

گزارش کار آزمایش اول آزمایشگاه سیستم عامل

نوید اخوان 810898036

روژین رستگارپور 810898042

زهرا فراهانی 810197550

آشنایی با سیستم عامل xv6

1: معماری سیستم عامل xv6 از نوع monolithic است (مانند بسیاری از سیستم عامل های دیگر Unix) در این نوع معماری ، هسته مسئول سیستم عامل است یعنی interface هسته مربوط به interface سیستم عامل است و هسته به طور کامل سیستم عامل را پیاده سازی می کند. در این مدل تمام بخش های privilege سخت افزاری در یک قسمت باهم قرار دارند

2: پردازنده xv6 از 2 قسمت user-space memory که شامل دستورات ، داده ها و استک است و per-process state private to the kernel شده است. xv6 می تواند زمان پردازش ها را به اشتراک بگذارد (time sharing) و به طور شفاف CPU های موجود را بین مجموعه ای از process ها که در انتظار اجرا هستند، تغییر می دهد. وقتی یک process اجرا نمی شود، xv6 رجیستر CPU خود را ذخیره می کند تا در اجرای بعدی، آن هارا بازیابی کند.

4: در exec system call ، حافظه process فراخوانی با یک تصویر حافظه جدید بارگیری شده از فایل ذخیره شده در سیستم فایل، جایگزین می شود. فایل باید دارای فرمت خاصی باشد که مشخص می کند کدام قسمت از فایل دستورات قرار دارند، کدام قسمت دیتا قرار دارد و در کدام قسمت دستورات آغاز میشوند و ... در xv6 از قالب ELF استفاده میشود. وقتی exec موفق شد، دیگر به calling program بر نمی گردد. در عوض دستورات بارگزاری شده از فایل در نقطه ی ورود اعالم شده و در ELF header شروع به اجرا می کنند.

یک فرآیند ممکن است با استفاده از فراخوانی سیستم fork، فرآیند جدیدی ایجاد کند. fork یک فرآیند جدید به نام فرزند فرزند ایجاد می کند که دقیقاً همان محتویات حافظه را در فرآیند فراخوانی دارد که به آن فرآیند والد می گویند. fork هم در والدین و هم در فرزند برمی گردد. fork در والد، pid فرزند را برمی گرداند.

اگر fork و exec ادغام نشوند، shell می تواند یک فرزند را fork کند، از open، close، dup در فرزند برای تغییر توصیفگرهای استاندارد فایل (file descriptors) ورودی و خروجی استفاده کند و سپس اجرا کند. هیچ تغییری در برنامه در حال اجرا مورد نیاز نیست. اگر fork و exec در یک فراخوانی سیستمی ترکیب شوند، طرح دیگری (احتمالاً پیچیده تر) برای تغییر مسیر ورودی و خروجی استاندارد مورد نیاز shell خواهد بود، یا خود برنامه باید نحوه تغییر مسیر I/O را درک کند.

8: عبارت UPROGS مخفف user programs می باشد و مقابل آن نام برنامه ها نوشته شده است. برای اضافه کردن برنامه به xv6 باید نام فایل آن را در Makefile در قسمت UPROGS اضافه کنیم تا امکان اجرای آن وجود داشته باشد. عبارت ULIB مخفف user library files می باشد و مقابل آن نام library ها نوشته می شود.

11: فایل مربوط به بوت از نوع entry.S است.

12: Objcopy محتویات یک object file را در یک object file دیگر کپی می‌کند. به طوری که فرمت فایل‌های مبدا و مقصد می‌تواند متفاوت باشد. زیرا از کتابخانه‌ی BFD استفاده می‌کند و فرمت‌های مختلف را می‌شناسد.

