به نام خدا

گزارش اول آزمایشگاه سیستم عامل

سید علیرضا میرشفیعی - ۸۱۰۱۰۱۵۳۲
امیر حسین صفری - ۸۱۰۱۰۱۵۷۱
محمد صدرا عباسی - ۸۱۰۱۰۱۴۶۹

بخش تشریحی

سوال ۱

سه وظیفه اصلی سیستم عامل:
۱. واسط بین نرم‌افزار و سخت‌افزار : سیستم‌عامل پلی میان برنامه‌های کاربردی و کاربران در لایه‌ی بالا و سخت‌افزار در لایه‌ی پایین است.
۲. مدیریت منابع سخت‌افزاری : اشتراک‌گذاری و تخصیص بهینه‌ی منابع.
۳. مدیریت نیازهای نرم‌افزاری و تعامل برنامه‌ها : پنهان‌سازی پیچیدگی‌های سخت‌افزار، ارائه‌ی سرویس‌های سطح بالا، و کنترل ارتباط بین برنامه‌ها.

سوال ۲

توضیح گروه بندی فایل های اصلی XV6 :

۱. Header Files :

- Basic Headers : تعریف انواع داده، ثابت‌ها و تنظیمات کلی (types.h, param.h).

۲. Kernel Files

- System Calls : پیاده‌سازی توابعی که برنامه‌ها برای تعامل با هسته استفاده می‌کنند (sysproc.c).
- Entering xv6 : مقداردهی اولیه و راه‌اندازی هسته (main.c).

- String Operations : توابع پردازش رشته‌ها (string.c).
- Low-Level Hardware : مدیریت ورودی/خروجی مانند صفحه‌کلید و کنسول (console.c, kbd.c).
- Locks : مدیریت همگام‌سازی پردازه‌ها (spinlock.c).
- Processes : شامل مدیریت حافظه مجازی، زمان‌بندی و تخصیص حافظه (proc.c, vm.c).
- Pipes : مدیریت ارتباط بین پردازه‌ها (pipe.c).
- Bootloader : بارگذاری هسته در حافظه (bootasm.S, bootmain.c).
- Link : مدیریت محل ذخیره کرنل در حافظه (entry.S).

۳. File System Files

- Filesystem : مدیریت فایل‌ها و دایرکتوری‌ها (fs.c, file.c).

۴. User-Level

- User-Level: برنامه‌های سطح کاربر (init.c, sh.c).

نام پوشه‌های اصلی در سیستم‌عامل لینوکس و محتویات آنها:

1. فایل‌های هسته : usr/src/linux/ یا lib/modules/$(uname -r)/build/
   شامل سورس‌کد کرنل، ماژول‌ها، و مدیریت سخت‌افزار، پردازنده و حافظه.

2. فایل‌های سرایند : /usr/include
   شامل فایل‌های .h برای تعریف انواع داده، توابع و سیستم‌کال‌ها.
3. فایل‌های سیستم‌فایل : /proc/, /home, /mnt, /var, /etc
   /etc (پیکربندی سیستم)، /var (لاگ‌ها و داده‌های متغیر)، /mnt (نقطه مانت)، /home (فایل‌های کاربران)، /proc (سیستم‌فایل مجازی کرنل).

سوال ۳

دستور make -n فقط دستورات کامپایل و لینک کردن را نمایش می‌دهد اما هیچ فایلی ایجاد نمی‌کند. هدف نهایی این دستورات، ساخت xv6.img است که شامل کرنل و بوت‌لودر می‌شود. برای ساخت فایل نهایی کرنل از دستور make kernel استفاده می‌شود.

```
sadra@sadra-ThinkPad-X1-Yoga-4th:~/Documents/Codes/xv6-public$ make -n
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -fno-pic -O -nostdinc -I. -c bo
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -fno-pic -nostdinc -I. -c boota
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
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o bio.o bio.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o console.o console.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o exec.o exec.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o file.o file.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o fs.o fs.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o ide.o ide.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o ioapic.o ioapic.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o kalloc.o kalloc.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o kbd.o kbd.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o lapic.o lapic.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o log.o log.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o main.o main.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o mp.o mp.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o picirq.o picirq.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o pipe.o pipe.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o proc.o proc.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o sleeplock.o sleeplock.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o spinlock.o spinlock.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o string.o string.c
gcc -m32 -gdwarf-2 -Wa,-divide   -c -o swtch.o swtch.S
```

سوال ۴

۱. متغیر UPROGS

این متغیر لیست برنامه‌های کاربری (User Programs) را مشخص می‌کند که باید برای اجرا در xv6 کامپایل شوند.

