (getc=0x801026f0 <kbdgetc>) at console.c:223
223      wakeup(&input.r);
(gdb)
```

```
pouya@pouya-VivoBook: ~/opeeating system/lab...
Continuing.
=> 0x80100321 <consoleread+161>:      mov     %eax,%edx

Thread 1 hit Hardware watchpoint 5: input.r

Old value = 40
New value = 41
0x80100321 in consoleread (ip=0x8010fa24 <icache+196>, dst=<optimized out>, n=<optimized out>) at console.c:253
253      c = input.buf[input.r++ % INPUT_BUF];
(gdb) c
Continuing.
=> 0x80100321 <consoleread+161>:      mov     %eax,%edx

Thread 1 hit Hardware watchpoint 5: input.r

Old value = 41
New value = 42
0x80100321 in consoleread (ip=0x8010fa24 <icache+196>, dst=<optimized out>, n=<optimized out>) at console.c:253
253      c = input.buf[input.r++ % INPUT_BUF];
(gdb) c
Continuing.
=> 0x80100321 <consoleread+161>:      mov     %eax,%edx

Thread 1 hit Hardware watchpoint 5: input.r

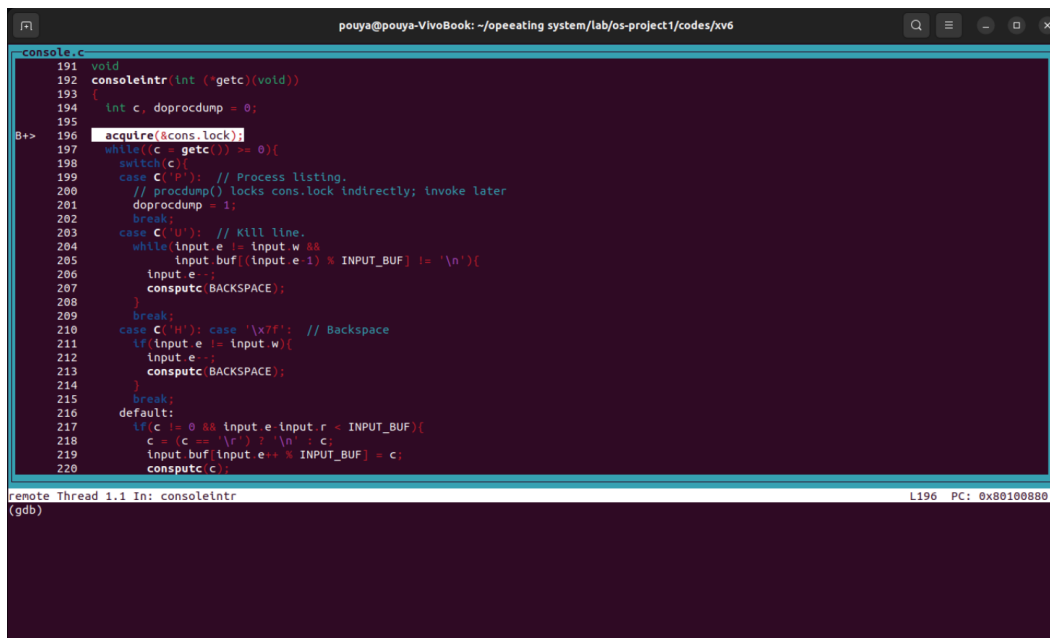
Old value = 42
New value = 43
0x80100321 in consoleread (ip=0x8010fa24 <icache+196>, dst=<optimized out>, n=<optimized out>) at console.c:253
253      c = input.buf[input.r++ % INPUT_BUF];
(gdb) c
Continuing.
```

برای تایید اینکه درنهایت (در حالت پایا) هر سه متغیر یاد شده به مقدار یکسان می‌رسند، مقدار هر سه را مشاهده می‌کنیم:

