گزارش پروژه (۱) آزمایشگاه سیستم عامل

نام استاد: دکتر کارگهی

امیرارسلان شهبازی ۸۱۰۱۰۱۴۵۱

محمدحسین مظهری ۸۱۰۱۰۱۵۲۰

محمدمهدی صمدی ۸۱۰۱۰۱۴۶۵

گروه ۱۱

لینک ریپوزیتوری گیتهاب: https://github.com/AMIRSH1383/OS-xv6-SMS.git

آخرین کامیت: f07ff84fa72b95c428fe40d497fe77e21a9793a6

مقدمه ای درباره سیستم عامل و xv6

۱) سه وظیفه اصلی سیستم عامل را نام ببرید.

۱-مدیریت منابع

یکی از اصلی‌ترین و مهم‌ترین کارهایی که سیستم عامل انجام می‌دهد مدیریت منابع کامپیوتر است. کامپیوتر مجموعه‌ای از منابع برای انتقال، ذخیره و پردازش داده‌ها و کنترل این اعمال است. سیستم عامل وظیفه مدیریت این منابع را به عنوان مدیر منابع کامپیوتر بر عهده دارد.

مدیریت حافظه:حافظه اصلی در کامپیوتر و سایر حافظه‌های جانبی، داده‌های ما را نگه‌داری می‌کنند. اینکه چه داده‌هایی باید از حافظه جانبی خوانده شده و درون حافظه اصلی (رَم) ریخته شود و تعیین زمان انجام این کارها اکثراً توسط سیستم عامل مدیریت می‌شود.

مدیریت دستگاه‌ها و دسترسی به درگاه‌هایI/O :هر دستگاه و ابزاری که به سیستم کامپیوتری متصل می‌شود به عنوان یک ابزار I/Oیا ابزار ورودی/خروجی شناخته می‌شود. از مانیتوری که تصاویر را نمایش می‌دهد گرفته تا کیبورد و پرینتری که از آن استفاده می‌کنیم.

مدیریت بن‌بست در استفاده از منابع

مدیریت میزان استفاده از منابع

۲-مدیریت فرآیند

در نسل‌های جدید کامپیوترها که چند پردازنده به طور موازی برای پردازش فرآیندها کار می‌کنند، مدیریت و هماهنگی بین آن‌ها یکی دیگر از وظایف سیستم عامل‌ها است.

علاوه بر تخصیص پردازنده‌ها به هر فرآیند که به نوعی مدیریت منابع به حساب می‌آید، سیستم عامل باید بتواند هماهنگی لازم بین فرآیندها و پردازنده‌ها را انجام دهد. زمان‌بندی پردازش و کنترل ترافیک هر پردازنده از جمله کارهایی است که برعهده سیستم عامل است.

۳-مدیریت عملکرد سیستم

سیستم عامل باید به طور مرتب وضعیت سلامت سیستم را بررسی کند. برای مثال:

- زمان پاسخ دادن سیستم به درخواست‌ها
- عملکرد پردازنده یا حافظه‌های سیستم در کار با داده‌ها
- خطاهایی که به طور مداوم تکرار می‌شود.

با بررسی و مانیتور کردن این‌گونه اطلاعات، سیستم عامل خواهد توانست وضعیت کلی سیستم را بسنجد و متناسب با آن تصمیم‌گیری کند. هدف این کارها، بهبود عملکرد سیستم کامپیوتری و افزایش کارآیی آن است.

داخل پرانتز: موارد دیگری از وظایف سیستم عامل نیز وجود دارد مثلا آنکه پل ارتباطی میان بخش نرم افزار و سخت افزار باشد و ... اما موارد اشاره شده در بالا از اصلی ترین وظایف سیستم عامل است.

۲) فایلهای اصلی سیستم عامل ۶ xvدر صفحه یک کتاب ۶ xvلیست شدهاند. به طور مختصر هر گروه را توضیح دهید. نام پوشه اصلی فایلهای هسته سیستم عامل، فایلهای سرایند و فایلسیستم در سیستم عامل لینوکس چیست؟ در مورد محتویات آن مختصرا توضیح دهید.

1-Basic headers

مقادیر ثابت که define شده اند و بعضی تعریف تایپ ها در این بخش قرار دارند.

types.h: شامل typdef های مورد نیاز

asm.h,param.h,memelayout.h: برخی تعاریف موارد ثابت را دارند

defs.h: تعاریف استراکت و توابع

X86.h: حاوی توابعی برای استفاده از اسمبلی موجود در معماری x86

mmu.h: استراکت و برخی مقادیر ثابت تعریف شده برای مدیریت حافظه

Elf.h,date.h: دو هدر باقی مانده

2-Entering xv6

امکان آغاز سیستم عامل و فراهم کردن امکانات لازم را مهیا می کند.

main.c: سیستم از اینجا شروع به کار می کند.

entry.c: هسته از اینجا شروع به کار می کند به این صورت که دستورات اسمبلی این بخش، برنامه را به بخشی که کد سی در آن اجرا می شود هدایت می کند.

entryother.c

3-Locks

در این بخش مدیریت دسترسی های مشترک با استفاده از lock صورت می گیرد. دو فایل موجود در این بخش پیاده سازی همین کار را انجام می دهند.

همچنین گرفتن و رها کردن قفل با استفاده از acquire , release صورت می گیرد.

4-Processes

این بخش وظیفه ی اختصاص دادن حافظه فیزیکی و مدیریت پردازه ها را دارد. همچنین context switching نیز در این بخش انجام می شود.

switch.s: در این بخش قابلیت context switching پیاده سازی شده است به این صورت که وضعیت فعلی رجیسترها ذخیره میشوند تا دوباره بعدا برای اجرا بتوانند بازیابی شوند.

proc.c,proc.h: قابلیت های مربوط به ایجاد و مدیریت پردازه ها.fork

kalloc.c: در این بخش پیاده سازی نحوه اختصاص یافتن حافظه فیزیکی به پردازه ها انجام شده است.

vm.c

5-System calls

در این بخشtrap ها وcall system ها تعریف شده اند.

traps.c,traps.h: انواعtrap ها و عدد متناظر آنها تعریف شده اند. همچنین توابع مربوط به trap نیز در این بخش پیاده سازی شده اند.

syscall.c,syscall.h: عدد متناظر با call system ها و توابع مرتبط پیاده سازی شده اند.

6-File system

هدف یک فایل سیستم چیدمان و ذخیره کردن دادههاست. معمولا فایل سیستم ها، اشتراک گذاری داده ها را میان یوزرها و اپلیکیشن ها پشتیبانی می کنند.

فایل سیستم ۶ xvاز ۶ لایه تشکیل شده است.

پایینیترین از طریق buffercache لایه بلوک هایی را از روی Disk IDE می خواند و می نویسد که تضمین می کند حداکثر یک process kernel در هر لحظه می تواند داده فایل سیستمی ذخیره شده در یک block را تغییر دهد.

