



## پروژه آزمایشگاه شماره ۱

دانشکده مهندسی برق و کامپیوتر

سیستم عامل - پاییز ۱۳۹۹

استاد : دکتر کارگهی

گروه ۱۷

اعضای گروه: دانشور امراللهی، علیرضا

توکلی، امین ستایش

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

سیستم عامل xv6 بر اساس Unix v6 پیاده‌سازی شده است. سیستم عامل Unix توسط Ken Thompson و Dennis Ritchie نوشته شده است. xv6 از ساختار Unix v6 استفاده می‌کند ولی با ANSI C برای پردازنده‌های مبتنی بر x86 پیاده‌سازی شده است.

از دلایل میتوان به موارد روبرو اشاره کرد: در فایل x86.h میتوانیم دستورات assembly مختص پردازنده‌های مبتنی بر x86 را مشاهده کنیم. در فایل asm.h نیز استفاده از معماری x86 ذکر شده است. در فایل mmu.h از x86 memory management unit استفاده شده است. در traps.h نیز میتوانیم مشاهده کنیم که trap ها مخصوص معماری x86 پیاده‌سازی شده‌اند.

۲. فایل‌های اصلی سیستم عامل xv6 در صفحه یک کتاب xv6 لیست شده‌اند. به طور مختصر هر گروه را توضیح دهید. نام پوشه اصلی فایل‌های هسته سیستم عامل، فایل‌های سرایند و فایل سیستم عامل لینوکس چیست؟ در مورد محتویات آن مختصراً توضیح دهید.

بخش‌های سیستم عامل xv6:

Basic Headers:

به طور کلی مقادیر ثابت که define شده‌اند و بعضی تعریف تایپ‌ها در این بخش قرار دارند.

- فایل types.h: شامل typedef های مورد نیاز
- param.h و memlayout.h و asm.h: حاوی define بعضی مقادیر ثابت
- defs.h: تعریف چند استراکت و توابع

- `x86.h`: توابعی برای استفاده از دستورات `assembly` در معماری `x86`
- `mmu.h`: بعضی استراکتها، مقادیر `define` شده برای مدیریت حافظه
- `date.h` و `elf.h`

## :Entering xv6

به طور کلی امکان آغاز سیستم عامل و فراهم کردن امکانات لازم را مهیا می‌کند.

- `main.c`: نقطه شروع سیستم است و سیستم از اینجا شروع به اجرا می‌کند.
- `entry.S`: کرنل از اینجا شروع به کار می‌کند و دستورات `assembly` این بخش برنامه را به بخش اجرای کد `c` منتقل می‌کنند.

- `entryother.c`

## :Locks

در این بخش مکانیزمی برای مدیریت دسترسی‌های مشترک با استفاده از `lock` پیاده‌سازی شده است. پیاده‌سازی این بخش در دو فایل `spinlock.h` و `spinlock.c` موجود است.

در این بخش امکاناتی برای گرفتن و رها کردن `lock` در نظر گرفته شده است. (توابع `acquire` و `release`)

## :Processes

این بخش وظیفه اختصاص دادن حافظه فیزیکی به پردازنده‌ها، مدیریت پردازنده‌ها (ایجاد و مدیریت پردازنده‌ها (Scheduling) و قابلیت `context switching` را برعهده دارد.

- `switch.S`: در این بخش قابلیت `context switching` پیاده‌سازی شده است به این صورت که وضعیت فعلی رجیسترها ذخیره می‌شوند تا دوباره بعداً برای اجرا بتوانند بازیابی شوند.
- `proc.h` و `proc.c`: قابلیت‌های مربوط به ایجاد و مدیریت پردازنده‌ها. برای مثال پیاده‌سازی `fork` در این بخش انجام شده است.
- `vm.c`
- `kalloc.c`: در این بخش پیاده‌سازی نحوه اختصاص یافتن حافظه فیزیکی به پردازنده‌ها انجام شده است.

## :System Calls

در این بخش `trap`ها و `system call`ها تعریف شده‌اند تا بتوان از آنها استفاده کرد.