```
pouya@pouya-VivoBook: ~/opeeating system/lab...
(gdb) print input.e
$4 = 48
(gdb) print input.w
$5 = 48
(gdb) print input.r
$6 = 48
(gdb) 
```

اشکال زدایی در سطح کد اسمبلی

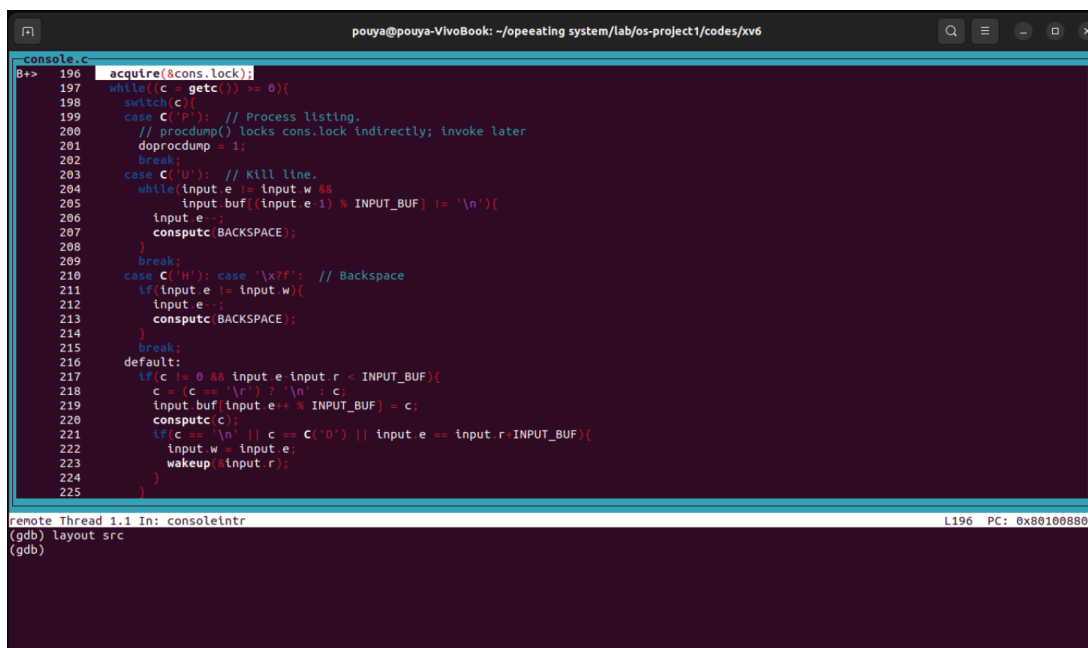
در این بخش، یک breakpoint برای خط ۱۹۶ در فایل console.c قرار می‌دهیم. بعد از تاج شدن بریک پوینت، با رفتن به محیط TUI، میتوان سورس کد محل تاج را مشاهده کرد (با توجه به ضبط شدن فریم های طی شده تا محل تاج، میتوان بین فریم ها نیز جابجا شد و سورس آنها را نیز مشاهده کرد:



```
191 void
192 consoleintr(int (*getc)(void))
193 {
194     int c, doprocdump = 0;
195
196     acquire(&cons.lock);
197     while((c = getc()) != 0){
198         switch(c){
199             case C('P'): // Process listing.
200                 // procdump() locks cons.lock indirectly; invoke later
201                 doprocdump = 1;
202                 break;
203             case C('U'): // Kill line.
204                 while(input.e != input.w &&
205                     input.buf[(input.e-1) % INPUT_BUF] != '\n'){
206                     input.e--;
207                     consputc(BACKSPACE);
208                 }
209                 break;
210             case C('H'): case '\x7f': // Backspace
211                 if(input.e != input.w){
212                     input.e--;
213                     consputc(BACKSPACE);
214                 }
215                 break;
216             default:
217                 if(c != 0 && input.e - input.r < INPUT_BUF){
218                     c = (c == '\r') ? '\n' : c;
219                     input.buf[input.e++ % INPUT_BUF] = c;
220                     consputc(c);
221                 }
222                 if(c == '\n' || c == C('D') || input.e == input.r + INPUT_BUF){
223                     input.w = input.e;
224                     wakeup(&input.r);
225                 }
226             }
```

سوال ۷- خروجی دستورهای layout asm, layout src در TUI چیست؟

با وارد کردن فرمان layout src، برنامه درحالت کد منبع آن (در اینجا، زبان C) مورد نمایش قرار میگیرد.



```
196     acquire(&cons.lock);
197     while((c = getc()) != 0){
198         switch(c){
199             case C('P'): // Process listing.
200                 // procdump() locks cons.lock indirectly; invoke later
201                 doprocdump = 1;
202                 break;
203             case C('U'): // Kill line.
204                 while(input.e != input.w &&
205                     input.buf[(input.e-1) % INPUT_BUF] != '\n'){
206                     input.e--;
207                     consputc(BACKSPACE);
208                 }
209                 break;
210             case C('H'): case '\x7f': // Backspace
211                 if(input.e != input.w){
212                     input.e--;
213                     consputc(BACKSPACE);
214                 }
215                 break;
216             default:
217                 if(c != 0 && input.e - input.r < INPUT_BUF){
218                     c = (c == '\r') ? '\n' : c;
219                     input.buf[input.e++ % INPUT_BUF] = c;
220                     consputc(c);
221                 }
222                 if(c == '\n' || c == C('D') || input.e == input.r + INPUT_BUF){
223                     input.w = input.e;
224                     wakeup(&input.r);
225                 }
226             }
```

سوال ۸- برای جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستورهایی استفاده میشود؟

با دستور up، میتوانیم به فریم قبلی یا بیرونی برویم و با دستور down، امکان جابه جایی به فریم بعدی با داخلی را خواهیم داشت (در فریم هایی که دستور bt نشان میدهد، با up به سمت فریم با شماره بیشتر و down به فریم با شماره کمتر در صورت وجود، جابه جا میشویم).