14:

- I. ثبات عام منظوره: در معماری x86، 8 نوع از این رجیستر وجود دارد: esp, ebp, esi, edi, edx, ecx, ebx, eax. برای محاسبات و ذخیره‌ی آدرس‌ها استفاده می‌شود.
- II. ثبات قطعه: 6 نوع از این رجیستر وجود دارد: cs, ds, es, fs, gs, ss. پوینتر به قطعه‌های مختلف حافظه process را نگهداری می‌کنند که از موارد استفاده‌ی آن‌ها می‌توان به مقدار پوینتر سر پشته اشاره کرد.
- III. ثبات وضعیت: مانند (ESR) برای آزمایش شرایط مختلف در عملیات، مانند "is the result zero"، "is the result negative" می‌شود. مانند FLAG.
- IV. ثبات کنترلی: 5 نوع از این رجیستر وجود دارد: cr0, cr1, cr2, cr3, cr4. حاوی مقداری به نام آدرس خطی خطا (PFLA) است. وقتی خطایی در صفحه رخ می‌دهد، آدرسی که برنامه سعی کرده به آن دسترسی داشته باشد در رجیستر CR2 ذخیره می‌شود.

18: در لینک زیر قابل مشاهده است:

<https://www.cs.montana.edu/courses/spring2005/518/Hypertextbook/jim/code/entry.S.htm>

19: اگر این آدرس فیزیکی نبود، برای یافتن آدرس فیزیکی‌اش به یک جدول نیاز داشتیم در حالی که در هنگام بوت شدن، مکانیزم ترجمه صفحه و table page غیر فعال است. پس باید در جای ثابتی از حافظه فیزیکی قرار بدهیم تا مسئولیت کنترل سایر آدرس‌های مجازی را داشته باشد.

22: برای مشخص شدن اینکه داده‌های موجود داده‌های سطح کاربر هستند و اجازه دسترسی به هسته برای آنها صادر نمیشود.

23:

مقدار حافظه‌ای که برنامه در سیستم دارد.	unit sz
پوینتر به page table ای که به برنامه اختصاص یافته.	pde_t* pgdir
اشاره گر به پایین kernel stack برای این برنامه.	char *kstack
وضعیت فرآیند‌های در حال اجرا را نگهداری میکند.	enum procstate state
id فرایند را نگهداری میکند.	int pid
اشاره گر به struct proc فرایند پدر.	struct proc *parent

اشاره گر به trap سیستم کال کنونی.	<code>struct trapframe *tf</code>
اشاره گری که باید برای اجرای ادامه برنامه باید به آن <code>swtch()</code> کرد	<code>struct context *context</code>
نشان دهنده رخ دادن <code>sleep(chan)</code> پردازنده. (اگر صفر نباشد رخ داده است)	<code>void *chan</code>
نشان میدهد که برنامه <code>kill</code> شده است یا خیر (اگر غیر صفر باشد رخ داده است)	<code>int killed</code>
بازفایل کردن	<code>struct file *ofile[NOFILE]</code>
اشاره گر برای دایرکتوری کنونی.	<code>struct inode *cwd</code>
نام فرایند در حال اجرا (که در <code>debugging</code> به کار میرود)	<code>char name[16]</code>

در لینوکس ساختار معادل `struct task_struct` نام دارد که در لینک زیر قابل مشاهده است:

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

27: زمانبند از طریق `p->context` اجرا میشود.

چاپ نام اعضای گروه

```

QEMU
Machine  View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ _

```

sort-string

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ sort_string oslabproj1
$ cat sort_string.txt
labjloopr
$ sort_string maoepasd
$ cat sort_string.txt
aademops
$
```

left arrow

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ OS Lab Group 21
```

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ OS Lab Group 21_
```

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ OS Lab Group 2 1
```

right arrow

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ OS Lab Group_2 1
```

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ OS Lab Group 2_1
```

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ OS Lab Group _1
```

up arrow

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ sort_string OperatingSystem
$ cat sort_string.txt
OSaeegimprstty
$
```

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ sort_string OperatingSystem
$ cat sort_string.txt
0Saeegimprstty
$ cat sort_string.txt_
```

```
QEMU
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

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 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #21:
1. Zahra Farahani
2. Rojin Rastegar Pour
3. Navid Akhavan
$ sort_string OperatingSystem
$ cat sort_string.txt
0Saeegimprstty
$ sort_string OperatingSystem_
```

Debugging

اشکال زدایی

1: تعداد و اطلاعات آن را پس از قرار دادن هر breakpoint نمایش می دهد. پس از نوشتن continue، فرایند را شروع می کند و اطلاعات breakpoint را نمایش می دهد. پس از هر استپ اینکه در کجای فرایند هستیم را نمایش می دهد و در صورت وجود توابع دیگر، می توان برای آن ها هم به همین صورت breakpoint قرارداد.

2: breakpoint·clear مورد نظر را پاک می کند و دستور delete تمامی breakpoint ها را پاک می کند.

3: خروجی، وضعیت stack را مشخص می کند و در صورتی که stack موجود نباشد:

```
Undefined command: "". Try "help".
(gdb) bt
No stack.
(gdb)
```

4:

X: حافظه را بررسی می کند. عبارت x/ FMT یک آدرس به ما خواهد داد و این آدرس جایی است که باید بررسی شود

Print: مقادیر عبارت EXP را چاپ می کند. می توان متغیرهایی که قابل دسترسی اند را در عبارت استفاده کرد و نتیجه را چاپ کرد و همچنین متغیرها یا شامل تمام آن هایی است که در فایل موجودند، یا به صورت global در آن اسکوپ تعریف شده اند.

-دستورات , x , d , u مقادیر را در فرمت خاص نمایش میدهند. برای مثال x در قالب هگزادسیمال مقادیر صحیح را نمایش میدهد. این دستور ادرس خانه حافظه را میگیرد. در مثال زیر در همان تابع merge sort ارایه به شکل زیر بوده است.

```
$ arr = {12, 11, 13, 5, 6, 7, 15, 84, 34, 62, 84}
```

```
$ x arr+0
```

```
$ 0x7fffffffddb0: 0x0000000c
```

```
$ x arr+5
```

```
$ 0x7fffffffddc4: 0x00000007
```

و اما دستور print در صورتی که نام ارایه را به آن بدهید کل ارایه را بصورت یکجا در فرمت دسیمال نمایش میدهد

```
. $ print arr
```

```
$1 = {12, 11, 13, 5, 6, 7, 15, 84, 34, 62, 84}
```

اما این دستور ویژگی های دیگری هم دارد و به شکل زیر نیز قابل استفاده است

```
. $ print [Expression]
```

```
$ print $[Previous value number]
```

```
$ print {[Type]}[Address]
```

```
$ print [First element]@[Element count]
```

```
$ print /[Format] [Expression]
```

در کل استفاده از print برای نمایش متغیر ها بهتر است.

در تصویر زیر چند استفاده از آن را میبینیم:

```
(gdb) info locals
   = {12, 11, 13, 5, 6, 7, 15, 84, 34, 62, 84}
   = 11
(gdb) print arr
$1 = {12, 11, 13, 5, 6, 7, 15, 84, 34, 62, 84}
(gdb) print arr[2]
$2 = 13
(gdb) print /x arr[2]
$3 = 0xd
(gdb) print /t arr[2]
$4 = 1101
(gdb) print (void *)arr[2]
Cannot access memory at address 0xd
(gdb) print (void *)arr@2
$5 = {
}
(gdb) print $2
$6 = 13
(gdb) print /s arr
$7 = {12, 11, 13, 5, 6, 7, 15, 84, 34, 62, 84}
```

:5

برای نمایش وضعیت و محتوای ثبات ها از info registers استفاده میکنیم.

```
(gdb) info registers
eax        0x1          1
ecx        0x1          1
edx        0x0          0
ebx        0x00112d20    -2146357984
esp        0x003bef70    0x003bef70
ebp        0x003bef78    0x003bef78
esi        0x00112850    -2146359216
edi        0x00112854    -2146359212
eip        0x00103ef5    0x00103ef5 <mycpu+21>
eflags     0x2          2 [ IOPL=0 ]
eip_base   0x0          0
cs_base     0x0          0
ds_base     0x0          0
ss_base     0x0          0
fs_base     0x0          0
gs_base     0x0          0
cr0        0xe0010011    [ PG CD NW WP ET PE ]
cr2        0x0          0
cr3        0x1fff000    [ PGB=0 PCID=0 ]
cr4        0x10         [ PSE ]
cr8        0x0          0
efer       0x0          [ ]
eax0       [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]
eax1       [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]
eax2       [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]
eax3       [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]
eax4       [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]
eax5       [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]
eax6       [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]
eax7       [v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <-repeats 12 times>, 0x00, 0x1f, 0x0, 0x0}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1f00, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x1f00}, v2_int64 = {0x0, 0x1f0000000000}, uint128 = 0x1f00000000000000000000000000000000]
eaxcsr     0x1f00         [ IN DN ZH GH LM PM ]
(gdb)
```

برای نمایش وضعیت و محتوای متغیر های محلی از دستور info locals استفاده میکنیم. در تصویر زیر دستور locals info را برای تابع main اجرا کرده ایم و مقادیر متغیر های محلی را میبینیم