لایه دوم به لایه های بالاتر اجازه میدهد که آپدیت هایی رو رویblock های بسیاری در یک transaction انجام دهد تا تضمین کند همه blockها اتوماتیک آپدیت می شوند.

لایه سوم فایلهای بدون نام provide می کند که هر کدام با یک inode و دنباله ایblock ها شامل داده های فایل نمایش داده می شوند .لایه چهارمdirectory ها را به عنوان یک inode خاص که محتویاتش دنباله ای ازentry های direcrotry هست که هر کدام یک اسم و رفرنس به inodeفایل است.

لایه پنجم سلسله مراتب name path ها مثل /c.fs6/xv/rtm/usr را با استفاده از ساختاری بازگشتی تامین می کند.

لایه آخر خیلی از منابع unix (مثل pipes, devices, files) را به کمک فایل systeminterface انتزاع سازی می کند و کار را برای programmer applicationها ساده تر می کند.

fs.c: روتین های سطح پایین مربوط به فایل سیستم را دارد.

log.c: یک ترنزکشن(تراکنش) را مدیریت می کند.

7-Pipes

در این بخش استراکت پایپ تعریف شده است و توابعی برای عملیات خواندن و نوشتن برای آن پیاده سازی شده است.

به طور کلی pipe برای این استفاده می شود که پردازه ها بتوانند بر روی pipe بنویسند یا از آن بخوانند و بتوانند با هم ارتباط برقرار کنند.

8-String operations

توابعی برای کارکردن با رشته ها در آن نوشته شده است.

9-Low−level hardware

mp.c,mp.h: تعاریف و پیاده سازی هایی برای پشتیبانی مولتی پروسسور

lapic.c: مدیریت اینتراپت های داخلی غیر از ورودی و خروجی

ioapic.c: مدیریت اینتراپت های سخت افزار برای یک سیستم SMP

kbd.c,kbd.h: تعریف دکمه های کیبورد

console.c: کدهایی برای کار کردن با ورودی و خروجی. که ورودی از طریق کیبورد یا سریال پورت است و خروجی در صفحه کنسول یا پورت نوشته می شود.

uart.c: سریال پورت intel 8250

10-User−level

در این بخش اولین برنامه سطح کاربر اجرا می شود و امکاناتی نظیر shell اجرایی می شوند.

initcode.s: کدهای اسمبلی برای اجرای برنامه سطح کاربر init

usys.s: حاوی تعریف سیستم کال ها در سطح کاربر

init.c: اولین برنامه سطح کاربر

sh.c: تعاریف و توابع برای اجرای دستورات در شل

11-Bootloader

در این بخش اعمال لازم برای بوت شدن سیستم انجام می شود.

bootasm.s: کد اسمبلی برای لود شدن کد BIOS از اولین سکتور حافظه و منتقل کردن اجرا به کد

bootmain.c: توابع مربوط به بوت

12-Link

یک linker script برای کرنل JOS است.

نکات پایانی:

فایل های مربوط به هسته لینوکس در پوشه ی boot/ قرار دارند.

فایل های سرایند لینوکس هم در usr/src/ قرار دارند.

فایل های فایل سیستم لینوکس از روت یا ریشه اصلی یعنی همان/ شروع می شوند.

در ادامه فایل های هسته سیستم عامل و سرایند و فایل سیستم آورده می شود.

هسته سیستم عامل لینوکس: https://github.com/torvalds/linux/tree/master/kernel

سرایند سیستم عامل لینوکس: https://github.com/torvalds/linux/tree/master/include

فایل سیستم لینوکس: https://github.com/torvalds/linux/tree/master/fs

کامپایل سیستم عامل xv6

۳) دستور make -n را اجرا نمایید . کدام دستور ، فایل نهایی هسته را می سازد ؟

```
hosein@hosein-VirtualBox:~/xv6-public/OS-Lab-Projects$ make -n
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
    -c -o console.o console.c
ld -m    elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o ioapic.o kalloc.o kbd.o lapic.o log.o main.o mp.o picirq.o
pipe.o proc.o sleeplock.o spinlock.o string.o swtch.o syscall.o sysfile.o sysproc.o trapasm.o trap.o uart.o vectors.o vm.o  -b binary initcode entryo
ther
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
dd if=bootblock of=xv6.img conv=notrunc
dd if=kernel of=xv6.img seek=1 conv=notrunc
hosein@hosein-VirtualBox:~/xv6-public/OS-Lab-Projects$
```

بخش نارنجی شده فایل نهایی هسته را می سازد .در حقیقت از linker استفاده می کند تا تمامی object file ها را به هم لینک کند و فایل نهایی هسته را بسازد.

۴) در Makefile متغیرهایی به نام های UPROGS و ULIB تعریف شده است . کاربرد آنها چیست ؟

در xv6 این دو متغیر برای تعریف برنامه های سطح کاربر و کتابخانه های آن مورد استفاده قرار می گیرند.

UPROGS معادل Program User و ULIB معادل User Libraries است که به ترتیب برنامه های کاربر و کتابخانه های کاربر محسوب می شوند.

UPROGS : برای برنامه های سطح کاربر مورد استفاده قرار می گیرد که قابلیت اجرا در xv6 را دارند . این متغیر لیستی از این برنامه ها را شامل می شود . برنامه یا دستور هایی که کاربر می تواند اجرا کند شامل :

cat_echo_forktest_grep_init_kill_ln_ls_mkdir_rm_sh\-stressfs\usertests_wc_zombie

هستند که همه ی آنها در UPROGS وجود دارند.

ULIB : به کتابخانه های سطح کاربر اختصاص یافته است و در واقع شامل تعدادی از کتابخانه های زبان c است

برنامه های سطح کاربر نیازمند این هستند که ULIB اجرا شود . فایل های ULIB شامل توابعی مانند موارد زیر هستند :

Printf/Strcmp/strcpy/malloc

و..

در بسیاری از کدهای xv6 توابع این کتابخانه ها استفاده شده اند و برای اجرایشان به کامپایل این فایل ها نیاز داریم.

اجرا بر روی شبیه سازQEMU

۵) دستور make qemu-n را اجرا نمایید. دو دیسک به عنوان ورودی به شبیه ساز داده شده است. محتوای آنها چیست؟ (راهنمایی: این دیسک ها حاوی سه خروجی اصلی فرایند بیلد هستند)

خروجی کامند بالا بدین صورت است.

```
daryl@daryl:~/Documents/xv6 - OS Lab/OS-xv6-SMS$ make qemu -n
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=
raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
daryl@daryl:~/Documents/xv6 - OS Lab/OS-xv6-SMS$
```

به این ران کردن اصطلاحا dry run می‌گویند. یعنی کامندهایی که قرار است ران شوند را نشان می‌دهد بدون آن که واقعا آن ها را اجرا کند.
دو دیسک فراهم شده fs.img (فایل سیستم) و xv6.img (کرنل) هستند.

Kernel image: خروجی کامپایل شده کدهای C و Assembly کرنل را حاوی است.

File System Image: حاوی فایل سیستمی است که xv6 استفاده می‌کند. در واقع شامل سیستم پروگرم‌ها، فایل‌های کانفیگ شده و دایرکتوری‌هایی که xv6استفاده می‌کند است.