- `traps.h` و `traps.c`: انواع `trap`ها و عدد متناظر آنها تعریف شده‌اند. همچنین توابع مربوط به `trap` نیز در این بخش پیاده‌سازی شده‌اند.
- `syscall.h` و `syscall.c`: عدد متناظر با `system call`ها و توابع مرتبط پیاده‌سازی شده‌اند.

## :File System

هدف یک فایل سیستم `organize` کردن و ذخیره کردن داده‌هاست. معمولاً `File System` ها به اشتراک‌گذاری داده‌ها را میان یوزرها و اپلیکیشن‌ها پشتیبانی می‌کنند.

فایل سیستم `xv6` از ۶ لایه تشکیل شده است.

پایینی‌ترین از طریق `buffercache` لایه بلوک‌هایی را از روی `IDE Disk` می‌خواند و می‌نویسد که تضمین می‌کند حداکثر یک `kernel process` در هر لحظه که می‌تواند داده فایل سیستمی ذخیره شده در یک `block` را تغییر دهد.

لایه دوم به لایه‌های بالاتر اجازه می‌دهد که آپدیت‌هایی رو روی `block` های بسیاری در یک `transaction` انجام دهد تا تضمین کند همه `block` ها اتماتیک آپدیت می‌شوند.

لایه سوم فایل‌های بدون نام `provide` می‌کند که هر کدام با یک `inode` و دنباله‌ای `block` ها شامل داده‌های فایل نمایش داده می‌شوند.

لایه چهارم `directory` ها را به عنوان یک `inode` خاص که محتویاتش دنباله‌ای از `entry` های `direcrotry` هست که هر کدام یک اسم و رفرنس به `inode` فایل است.

لایه پنجم سلسله مراتب `path name` ها (مثل `usr/rtn/xv6/fs.c/`) را با استفاده از ساختاری بازگشتی تامین می‌کند.

لایه آخر خیلی از منابع `unix` (مثل `files, devices, pipes, ...`) را به کمک فایل `systeminterface` انتزاع‌سازی می‌کند و کار را برای `application programmer` ها ساده‌تر می‌کند.

بعضی از فایل‌های آن به نام‌های زیر هستند که اعمال بالا را مدیریت می‌کنند:

- `fs.c`: روتین‌های `low level` مربوط به `file system` را داراست
- `log.c`: حداکثر یک `transaction` در لحظه را مدیریت می‌کند

`Pipes`:

در این بخش `struct pipe` تعریف شده است و توابعی برای عملیات خواندن و نوشتن برای آن پیاده‌سازی شده است. به طور کلی `pipe` برای این استفاده می‌شود که پردازنده‌ها بتوانند بر روی `pipe` بنویسند یا از آن بخوانند و بتوانند با هم ارتباط برقرار کنند.

`String Operations`:

توابع کمکی لازم برای کار کردن با `string`.

`Low level hardware`:

- `mp.h`: تعاریف مربوط به فایل `mp.c`
- `mp.c`: پیاده‌سازی پشتیبانی مولتی پراسسور
- `lapic.c`: مدیریت `interrupt` های داخلی (غیر `I/O`)
- `ioapic.c`: مدیریت `Interrupt` های سخت‌افزار برای یک سیستم `SMP`
- `kbd.h` و `kbd.c`: تعریف ثابت‌های دکمه‌های کیبورد

- `console.c`: کدهای ورودی و خروجی. ورودی از طریق کیبورد یا سریال پورت است و خروجی در صفحه و سریال پورت نوشته می‌شود.

- `Uart.c`: سریال پورت intel 8250

:User level

در این بخش اولین برنامه سطح‌کاربر اجرا می‌شود و امکاناتی نظیر `shell` اجرایی می‌شوند.

- `initcode.S`: کدهای `asm` برای اجرای برنامه سطح کاربر `init`.

- `usys.S`: تعریف سیستم‌کال‌ها در سطح کاربر.

- `init.c`: اولین برنامه سطح کاربر.

- `sh.c`: توابع و تعریف‌ها برای اجرای دستورات در `shell`.

:Boot Loader

این بخش عملیات‌های لازم برای `boot` شدن سیستم را انجام می‌دهد.

