	به نام خدا	

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

آزمایشگاه سیستم عامل

آزمایش اول

نام و نام خانوادگی	آناهیتا مشتاق کهنموئی-مونا محدث مجتهدی-ریحانه احمدپور
شماره دانشجویی	810198494-810198557-810198473
تاریخ ارسال گزارش	1400/12/22 – یکشنبه

فهرست سوالات و مطالب آشنایی با سیستم عامل(xv6)

سوال یک	۳
سوال دو	۳
سوال چهار	۳
سوال هشت	۸
سوال یازده	۹
سوال دوازده	۹
سوال چهارده	۹
سوال هجده	۹
سوال نوزده	۱۰
سوال بیست و دو	۱۰
سوال بیست و سه	۱۰
سوال بیست و هفت	۱۰

آشنایی با سیستم عامل xv6

سوال 1. معماری سیستم عامل **xv6** چیست؟ چه دلایلی در دفاع از نظر خود دارید؟

معماری سیستم عامل xv6 بر پایه monolithic kernel است.(همانگونه که اکثر سیستم عامل‌های مبتنی بر UNIX پیاده سازی شده اند)، زیرا تمامی سیستم عامل در هسته قرار گرفته و تمامی کارها در سیستم عامل با تمامی اختیارات سخت‌افزاری انجام می‌شود،یعنی تمامی سیستم کال‌ها در سطح کرنل(kernel) اجرا می‌شود. این موضوع(سطح کاربری) باعث می‌شود که پیاده سازی سیستم عامل ساده تر شود اما ممکن است عواقبی داشته باشد. به طور مثال، در صورتی که یک اشتباه در سیستم عامل رخ دهد، این اشتباه میتواند بسیار مخرب باشد(زیرا در سطح کرنل جابه‌جا می‌شود و در کرنل به آدرس‌های اینتراپت می‌پرد) و منجر به توقف فعالیت سیستم عامل شود.

رفرنس:book-rev11.pdf

سوال 2. یک پردازه در سیستم عامل **xv6** از چه بخش‌هایی تشکیل شده است؟ این سیستم عامل به طور کلی چگونه پردازنده را به پردازه های مختلف اختصاص می‌دهد؟

به طور کلی یک پردازه در xv6 شامل حافظه‌ای است(user space memory) که این حافظه خود شامل دستورات، داده و پشته(instructions, data, and stack) است و یک حالت برای هر پردازه که به صورت خصوصی در اختیار هسته(kernel) است، می‌باشد.

همچنین هر پردازه(process) با یک شناسه برای هسته سیستم عامل قابل شناسایی است(identifier or pid). به طور کلی، سیستم عامل از روش اشتراک گذاری زمان و زمان بندی (time-sharing) بین پردازه ها استفاده می‌کند، یعنی بطور مداوم و بدون اینکه کاربر متوجه شود (transparency)، پردازنده های موجود را بین پردازه های آماده اجرا شدن پخش می‌کند و زمانی که یک پردازه در حال اجرا نیست، سیستم عامل مقدار ثبات‌های مربوط به پردازنده را ذخیره کرده(CPU registers) تا برای اجرا مجدد آماده باشد و در اجرای بعدی آن‌ها را بازیابی می‌کند.

سوال 3. مفهوم file descriptor درسیستم عامل های مبتنی بر UNIX چیست؟ عملکرد pipe درسیستم عامل xv6 چگونه است و به طور معمول برای چه هدفی استفاده می شود؟

به طور کلی این مفهوم، نشان دهنده یک شیء مدیریت شده توسط کرنل است که یک پردازه میتواند از بخواند یا در آن بنویسد و پردازه با باز کردن یک directory ، file، و pipe، device یا کپی کردن یک توصیفگر موجود file descriptor بدست آورد و تفاوت های بین pipes ،files و devices را خلاصه می‌کند و آنها را استریم‌هایی از بایت ها در نظر می‌گیرد. همچنین در هسته این سیستم عامل ها هر پردازه یک فضای خصوصی برای file descriptor دارد که از صفر شروع می‌شود.

پایپ یک بافر کوچک است که به عنوان یک جفت file descriptor در معرض پردازه ها قرار می‌گیرد یکی برای خواندن و یکی برای نوشتن.نوشتن دیتا تا انتهای پایپ باعث می‌شود که دیتا قابل خواندن از آخر پایپ دیگر شود و پایپ ها راهی را برای ارتباط برقرار کردن پردازه ها ایجاد کردند.