خروجی آخر build علاوه بر دو مورد توضیح داده شده bootloader است که سیستم را Initialize می‌کند و کرنل را لود می‌کند.

مراحل بوت سیستم عامل xv6

اجرای بوت لودر

۷) 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. (راهنمایی: از ابزار objdump استفاده کنید . باید بخشی از آن مشابه فایل bootasm.S باشد.)

این فایل از نوع ELF می باشد.

در ابتدای بوت سیستم ، فایل S.bootasm که یک کد به زبان اسمبلی است ، اجرا می شود و پردازنده را در حالت ۳۲ بیت protected-mode قرار داده که این حالت باعث می شود رجیسترها ، آدرسهای مجازی و اکثر محاسبات عددی به جای ۱۶ بیت با ۳۲ بیت هندل شوند . برای این که این تغییر مد انجام شود نیاز به زبان اسمبلی داریم زیرا processor در ابتدا تنها زبان اسمبلی را می فهمد.

تفاوت این فایل با دیگر فایل های دودویی در نداشتن header می باشد . چون cpu وظیفه ی اجرای instruction ها را بر عهده دارد و هدرهای فایل های دودویی instruction نیستند و cpu توانایی فهمیدن آنها را ندارد . به همین علت از این نوع فایل دودویی استفاده می شود .

تبدیل فایل دودویی به اسمبلی :

-d -> disassemble -b -> binary format(raw) 16 -> bootsector is 16 bit(by default)

```
$ hosein@hosein-VirtualBox:~/xv6-public/OS-Lab-Projects$ objdump -S bootblock.o bootblock.asm
hosein@hosein-VirtualBox:~/xv6-public/OS-Lab-Projects$ objdump -D -b binary -m i386 -M addr16,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 64                   in     $0x64,%al
   b:   a8 02                   test   $0x2,%al
   d:   75 fa                   jne    0x9
   f:   b0 d1                   mov    $0xd1,%al
  11:   e6 64                   out    %al,$0x64
  13:   e4 64                   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 08 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 f0 00                call   0x13b
  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    $0xef668ae0,%eax
  5c:   eb fe                   jmp    0x5c
  5e:   66 90                   xchg   %eax,%eax
        ...
  68:   ff                      (bad)
  69:   ff 00                   incw   (%bx,%si)
  6b:   00 00                   add    %al,(%bx,%si)
  6d:   9a cf 00 ff ff          lcall  $0xffff,$0xcf
  72:   00 00                   add    %al,(%bx,%si)
  74:   00 92 cf 00             add    %dl,0xcf(%bp,%si)
  78:   17                      pop    %ss
  79:   00 60 7c                add    %ah,0x7c(%bx,%si)
  7c:   00 00                   add    %al,(%bx,%si)
  7e:   ba f7 01                mov    $0x1f7,%dx
  81:   00 00                   add    %al,(%bx,%si)
  83:   ec                      in     (%dx),%al
  84:   83 e0 c0                and    $0xffc0,%ax
  87:   3c 40                   cmp    $0x40,%al
  89:   75 f8                   jne    0x83
  8b:   c3                      ret
  8c:   55                      push   %bp
  8d:   89 e5                   mov    %sp,%bp
  8f:   57                      push   %di
  90:   53                      push   %bx
  91:   8b 5d 0c                mov    0xc(%di),%bx
  94:   e8 e5 ff                call   0x7c
  97:   ff                      (bad)
  98:   ff                      (bad)
  99:   b8 01 00                mov    $0x1,%ax
```

```
 9c:   00 00                   add    %al,(%bx,%si)
 9e:   ba f2 01                mov    $0x1f2,%dx
 a1:   00 00                   add    %al,(%bx,%si)
 a3:   ee                      out    %al,(%dx)
 a4:   ba f3 01                mov    $0x1f3,%dx
 a7:   00 00                   add    %al,(%bx,%si)
 a9:   89 d8                   mov    %bx,%ax
 ab:   ee                      out    %al,(%dx)
 ac:   89 d8                   mov    %bx,%ax
 ae:   c1 e8 08                shr    $0x8,%ax
 b1:   ba f4 01                mov    $0x1f4,%dx
 b4:   00 00                   add    %al,(%bx,%si)
 b6:   ee                      out    %al,(%dx)
 b7:   89 d8                   mov    %bx,%ax
 b9:   c1 e8 10                shr    $0x10,%ax
 bc:   ba f5 01                mov    $0x1f5,%dx
 bf:   00 00                   add    %al,(%bx,%si)
 c1:   ee                      out    %al,(%dx)
 c2:   89 d8                   mov    %bx,%ax
 c4:   c1 e8 18                shr    $0x18,%ax
 c7:   83 c8 e0                or     $0xffe0,%ax
 ca:   ba f6 01                mov    $0x1f6,%dx
 cd:   00 00                   add    %al,(%bx,%si)
 cf:   ee                      out    %al,(%dx)
 d0:   b8 20 00                mov    $0x20,%ax
 d3:   00 00                   add    %al,(%bx,%si)
 d5:   ba f7 01                mov    $0x1f7,%dx
 d8:   00 00                   add    %al,(%bx,%si)
 da:   ee                      out    %al,(%dx)
 db:   e8 9e ff                call   0x7c
 de:   ff                      (bad)
 df:   ff 8b 7d 08             decw   0x87d(%bp,%di)
 e3:   b9 80 00                mov    $0x80,%cx
 e6:   00 00                   add    %al,(%bx,%si)
 e8:   ba f0 01                mov    $0x1f0,%dx
 eb:   00 00                   add    %al,(%bx,%si)
```

```
 ed:   fc                      cld
 ee:   f3 6d                   rep insw (%dx),%es:(%di)
 f0:   5b                      pop    %bx
 f1:   5f                      pop    %di
 f2:   5d                      pop    %bp
 f3:   c3                      ret
 f4:   55                      push   %bp
 f5:   89 e5                   mov    %sp,%bp
 f7:   57                      push   %di
 f8:   56                      push   %si
 f9:   53                      push   %bx
 fa:   83 ec 0c                sub    $0xc,%sp
 fd:   8b 5d 08                mov    0x8(%di),%bx
100:   8b 75 10                mov    0x10(%di),%si
103:   89 df                   mov    %bx,%di
105:   03 7d 0c                add    0xc(%di),%di
108:   89 f0                   mov    %si,%ax
10a:   25 ff 01                and    $0x1ff,%ax
10d:   00 00                   add    %al,(%bx,%si)
10f:   29 c3                   sub    %ax,%bx
111:   c1 ee 09                shr    $0x9,%si
114:   83 c6 01                add    $0x1,%si
117:   39 df                   cmp    %bx,%di
119:   76 1a                   jbe    0x135
11b:   83 ec 08                sub    $0x8,%sp
11e:   56                      push   %si
11f:   53                      push   %bx
120:   e8 67 ff                call   0x8a
123:   ff                      (bad)
124:   ff 81 c3 00             incw   0xc3(%bx,%di)
128:   02 00                   add    (%bx,%si),%al
12a:   00 83 c6 01             add    %al,0x1c6(%bp,%di)
12e:   83 c4 10                add    $0x10,%sp
131:   39 df                   cmp    %bx,%di
133:   77 e6                   ja     0x11b
135:   8d 65 f4                lea    -0xc(%di),%sp
```

```
 138:   5b                      pop    %bx
 139:   5e                      pop    %si
 13a:   5f                      pop    %di
 13b:   5d                      pop    %bp
 13c:   c3                      ret
 13d:   55                      push   %bp
 13e:   89 e5                   mov    %sp,%bp
 140:   57                      push   %di
 141:   56                      push   %si
 142:   53                      push   %bx
 143:   83 ec 10                sub    $0x10,%sp
 146:   6a 00                   push   $0x0
 148:   68 00 10                push   $0x1000
 14b:   00 00                   add    %al,(%bx,%si)
 14d:   68 00 00                push   $0x0
 150:   01 00                   add    %ax,(%bx,%si)
 152:   e8 9d ff                call   0xf2
 155:   ff                      (bad)
 156:   ff 83 c4 10             incw   0x10c4(%bp,%di)
 15a:   81 3d 00 00             cmpw   $0x0,(%di)
 15e:   01 00                   add    %ax,(%bx,%si)
 160:   7f 45                   jg     0x1a7
 162:   4c                      dec    %sp
 163:   46                      inc    %si
 164:   75 21                   jne    0x187
 166:   a1 1c 00                mov    0x1c,%ax
 169:   01 00                   add    %ax,(%bx,%si)
 16b:   8d 98 00 00             lea    0x0(%bx,%si),%bx
 16f:   01 00                   add    %ax,(%bx,%si)
 171:   0f b7 35                movzww (%di),%si
 174:   2c 00                   sub    $0x0,%al
 176:   01 00                   add    %ax,(%bx,%si)
 178:   c1 e6 05                shl    $0x5,%si
 17b:   01 de                   add    %bx,%si
 17d:   39 f3                   cmp    %si,%bx
 17f:   72 15                   jb     0x196
```

```
 181:   ff 15                   call   *(%di)
 183:   18 00                   sbb    %al,(%bx,%si)
 185:   01 00                   add    %ax,(%bx,%si)
 187:   8d 65 f4                lea    -0xc(%di),%sp
 18a:   5b                      pop    %bx
 18b:   5e                      pop    %si
 18c:   5f                      pop    %di
 18d:   5d                      pop    %bp
 18e:   c3                      ret
 18f:   83 c3 20                add    $0x20,%bx
 192:   39 de                   cmp    %bx,%si
 194:   76 eb                   jbe    0x181
 196:   8b 7b 0c                mov    0xc(%bp,%di),%di
 199:   83 ec 04                sub    $0x4,%sp
 19c:   ff 73 04                push   0x4(%bp,%di)
 19f:   ff 73 10                push   0x10(%bp,%di)
 1a2:   57                      push   %di
 1a3:   e8 4c ff                call   0xf2
 1a6:   ff                      (bad)
 1a7:   ff 8b 4b 14             decw   0x144b(%bp,%di)
 1ab:   8b 43 10                mov    0x10(%bp,%di),%ax
 1ae:   83 c4 10                add    $0x10,%sp
 1b1:   39 c1                   cmp    %ax,%cx
 1b3:   76 da                   jbe    0x18f
 1b5:   01 c7                   add    %ax,%di
 1b7:   29 c1                   sub    %ax,%cx
 1b9:   b8 00 00                mov    $0x0,%ax
 1bc:   00 00                   add    %al,(%bx,%si)
 1be:   fc                      cld
 1bf:   f3 aa                   rep stos %al,%es:(%di)
 1c1:   eb cc                   jmp    0x18f
        ...
 1fb:   00 00                   add    %al,(%bx,%si)
 1fd:   00 55 aa                add    %dl,-0x56(%di)
hosein@hosein-VirtualBox:~/xv6-public/OS-Lab-Projects$
```