- `bootasm.S`: کد اسمبلی برای لود شدن کد BIOS از اولین سکتور حافظه و منتقل کردن اجرا به کد `c`

- `bootmain.c`: توابع مرتبط برای عملیات‌های `boot`.

:Link

یک `linker script` برای `JOS kernel` است.

فایل‌های مربوط به هسته لینوکس در پوشه `boot/` قرار دارند.

فایل‌های سراینده لینوکس در `usr/src/` قرار دارند.

فایل‌های فایل‌سیستم در سیستم‌عامل لینوکس از `root` اصلی یا همان `/` شروع می‌شوند.

فایل هسته سیستم‌عامل لینوکس: <https://github.com/torvalds/linux/tree/master/kernel>

فایل‌های سراینده سیستم‌عامل لینوکس: <https://github.com/torvalds/linux/tree/master/include>

فایل‌های فایل‌سیستم در سیستم‌عامل لینوکس: <https://github.com/torvalds/linux/tree/master/fs>

۶. در `xv6` در سکتور نخست دیسک قابل بوت، محتوای چه فایلی قرار دارد؟

```
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror  
-fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -fno-pic -O -nostdinc -l. -c  
bootmain.c
```

```
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror
-fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -fno-pic -nostdinc -l. -c
bootasm.S
ld -m elf_i386 -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o
objdump -S bootblock.o > bootblock.asm
objcopy -S -O binary -j .text bootblock.o bootblock
./sign.pl bootblock
```

دو فایل bootmain.c و bootasm.S وجود دارد.

۹. بوت سیستم توسط فایل‌های **bootmain.c** و **bootasm.S** صورت می‌گیرد. چرا تنها از کد C استفاده نشده است؟  
 زیرا برای بعضی ویژگی‌ها نیاز داریم از قابلیت‌های سطح سیستم استفاده کنیم. همچنین اجرای کد اسمبلی کم‌حجم‌تر و سریع‌تر است. کد bootasm.S پردازنده را به حالت محافظت شده ۳۲ بیت برده و پس از آن تابع bootmain از فایل bootmain.c صدا زده می‌شود.

۱۰. یک ثابت عام‌منظوره، یک ثابت قطعه، یک ثابت وضعیت و یک ثابت کنترلی در معماری x86 نام برده و وظیفه هر یک را به طور مختصر توضیح دهید.

- **ثبات عام منظوره:** همانطور که در کتاب هم اشاره شده، x86 دارای ۸ ثبات عام‌منظوره می‌باشد. این ثبات‌ها عبارتند از %eax, %ebx, %ecx, %edx, %edi, %esi, %ebp, %esp. حرف e در این ثبات‌ها نشان‌دهنده extended است زیرا ۳۲ بیت هستند. این ثبات‌ها نگهداری بعضی اشاره‌گرها، داده و برای نگهداری عملیات‌های ریاضی استفاده می‌شوند.
- **ثبات قطعه:** آدرس استک، کد و داده در این ثبات‌ها نگهداری می‌شود. برای مثال SS پوینتر به استک، CS پوینتر به کد و DS پوینتر به داده را نگه می‌دارد.
- **ثبات وضعیت:** شامل اطلاعات راجع به وضعیت پردازنده است. EFLAGS در این بخش محسوب می‌شود و اطلاعات فلگ‌هایی نظیر carry, sign, zero و غیره را مشخص می‌کند.
- **ثبات کنترلی:** کنترل CPU یا دستگاه‌های دیجیتال دیگر را در دست دارد. cr0, %cr2, %cr3, %cr4 از این ثبات‌ها هستند. این ثبات‌ها وظیفه تغییر مدل آدرس‌دهی، کنترل interrupt، کنترل paging و هم‌پردازنده‌ها<sup>۱</sup> را دارند.