سوال 4. فراخوانی های سیستمی exec و fork چه عملی انجام می‌دهند؟ از نظر طراحی ادغام نکردن این دو چه مزیتی دارند؟

با فراخوانی سیستمی fork یک پردازه (parent)میتواند یک پردازه دیگر(child)با محتوای مموری یکسان بسازد. فراخوانی سیستمی exec حافظه پردازه فراخوانی را با حافظه جدید تصویر لود شده از یک فایل ذخیره شده در سیستم فایل،جایگزین می‌کند وفایل باید فرمت خاصی داشته باشد تا مشخص کند که کدام قسمت فایل،دستورها کدام،داده ها و از کدام دستور شروع شود و غیره.

ادغام نکردن این دو باعث میشود که shell بتواند یک child را fork کند،از open و close و dup در فرزند استفاده می‌کند تا ورودی و خروجی استاندارد file descriptors را تغییر دهد و سپس exec کند.اگردر فراخوانی ادغام می‌شدند،یک طرح احتمالا پیچیده تر برای shell لازم میشد تا ورودی و خروجی استاندارد بتوانند تغییر مسیر دهند یا اینکه خود برنامه باید می‌فهمید که چگونه I/O تغییر مسیر دهد.

سوال 5. سه وظیفه اصلی سیستم عامل را نام ببرید.

۱-سیستم عامل باید بتواند فعالیت های مختلف را همزمان انجام دهد. ۲- وظیفه دارد که time-share بین منابع کامپیوتری برای پردازه ها انجام دهد. ۳-وظیفه ایزوله کردن شرایط برای پردازه ها هم دارد یعنی اگر یک پردازه باگ داشته باشد و شکست بخورد، این نباید روی بقیه پردازه هایی که به این پردازه بستگی ندارند،تاثیر بگذارد.

Multiplexing-Isolation-Interaction

سوال 6. فایل های 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 در سیستم عامل لینوکس چیست؟ در مورد محتویات آن مختصراً توضیح دهید.

سیستم عامل xv6 شامل بخش های زیر است:
تعریف تایپ های مختلف و برخی مقادیر ثابت اینجا تعریف می‌شوند:

#Basic Headers:

```
# basic headers
01 types.h
01 param.h
02 memlayout.h
02 defs.h
04 x86.h
06 asm.h
07 mmu.h
09 elf.h
09 date.h
```

Notice that things are grouped by their relevance. “Basic Headers” lays out a lot of constants and conventions for XV6.
We don’t need to look at those right away.

types.h شامل typedef ها است.
param.h memlayout.h asm.h حاوی یکسری از مقادیر define شده هستند.
x86.h تعریف توابع برای استفاده از assembly.
mmu.h دارای مقادیر ثابت و استراکت ها برای مدیریت حافظه.
executable and linkable format شامل ساختار و بدنه فایل‌های elf.h.
شامل ساختار نمایش زمان date.h.

#Boot Loader:

```
# bootloader
91 bootasm.S
92 bootmain.c
```

When an x86 PC boots, it starts executing a program called the BIOS (Basic Input /Output System), which is stored in
non-volatile memory on the motherboard. The BIOS’s job is to prepare the hardware and then transfer control to the
operating system. Specifically, it transfers control to code loaded from the boot sector, the first 512-byte sector of the
.boot disk. The boot sector contains the boot loader: instructions that load the kernel into memory

دارای توابع لازم برای boot شدن سیستم.
bootasm.S هم دارای کد اسمبلی برای بالا آمدن کد BIOS از اولین بخش حافظه و اجرای کد C است.
bootmain.c شامل تابع bootmain است که یک ELF kernel image را از دیسک می خواند و سپس به اول کرنل می رود.

#Entering xv6:

```
# entering xv6
10 entry.S
11 entryother.S
12 main.c
```

Is the code that starts XV6. Note that XV6, the operating system, is a program just like any other program. It does have
some special requirements, though. Since it is the first code that the CPU runs, it needs some special machinery to set
.up and launch the kernel itself; this is in entry.S. main.c contains the main function for the OS

سیستم را شروع می‌کند و امکانات لازم را فراهم می‌کند.
main.c اولین فایلی هست که اجرا می‌شود و سیستم عامل را شروع می‌کند.
در entry.S کرنل شروع به کار می‌کند و دستورات اسمبلی را به بخش اجرای کد C انتقال می‌دهد. entryother.S بخش های entry.S و bootasm.S را در کنار هم قرار می‌دهد.

#Link:

```
# link
93 kernel.ld
```

.This code connects to kernel JOS

یک اسکریپت linker برای اتصال به کرنل JOS است.

#Locks:

```
# locks
15 spinlock.h
15 spinlock.c
```

File locking is a mechanism to restrict access to a file among multiple processes. It allows only one process to access the
file in a specific time, thus avoiding the interceding update problem.