```
pouya@pouya-VivoBook: ~/opeeating system/lab/os-project1/codes/xv6
Breakpoint 1 at 0x80100880: file console.c, line 196.
(gdb) c
Continuing.
The target architecture is set to "i386".
=> 0x80100880 <consoleintr>:    push    %ebp

Thread 1 hit Breakpoint 1, consoleintr (getc=0x801026f0 <kbdgetc>) at console.c:196
196      acquire(&cons.lock);
(gdb) bt
#0  consoleintr (getc=0x801026f0 <kbdgetc>) at console.c:196
#1  0x801027e0 in kbdintr () at kbd.c:49
#2  0x80105af5 in trap (tf=0x80115418 <stack+3912>) at trap.c:67
#3  0x8010584f in alltraps () at trapasm.S:20
#4  0x80115418 in stack ()
#5  0x801117a4 in cpus ()
#6  0x801117a0 in ?? ()
#7  0x8010303f in mpmain () at main.c:57
#8  0x8010318c in main () at main.c:37
(gdb)
```

```
pouya@pouya-VivoBook: ~/opeeating system/lab/os-project1/cod...
trap.c
58      break;
59      case T_IRQ0 + IRQ_IDE:
60          ideintr();
61          lapiceoi();
62          break;
63      case T_IRQ0 + IRQ_IDE+1:
64          // Bochs generates spurious IDE1 interrupts.
65          break;
66      case T_IRQ0 + IRQ_KBD:
67          kbdintr();
68          lapiceoi();
69          break;
70      case T_IRQ0 + IRQ_COM1:
71          uartintr();
72          lapiceoi();
73          break;
74      case T_IRQ0 + 7:
75      case T_IRQ0 + IRQ_SPURIOUS:
76          cprintf("cpu%d: spurious interrupt at %x:%x\n",
77                  cpuid(), tf->cs, tf->eip);

remote Thread 1.1 In: trap L67 PC: 0x80105af5
(gdb) up
#1  0x801027e0 in kbdintr () at kbd.c:49
(gdb) up
#2  0x80105af5 in trap (tf=0x80115418 <stack+3912>) at trap.c:67
(gdb)
```

```
pouya@pouya-VivoBook: ~/opeeating system/lab/os-project1/cod...
kbd.c
40     else if('A' <= c && c <= 'Z')
41         c += 'a' - 'A';
42     }
43     return c;
44 }
45
46 void
47 kbdintr(void)
48 {
> 49     consoleintr(kbdgetc);
50 }
51
52
53
54
55
56
57
58
59

remote Thread 1.1 In: kbdintr L49 PC: 0x801027e0
(gdb) up
#1  0x801027e0 in kbdintr () at kbd.c:49
(gdb) up
#2  0x80105af5 in trap (tf=0x80115418 <stack+3912>) at trap.c:67
(gdb) down
#1  0x801027e0 in kbdintr () at kbd.c:49
(gdb)
```

اضافه کردن یک متن به boot message

برای نوشتن نام اعضای گروه به فایل init.c مراجعه می کنیم و نام اعضای گروه را چاپ می کنیم:

```

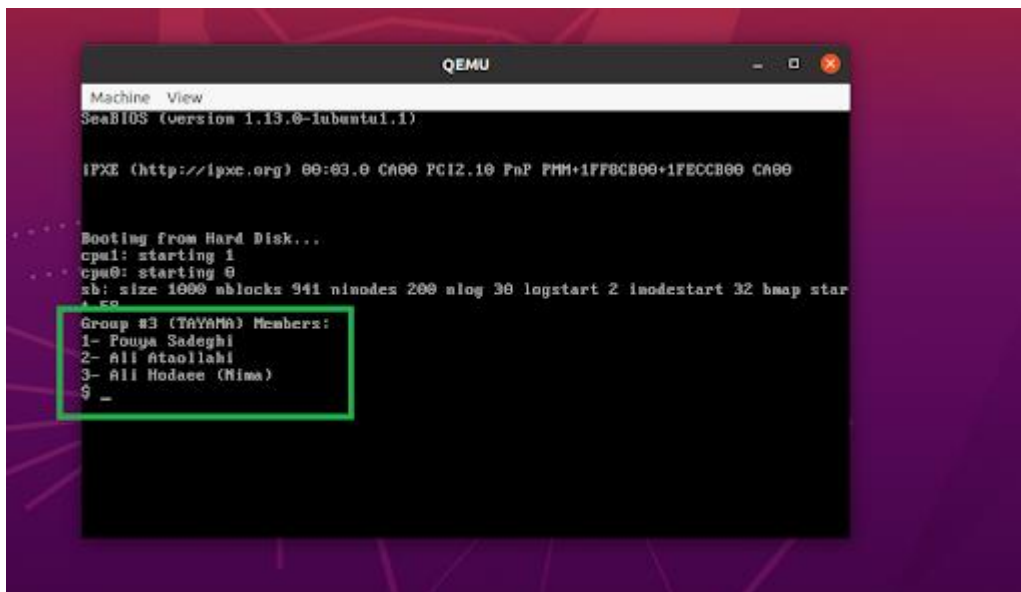
int
main(void)
{
    int pid, wpid;