دستور info registers:

---

<sup>۱</sup> coprocessor

```
(gdb) info registers
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd110      0xffffd110
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x10000c        0x10000c
eflags         0x10202        [ IF RF ]
cs             0x23            35
ss             0x2b            43
ds             0x2b            43
es             0x2b            43
fs             0x0            0
gs             0x0            0
```

```
(qemu) info registers
EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000663
ESI=00000000 EDI=00000000 EBP=00000000 ESP=00000000
EIP=0000ffff EFL=00000002 [-----] CPL=0 II=0 A20=1 SMM=0 HLT=0
ES =0000 00000000 0000ffff 00009300
CS =f000 fffff000 0000ffff 00009b00
SS =0000 00000000 0000ffff 00009300
DS =0000 00000000 0000ffff 00009300
FS =0000 00000000 0000ffff 00009300
GS =0000 00000000 0000ffff 00009300
LDT=0000 00000000 0000ffff 00008200
TR =0000 00000000 0000ffff 00008b00
GDT=      00000000 0000ffff
IDT=      00000000 0000ffff
CR0=60000010 CR2=00000000 CR3=00000000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0fff DR7=00000400
EFER=0000000000000000
FCW=037f FSW=0000 [ST=0] FTW=00 MXCSR=00001f80
FPR0=0000000000000000 0000 FPR1=0000000000000000 0000
FPR2=0000000000000000 0000 FPR3=0000000000000000 0000
FPR4=0000000000000000 0000 FPR5=0000000000000000 0000
FPR6=0000000000000000 0000 FPR7=0000000000000000 0000
XMM0=00000000000000000000000000000000 XMM01=00000000000000000000000000000000
XMM02=00000000000000000000000000000000 XMM03=00000000000000000000000000000000
XMM04=00000000000000000000000000000000 XMM05=00000000000000000000000000000000
XMM06=00000000000000000000000000000000 XMM07=00000000000000000000000000000000
```

۱۴. کد معادل `entry.S` در هسته لینوکس را بیابید.

کد متناظر در لینوکس در فایل زیر است:

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

۱۵. چرا این آدرس فیزیکی است؟

اگر این بخش را به صورت مجازی در نظر می‌گیریم باز باید یک بخش فیزیکی در نظر می‌گیریم تا این بخش مجازی را مشخص کند، یعنی در نهایت نیاز به بخش فیزیکی بود.

۱۹. جهت نگهداری اطلاعات مدیریتی برنامه‌های سطح کاربر ساختاری تحت عنوان **proc struct** ارائه شده است. اجزای آن را توضیح داده و ساختار معادل آن در سیستم عامل لینوکس را بیابید.

- **SZ**: سایز حافظه متعلق به پردازنده به **byte**
- **pgdir**: پوینتر به **page table** است.
- **kstack**: پایین استک کرنل برای این پردازنده را مشخص می‌کند.
- **state**: وضعیت این پردازنده را مشخص می‌کند.
- **pid**: عدد اختصاص داده شده به این پردازنده.
- **parent**: پدر این پردازنده یا به عبارت دیگر سازنده این پردازنده را مشخص می‌کند.
- **tf**: چارچوب **trap** برای **system call** حال حاضر
- **context**: برای **context switching** نگهداری شده است.
- **chan**: اگر صفر نباشد به معنای خوابیدن پردازنده است.
- **killed**: اگر غیر صفر باشد یعنی پردازنده **kill** شده است.
- **ofile**: فایل‌های باز شده توسط این پردازنده.
- **cwd**: پوشه‌ی کنونی را مشخص می‌کند.
- **name**: نام این پردازنده.

معادل این استراکت در کرنل لینوکس در لینک زیر آمده است: (**task\_struct**)

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

۲۱. در **kvmalloc** یک صفحه برای آدرس‌های هسته برای مدیریت پراسس‌ها ساخته می‌شود و برای کل سیستم است در صورتی که **setupkvm** یک صفحه برای هر پراسس می‌سازد که مثلاً در **userinit** برای اولین پراسس این صفحه ساخته می‌شود.

۲۴. کد متناظر در لینوکس در فایل زیر است:

<https://github.com/torvalds/linux/blob/master/arch/arm64/mm/proc.S>

### اشکال زدایی

۱. برای مشاهده **BreakPoint**‌ها از چه دستوری استفاده می‌شود؟

با استفاده از دستور **maint info breakpoints** این کار امکان‌پذیر است.

۲. برای حذف یک **BreakPoint** از چه دستوری و چگونه استفاده می‌شود؟

با استفاده از دستور **clear filename:line** این کار امکان‌پذیر است. که در آن **filename** نام فایل مورد نظر و **line** نیز نشان‌دهنده خط مورد نظر است.