۸) علت استفاده از دستور objcopy در حین اجرای عملیات make چیست ؟

دلیل استفاده از دستور objcopy این است که تنها قسمت text فایل دودویی را از روی bootblock.o کپی کند و bootblock که یک فایل دودویی خام (raw binary file) می باشد را تولید کند. بخش textفایل همان instruction ها هستند که برای cpu قابل اجرا می باشند .

این دستور محتوای یک object file را در یک فایل دیگر کپی می کند و برای خواندن و نوشتن object file از کتابخانه BFD GNU استفاده می کند و می تواند object file مقصد را با فرمتی متفاوت از object file مبدا بنویسد . objcopy ، فایل های موقت درست میکند تا ترجمه هایش را انجام دهد و سپس آنها را حذف می کند . به تمامی فرمت های توصیف شده در BFD دسترسی دارد و به همین دلیل می تواند بیشتر فرمت ها رو بدون اینکه صریحا به آن اعلام شود تشخیص دهد.

۱۳)کد bootmain.c هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس 0x100000 قرار می دهد. علت انتخاب این آدرس چیست ؟

بارگذار هسته xv6 را در حافظه در آدرس فیزیکی ۰x100000بارگذاری می‌کند. دلیل اینکه هسته را در آدرس ۰x80100000که هسته انتظار دارد دستورالعمل‌ها و داده‌هایش را در آن پیدا کند بارگذاری نمی‌کند، این است که ممکن است در یک ماشین کوچک، هیچ حافظه فیزیکی در چنین آدرس بالایی وجود نداشته باشد. دلیل اینکه هسته را در ۰x100000به جای ۰x0قرار می‌دهد این است که دامنه آدرس ۰xa0000:0x100000شامل دستگاه‌های ورودی/خروجی است.

