



به نام خدا  
آزمایشگاه سیستم عامل



## پروژه اول آزمایشگاه سیستم عامل (آشنایی با هسته سیستم عامل xv6)

### گروه ۶

اعضای گروه :

۸۱۰۱۹۸۴۱۰

محمد سعادت

۸۱۰۱۹۸۴۳۶

محمد عراقی

۸۱۰۱۹۸۵۲۷

سید عماد امامی

## آشنایی با سیستم عامل xv6

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

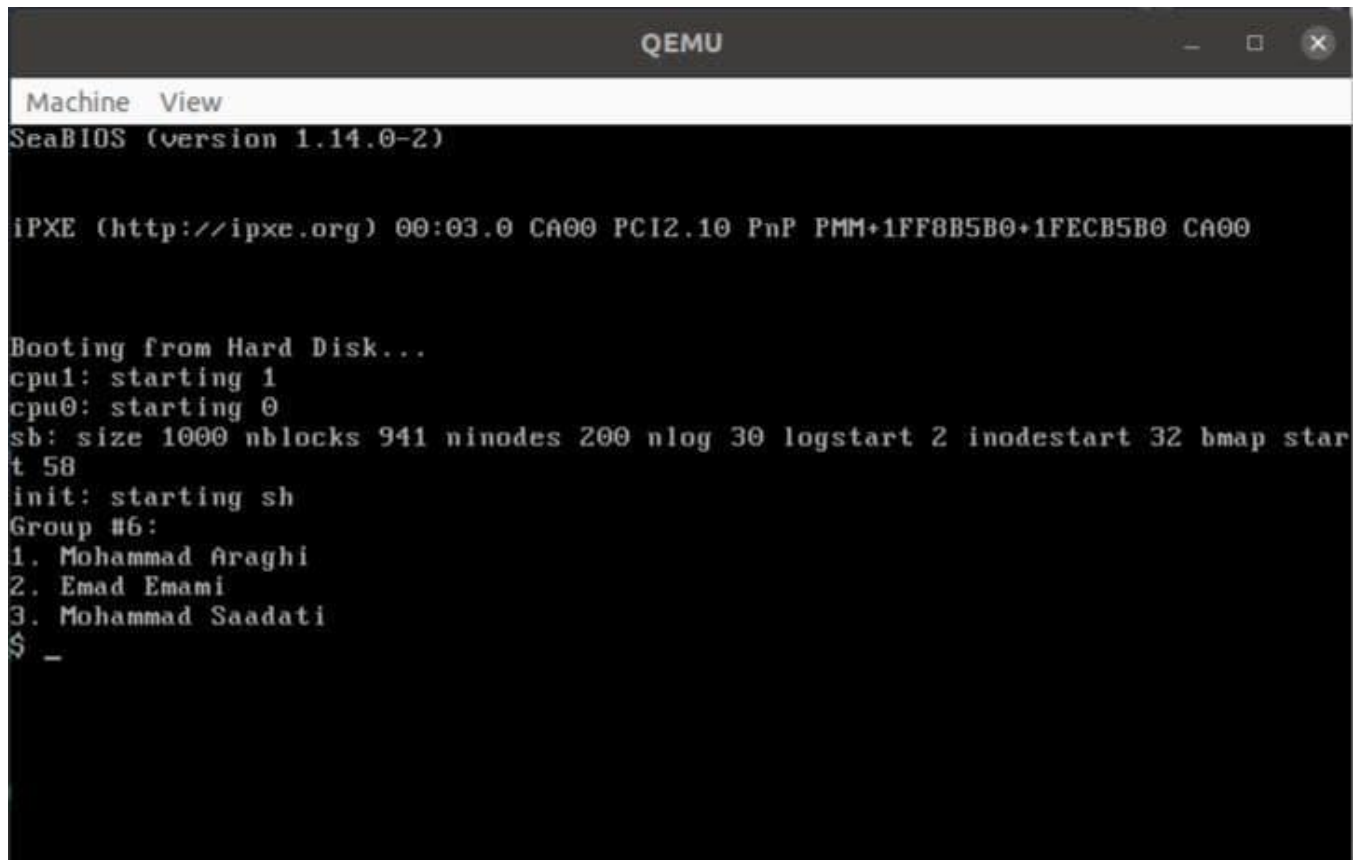
سیستم عامل xv6 به زبان C و بر اساس Unix Version 6 توسط Ken Thompson's و Dennis Ritchie's نوشته شده است. xv6 از معماری و استایل Unix v6 استفاده میکند ولی با ANSI C برای سیستم های چندپردازنده ای مبتنی بر x86 پیاده سازی شده است.