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

این دستور که مخفف backtrace است سلسله توابعی که فراخوانی شده و در استک اضافه شده‌اند را نشان می‌دهد. به طور خلاصه این دستور نشان می‌دهد که برنامه چطور به جایی که اکنون در آن قرار دارد رسیده است.

```
(gdb) target remote tcp::26000
Remote debugging using tcp::26000
0x80103e05 in ?? ()
(gdb) b cat.c:12
Note: breakpoint 1 also set at pc 0x97.
Breakpoint 2 at 0x97: file cat.c, line 12.
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, cat (fd=3) at cat.c:12
12      while((n = read(fd, buf, sizeof(buf))) > 0) {
(gdb) bt
#0  cat (fd=3) at cat.c:12
#1  0x000000054 in main (argc=<optimized out>, argv=<optimized out>) at cat.c:39
(gdb) □
```

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

دستور print می‌تواند یک expression دریافت کند و مقدار آن را نمایش دهد اما دستور x بر اساس آدرس کار می‌کند و مقدار را نمایش می‌دهد.

همچنین این دو دستور از جهت نحوه نمایش اطلاعات با یکدیگر متفاوت هستند.

با استفاده از دستور info register نام یک ثبات خاص را نمایش داد، که در آن نام آن ثبات به عنوان آرگومان پاس داده شده است.

### ۵. برای نمایش وضعیت ثباتها از چه دستوری استفاده میشود؟ متغیرها محلی چطور؟ نتیجه این دستور را در گزارش کار خود

بیابید. همچنین در گزارش خود توضیح دهید که در معماری x86 رجیسترهای edi و esi نشانگر چه چیزی هستند؟

مشاهده وضعیت ثباتها با دستور info registers:



```

(qdb) info registers
eax      0x3      12256
ecx      0x2fe0   40060
edx      0xbfac   12272
ebx      0x2ff0   12272
esp      0x2f88   0x2f88
ebp      0x2f88   0x2f88
esi      0x2      2
edi      0x3      3
eip      0x97     0x97 <cat-7>
eflags   0x216    [ IOPL=0 IF AF PF ]
cs       0x1b     27
ss       0x23     35
ds       0x23     35
es       0x23     35
fs       0x0      0
gs       0x0      0
fs_base  0x0      0
gs_base  0x0      0
k_gs_base 0x0      0
cr0      0xb8010011 [ PG WP ET PE ]
cr2      0x0      0
cr3      0xdfef000 [ PDBR=0 PCID=0 ]
cr4      0x10     [ PSE ]
cr8      0x0      0
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_int64 = [0x0, 0x0], v2_int128 = [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_int64 = [0x0, 0x0], v2_int128 = [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_int64 = [0x0, 0x0], v2_int128 = [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_int64 = [0x0, 0x0], v2_int128 = [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_int64 = [0x0, 0x0], v2_int128 = [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_int64 = [0x0, 0x0], v2_int128 = [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_int64 = [0x0, 0x0], v2_int128 = [0x0]}
xmm7     {v4_float = [0x0, 0x0, 0x0, 0x0], v2_double = [0x0, 0x0], v16_int8 = [0x0 <repeats 12 times>], v8_int16 = [0x0, 0x0, 0x0, 0x0], v8_int16 = [0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0], v4_int32 = [0x0, 0x0, 0x0, 0x0], v2_int64 = [0x0, 0x0], v2_int128 = [0x0, 0x0, 0x1f80000000000000, 0x1f8000000000000000]}
xmm8     {v4_float = [0x0, 0x0, 0x0, 0x0], v2_double = [0x0, 0x0], v16_int8 = [0x0 <repeats 12 times>], v8_int16 = [0x0, 0x0, 0x0, 0x0], v8_int16 = [0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0], v4_int32 = [0x0, 0x0, 0x0, 0x0], v2_int64 = [0x0, 0x0], v2_int128 = [0x0, 0x0, 0x1f80, 0x0]}
mcses    [ ]
(qdb) []