اجرای هسته xv6

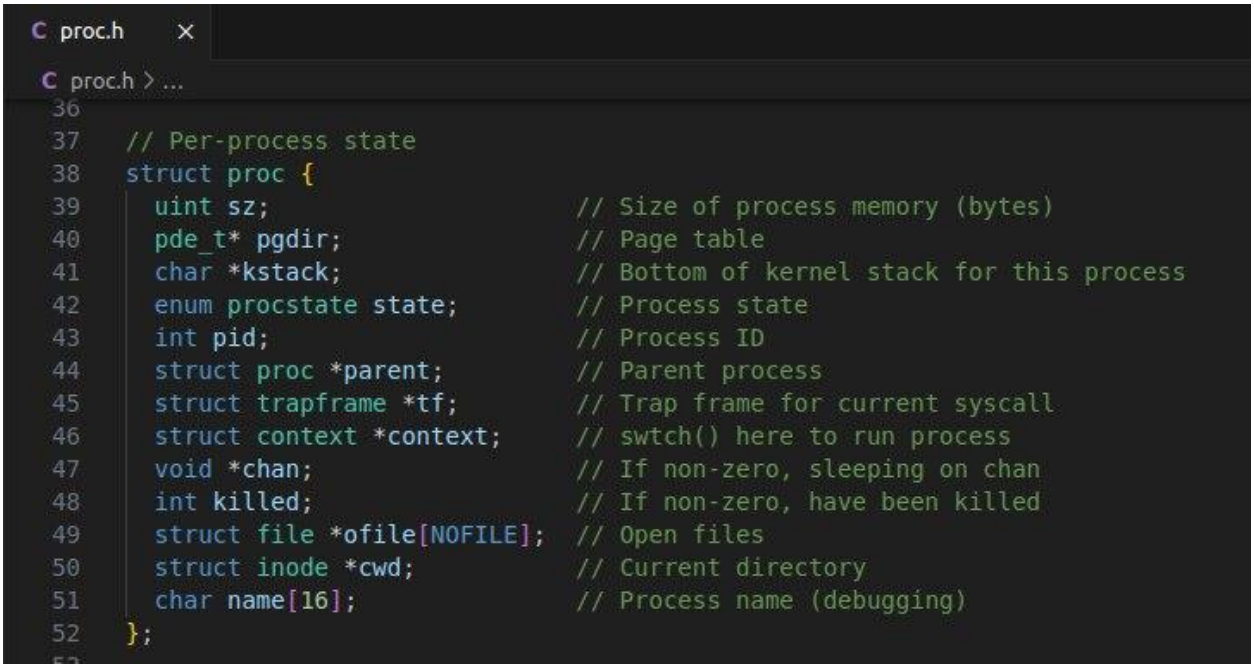
سوال ۱۸) علاوه بر صفحه بندی در حد ابتدایی از قطعه بندی به منظور حفاظت هسته استفاده خواهد شد.این عملیات توسط ()seginit انجام می گردد. همان طور که ذکر شد، ترجمه قطعه تأثیری بر ترجمه آدرس منطقی نمی گذارد. زیرا تمامی قطعه ها اعم از کد و داده روی یکدیگر می افتند. با این حال برای کد و داده های سطح کاربر پرچم SEG_USERتنظیم شده است. چرا؟ (راهنمایی: علت مربوط به ماهیت دستور العمل ها و نه آدرس است .)

جهت تمایز میان پردازنده های سطح کاربر و پردازنده های سطح هسته از فلگ SEG_USER استفاده می شود .

اجرای نخستین برنامه سطح کاربر

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

ساختار اشاره شده در فایل proc.h قرار دارد و به صورت زیر است.



```c
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

Sz: سایز ممووری این پراسس (در واحد بایت).

معادل لینوکس: mm->total_vm

Pgdir: پوینتر به page table.

معادل لینوکس: mm->pgd

Kstack: پوینتر به ته استک مختص به این پراسس.

معادل لینوکس: thread_info->task->stack

State: استیت فعلی پراسس (RUNNING, SLEEPING ...).

معادل لینوکس: state

Pid: آی‌دی پراسس.

معادل لینوکس:pid

Parent: پوینتر به پراسس پدر (هر پراسس از پراسس دیگری ناشی می‌شود، به غیر از اولین پراسس).

معادل لینوکس:real_parent

Tf: پوینتر به trap frame برای سیستم کال فعلی.

معادل لینوکس:Part of thread_struct within task_struct

Context: پوینتر به کانتکستی که بین پراسس‌ها سوییچ می‌کند.

معادل لینوکس: thread_struct

Chan: اگر صفر نباشد بدین معناست که پراسس روی یک چنل خوابیده است.

معادل لینوکس: Condition variables or wait queues

Killed: اگر صفر نباشد بدین معانست که پراسس کیل شده است.

معادل لینوکس: Signals handling (signal->flags)

Ofile: آرایه‌ای از پوینتر ها برای باز کردن فایل‌ها.

معادل لینوکس: files_struct

Cwd: پوینتر به دایرکتوری در حال انجام فعلی (current working directory).

معادل لینوکس: fs->pwd

Name: آرایه‌ای از کاراکتر ها برای ذخیره اسم پراسس (بیشتر برای دیباگ استفاده می‌شود).

معادل لینوکس:comm

معادل ساختار proc در لینوکس، ساختار task_struct است.

۲۳) کدام بخش از آماده‌سازی سیستم، بین تمام هسته‌های پردازنده مشترک و کدام بخش اختصاصی است؟ (از هر کدام یک مورد با ذکر دلیل توضیح دهید). زمان‌بند روی کدام هسته اجرا می‌شود؟

بخش مشترک: راه اندازی Interrupt Controllerها باید مشترک باشد تا همه کرنل ها بتوانند Interruptها را مدیریت کنند. اگر بدین صورت نباشد، مدیریت و رفع Interrupt ها به مشکل می‌خورد.

بخش اختصاصی: Seginit برای اختصاصی است تا هر کرنل تنظیمات مختص خود را داشته باشد.

زمان‌بند به صورت چرخشی روی تمام کرنل‌ها اجرا می‌شود تا آن‌ها بتوانند پراسس‌های خود را پیش ببرند.

اشکال زدایی

روند اجرایGDB

۱) برای مشاهده breakpointها از چه دستوری استفاده می‌شود؟

همانطور که در تصویر زیر مشاهده می‌کنیم بعد از افزودن breakpoint(ها) با دستوری info breakpoints می‌توان آن‌ها را مشاهده کرد.

```
(gdb) break test_code_for_gdb.c:5
No symbol table is loaded.  Use the "file" command.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (test_code_for_gdb.c:5) pending.
(gdb) break test_code_for_gdb.c:9
No symbol table is loaded.  Use the "file" command.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (test_code_for_gdb.c:9) pending.
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   <PENDING>  test_code_for_gdb.c:5
2       breakpoint     keep y   <PENDING>  test_code_for_gdb.c:9
(gdb)
```

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

با دستور del idx که idx شماره breakpoint مدنظر در لیست breakpointها است می‌توان آن را حذف کرد.

```
(gdb) del 1
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
2       breakpoint     keep y   <PENDING>  test_code_for_gdb.c:9
(gdb)
```

کنترل روند اجرا و دسترسی به حالت سیستم

۳) دستور bt را اجرا کنید. خروجی آن چه چیزی را نمایش می‌دهد؟

دستور backtrace مانند یک stack است که در هر لحظه وضعیت توابع کال‌شده و به پایان نرسیده را نشان می‌دهد. در این کد یک تابع ساده داشتیم اما اگر از توابع بازگشتی استفاده کنیم استک به خوبی وضعیت همه را نمایش می‌دهد.

```
daryl@daryl:~/Documents/xv6 - OS Lab$ gcc -g -o out test_code_for_gdb.c
daryl@daryl:~/Documents/xv6 - OS Lab$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file out
Reading symbols from out...
(gdb) break 10
Breakpoint 1 at 0x1155: file test_code_for_gdb.c, line 10.
(gdb) run
Starting program: /home/daryl/Documents/xv6 - OS Lab/out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at test_code_for_gdb.c:10
10          int plus_val = 10;
(gdb) bt
#0  main () at test_code_for_gdb.c:10
(gdb) break 11
Breakpoint 2 at 0x55555555515c: file test_code_for_gdb.c, line 11.
(gdb) continue
Continuing.