این بخش مدیریت دسترسی های مشترک را با کمک lock انجام می‌دهد. به طور دقیق تر، سیستم عامل با استفاده از lock می تواند از interrupt handler هایی که در یک CPU در حال اجرا هستند، در مقابل non-interrupt code هایی که در CPU دیگر در حال اجرا هستند، جلوگیری کند.
با کمک توابع aquire و release میتوان lock کردن را مدیریت کرد.

#Processes:

```
# processes
17 vm.c
23 proc.h
24 proc.c
30 swtch.S
31 kalloc.c
```

The “processes” section is also important - these files contain the key machinery for running user programs

and enabling
OS features like multitasking and multiprogramming. We will not go to a low-enough level in this course to investigate
the low-level hardware section. Finally, in the user-level section, init.c contains the code for the first user process. init is
always running, until the OS is shut down. An important missing file from this list is user.h, containing the definitions for
.user-level functionality

این بخش پردازه ها را مدیریت می‌کند:
vm.c: توابع مدیریت page table ها.
proc: توابع مربوط به ایجاد و مدیریت پردازه ها مثل fork.
swtch.S: توابع مربوط به قابلیت context switching که با ذخیره کردن وضعیت فعلی رجیستر ها. قابلیت بازیابی و استفاده دوباره از آنها در آینده را فراهم می‌کند.
kalloc.c: مدیریت اختصاص دادن حافظه فیزیکی به پردازه ها.

```
# system calls
32 traps.h
32 vectors.pl
33 trapasm.S
33 trap.c
35 syscall.h
35 syscall.c
37 sysproc.c
```

#System Calls:

A computer program makes system call when it makes request to operating system's kernel. System calls ,are used for hardware services, to create or execute process, and for communicating with kernel services including application and process scheduling.

در این بخش system call ها و trap تعریف شده اند.
traps: انواع trap ها و توابع مربوط به آنها را تعریف کرده است.
syscall: انواع systemcall ها و توابع مربوط به آنها را تعریف کرده است.
همه system call ها و trap ها به یک عدد خاصی map شده اند. تا در فایل های دیگر قابل شناسایی باشند.

#File System:

```
# file system
38 buf.h
39 sleeplock.h
39 fcntl.h
40 stat.h
40 fs.h
41 file.h
42 ide.c
44 bio.c
46 sleeplock.c
47 log.c
49 fs.c
58 file.c
60 sysfile.c
66 exec.c
```

Files are stored on disk or other storage and do not disappear when a user logs off. Files have names and are associated
with access permission that permits controlled sharing. Files could be arranged or more complex structures to reflect
the relationship between them.

در این بخش توابعی برای مرتب سازی، مدیریت و ذخیره داده ها وجود دارد.
fs: دارای برخی داده ساختارها برای تعریف فرمت فایل سیستم و روتین های سطح پایین مربوط به آنها است.
log.c: یک logging ساده برای فراخوانی سیستم های مربوط به فایل سیستم ها.
fcntl.h: دارای برخی مقادیر تعریف شده برای باز کردن فایل ها.

#Pipes:

```
# pipes
67 pipe.c
```

A pipe is a small kernel buffer exposed to processes as a pair of file descriptors, one for reading and one for writing.
Writing data to one end of the pipe makes that data available for reading from the other end of the pipe. Pipes provide a
way for processes to communicate.

در این بخش ساختار pipe و توابع مربوط به خواندن و نوشتن آن تعریف شده است.
pipe برای برقراری ارتباط بین پردازه های مختلف استفاده میشود.

#String Operations:

```
# string operations
69 string.c
```

.Functions related to work with strings

توابع کمکی برای کار با string ها.

#Low-Level Hardware:

```
# low-level hardware
70 mp.h
72 mp.c
73 lapic.c
76 ioapic.c
77 kbd.h
78 kbd.c
79 console.c
83 uart.c
```

The job of an operating system is to share a computer among multiple programs and to provide a more useful set of
services than the hardware alone supports. The operating system manages and abstracts the low-level hardware, so
.that, for example, a word processor need not concern itself with which type of disk hardware is being used

این بخش شامل پشتیبانی از ساختار Multiprocessing و جستجو در آنها است.
mp: برای پشتیبانی قابلیت مولتی پروسسینگ.
lapic.c: مدیریت interrupt های داخلی بجز I/O
ioapic.c: مدیریت interrupt های سخت افزاری برای سیستم SMP
kbd: تعریف ثابت های کیبورد
console.c: برای مدیریت ورودی و خروجی از طریق کیبورد یا سریال پورت و صفحه کنسول.
uart: Intel 8250 serial port