```
(gdb) c
The program is not being run.
(gdb) c
The program is not being run.
(gdb) clear
Deleted breakpoint 1
(gdb) b mergeSort
Breakpoint 2 at 0x00103ef5 : file mergeSort.c, line 60.
(gdb) r
Starting program: /home/alighadyani/UT/DA/a.out
Given array is

Breakpoint 2, 0x00103ef5 in mergeSort (arr=0x7ffffffffffddb0, arr_size=0, n=10) at mergeSort.c:60
60      begin = end
(gdb) bt
#0  0x00103ef5 in mergeSort (arr=0x7ffffffffffddb0, arr_size=0, n=10) at mergeSort.c:60
#1  0x00103ef5 in mergeSort (arr=0x7ffffffffffddb0, arr_size=0, n=10) at mergeSort.c:86
(gdb) up
#1  0x00103ef5 in mergeSort (arr=0x7ffffffffffddb0, arr_size=0, n=10) at mergeSort.c:86
86      mergeSort(arr, arr_size, n/2);
(gdb) info locals
arr = {12, 11, 13, 5, 6, 7, 15, 84, 34, 62, 84}
arr_size = 11
n = 11
(gdb)
```

در معماری x86 ثبات های EDI و ESI به ترتیب نشانگر اندیس مقصد و مبدا در عملیات هایی است که بر روی رشته ها انجام میشود.

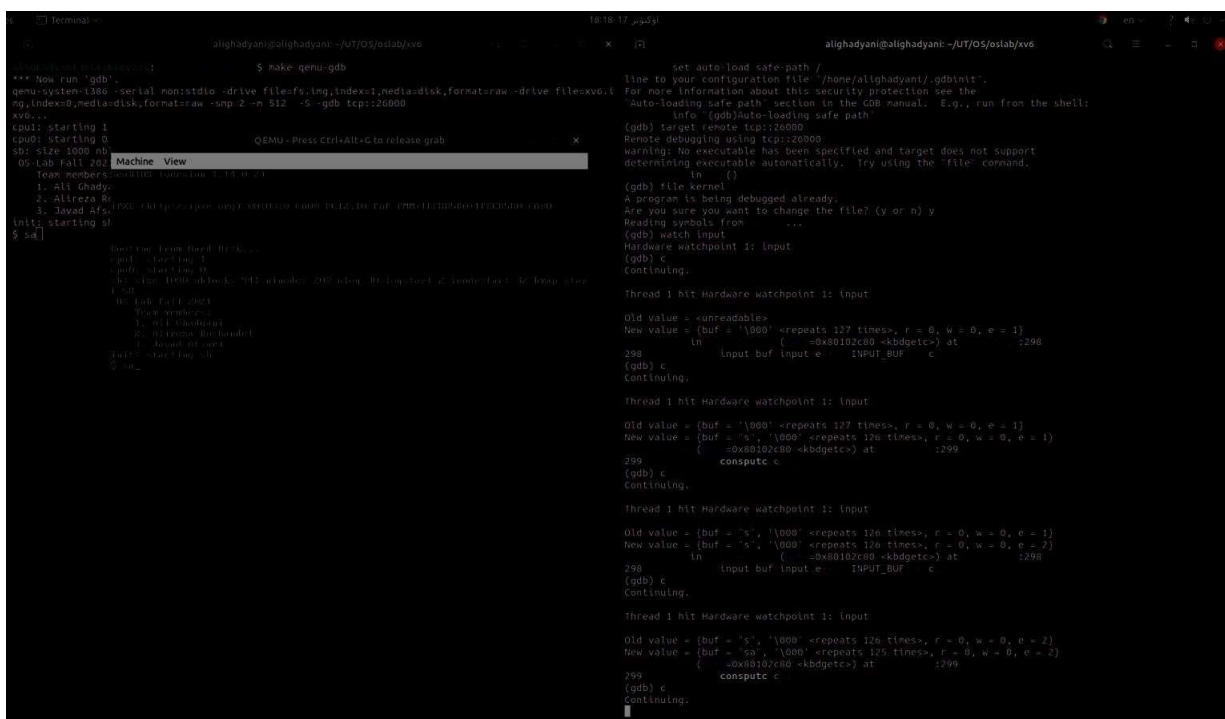
6: در input یک آرایه از کاراکتر به نام buf به اندازه BUF_SIZE قرار دارد که مانند یک پشته کاراکتر هایی که کاربر در ترمینال وارد میکند در آن push میشود. متغیری که به بالای پشته اشاره میکند input.e است.

هر بار که کلید Enter را می‌فشاریم کاراکتر “\n” در پشته push میشود و دستورات اجرا میشوند و دیگر نیازی به کاراکتر های وارد شده تا وارد کردن کلید Enter نیست اما خالی کردن پشته زمانبر است. برای همین بجای خالی کردن آن تعداد کاراکتر هایی که حذف باید بشوند را در input.r ذخیره کرده و ابتدای پشته را در input.w ذخیره میکنیم.

به این ترتیب کاراکتر های ورودی در خط جدید از خانه شماره input.w به بعد buf نوشته میشوند و خانه های قبل آن به تعداد input.r در صورت نیاز جایگزین میشوند.

اندازه buf حداکثر تعداد کاراکتر قابل وارد کردن در ترمینال است.

در تصویر زیر با استفاده از GDB و دستور watch input تغییرات آنرا هنگام وارد کردن دستورات در ترمینال مانیتور کرده ایم.