Breakpoint 2, main () at test_code_for_gdb.c:11
11          adder(a, plus_val);
(gdb) bt
#0  main () at test_code_for_gdb.c:11
(gdb) continue
Continuing.
[Inferior 1 (process 8726) exited normally]
(gdb) bt
No stack.
(gdb)
```

کنترل روند اجرا و دسترسی به حالت سیستم

۴) تفاوت دستور های print , x را توضیح دهید.چگونه می توان محتوای یک ثبات خاص را چاپ کرد؟(راهنمایی : می توانید از دستور help استفاده نمایید .)(help x , help print)

دستور print به عنوان ورودی یک expression را دریافت کرده و مقدار آن را نمایش میدهد ولی دستور x یک آدرس گرفته و مقدار ذخیره شده در آن را نمایش میدهد. همچنین نوع نمایش خروجی این x دستور دو دستور هم متفاوت است.

همچنین برای دریافت محتوای یک ثبات خاص میتوان از دستور info register<register name> استفاده کرد.

۵) برای نمایش وضعیت ثبات ها از چه دستوری استفاده می شود ؟ متغیرهای محلی چطور ؟ نتیجه ی این دستور را در گزارش کار خود بیاورید. همچنین در گزارش خود توضیح دهید که در معماری x86 رجیسترهای edi, esi نشانگر چه چیزی هستند .

برای نمایش وضعیت ثبات ها می توان از info register استفاده کرد . علاوه بر آن از مخفف این دستور یعنی i r نیز می توان استفاده کرد . باری نمایش متغیرهای محلی از طریق info locals اقدام می شود .

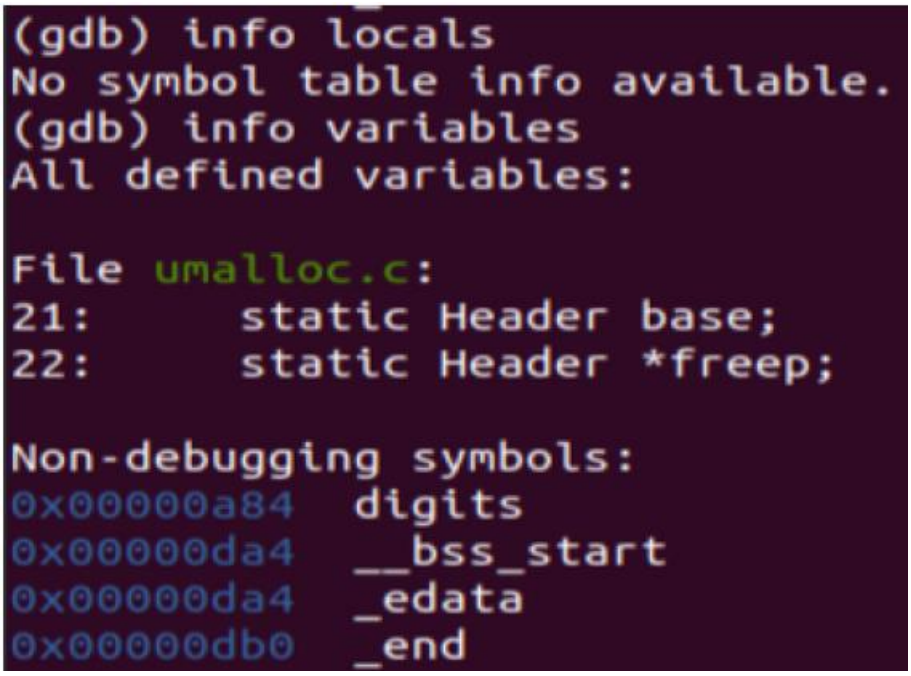
E در ابتدای اسامی این ثبات ها به معنای extended بوده و در حالت ۳۲ بیت به کار می رود.

EDI : مخفف Extended Destination Index است و برای اشاره به یک مقصد در عملیات stream به کار می رود .

ESI : مخفف Extended Source Index است و برای اشاره به مبدا در عملیات stream به کار می رود .

(SI به عنوان نشانگر داده و به عنوان مبدا در برخی عملیات مربوط به رشته ها استفاده می شود .)

```
(gdb) info registers
eax            0x1                 1
ecx            0xb0                176
edx            0x1                 1
ebx            0x200               512
esp            0x803bef70          0x803bef70
ebp            0x803bef78          0x803bef78
esi            0x80113a70          -2146354576
edi            0x80113a74          -2146354572
eip            0x80103e25          0x80103e25
eflags         0x2                 [ IOPL=0 ]
cs             0x8                 8
ss             0x10                16
ds             0x10                16
es             0x10                16
fs             0x0                 0
gs             0x0                 0
fs_base        0x0                 0
gs_base        0x0                 0
k_gs_base      0x0                 0
cr0            0xe0010011          [ PG CD NW WP ET PE ]
cr2            0x0                 0
cr3            0x3ff000            [ 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 = {0x
0, 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 = {0x
0, 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 = {0x
0, 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 = {0x
0, 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 = {0x
0, 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 = {0x
0, 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 = {0x
0, 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, 0x1
f80, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x1f80}, v2_int64 = {0x0, 0x1f8000000000}, uint128 = 0x1f80000000000000000000000000
```



۶) به کمک استفاده از GDB، درباره ساختار struct input موارد زیر را توضیح دهید: توضیح کلی این struct و متغیرهای درونی آن و نقش آن ها نحوه و زمان گیری مقدار متغیرهای درونی(برای مثال ، input.e در چه حالتی تغییر می کند و چه مقداری را می گیرد ؟)

این استراکت دارای یک آرایه ۱۲۸ تایی از کاراکتر(بافر) و ۳ متغیر w,r,e است . هر سه متغیر نشان دهنده اندیس در بافر هستند . r اندیسی است که دستورات تا آنجا انجام شده و توسط سیستم عامل مدیریت شده اند.

W اندیسی است که تا آنجا در بافر ذخیره شده است (یعنی مثلا در هنگام وارد کردن دستور جدید ، اندیس اول خط نمایش داده می شود .) و پس از وارد کردن کامند جدید و فشردن enter ، r و w آپدیت می شوند و مقدار e را می گیرند . e اندیسی است که محلی که در حال تایپ در آن هستیم را نشان می دهد.(یعنی مثلا با وارد کردن یک کاراکتر جدید در کنسول یک واحد زیاد می شود .) و کامند وارد شده در بافر ذخیره می شود .

اشکال زدایی در سطح کد اسمبلی

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

در نمای layout src کد خود را در کنار رابط GDB می بینید. کد برای نشان دادن خط فعلی در حال اجرا برجسته شده است . این به شما اجازه می دهد تا به صورت تعاملی از طریق کد منبع خود به هنگام اشکال زدایی برنامه خود حرکت کنید .

از این نما میتوان break point ها را تعیین ، مقادیر متغیر ها را بررسی و اجرای برنامه را کنترل کرد .

این نما به ویژه برای درک منطق سطح بالای برنامه ی شما در حین اشکال زدایی مفید است .

در نمای layout asm کد اسمبلی جدا شده برنامه خود را مشاهده می کنید.

این نما دستورالعمل های واقعی ماشین را نشان می دهد که با کد شما مطابقت دارد . این به شما این امکان را می دهد که نحوه ترجمه ی کد خود را به دستور العمل های اسمبلی سطح پایین بررسی کنید .

در این نما می توانید break point ها را تعیین ، رجیستر ها را بررسی و کد را اجرا کنید .

این نما زمانی مفید است که نیاز دارید در جزئیات اجرای برنامه غوطه ور شوید و بفهمید که چگونه خطوط خاصی از کد منبع به اسمبلی ترجمه می شوند .

۸) برای جابه جایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستوراتی استفاده می شود؟

از دو دستور up , down میتوان استفاده کرد. هر دوی این دستورات می توانند یک عدد به عنوان ورودی بگیرند (که به صورت پیش فرض ۱ است) که میتوان با استفاده از آن چندین لایه حرکت کرد .

شرح پروژه

اضافه کردن یک متن به Boot message

نمایش نام اعضای تیم در Boot Message

به فایل init.c قطعه کد زیر اضافه شده که نام سه عضو گروه را پرینت می‌کند.