```

با استفاده از دستور `info locals` می‌توانیم وضعیت متغیرهای محلی را مشاهده کنیم. اگر بخواهیم تمام متغیرها به صورت گلوبال هم مشاهده کنیم می‌توانیم از دستور `info variables` استفاده کنیم.

```
(gdb) info locals
n = <optimized out>
(gdb) info variables
All defined variables:

File cat.c:
5:      char buf[512];

File umalloc.c:
21:      static Header base;
22:      static Header *freep;

Non-debugging symbols:
0x000000878  digits
0x000000b5c  __bss_start
0x000000b5c  _edata
0x000000d80  _end
(gdb) □
```

بعضی عملیات‌ها هستند که فقط با استفاده از `esi` و `edi` قابل انجام هستند. `Esi` در این عملیات‌ها به `source` و `edi` به مکان `destination` اشاره می‌کند.

۶. به کمک استفاده از GDB، درباره ساختار `input struct` موارد زیر را توضیح دهید.

در این استراکت سه متغیر `e`, `r`, `w` وجود دارد که دو متغیر `r` و `w` ابتدای این خط ورودی را در `buffer` نشان می‌دهند. با استفاده از `print` مقدار این دو را مشاهده کردیم.

برای متغیر `e` نیز در ادامه توضیح داده شده است.

در فایل `console.c` روی خط ۴۰۸ breakpoint قرار دادیم و همانطور که در تصویر مشخص است بعد از اینکه یکی cursor را به راست برده‌ایم مقدار `input.e` یکی بیشتر شده است.

پس `input.e` نمایانگر مقدار انتهایی خط در حال تایپ شدن در `buffer` است.

```
(gdb) b console.c:408
Breakpoint 1 at 0x80100cf0: file console.c, line 409.
(gdb) print input.e
$1 = 0
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, consoleintr (getc=0x80115c80 <tickslock>) at console.c:409
409      release(&cons.lock);
(gdb) print input.e
$2 = 1
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, consoleintr (getc=0x80115c80 <tickslock>) at console.c:409
409      release(&cons.lock);
(gdb) print input.e
$3 = 2
(gdb) □
```

#### ۷. خروجی دستورهای `layout src` و `layout asm` در TUI چیست؟

با استفاده از دستور `layout asm` می‌توان برنامه را در حالت کد اسمبلی مشاهده کرد.

با استفاده از دستور `layout src` می‌توان برنامه را در حالت کد سورس آن مشاهده کرد.

```

0x80100ce8 <consoleintr+88>    call    *%eax
0x80100cea <consoleintr+90>    mov     %eax,%esi
0x80100cec <consoleintr+92>    test   %eax,%eax
0x80100cee <consoleintr+94>    jns     0x80100cc2 <consoleintr+50>
B+> 0x80100cf0 <consoleintr+96>    sub     $0xc,%esp
0x80100cf3 <consoleintr+99>    push   $0x8010b540
0x80100cf8 <consoleintr+104>    call   0x80104b40 <release>
0x80100cfd <consoleintr+109>    add     $0x10,%esp
0x80100d00 <consoleintr+112>    test   %ebx,%ebx
0x80100d02 <consoleintr+114>    jne     0x80100f10 <consoleintr+640>
0x80100d08 <consoleintr+120>    lea     -0xc(%ebp),%esp
0x80100d0b <consoleintr+123>    pop     %ebx
0x80100d0c <consoleintr+124>    pop     %esi
0x80100d0d <consoleintr+125>    pop     %edi
0x80100d0e <consoleintr+126>    pop     %ebp
0x80100d0f <consoleintr+127>    ret
0x80100d10 <consoleintr+128>    call   0x80100aa0 <jump_right_cursor>
0x80100d15 <consoleintr+133>    jmp     0x80100cb7 <consoleintr+39>
0x80100d17 <consoleintr+135>    call   0x801009e0 <jump_left_cursor>
0x80100d1c <consoleintr+140>    jmp     0x80100cb7 <consoleintr+39>
0x80100d1e <consoleintr+142>    call   0x80100b70 <delete_line_until_here>
0x80100d23 <consoleintr+147>    jmp     0x80100cb7 <consoleintr+39>
0x80100d25 <consoleintr+149>    cmp     $0x7f,%esi
0x80100d28 <consoleintr+152>    jne     0x80100d57 <consoleintr+199>
0x80100d2a <consoleintr+154>    mov     0x80110fc8,%eax
0x80100d2f <consoleintr+159>    cmp     0x80110fc4,%eax
0x80100d35 <consoleintr+165>    je      0x80100cb7 <consoleintr+39>
0x80100d37 <consoleintr+167>    sub     $0x1,%eax
0x80100d3a <consoleintr+170>    mov     %eax,0x80110fc8
0x80100d3f <consoleintr+175>    mov     0x8010b578,%eax
0x80100d44 <consoleintr+180>    test   %eax,%eax
0x80100d46 <consoleintr+182>    je      0x80100ef0 <consoleintr+608>
0x80100d4c <consoleintr+188>    cli