```
alighadyani@alighadyani: ~/UT/OS/oslab/xv6
$ make qemu-gdb
*** Now run 'gdb' ***
gdb> set auto-load safe-path /
gdb> line to your configuration file "/home/alighadyani/.gdbinit".
gdb> For more information about this security protection see the
gdb> "Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
gdb> info "(gdb)Auto-loading safe path"
gdb> (gdb) target remote tcp://26000
gdb> Remote debugging using tcp://26000
warning: no executable has been specified and target does not support
gdb> determining executable automatically.  Try using the 'file' command.
gdb> file /bin/ls
gdb> A program is being debugged already.
gdb> Are you sure you want to change the file? (y or n) y
gdb> Reading symbols from /bin/ls...
gdb> (gdb) watch input
gdb> Hardware watchpoint 1: input
gdb> c
gdb> Continuing.

Thread 1 hit Hardware watchpoint 1: input

Old value = <unreadable>
New value = (buf = '\000' <repeats 127 times>, r = 0, w = 0, e = 1)
  298     ln      ( _0x80102c80 <+kdbggetc>) at      :298
        input buf input.e = INPUT_BUF  c
gdb> c
gdb> Continuing.

Thread 1 hit Hardware watchpoint 1: input

Old value = (buf = '\000' <repeats 127 times>, r = 0, w = 0, e = 1)
New value = (buf = 's', '\000' <repeats 126 times>, r = 0, w = 0, e = 1)
  298     ln      ( _0x80102c80 <+kdbggetc>) at      :298
        consputc c
gdb> c
gdb> Continuing.

Thread 1 hit Hardware watchpoint 1: input

Old value = (buf = 's', '\000' <repeats 126 times>, r = 0, w = 0, e = 1)
New value = (buf = 's', '\000' <repeats 126 times>, r = 0, w = 0, e = 2)
  298     ln      ( _0x80102c80 <+kdbggetc>) at      :298
        input buf input.e = INPUT_BUF  c
gdb> c
gdb> Continuing.