```
for(;;){
  // ----------------------------------------
  printf(1, "\nMembers of Team SMS:\n");
  printf(1, "  Mohamad Mahdi Samadi\n");
  printf(1, "   Amir Arsalan Shahbazi\n");
  printf(1, "    Mohammad Hosein Mazhari\n");
  // ----------------------------------------
```

اضافه کردن قابلیت های جدید به کنسول

۱. برای این بخش یکسری توابع در فایل console.c اضافه می کنیم. ایده ی اصلی انجام آن به این صورت است که می گوییم هروقت که روی کنسول می خواهیم چیزی بنویسیم مکانی که نشانگر آنجا قرار دارد را به دست می آوریم. سپس با توجه به اینکه مکان ابتدا و انتهای آن دستوری که داریم می نویسیم کجاست می توانیم بین کاراکترهای موجود جا به جا شویم.

همچنین از مرز بندی برای اینکه اولا بیشتر از تعداد کاراکتر های موجود پیش نرویم و ثانیا بیش از حد به ابتدای خط و حتی خط های بالا نرویم استفاده می کنیم.

تغییر بعدی این بخش اضافه کردن توابعی برای نوشتن بین جمله و پاک کردن بین جمله است.

۲.در این بخش ابتدا دو کلید بالا و پایین را پیاده سازی می کنیم. ایده ی انجام به این صورت است که یک آرایه به سایز حداکثر تعداد پیام ها در نظر گرفته و هر دستوری که کاربر می دهد ما دستور را در آن آرایه ذخیره می کنیم. هر وقت کاربر کلید بالا یا پایین را فشار داد ما روی آرایه ای که داشتیم پیمایش انجام داده و دستور موجود در آن خانه را روی کنسول نمایش می دهیم.

برای هر یک از کامند ها یک آیدی لحاظ کردیم که به کمک آن آیدی می توانیم کامند مد نظر را پیدا کنیم.

دستور بعدی کامند history است. برای اعمال، آن را مانند یک دستور سیستم عاملی مثل اکو یا ... در نظر می گیریم. برای پیاده سازی آن ابتدا یک فایل جدید به همین نام می سازیم. سپس آن را به میک فایل اضافه می کنیم. پیاده سازی آن نیز به این صورت است که در فایل sh.c می گوییم هر کامند جدیدی که آمد آن را در آرایه ای ذخیره کن. سپس هر وقت که دستور تاریخچه را دیدی دستورهای پیشین را به عنوان آرگومان به تابعی که در فایل هیستوری است پاس بده. در فایل هیستوری تنها کاری که می کنیم این است که تعداد پیام ها را در ابتدا یافته و سپس با پیمایش روی آن ها، در کنسول نمایش می دهیم.

۳. در ابتدا چند متغیر تعریف می‌کنیم. توضیحات مربوط به هر متغیر جلوی آن کامنت شده است.

```
// variables related to the ctrl+s / ctrl+f feature
static int copybuf[INPUT_BUF]; // buffer to store the copied inputs
static int copying = 0;  // flag to check if copying is active
static int insert_copied_commands = 0; // activated after ctrl+f until all the copied commands are runned
static int current_copied_command_to_run_idx = 0; // index of current copied command ro run
static int num_copied_commands = 0;  // length of the copied content
```

تابع زیر در زمانی که ctrl+s زده شده و تا قبل فشردن ctrl+f کار می‌کند و دستورها را ذخیره می‌کند.

```
void store_command(int c) {
  if (num_copied_commands < INPUT_BUF) {
    copybuf[num_copied_commands] = c; // save the current command
    num_copied_commands++; // increase the number of copied commands by 1
  }
  return;
}
```

دو کیس جدید به سوییچ کیس تابع consoleintr فایل console.c اضافه شده است. در این دو کیس سیگنال‌هایی فعال و خاموش شده و یا متغیرهای مربوط به سایز و اندیس کپی کردن ریست می‌شوند.

```
    case C('S'):  // Start copying on Ctrl+S
      copying = 1; // enable copying flag
      num_copied_commands = 0;
      insert_copied_commands = 0;
      break;
    case C('F'):  // Paste copied content on Ctrl+F
      copying = 0; // disable copying flag
      insert_copied_commands = 1; // send the signal to start running the copied commands
      current_copied_command_to_run_idx = 0; // set the idx to zero
      break;
```

در ابندای تابع consoleintr این تغییراتی داده‌ایم.

شرط insert_copied_commands به حلقه وایل اضافه شده. این شرط زمان فشردن ctrl+f true می‌شود و تا زمانی که تمام دستورهای ذخیره شده اجرا نشدند false نمی‌شود.

کلید وارد شده در متغیر temp_c ریخته می‌شود و در ابتدای حلقه وایل متغیر c یا توسط کلید فشرده شده توسط کاربر یا توسط دستوری ذخیره شده پر می‌شود.

اگر اجرای دستورهای ذخیره‌شده به انتها برسد، سیگنال insert_copied_commands غیر فعال می‌شود.

```
void
consoleintr(int (*getc)(void))
{
  int temp_c, doprocdump = 0;

  char current_command[MAX_LEN_OF_COMMAND];

  acquire(&cons.lock);
  while(((temp_c = getc()) >= 0) || (insert_copied_commands)){
    // determine if the next command to run is a new command or one of the copied commands
    int c = 0;
    if (insert_copied_commands) {
      c = copybuf[current_copied_command_to_run_idx];
      current_copied_command_to_run_idx++;
    } else {
      c = temp_c;
    }

    if (current_copied_command_to_run_idx == (num_copied_commands - 1)) {
        insert_copied_commands = 0; // turn off this signal
    }
```

تغییراتی نیز در حالت دیفالت سوییچ کیس اعمال شده است که در تصویرهای زیر مشاهده می‌کنیم.

۴. متغیرهای تعریف شده مرتبط به این بخش.