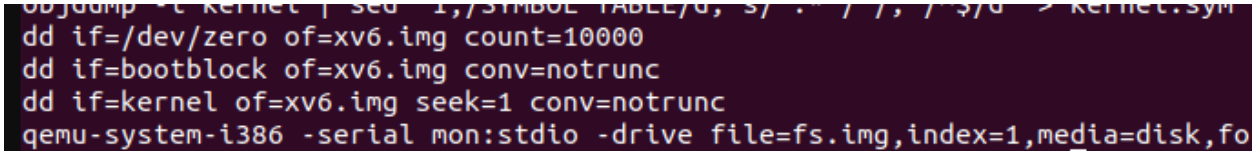
۲. متغیر ULIB

این متغیر شامل کتابخانه‌های موردنیاز برای برنامه‌های کاربری است.

سوال ۵

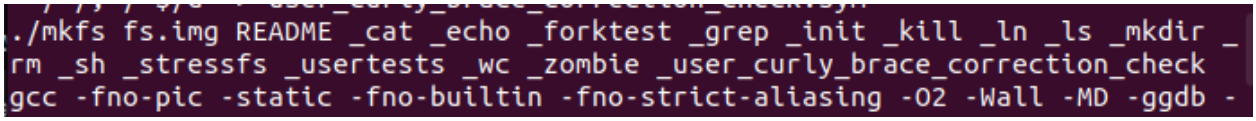
هنگام اجرای make -n qemu، شبیه‌ساز QEMU با دو دیسک ورودی اجرا می‌شود:

xv6.img : شامل بوت‌لودر و کرنل xv6.



خط اول: یک دیسک خام ۱۰,۰۰۰ بلاکی (xv6.img) ایجاد می‌کند.
خط دوم: بوت‌لودر (bootblock) را در ابتدای دیسک کپی می‌کند.
خط سوم: کرنل (kernel) را از بلاک ۱ به بعد در دیسک ذخیره می‌کند.

fs.img : شامل فایل‌سیستم و برنامه‌های کاربری



این دیسک شامل فایل‌های مهم مانند README و برنامه‌های سطح کاربر (sh, ls, echo, wc و...) می‌شود

سوال ۸

objcopy برای حذف اطلاعات اضافی (مانند هدر ELF و متادیتاهای اشکال‌زدایی) و تبدیل فایل‌ها به فرمت باینری خام استفاده می‌شود. این کار باعث می‌شود که فایل‌های تولیدشده، مستقیماً توسط بوت‌لودر یا کرنل در حافظه بارگذاری و اجرا شوند

سوال ۱۳

۱. به عنوان قراردادی میان بسیاری از سیستم های عامل آنها را در این آدرس load می کنند

۲. رزرو بودن آدرس‌های پایین برای BIOS و Bootloader
حافظه زیر 1MB معمولاً توسط BIOS، کارت گرافیک و سخت‌افزارها اشغال می‌شود و آدرس 0x100000 اولین بخش خالی و ایمن برای بارگذاری کرنل است

۳. وجود حافظه کافی برای load کردن kernel که بتواند عملیات های مورد نظرش را به راحتی اجرا کند.

سوال ۱۸

-در xv6، پردازنده از سیستم سطح‌بندی (Privilege Levels) در معماری x86 برای جداسازی فضای کاربر و کرنل استفاده می‌کند. این سطح‌بندی تضمین می‌کند که برنامه‌های کاربر نتوانند مستقیماً به حافظه کرنل دسترسی داشته باشند.

-GDT در xv6 برای تعریف قطعه‌های حافظه استفاده می‌شود. هر قطعه دارای سطح دسترسی مشخصی است که در اجرای برنامه‌ها نقش دارد. پردازنده هنگام اجرای هر دستورالعمل، GDT را بررسی می‌کند تا سطح دسترسی آن را مشخص کند.