    if(open("console", O_RDWR) < 0){
        mknod("console", 1, 1);
        open("console", O_RDWR);
    }
    dup(0); // stdout
    dup(0); // stderr

    for(;;){
        printf(1, "Group #3 (TAYAMA) Members:\n1- Pouya Sadeghi\n2- Ali Ataollahi\n3- Ali Hodaee (Nima)\n");
        pid = fork();
        if(pid < 0){
            printf(1, "init: fork failed\n");
            exit();
        }
        if(pid == 0){
            exec("sh", argv);
            printf(1, "init: exec sh failed\n");
            exit();
        }
        while((wpid=wait()) >= 0 && wpid != pid)
            printf(1, "zombie!\n");
    }
}

```

در ادامه برای مشاهده نتیجه با make qemu دوباره عملیات شبیه سازی را انجام می دهیم:



```

QEMU
Machine View
SeaBIOS (version 1.13.0-tarcent1.1)

IPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00

Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 mblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
A 00

Group #3 (TAYAMA) Members:
1- Pouya Sadeghi
2- Ali Ataollahi
3- Ali Hodaee (Nima)
$ _

```

اضافه کردن چند قابلیت به کنسول xv6

برای نوشتن دستورات به فایل console.c مراجعه می کنیم.

ابتدا برای سادگی کد نویسی مقداری ریفتور و تغییر روی کد اصلی انجام می دهیم تا فرایند کد نویسی تسریع گردد. ابتدا دو تابع زیر برای تغییر مکان cursor به کد اضافه شد.

```
int get_pos() {
    int pos;

    outb(CRTPORT, 14);
    pos = inb(CRTPORT+1) << 8;
    outb(CRTPORT, 15);
    pos |= inb(CRTPORT+1);

    return pos;
}

static void change_pos(int pos) {
    outb(CRTPORT, 14);
    outb(CRTPORT+1, pos>>8);
    outb(CRTPORT, 15);
    outb(CRTPORT+1, pos);
}
```

انتهای تابع زیر کمی ریفتور شد.

```
static void cgaputc(int c)
{
    int pos;

    // Cursor position: col + 80*row.
    pos = get_pos();

    if(c == '\n')
        pos += 80 - pos%80;
    else if(c == BACKSPACE){
        if(pos > 0) --pos;
    } else
        crt[pos++] = (c&0xff) | 0x0700; // black on white

    if(pos < 0 || pos > 25*80)
        panic("pos under/overflow");

    if((pos/80) >= 24){ // Scroll up.
        memmove(crt, crt+80, sizeof(crt[0])*23*80);
        pos -= 80;
        memset(crt+pos, 0, sizeof(crt[0])*(24*80 - pos));
    }

    change_pos(pos);
    if(c == BACKSPACE)
        crt[pos] = ' ' | 0x0700;
}
```

برای آنکه خوانایی کد بیشتر شود و قابلیت های struct input بیشتر شود یک عدد end به آن اضافه شد.

```

#define INPUT_BUF 128
struct {
    char buf[INPUT_BUF];
    uint r; // Read index
    uint w; // Write index
    uint e; // Edit index
    uint end; //end index
} input;

```

تابع شیفت به سمت چپ و شیفت به سمت راست اضافه شد.

در تابع شیفت به راست ابتدا پوزیشن کورسر گرفته شده سپس در بافر هر کارکتر در خانه بعد نوشته شده و سپس در خروجی با consputc قرار داده می شود. کورسر نیز یک عدد به سمت جلو می رود و input.end یک عدد زیاد می شود.