```
// variables related to the NON=? feature
#define PRECISION 2
#define isdigit(c) (c >= '0' && c <= '9') // determine if an input char is digit or not
const int math_exp_len = 5;
```

برای این بخش توابعی تعریف شده که در حالت دیفالت سوییچ کیس کال شده‌اند. ابتدا توابع را توضیح می‌دهیم.

تابع زیر با وارد شدن کاراکتر جدید و قبل از قرار دادن آن روی کنسول چک می‌کند که آیا ۵ کاراکتر اخیر با هم تشکیل عبارت ریاضی می‌دهند یا خیر.

```
int detect_math_expression(char c) {
  if (c == '?' && input.buf[(input.e-1) % INPUT_BUF] == '=') {
    if (isdigit(input.buf[(input.e-2) % INPUT_BUF])) {
      char operator = input.buf[(input.e-3) % INPUT_BUF];
      if ((operator == '+') || (operator == '-') || (operator == '*') || (operator == '/')) {
        if (isdigit(input.buf[(input.e-4) % INPUT_BUF])) {
          return 1;
        }
      }
    }
  }
  return 0;
}
```

تابع زیر در ابتدا عبارت ریاضی را decode کرده و سپس حاصل آن را پیدا می‌کند. همچنین اگر operator تقسیم بود سیگنالی را فعال می‌ند که جلو تر مورد نیاز است.

```
367  float calculate_result_math_expression(int* is_divide) {
368    char operator = input.buf[(input.e-3) % INPUT_BUF];
369    float first_operand = input.buf[(input.e-4) % INPUT_BUF] - '0';
370    float second_operand = input.buf[(input.e-2) % INPUT_BUF] - '0';
371    float result = 0;
372
373    switch (operator) {
374      case '+':
375        result = first_operand + second_operand;
376        break;
377      case '-':
378        result = first_operand - second_operand;
379        break;
380      case '*':
381        result = first_operand * second_operand;
382        break;
383      case '/':
384        result = first_operand / second_operand;
385        *is_divide = 1;
386        break;
387      default:
388        break;
389    }
390
391    return result;
392  }
```

بعد از محاسبه حاصل عبارت، باید آن را به صورت آرایه‌ای از کاراکترها در بیاریم تا روی کنسول نمایش داده شود. تابع زیر حاصل جمع، تفریق و ضرب را که عددی صحیح هست را به استرینگ تبدیل می‌کند.

```
int int_to_str(int result, char* res_str) {
  int res_len = 0;
  int is_neg = 0;

  if (result == 0) {
    res_str[res_len] = '0';
    res_len += 1;
  } else {
      if (result < 0) {
        is_neg = 1;
        result = -result;
      }
      int temp_result = result;
      do {
        res_str[res_len] = (temp_result % 10) + '0';
        res_len += 1;
        temp_result /= 10;
        }
       while (temp_result > 0);

  }

  if (is_neg) {
    res_str[res_len] = '-';
    res_len += 1;
  }

  // Reverse the string
  for (int i = 0; i < res_len / 2; ++i) {
    char temp = res_str[i];
    res_str[i] = res_str[res_len - i - 1];
    res_str[res_len - i - 1] = temp;
  }

  for (int i = 0; i < res_len; i++) {
    input.buf[input.e % INPUT_BUF] = res_str[i];
    input.e++;
    input.position++;
    consputc(res_str[i]);
  }
  return res_len;
}
```

تابع زیر هم برای حاصل تقسیم به کار می‌رود و عدد فلوت را به استرینگ تبدیل می‌کند.

```
int float_to_str(float result, char* res_str, int precision) {
  int res_len = 0;
  int is_neg = 0;
  int is_less_than_one = 0;

  if (result == 0) {
    res_str[res_len] = '0';
    res_len += 1;
  } else {
      if (result < 0) {
        is_neg = 1;
        result = -result;
      }
      if ((result > 0) && (result < 1)) {
        is_less_than_one = 1;
      }
      // shift the number by the given number of digits
      for (int i=0; i<precision; i++) {
        result *= 10;
      }
      int temp_result = result;
      int point = 0;
      do {
        if (point == precision) {
          res_str[res_len] = '.';
          res_len += 1;
        } else {
          res_str[res_len] = (temp_result % 10) + '0';
          res_len += 1;
          temp_result /= 10;
        }
        point++;
      } while (temp_result > 0);
  }

  if (is_less_than_one) {
    res_str[res_len] = '.';
    res_len += 1;
    res_str[res_len] = '0';
    res_len += 1;

  }

  if (is_neg) {
    res_str[res_len] = '-';
    res_len += 1;
  }
```

این قطعه کد به ابتدای حالت دیفالت اضافه شده. در ابتدای ورود به هر کیس چک می‌شود که اگر در حال کپی بودیم دستور اجرا نشود. اما اگر در حال کپی بودیم صرفا دستور ذخیره شود تا بعدا اجرا شود (البته به جز حالاتی مثل کلید بالا و پایین کیبورد که فرض شده کاربر در هنگام کپی آن‌ها را نمی‌زند)

```
673     default:
674       if (copying) {
675         store_command(c);
676       } else {
677         if(c != 0 && input.e-input.r < INPUT_BUF) {
678           // Check for pattern "NON=?"
679           int is_divide = 0;
680           int is_math_expression = detect_math_expression(c);
681           char res_str[12];
682           int res_len = 0;
683
684           if (is_math_expression) {
685             float result = calculate_result_math_expression(&is_divide);
686             if (!is_divide) {
687               res_len = int_to_str((int)result, res_str);
688             } else {
689               res_len = float_to_str(result, res_str, PRECISION);
690             }
691
692             // Clear the input buffer for this expression
693             for(int i = 0; i < math_exp_len + (res_len - 1); i++) {
694                 input.e--;
695                 input.position--;
696                 consputc(BACKSPACE);
697             }
698
699             // Put the result on console
700             for (int i = 0; i < res_len; i++) {
701               input.buf[input.e % INPUT_BUF] = res_str[i];
702               input.e++;
703               input.position++;
704               consputc(res_str[i]);
705             }
```

برنامه سطح کاربر

با نوشتن دو فایل encode.c و decode.c و انجام تغییرات در Makefile (همانند دستورهای echo , cat) ،دستورات مربوطه را به قابلیت های سیستم عامل اضافه میکنیم .

مثالی از یک عبارت رمزگذاری و رمزگشایی شده:

```
$ encode salam23 bar0 10shoma CheTor Ast AhVALAt ShaRIF?
$ cat result.txt
ygrgs23 hgx0 10ynusg InkZux Gyz GnBGRGz YngXOL?
$ decode ygrgs23 hgx0 10ynusg InkZux Gyz GnBGRGz YngXOL?
$ cat result.txt
salam23 bar0 10shoma CheTor Ast AhVALAt ShaRIF?
```