دلایل به شرح زیر است: در فایل traps.h می توان دید که trap های مخصوص معماری x86 پیاده سازی شده اند. در فایل mmu.h از x86 memory management unit استفاده شده است. در فایل asm.h استفاده از معماری x86 ذکر شده است. در فایل x86.h دستورات assembly مختص پردازنده های مبتنی بر x86 قابل مشاهده است. در فایل lapic.c استفاده از APIC قابل مشاهده است که مخصوص پردازنده intel است. همچنین ساختار این سیستم عامل از نوع Monolithic است.

### ۲) یک پردازنده در سیستم عامل xv6 از چه بخش هایی تشکیل شده است؟ این سیستم عامل به طور کلی چه گونه پردازنده را به پردازنده های مختلف اختصاص می دهد؟

هر پردازنده در xv6 از user-space memory که شامل stack, data, instructions و per-process state می باشد تشکیل شده که مخصوص kernel است. این سیستم عامل زمان پردازنده را بین کارها تقسیم میکند بدین صورت که پردازنده ها را بین فرآیند هایی که در صف اجرا هستند قرار میدهد. هنگامی که فرآیندی متوقف شد xv6 رجیستر های پردازنده را ذخیره میکند تا برای اجرای بعدی بازایی کند. همچنین هسته برای هر فرآیند یک pid ویژه متناظر میکند.

## اضافه کردن یک متن به Message Boot



```
Machine View
SeaBIOS (version 1.14.0-2)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B5B0+1FECB5B0 CA00

Booting from Hard Disk...
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
Group #6:
1. Mohammad Araghi
2. Emad Emami
3. Mohammad Saadati
$ _
```

## مقدمه ای درباره سیستم عامل و xv6

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

بخش های سیستم عامل xv6 :

: Basic Headers

شامل مقادیر ثابت define شده و برخی تعریف type ها

mmu.h : شامل مقادیر define شده برای مدیریت حافظه و برخی استراکت ها

types.h : حاوی typedef های مورد نیاز

param.h, memlayout.h, asm.h : شامل define برخی مقادیر ثابت

defs.h : تعریف چند استراکت و تابع

x86.h : شامل توابعی برای استفاده از دستورات assembly

: Entering xv6

فراهم کردن امکانات لازم برای شروع برای آغاز به کار سیستم عامل

entry.s : هسته کرنل در این فایل که در این بخش است شروع به کار میکند. در این بخش دستور های

assembly نوشته شده که برنامه را به بخشی که با کد C زده شده منتقل میکند.

main.c : در کد C این فایل برنامه تمام موارد لازم را شروع میکند و از حافظه مورد نیاز استفاده میکند.

: Locks

مدیریت lock را بر عهده دارد. در این بخش قابلیت هایی مثل acquire و release برای lock ها قرار میدهد. در ضمن این بخش راهکاری برای debugging نیز در نظر گرفته است.

: Processes

در این بخش مدیریت پردازنده ها، دادن حافظه به پردازنده ها، ویژگی context switching و مدیریت و ایجاد پردازنده ها صورت میگیرد.

Proc.h, proc.c: مدیریت و ایجاد پردازنده ها در این قسمت انجام میگردد

kalloc.c : چگونگی قرار دادن حافظه به پردازنده ها پیاده سازی شده است.

swtch.s : در این بخش قابلیت context switching برای ذخیره کردن وضعیت حال حاضر ریجیستر

ها برای استفاده مجدد و بازیابی در اجراهای بعدی پیاده سازی شده است

: File System

تعریف ساختار buf ، sleeplock جهت قفل های طولانی مدت برای پردازنده ها ، همچنین تعریف ساختار stat برای ذخیره سازی اطلاعات فایل ها صورت گرفته است. Log کردن ساده که سیستم کال های همزمان فایل سیستم ها را قابل انجام میکند. همچنین پیاده سازی سطح پایین سیستم فایل ها و بررسی سطح بالای سیستم کال های سیستم فایل و اجرای برنامه مثل گرفتن حافظه مناسب و ... در این بخش انجام می شود.

log.c: در هر لحظه حداکثر یک transaction را مدیریت کند.

fs.c : شامل low level های file system می باشد

: Pipes

در این بخش تعریف ساختار pipes و تعریف توابع مربوط به open , close , read , write ... برای کار کردن با pipe قرار دارد.