```

void shift_input_right() {
    int index, next_char, pos;
    pos = get_pos();
    change_pos(pos + 1);
    index = input.e;
    next_char = input.buf[index % INPUT_BUF];
    while(index < input.end) {
        int temp = next_char;
        next_char = input.buf[(index + 1) % INPUT_BUF];
        input.buf[(index + 1) % INPUT_BUF] = temp;
        consputc(input.buf[(index + 1) % INPUT_BUF]);
        index++;
    }
    input.end++;
    change_pos(pos);
}

```

در تابع شیفت به راست ابتدا پوزیشن کورسر گرفته شده سپس در بافر هر کارکتر در خانه قبل نوشته شده و سپس در خروجی با consputc قرار داده می شود. کورسر نیز یک عدد به سمت قبل می رود و input.end یک عدد کم می شود. کارکتر ابتدایی هم طبعاً پاک می شود.

```

void shift_input_left() {
    int index, pos;
    pos = get_pos();
    index = input.e - 1;
    while(index < input.end) {
        input.buf[index % INPUT_BUF] = input.buf[(index + 1) % INPUT_BUF];
        consputc(input.buf[index % INPUT_BUF]);
        index++;
    }
    consputc(' ');
    input.end--;
    change_pos(pos);
}

```

همچنین کنترلر Ctrl+U به صورت زیر ریفتور شد.

در این تابع تا زمانی که اینپوت ادیتور به ابتدای خط نرسد یک واحد به سمت چپ ورودی را شیفت می دهد تا جایی که تمام خط پاک بشود و در هر بار هم این اینپوت ادیتور یک عدد کم می شود.

```
void kill_line(){
    while(input.e != input.w && input.buf[(input.e-1) % INPUT_BUF] != '\n') {
        if(input.e != input.w){
            consputc(BACKSPACE);
            shift_input_left();
            input.e--;
        }
    }
}

case C('U'):
    kill_line();
    break;
```

و back_space به صورت زیر:

در این تابع یک واحد به سمت چپ شیفت می کنیم و input.e یک عدد کم می شود.

```
case C('H'): case '\x7f':
    back_space();
    break;
```

```
void back_space() {
    if(input.e != input.w){
        consputc(BACKSPACE);
        shift_input_left();
        input.e--;
    }
}
```

با توجه به اضافه شدن end به input باید دیفالت سوئیچ نوشته شده در consoleintr دچار تغییر شود. برای اینکار این قسمت را به بدین شکل در می آوریم:

```

default:
    if(c != 0 && input.e-input.r < INPUT_BUF){
        c = (c == '\r') ? '\n' : c;
        if(c == '\n' || c == C('D'))
            input.buf[input.end++ % INPUT_BUF] = c;
        else {
            shift_input_right();
            input.buf[input.e++ % INPUT_BUF] = c;
        }

        consputc(c);

        if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
            char* key;
            key = input.buf + input.w;
            input.e = input.end;
            input.w = input.e;
            wakeup(&input.r);
        }
    }
    break;
}

```

: Cntrl+n (۱)

مربوط به پاک کردن اعداد در خط:

در اینجا اول به ابتدای خط می رویم سپس تا زمانی که اینپوت ادیتور با انتهای خط برابر نشود یک فرایندی را انجام می‌دهیم که در آن اگر کارکتر قرار گرفته شده در ایندکس اینپوت ادیتور عدد بود یک واحد به سمت چپ آن را شیفت می‌دهیم در غیر این صورت خود آن را می‌گذاریم.

```

case C('N'):{
    delete_num_of_line();
    break;
}

```

```

void delete_num_of_line() {
    go_to_first_of_line();
    while (input.e < input.end){
        int pos = get_pos();
        if(is_num(input.buf[(input.e) % INPUT_BUF])){
            int pos=get_pos();
            change_pos(++pos);
            input.e++;
            consputc(BACKSPACE);
            shift_input_left();
            input.e--;
        }
        else{
            input.buf[(input.e) % INPUT_BUF] = input.buf[(input.e) % INPUT_BUF];
            consputc(input.buf[(input.e) % INPUT_BUF]);
            change_pos(++pos);
            input.e++;
        }
    }
}

```

```

void go_to_first_of_line() {
    int pos;
    pos = get_pos();
    int change;
    change = pos%80 - 2;
    input.e -= change;
    change_pos(pos - change);
}

```

```

int is_num(int c) {
    return (c >= '0' && c <= '9') ? 1 : 0;
}

```

Ctrl+r (۲):

مربوط به ریورس کردن:

در این قسمت ابتدا محل قرار گرفتن کارکترهای تایپ شده در ترمینال را به عنوان ورودی به تابع `print_word_reverse` می دهیم و در آنجا آن را در یک آرایه می ریزیم و سپس به صورت معکوس روی آن حلقه زده و آنجا را چاپ می کنیم.