Thread 1 hit Hardware watchpoint 1: input

Old value = (buf = 's', '\000' <repeats 126 times>, r = 0, w = 0, e = 2)
New value = (buf = 'sa', '\000' <repeats 125 times>, r = 0, w = 0, e = 2)
  298     ln      ( _0x80102c80 <+kdbggetc>) at      :298
        consputc c
gdb> c
gdb> Continuing.
```

در تصویر فوق مراحل اضافه کردن کاراکتر ها به پشته را مشاهده میکنید که در آن ابتدا input.e که نشانگر بالای پشته است آپدیت میشود و سپس کاراکتر ورودی به پشته اضافه میشود.

در تصویر زیر با وارد کردن BACKSPACE مشاهده میشود که تنها input.e یک واحد کاسته میشود.

در تصویر زیر به تعدادی کاراکتر "a" وارد شد تا buf پر شود و سپس کلید Enter فشرده شد و تعدادی کاراکتر "s" همطور که میبینید در buf جایگزین کاراکتر های قبل شد و سپس کلید Enter فشرده شد و کاراکتر "k" وارد شده است. طی این مراحل نکته حایز اهمیت آن است که هیچ یک از input.e, input.w, input.r به نحوی که در محدوده اندازه buf قرار بگیرند تغییر نکرده است و مقداری بیش از اندازه buf اختیار کرده اند. پس این بسیار مهم است که هنگام استفاده از این متغیر ها حتما MOD BUF_SIZE آنها را بگیریم تا با مشکل segmentation error برنخوریم

7: دستور layout src رابط کاربری متنی(GDB(TUI را باز میکند و همچنین source code قابل مشاهده خواهد بود. دستور layout asm مشابه دستور قبل است با این تفاوت که assembly code برنامه نوشته شده را نمایش میدهد. در هر دوی این دستورات در صورتی که breakpoint داشته باشیم خط مربوط به آن highlight میشود. دستور layout دارای mode های دیگری مانند split که ترکیب دو دستور بالا است نیز است.

8: دستورات up و down برای حرکت میان توابعی است که یکدیگر را فراخوانی کرده اند. در مثال merge sort مثلا بعد از چند فراخوانی نیاز به چک کردن وضعیت تابع مرحله قبل داریم با دستور up به تابعی که در آن تابع فعلی را فراخوانی کرده ایم برمیگردیم و بالعکس با دستور down

```
Breakpoint 1, 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=1) at mergeSort.c:60
60      begin = end;
(gdb) bt
#0 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=1) at mergeSort.c:60
#1 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=2)
    at mergeSort.c:64
#2 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=5)
    at mergeSort.c:64
#3 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=10)
    at mergeSort.c:64
#4 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=10)
    at mergeSort.c:86
(gdb) up
#1 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=2)
    at mergeSort.c:64
64      mergeSort(array, begin, mid);
(gdb) up
#2 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=5)
    at mergeSort.c:64
64      mergeSort(array, begin, mid);
(gdb) down
#1 0x7ffffffffffdb0 (array=0x7ffffffffffdb0, begin=0, end=2)
    at mergeSort.c:64
64      mergeSort(array, begin, mid);
(gdb)
```

در تصویر زیر نمونه ای از کاربرد این دستورات را مشاهده میکنید. پس از اجرا هر کدام از دستورات up یا down از دستور info locals استفاده کردیم تا وضعیت متغیر های محلی را نمایش دهیم.

```

#1      in mergeSort (array=0x7fffffffddb0, begin=0, end=2)
    at mergeSort.c:64
#2      in mergeSort (array=0x7fffffffddb0, begin=0, end=5)
    at mergeSort.c:64
#3      in mergeSort (array=0x7fffffffddb0, begin=0, end=10)
    at mergeSort.c:64
#4      in main () at mergeSort.c:86
(gdb) up
#1      in mergeSort (array=0x7fffffffddb0, begin=0, end=2)
    at mergeSort.c:64
#4      mergeSort array, begin, mid);
(gdb) up
#2      in mergeSort (array=0x7fffffffddb0, begin=0, end=5)
    at mergeSort.c:64
#4      mergeSort array, begin, mid);
(gdb) info locals
mid = 2
(gdb) down
#1      in mergeSort (array=0x7fffffffddb0, begin=0, end=2)
    at mergeSort.c:64
#4      mergeSort array, begin, mid);
(gdb) info locals
mid = 1
(gdb)

```

<https://gitlab.com/NavidAkhavan/os-lab>