: String Operations

شامل توابع مورد نیاز برای کار با استرینگ ها.

: Low level hardware

mp.h: مدیریت چندین پردازنده و جدول ورودی و خروجی

mp.c: پشتیبانی multi processor

ioapic.c : مدیریت وقفه های سخت افزاری و I/O برای سیستم های SMP

lapic.c : کنترلر پیشرفته وقفه های قابل برنامه نویسی غیر از I/O

Kbd.c, kbd.h: تعریف ثابت های خاص کیبورد

console.c: تعریف کنسول برای ورودی گرفتن با کیبورد یا پورت سریال و خروجی دادن روی صفحه نمایش

Uart.c : تعریف پورت سریال Intel 8250

**: User level**

در این قسمت قابلیت هایی مثل shell و اولین برنامه سطح کاربر اجرا می شود.

**init.c** : اولین برنامه سطح کاربر

**sh.c** : راه اندازی shell / command line

**usys.s** : تعریف سیستم کال های سیستم عامل در سطح یوزر

**initcode.s** : شامل کد های assembly جهت اجرا برنامه سطح کاربر موجود در فایل **init**.

**: Boot Loader**

عملیات های مورد نیاز جهت بوت شدن سیستم عامل در این بخش انجام می شود.

**bootmain.c** : توابع مورد نیاز برای عملیات بوت

**bootasm.s** : کد سطح ماشین که BIOS از اولین سکتور لود میکند که cpu را به حالت **protected**

**32-bit mode** می برد

**: Link**

کد اتصال دهنده ساده به هسته JOS

\*\*\*\*\*

فایل های مربوط به هسته سیستم عامل لینوکس در پوشه **/boot** قرار دارند.

فایل های سرایند لینوکس در پوشه **/usr/src** قرار دارند.

فایل های فایل سیستم در سیستم عامل لینوکس از **root ( / )** شروع میشوند.

**اجرای بوت لودر**

۸) در **xv6** در سکتور نخست دیسک قابل بوت، محتوای چه فایلی قرار دارد ؟

محتوای دو فایل **bootmain.c** و **bootasm.S**

خروجی دستور **make -n** به صورت زیر میباشد :

```
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
```

(۱۱) بوت سیستم توسط فایل‌های **bootasm.S** و **bootmain.c** صورت می‌گیرد. چرا تنها از کد C استفاده نشده است؟

برای اجرا کردن یک کد به زبان C نیاز به یک محیط اجرا داریم تا کد کامپایل شود و تبدیل به کد قابل اجرا اسمبلی برای سخت افزار بشود بنابراین ابتدا سیستم عامل کدی به زبان اسمبلی را اجرا میکند تا محیط لازم برای اجرا کد C فراهم شود و پردازنده را به حالت محافظت شده ۳۲ بیت برده و سپس به کد **bootmain** پرش میکند.

(۱۲) یک ثبات عام منظوره، یک ثبات قطعه، یک ثبات وضعیت و یک ثبات کنترلی در معماری x86 را نام برده و وظیفه هر یک را به طور مختصر توضیح دهید.

**ثبات عام منظوره:** xv6 دارای ۸ ثبات عام منظوره است که به نام‌های `eax, %ebx, %ecx, %edx, %edi, %esi, %ebp, %esp` می‌باشند این ثبات‌ها برای نگهداری دیتا یا آدرس به کار می‌روند.

**ثبات قطعه:** آدرس شروع دیتا را ذخیره میکند که بعداً با یک `offset` جمع می‌شود تا محل دقیق دیتا مشخص شود شامل `cs, %ds, %es, %fs, %gs, %ss`

**ثبات وضعیت:** حاوی اطلاعات مربوط به پردازنده است که بیت‌های آن توسط ماشینی که روی پردازنده اجرا می‌شود، نوشته یا خوانده می‌شود. `flag`‌هایی مثل `zero, carry, sign` در x86 نمونه‌ای برای این ثبات‌ها می‌باشند.