```
case C('R'): {
    reverse_line();
    break;
}
```

```
void reverse_line() {
    char* key = input.buf + input.w;
    print_word_reverse(key);
}
```

```
void print_word_reverse(char* key) {
    char temp[INPUT_BUF];
    strncpy(temp, key, INPUT_BUF);
    kill_line();
    for (int i = strlen(temp)-1; i >= 0; i--) {
        input.buf[input.e % INPUT_BUF] = temp[i];
        consputc(input.buf[input.e % INPUT_BUF]);
        input.e++;
        input.end++;
    }
}
```

Tab (۳

برای آنکه تاریخچه دستورات زده شده ذخیره شود یک فایل به نام prefix_predict.c اضافه شد که دو تابع update_history و starts_with در آن قرار دارد.

در تابع update_history دستور وارد و ساینز آن را به عنوان ورودی می گیریم و در یک آرایه به نام command ذخیره می کنیم تا یک هیستوری از دستورات وارد شده داشته باشیم.

در starts_with هم بررسی میکنیم که کلمه وارد شده یک prefix برای کلمه دیگر هست یا نه. این تابع برای آن است که بتوانیم با توجه به ورودی زده شده در آرایه command پیش بینی دستور را انجام دهیم.

```

void update_history(char* inp,int size_command) {
    strcpy(command[command_num % MAX_COMMAND_NUM ], inp , size_command);
    command_num++;
    if(size_command < MAX_COMMAND_NUM) size_command++;
}
int starts_with(char *pre, char *str)
{
    if(strncmp(str, pre, strlen(pre)) == 0) return 1;
    return 0;
}

```

همچنین با توجه به اینکه به فایل user.h در این فایل نیاز بود و strcpy احتیاج بود که در defs.h قرار داشت که با user.h همزمان نمی توانستیم آن را اینکلود کنیم پس تابع strcpy به این فایل اضافه شد.

```

static char* strcpy(char *s, const char *t,int size)
{
    char *os;

    os = s;
    while((*s++ = *t++) != 0)
        ;
    return os;
}

```

همچنین باید هر دستور در هیستوری ذخیره شود که دیفالت در consoleintr را بدین صورت تغییر می دهیم تا دستور وارد شده وارد هیستوری دستورات شود.

```

default:
    if(c != 0 && input.e-input.r < INPUT_BUF){
        c = (c == '\r') ? '\n' : c;
        if(c == '\n' || c == C('D'))
            input.buf[input.end++ % INPUT_BUF] = c;
        else {
            shift_input_right();
            input.buf[input.e++ % INPUT_BUF] = c;
        }

        consputc(c);

        if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
            char* key;
            key = input.buf + input.w;
            updatehistory(key, input.e - input.w);
            sizeCommand++;
            command_num++;
            input.e = input.end;
            input.w = input.e;
            wakeup(&input.r);
        }
    }
    break;

```

حال سراغ خود کد اصلی در کنسول میرویم:

در اینجا ابتدا یک پیش بینی از دستور در لیست دستورات انجام می دهیم و به عنوان خروجی ایندکس دستور در تاریخچه را نگه می داریم تا بتوانیم آن را چاپ کنیم.

در خود تابع predic_in_list_command هم ورودی را می خوانیم.

```

case '\t': {
    predict_process();
    break;
}

```

```

#define MAX_COMMAND_NUM 15
extern int command_num ,size_command ;
extern char command[MAX_COMMAND_NUM][128];

```



```

void predict_process() {
    int index = predict_in_command_list();
    if(index != -1)
        print_word(command[index]);
}

```

```

int predict_in_command_list() {
    int index = -1;
    char* key = input.buf + input.w;
    if(size_command < MAX_COMMAND_NUM)
        for (int i = size_command; i >= 0; i--)
            if(starts_with(key, command[i]))
                index = i;
    else {
        int endIndex = ((command_num % MAX_COMMAND_NUM) + (MAX_COMMAND_NUM - 1))
            % MAX_COMMAND_NUM;
        int i = (command_num % MAX_COMMAND_NUM);
        while (i != endIndex)
        {
            if(starts_with(key, command[i])) {
                index = i;
            }
            i++;
            if(i == MAX_COMMAND_NUM) i = 0;
        }
        if(starts_with(key, command[i])) index = i;
    }
    return index;
}

```