```

emote Thread 1.1 In: consoleintr

gdb) layout asm

gdb) ☐

```
console.c
394     }
395     break;
396     default:
397         if(c != 0 && input.e-input.r < INPUT_BUF){
398             c = (c == '\r') ? '\n' : c;
399             input.buf[input.e++ % INPUT_BUF] = c;
400             consputc(c);
401             if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
402                 input.w = input.e;
403                 wakeup(&input.r);
404             }
405         }
406         break;
407     }
408 }
B+>409 release(&cons.lock);
410 if(doprocdump) {
411     procdump(); // now call procdump() wo. cons.lock held
412 }
413 }
414
415 int
416 consoleread(struct inode *ip, char *dst, int n)
417 {
418     uint target;
419     int c;
420
421     iunlock(ip);
422     target = n;
423     acquire(&cons.lock);
424     while(n > 0){
425         while(input.r == input.w){
426             if(myproc()->killed){
```

```
remote Thread 1.1 In: consoleintr
(gdb) layout asm
(gdb) layout src
(gdb) □
```

۸. برای جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستوراتی استفاده میشود؟

به ترتیب از دستورات up و down می‌توان به این منظور استفاده کرد، نمونه‌ای از استفاده از این دستورات در عکس زیر آمده است.

```

406         break;
407     }
408 }
B+>409 release(&cons.lock);
410 if(doprocdump) {
411     procdump(); // now call procdump() wo. cons.lock held
412 }
413 }
414
415 int
416 consoleread(struct inode *ip, char *dst, int n)
417 {
418     uint target;
419     int c;
420
421     iunlock(ip);
422     target = n;
423     acquire(&cons.lock);
424     while(n > 0){
425         while(input.r == input.w){
426             if(myproc()->killed){

```

remote Thread 1.1 In: consoleintr

```

#19 0x8010353f in mpmain () at main.c:57
#20 0x8010368c in main () at main.c:37
(gdb) up
#1 0x801137a0 in ?? ()
(gdb) up
#2 0x80102cb4 in kbdintr () at kbd.c:49
(gdb) down 2
#0 consoleintr (getc=0x80115c80 <tickslock>) at console.c:409
(gdb) up 2
#2 0x80102cb4 in kbdintr () at kbd.c:49
(gdb) up
#3 0x80102bc0 in ?? ()
(gdb) up
#4 0x80105fca in trap (tf=0x80105f4d <trap+221>) at trap.c:67
(gdb) down 4
#0 consoleintr (getc=0x80115c80 <tickslock>) at console.c:409
(gdb) 

```



```
49 consoleintr(kbdgetc);
50 }
```

remote Thread 1.1 In: kbdintr

```
(gdb) up
#1 0x801137a0 in ?? ()
(gdb) up
#2 0x80102cb4 in kbdintr () at kbd.c:49
(gdb) down 2
#0 consoleintr (getc=0x80115c80 <tickslock>) at console.c:409
(gdb) up 2
#2 0x80102cb4 in kbdintr () at kbd.c:49
(gdb) up
#3 0x80102bc0 in ?? ()
(gdb) up
#4 0x80105fca in trap (tf=0x80105f4d <trap+221>) at trap.c:67
(gdb) down 4
#0 consoleintr (getc=0x80115c80 <tickslock>) at console.c:409
(gdb) up 2
#2 0x80102cb4 in kbdintr () at kbd.c:49
```