**ثبات کنترلی:** نقش تغییر یا کنترل رفتار کلی پردازنده یا دستگاه‌های دیجیتال را ایفا می‌کنند. `cr0, %cr2, %cr3, %cr4` از نمونه این ثبات‌ها هستند. وظیفه تغییر مدل آدرس دهی، کنترل `interrupt` کنترل `paging` و هم پردازنده‌ها را دارند.

۱۴) مدهای مختلف در پردازنده های x86 را مختصراً توضیح دهید.

مد های اصلی:

16-bit real mode

در این مد آدرس مورد دسترسی در برنامه مستقیم به آدرس فیزیکی نگاشت پیدا میکند.

16-bit protected mode and 32-bit protected mode

در این مد آدرس مورد دسترسی در برنامه به واسطه یک جدول به آدرس فیزیکی حافظه نگاشت پیدا میکند.

مد ۳۲بیتی قابلیت صفحه گذاری و حافظه مجازی را بصورت ۳۲بیتی برای ما فراهم می کند.

16-bit long mode

در این مد رجیسترهای ۶۴بیتی استفاده می شوند. این مد یک روش آدرس دهی جدید به نام RIP-relative هم دارد.

۱۸) چرا از همان ابتدا هسته در حافظه توسط BIOS بارگذاری نمیشود؟ علت اصلی را بیان نمایید.

چون در این حالت با تغییر دیسک با file system نیاز به تغییر دستورات در BIOS است . با این کار BIOS نیاز ندارد که بداند هسته کجاست و اینکار توسط entry.S انجام میشود.

## اجرای هسته xv6

۱۹) چرا آدرس اشاره شده فیزیکی است؟

زیرا اگر این بخش بصورت مجازی بود ترجمه صفحه در کد بوت اجرا نمی شد و باید یک بخش فیزیکی در نظر میگرفتیم تا این بخش مجازی را مشخص کند به همین دلیل به ادرس فیزیکی نیاز است.



۲۲) علاوه بر صفحه بندی در حد ابتدایی از قطعه بندی به منظور حفاظت هسته استفاده خواهد شد. این عملیات توسط `seginet()` انجام میگردد. همانطور 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 داده‌های سطح کاربر پرچم `SEG_USER` تنظیم شده است. چرا؟

زیرا بتوان تفاوتی بین پردازنده های سطح کاربر و پردازنده های سطح هسته ایجاد نمود. از انجایی که محتوای هر دو این پردازنده ها در یک فضای فیزیکی قرار گرفته اند، با این کار می توان تشخیص داد که آن داده ها، داده های سطح کاربر می باشند و اجازه دسترسی به هسته را ندارند.

### اجرای نخستین برنامه سطح کاربر

۲۳) مدیریت برنامه های سطح کاربر مستلزم ارائه انتزاعاتی برای ایجاد تمایز میان این برنامه ها و برنامه مدیریت آنها است. جهت نگهداری اطلاعات مدیریتی برنامه های سطح کاربر ساختاری تحت عنوان `struct proc` (خط ۲۳۳۶) ارائه شده است. اجزای آن را توضیح داده و ساختار معادل آن در سیستم عامل لینوکس را بیابید.

`chan`: به معنای خوابیدن پردازنده در حالت صفر

`context`: نگهداری `context switching`

`cwd`: پوشه فعلی

`killed`: به معنای `kill` شدن پردازنده در حالت صفر

`Kstack`: پایین استک کرنل برای پردازنده

`name`: نام پردازنده

`ofile`: فایل های باز شده توسط پردازنده

`parent`: سازنده (پدر) پردازنده

`Pgdir`: پوینتر به `page table`

`pid`: عدد نظیر شده به پردازنده

state : وضعیت پردازنده

SZ: سایز حافظه پردازنده به بایت

tf: trap برای system call فعلی

ساختار معادل این استراک در linux در لینک زیر آمده است:

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

## ۲۷) فراخوانی سیستمی exec() در لینوکس چه وظیفه ای دارد؟

دستور exec در لینوکس برای اجرای دستور از خود bash استفاده می شود. این دستور فرآیند جدیدی ایجاد نمی کند ، بلکه دستور bash را با دستور جایگزین می کند. اگر دستور exec موفقیت آمیز باشد ، به فرایند فراخوانی بر نمی گردد. دستور exec فرآیند جدیدی ایجاد نمی کند. وقتی دستور exec را از ترمینال اجرا می کنیم ، فرایند ترمینال در حال اجرا با فرمان جایگزین می شود که به عنوان آرگومان دستور exec ارائه شده است. وقتی اسکریپت های Bash ایجاد می کنیم ، ممکن است بخواهیم خروجی همه دستورات echo را به یک فایل log بدون تغییر صریح اپراتور تغییر مسیر و نام فایل log بعد از هر دستور echo تغییر مسیر دهیم. دستور Bash exec یک ابزار قدرتمند داخلی است که می تواند برای این منظور استفاده شود.

## اشکال زدایی

### روند اجرای GDB

۱) برای مشاهده Breakpoint ها از چه دستوری استفاده میشود؟

دستور maint info breakpoints

۲) برای حذف یک Breakpoint از چه دستوری و چگونه استفاده میشود؟

دستور clear filename:line که filename نام فایل و line نشان دهنده خط مورد نظر است.

## کنترل روند اجرا و دسترسی به حالت سیستم

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

این دستور نشان می دهد که برنامه چگونه به وضعیت فعلی رسیده است و توابعی که صدا زده شده و در استک اضافه شده اند را نشان می دهد.

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

دستور x طبق آدرس کار می کند و مقدار را نمایش می دهد ولی دستور print می تواند یک expression بگیرد و محتوای آن را نشان دهد. این دو دستور در نحوه نمایش اطلاعات نیز با یکدیگر تفاوت دارند.

با استفاده از دستور info register name که در آن نام ثبات به عنوان آرگومان داده می شود.

### ۵) برای نمایش وضعیت ثباتها از چه دستوری استفاده میشود؟ متغیرهای محلی چطور؟ همچنین در گزارش خود توضیح دهید که در معماری x86 رجیسترهای edi و esi نشانگر چه چیزی هستند؟

دستور info registers

```
(gdb) info registers
eax      0x0
ecx      0x0
edx      0x603
ebx      0x0
esp      0x0
ebp      0x0
esi      0x0
edi      0x0
ebp      0xffff0
eflags   0x2 [ IOPL=0 ]
cs       0xf000 61440
ss       0x0
ds       0x0
es       0x0
fs       0x0
gs       0x0
fs_base  0x0
gs_base  0x0
k_gs_base 0x0
cr0      0x6000010 [ CD NW ET ]
cr2      0x0
--Type <RET> for more, q to quit, c to continue without paging--
cr3      0x0 [ POBR=0 PCID=0 ]
cr4      0x0
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_int64 = {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_int64 = {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_int64 = {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_int64 = {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_int64 = {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_int64 = {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_int64 = {0x0, 0x0}, uint128 = 0x0
xmm7     {v4.float = {0x0, 0x0, 0x0, 0x0}, v2.double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 12 times>, 0x80, 0x1f, 0x0, 0x0}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}}
, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
mxcsr    0x1f80 [ IM DM ZM OM UM PM ]
(gdb)
```

دستور info locals

```
(gdb) info locals
n = <optimized out>
```

esi نشان دهنده source و edi نشان دهنده destination در عملیات هایی که فقط با این دو دستور انجام می شود می باشند.

## اشکال زدایی در سطح کد اسمبلی

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

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

۲ - نحوه و زمان تغییر مقدار متغیرهای درونی

در این ساختار سه متغیر W و r و e وجود دارد که دو متغیر W و r ابتدای خط ورودی را در بافر نشان میدهد.

متغیر e نیز نشان دهنده مقدار انتهایی خط در حال تایپ شدن در بافر است.

۷) خروجی دستورهای layout src و layout asm در TUI چیست؟

دستور layout src برنامه را در حالت کد سورس آن نشان میدهد.

دستور layout asm برنامه را در حالت کد اسمبلی نشان میدهد.

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

با استفاده از دستورات down و up