```

void print_word(char* key) {
    char temp[INPUT_BUF];
    strncpy(temp, key, INPUT_BUF);
    kill_line();
    for (int i = 0; i < strlen(temp)-1; i++){
        input.buf[input.e % INPUT_BUF] = temp[i];
        consputc(input.buf[input.e % INPUT_BUF]);
        input.e++;
        input.end++;
    }
}

```

اجرا و پیاده سازی یک برنامه سطح کاربر

در ابتدا، باید کد برنامه را بنویسیم. الگوریتم که مشخص است و ارتباطی به سیستم عامل ندارد. از آنجا که در اینجا کد عادی C نمی‌نویسیم و کد را برای سیستم عامل می‌نویسیم، باید نکاتی را در نظر بگیریم. اولین نکته این است که ممکن است خیلی از

توابع استاندارد زبان در اینجا موجود نباشند و یا بعضاً header فایل با نام مشابه وجود داشته باشد(هر دو اتفاق اینجا افتاده)؛ پس باید برنامه را با توجه به این توابع توسعه بدیم. در ابتدا توابع استفاده شده را شرح میدهم.

توجه شود که تابع `atoi` بطور پیشفرض وجود دارد و برای ما قابل استفاده است، اما یک تابع با قابلیت مشابه توسعه دادیم که بتوانیم ارور های برنامه را بهتر کنترل کنیم(یک عدد منفی در صورت ایجاد مشکل برگردانیم و مشکلات مورد نظرمان را همراه با تبدیل به عدد چک کنیم). این ارور ها، شامل ارور در صورت وارد کردن عدد منفی یا عدد با فرمت غلط میباشد که برنامه بدرستی کاربر را از ایراد با خبر میسازد.

تابع بعدی، `printf` میباشد که عملکرد آن مشابه با تابع ساخته شده در `c` میباشد، با این تفاوت که یک آرگومان دیگر به عنوان شناسه محلی که باید در آن بنویسید را در ابتدا دریافت میکند. از ابتدا آدرس ۱ به خروجی استاندارد و ۲ مربوط به ارور استاندارد میباشد(توجه شود که شناسه یاد شده، مربوط به اندیس پردازش میباشد که دستور متوجه شود در کدام آدرس باید بنویسد). هم چنین این شناسه، نیابد لزومه به خروجی های ترمینال منتهی شود و میتواند مربوط به یک فایل باشد و در فایل عملیات نوشتن را انجام دهد. موضوع قابل توجه، این است که این تابع، در مراحل پایین تر، دارد فراخوانی سیستمی `write` را صدا میزند و فقط رابط بهتری را در اختیارمان قرار میدهد.

مابقی توابع نیز بصورت عادی میباشد و همانند چیزی که قبلاً دیده ایم، عمل میکنند.

حال که برنامه مورد نظر را نوشتیم، برنامه را درکنار سایر کدمنبع های سیستم عامل قرار میدهم. نکته قابل توجه نام فایل است که بعداً برنامه به همین نام پردازش شده و در آینده نیز کاربر باید برنامه را به همین نام صدا کند.

قبل از کامپایل و اجرای سیستم عامل، برای اینکه برنامه ما در سطح کاربر در سیستم عامل در دسترس قرار بگیرد، باید تغییراتی را در `Makefile` سیستم عامل ایجاد کنیم. باید به `EXTRA` به فرمت `<source_name>.c` و در `UROGS` به فرمت `<source_name>_` اضافه کنیم که برنامه در دسترس کاربر قرار بگیرد.

`UROGS` برنامه را در فایل سیستم قرار میدهد (که بعداً به `qemu` داده میشود) تا برنامه لود شده و در دسترس قرار بگیرد و یوزر بتواند آن دستورها را اجرا کند. `EXTRA` نیز موجب کپی شدن سورس به دایرکتوری `dist` میشود و مقدمات آماده سازی و افزودن برنامه را فراهم میکند.

و تمام. فقط کافی است که به شمل عادی سیستم عامل را کامپایل کرده و در `qemu` اجرا کنیم. برنامه ما آماده اجرا شدن میباشد.

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
Group #3 (TAYAMA) Members:
1- Pouya Sadeghi
2- Ali Ataollahi
3- Ali Hodaee (Nima)
$ prime_numbers 12 81
$ cat prime_numbers.txt
13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
$ prime_numbers 94 31
$ cat prime_numbers.txt
31 37 41 43 47 53 59 61 67 71 73 79 83 89
$
```