-پرچم SEG_USER در قطعه‌بندی، مشخص می‌کند که کد و داده‌های یک برنامه در سطح کاربر اجرا شوند. اگر این پرچم تنظیم نشود، پردازنده اجرای آن را محدود کرده و از دسترسی غیرمجاز به کرنل جلوگیری می‌کند.

-پردازنده همچنین از CPL و DPL استفاده می‌کند تا سطح دسترسی فرآیندها را بررسی کند. اگر یک برنامه کاربر بخواهد بدون مجوز مناسب به کرنل دسترسی پیدا کند، CPU خطای "General Protection Fault" ایجاد می‌کند و برنامه متوقف می‌شود.

می توان گفت نهایتا دلیل اصلی, سطح بندی و تمایز بین سطح کاربر و کرنل برای ثبات و امنیت بیشتر در سیستم است

سوال ۱۹

فیلد در proc	معادل در لینوکس	توضیح
uint sz	mm->total_vm	سایز حافظه پردازه (bytes)

pde_t* pgdir	mm->pgd	جدول صفحه (Page Table)
char *kstack	thread_info->task->stack	اشاره‌گر به استک کرنل پردازه
enum procstate state	state	وضعیت پردازه (RUNNING, SLEEPING, etc.)
int pid	pid	شناسه پردازه (Process ID)
struct proc *parent	parent	اشاره‌گر به پردازه والد
struct trapframe *tf	trapframe	ذخیره وضعیت رجیسترها هنگام وقفه
struct context *context	thread_struct	کانتکست ذخیره‌شده برای سوییچ پردازه
int killed	signal->flags	مشخص می‌کند که پردازه کشته شده است یا نه
struct file *ofile[NOFILE]	files_struct	فایل‌های باز شده توسط پردازه
struct inode *cwd	fs->pwd	دایرکتوری جاری پردازه
char name[16]	comm	نام پردازه (برای دیباگینگ)

سوال ۲۳

- بخش مشترک بین تمام پردازنده‌ها: مقداردهی کنترلر وقفه (interrupt controller)
  دلیل: کنترلر وقفه، از جمله IOAPIC و PIC، در کل سیستم مشترک است و تمامی پردازنده‌ها از آن برای مدیریت وقفه‌های خارجی استفاده می‌کنند. این مقداردهی باید یک‌بار در هنگام بوت انجام شود تا پردازنده‌ها بتوانند وقفه‌های مشترک را دریافت کنند.
- بخش اختصاصی برای هر پردازنده: مقداردهی قطعه‌بندی (segnit)
  دلیل: هر پردازنده نیاز دارد که جدول (GDT) مخصوص خودش را تنظیم

کند تا بتواند بین سطح کاربر و کرنل تمایز قائل شود. این مقداردهی فقط برای پردازنده خاص انجام می‌شود و بین هسته‌ها مشترک نیست.

- زمان‌بند روی هر هسته پردازنده اجرا می‌شود، اما CPU هسته شماره 0 ابتدا زمان‌بند را راه‌اندازی می‌کند

اشکال زدایی

سوال 1

برای مشاهده breakpoint ها می توان از دستور زیر استفاده کرد

info break

یا

info breakpoints

```
(gdb) break cat.c:8
Breakpoint 1 at 0xf0: file cat.c, line 8.
(gdb) break cat.c:20
Breakpoint 2 at 0xfe: file cat.c, line 20.
(gdb) break cat.c:27
Breakpoint 3 at 0x0: file cat.c, line 27.
(gdb) break cat.c:40
Breakpoint 4 at 0x54: file cat.c, line 40.
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000000f0 in cat at cat.c:8
2       breakpoint     keep y   0x000000fe in cat at cat.c:20
3       breakpoint     keep y   0x00000000 in main at cat.c:27
4       breakpoint     keep y   0x00000054 in main at cat.c:40
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000000f0 in cat at cat.c:8
2       breakpoint     keep y   0x000000fe in cat at cat.c:20
3       breakpoint     keep y   0x00000000 in main at cat.c:27
4       breakpoint     keep y   0x00000054 in main at cat.c:40
```