گزارش کار افزودن ماژول به هسته لینوکس

در اولین مرحله برای ساختن یک ماژول/درایو برای لینوکس، باید کد مورد نظر رو بنویسیم. باید توجه داشت که این کد با کدهای عادی که به زبان C نوشته میشوند، تفاوت دارد و این تفاوت از جهت کتابخانه ها و ساختاری میباشد. در وهله اول، باید کتابخانه های مربوط به لینوکس را اینکلود کنیم (خط های ۱-۳). همچنین شایان ذکر است که در این کد، تعدادی کال میبینیم که خارج از توابع انجام شده اند (خطوط ۵-۷ و ۱۸ و ۱۹) که مهم ترین آنها، خط پنج است که لایسنس و استاندارد مارا مشخص میکند. همچنین باید مشخص کنیم که هنگام لود کردن درایو یا ریموو کردن آن، چه تابعی باید اجرا شود (خطوط ۱۸ و ۱۹).

```

1  #include <linux/module.h>
2  #include <linux/kernel.h>
3  #include <linux/init.h>
4
5  MODULE_LICENSE("GPL");
6  MODULE_AUTHOR("Group#3 members");
7  MODULE_DESCRIPTION("this module print group#3's members name");
8
9  static int __init print_members_name(void) {
10     printk(KERN_INFO "Pouya Sadeghi, Ali Ataollahi, Ali Hodaee\n");
11     return 0;
12 }
13
14 static void __exit outro(void) {
15     printk(KERN_INFO "we are group#3\n");
16 }
17
18 module_init(print_members_name);
19 module_exit(outro);

```

حال، باید این کد را کامپایل کنیم. برای این منظور از Makefile استفاده میکنیم و آنرا با محتوای زیر ایجاد میکنیم. حال با اجرای دستور make در ترمینال، برنامه ما کامپایل میشود. از بین فایل های ایجاد شده، فایلی به نام driver.ko برای ما مهم ترین فایل میباشد که ماژول هسته کامپایل شده ما میباشد (از آنجا که نام فایل ما driver.c بود، این نام به فایل مذکور نسبت داده شد).

```

1  obj-m += driver.o
2
3  all:
4      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5  clean:
6      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

```

در ادامه برای اضافه کردن این ماژول به کرنل، دستور sudo insmod driver.ko را اجرا میکنیم. برای اینکه چک کنیم آیا ماژول به درستی در کرنل لود شده است، از دستور lsmod استفاده میکنیم:

```
pouya@pouya-VivoBook:~/linux_kernel_mod$ lsmod
Module                  Size  Used by
driver                  16384  0
ccm                     20480  6
rfcomm                  81920  4
cmac                    16384  2
algif_hash              16384  1
algif_skcipher          16384  1
af_alg                  32768  6 algif_hash,algif_skcipher
snd_hda_codec_hdmi      77824  1
bnep                    28672  2
snd_hda_codec_realtek   159744  1
snd_hda_codec_generic  102400  1 snd_hda_codec_realtek
ledtrig_audio           16384  1 snd_hda_codec_generic
joydev                  32768  0
intel_tcc_cooling        16384  0
x86_pkg_temp_thermal    20480  0
intel_powerclamp        20480  0
snd_soc_skl             172032  0
snd_soc_hdac_hda        24576  1 snd_soc_skl
snd_hda_ext_core        32768  2 snd_soc_hdac_hda,snd_soc_skl
snd_soc_sst_ipc         20480  1 snd_soc_skl
```

با مشاهده نام ماژول در لیست ماژول های لود شده در کرنل، متوجه میشویم که برنامه ب درستی رد کرنل لود شده است.

برای حذف ماژول از کرنل نیز، دستور `sudo rmmod driver` را اجرا میکنیم.

حال برای چک کردن خروجی، از دستور `sudo dmesg` استفاده میکنیم و میبینیم که خروجی ها بدرستی چاپ شده اند و ماژول عملکرد مورد انتظار را داشته و بدستی به کرنل اضافه و از آن حذف شده است:

```
exe= /usr/bin/odps-daemon sudo=102 hostname=1 addr=1 terminate=1
[ 764.336907] audit: type=1420 audit(1665405870.947:103): subj_apparmor=unconfined
[ 764.635695] audit: type=1400 audit(1665405871.243:104): apparmor="DENIED" operation="open" profile="snap.s
sted_mask="r" denied_mask="r" fsuid=1000 ouid=0
[ 839.250000] kauditd_printk_skb: 1 callbacks suppressed
[ 839.250002] audit: type=1326 audit(1665405945.865:106): auid=1000 uid=1000 gid=1000 ses=3 subj=? pid=2901
ll=93 compat=0 ip=0x7fe01749239b code=0x50000
[ 1116.527460] nouveau 0000:01:00.0: Enabling HDA controller
[ 1116.674915] nouveau 0000:01:00.0: bus: MMIO read of 00000000 FAULT at 6013d4 [ PRIVRING ]
[ 1125.050926] nouveau 0000:01:00.0: Enabling HDA controller
[ 1192.972190] driver: loading out-of-tree module taints kernel.
[ 1192.972234] driver: module verification failed: signature and/or required key missing - tainting kernel
[ 1192.972493] Pouya Sadeghi, Ali Ataollahi, Ali Hodaee ← گزارش مربوط به لود و مقدار دهی اولیه شدن ماژول
[ 1204.082108] we are group#3 ← گزارش ایجاد شده در زمان حذف ماژول از هسته
```