سوال 2

برای حذف breakpoint ها می توان از دو دستور زیر استفاده کرد

d i

که i شماره breakpoint ای است که باید حذف شود

delete i

که i شماره breakpoint ای است که باید حذف شود

```
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000000f0 in cat at cat.c:8
2       breakpoint     keep y   0x000000fe in cat at cat.c:20
3       breakpoint     keep y   0x00000000 in main at cat.c:27
4       breakpoint     keep y   0x00000054 in main at cat.c:40
(gdb) d 2
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000000f0 in cat at cat.c:8
3       breakpoint     keep y   0x00000000 in main at cat.c:27
4       breakpoint     keep y   0x00000054 in main at cat.c:40
(gdb) delete 4
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000000f0 in cat at cat.c:8
3       breakpoint     keep y   0x00000000 in main at cat.c:27
```

سوال 3:

درواقع bt مخفف backtrace می باشد.این دستور نشان دهنده call stack تا نقطه ای می باشد که برنامه متوقف شده.

خطوطی که در خروجی نیز نمایش داده شده اند درواقع توابع صدا شده اند که در بالا ترین خط تابعی است که برنامه در آن متوقف شده.

```
(gdb) break cat.c:8
Breakpoint 1 at 0xf0: file cat.c, line 8.
(gdb) break cat.c:18
Breakpoint 2 at 0xd3: file cat.c, line 18.
(gdb) continue
Continuing.

Thread 1 hit Breakpoint 2, cat (fd=3) at cat.c:18
18          if(n < 0){
(gdb) bt
#0  cat (fd=3) at cat.c:18
#1  0x00000054 in main (argc=2, argv=0x2fe4) at cat.c:39
```

در این مثال تابع بالایی یعنی cat تابعی است که برنامه متوقف شده و با #0 نشان داده شده

تابعی که cat را صدا زده main است که با #1 نمایش داده می شود

سوال 4

از دستور x برای مشاهده یک آدرس در حافظه استفاده می شود.

Syntax:x /format address

از دستور print برای نمایش یک variable یا expression استفاده می شود

Syntax:print expression یا print variable

برای مشاهده مقدار یک رجیستر خاص می توان از دستور زیر استفاده کرد

```
(gdb) x /d 0x00ff
0xff:   0
(gdb) print 2*3
$1 = 6
```

Info registers <register name>

```
(gdb) info registers edi
edi            0x0                 0
```

سوال 5

برای مشاهده وضعیت رجیستر ها می توان از دستور زیر استفاده کرد

info registers

```
(gdb) info registers
eax            0x0                 0
ecx            0x0                 0
edx            0x663               1635
ebx            0x0                 0
esp            0x0                 0x0 <main>
ebp            0x0                 0x0 <main>
esi            0x0                 0
edi            0x0                 0
eip            0xfff0              0xfff0
eflags         0x2                 [ IOPL=0 ]
cs             0xf000              61440
ss             0x0                 0
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
fs_base        0x0                 0
gs_base        0x0                 0
k_gs_base      0x0                 0
cr0            0x60000010          [ CD NW ET ]
cr2            0x0                 0
cr3            0x0                 [ PDBR=0 PCID=0 ]
cr4            0x0                 [ ]
--Type <RET> for more, q to quit, c to continue without paging--c
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}, 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 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0,
 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
mxcsr          0x1f80              [ IM DM ZM OM UM PM ]
```

برای متغیر های محلی نیز از دستور زیر استفاده می شود

info locals

```
(gdb) break cat.c:8
Breakpoint 1 at 0xf0: file cat.c, line 8.
(gdb) break cat.c:18
Breakpoint 2 at 0xd3: file cat.c, line 18.
```

```
(gdb) continue
Continuing.

Thread 1 hit Breakpoint 2, cat (fd=3) at cat.c:18
18          if(n < 0){
(gdb) info locals
n = 0
(gdb)
```

edi(extended destination index)

این رجیستر عنوان pointer به مقصد در اعمال string عمل می کند.در برخی از عملیات ها مانند MOVS و LODS رجیستر edi به buffer مقصد اشاره می کند

esi(extended source index)

به عنوان رجیستر مبدا در اعمال string استفاده می شود.این رجیستر معمولا به buffer مبدا اشاره می کند که داده ها خوانده می شوند

سوال 6

struct input دارای یک بافر (input.buf) و 3 نشانگر بر روی اخرین مقدار وارد شده به بافر هستند. از انجایی که تمامی ورودی ها داخل بافر مانند یک log ذخیره میشوند (اگر از ماکزیمم حافظه input.buf بیشتر باشد overwrite میشود) برای دسترسی به اخرین مقدار وارد شده درون این بافر نیازمند این 3 نشانگر هستیم.

مثال: system_

Inpu.rs: به اولین کرکتر وارد شده در کنسول اشاره دارد (در این مثال s)
Inpu.wm: به اخرین کرکتر وارد در کنسول اشاره دارد (در این مثال m)
Inpu.e: به عنوان ایندکس ادیتور است، یعنی کرکتر بعدی که در کنسول وارد شود داخل این ایندکس ذخیره خواهد شد(_)

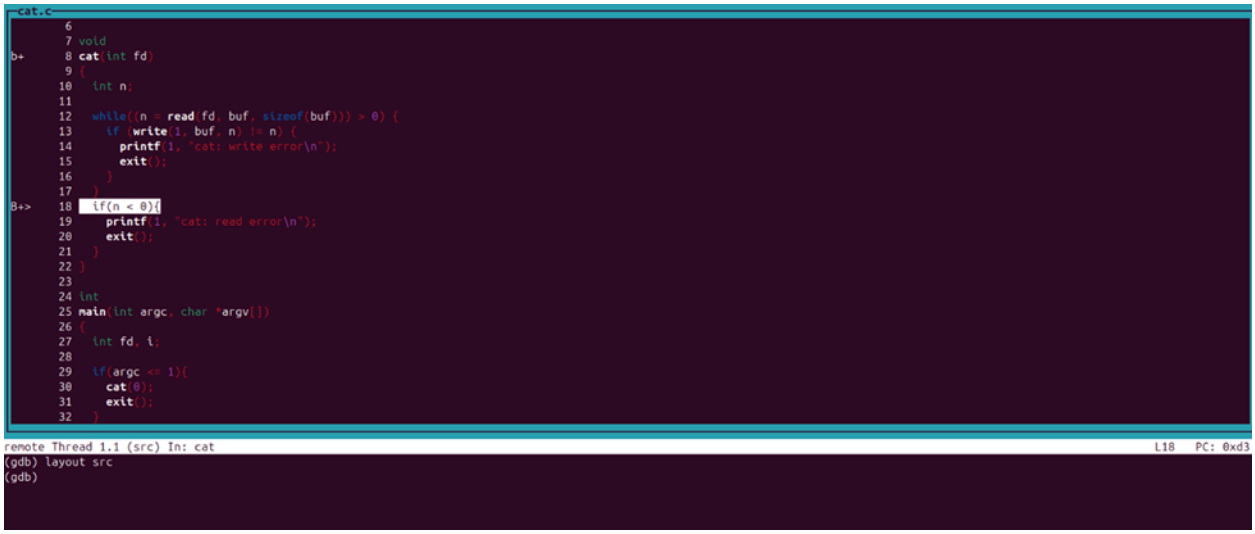
```
(gdb) ptype input
type = struct {
    char buf[128];
    uint r;
    uint w;
    uint e;
}
```

همانطور که در بخش قبل گفته شد input.buf همواره در ایندکس input.e ورودی میگیرد. تنها نکته ای که باید مورد توجه واقع شود lock کرد کنسول هنگام وارد کردن یک ورودیست. به این دلیل که اگر کنسول در زمان اینتراپت با یک پردازنده، از طرف همان پردازنده لاک نشود. احتمال تغییر بافر هنگام پردازش وجود دارد.
3 نشانگر دیگر نیز همواره با تغییر ورودی کنسول به شکل گفته شده خواهند کرد.

```
$2 = {buf = "lgcghjefreg", '\000' <repeats 116 times>, r = 0, w = 0, e = 11}
```

سوال 7

خروجی دستور layout src:



خروجی دستور layout asm:

سوال 8

از دو دستور می توان استفاده کرد.یکی دستور up است که از تابع کنونی خارج و وارد تابعی می شود که تابع کنونی را فراخوانی کرده.دستور down از تابع کنونی خارج شده و وارد تابع بعدی که این تابع آن را فراخوانی می کند می شود