#User-Level:

```
# user-level
84 initcode.S
84 usys.S
85 init.c
85 sh.c
```

.init.c contains the code for the first user process

این بخش شامل برنامه های سطح کاربر و فضای مربوط به کاربر می باشد.
init.c: شامل اولین برنامه سطح کاربر که اجرا میشود میباشد.
این بخش برخی قابلیت ها مثل shell را اجرا میکنند.
initcode.S: کدهای اسمبلی برای اجرای برنامه init.c
usys.S: تعریف فراخوانی های سیستمی در سطح کاربر
sh.c: توابع و تعاریف برای اجرای دستورات در shell

سیستم عامل **linux**:

/Linux kernel file, Header file and file systems are stored in /boot
The Boot File System (named BFS on Linux, but BFS also refers to the Be File System) was used on UnixWare to store files
necessary to its boot process. It does not support directories, and only allows contiguous allocation for files, to make it
.simpler to be used by the boot loader

فایل های هسته در /boot هستند.
فایل های سرایند کرنل در /usr/src هستند. سرایندهای استاندارد کتابخانه ها نیز در /usr/include ذخیره شده اند.
فایل های فایل سیستم در root یا / هستند.

کامپایل سیستم عامل xv6

سوال 8.در Makefile متغیر هایی به نام های UPROGS و ULIB تعریف شده.کاربرد آنها چیست؟

UPROGES: list of user program.
ULIB: user mode standard library, including printf, malloc, free ,…

کامپایل سیستم عامل xv6

سوال 10. در **xv6** در سکتور نخست دیسک قابل بوت، محتوای چه فایلی قرار دارد؟
با اجرای دستور make -n متوجه میشویم که یک فایل بنام xv6.img داریم. اگر به Makefile مراجعه کنیم میبینیم که این فایل با دستورات زیر ساخته شده است:

```
xv6.img: bootblock kernel
        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
```

به دنبال bootblock میرویم و دستورات زیر را میبینیم:

```
bootblock: bootasm.S bootmain.c
fno-pic -O -nostdinc -I. -c bootmain.c- (CFLAGS)$ (CC)$
fno-pic -nostdinc -I. -c bootasm.S- (CFLAGS)$ (CC)$
N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o- (LDFLAGS)$ (LD)$
S bootblock.o > bootblock.asm- (OBJDUMP)$
S -O binary -j .text bootblock.o bootblock- (OBJCOPY)$
sign.pl bootblock/.
```

در نتیجه میتوان گفت فایل های bootmain.c و bootasm.S در سکتور نخست هستند.

سوال 11.

Boot loader دو فایل source که یکی bootasm.S به زبان اسمبلی و دیگری bootmain.c به زبان c هستند.بعد از کامپایلر هر کدام از این فایل ها به فایل با نام اصلیشان و پسوند .o تبدیل می شود.فایل با پسوند .o فایل object code file است که شامل ورژن باینری source کد مورد نظر می باشد.
تفاوت فایل های بوت با بقیه فایل های کامپایل شده این است که حتما در sector اول (first block on first track) قرارمیگیرد و محل ان همیشه ثابت است ولی بقیه فایل ها در ادرس های با شماره متفاوت می توانند قرار بگیرند
محتویات این فایل کد مخصوصی جهت راه اندازی سیستم عامل را شامل می شود و عملیات locate load و initialize کردن os را انجام می دهد.
objdump -D bootmain.o فایل را به زبان اسمبلی تبدیل می نماید.

سوال 12.

دستور objcopy قابلیت تبدیل فرمت فایل مبدا به s-record اسمبلی و raw bianary file را دارد بنابر این در makefile از این دستور برای تبدیل فایل های bootmain.c و bootasm.S (فایل های source بوت)و فایل های initial و بقیه فایل ها که هنگام بوت لازم می شود به فایل باینری خام (مثل boot block) که در sector اول می نشید و به زبان پایه ماشین است استفاده می شود تا بدون نیاز به مترجم و کامپایلر در زمان بوت ماشین بتواند این فایل ها را اجرا کند .دستور مربوطه

```
S -O binary -j .text bootblock.o bootblock- (OBJCOPY)$
```

سوال 13. بوت سیستم توسط bootmain.c و bootasm.S صورت میگیرد. چرا تنها از کد C استفاده نشده است؟
علت این کار این است که در این مرحله نیاز به برخی قابلیت هایی داریم که در سطح سیستم اجرا شوند. اما زبان C در سطح بالاتری هست. همچنین سرعت اجرای کد اسمبلی بالاتر است و با اینکار سیستم سریعتر بوت میشود. کد اسمبلی پردازنده را برای کد C فراهم میکند به این صورت که آنرا به حالت محافظت شده ۳۲بیت میبرد و بعد bootmain را صدا میزند.

سوال 14. 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.
معماری x86 دارای ۸ ثبات عام منظوره میباشد. esi و edi دو تا از این ثبات ها هستند که معمولا به عنوان پوینتر استفاده میشوند. در همه این ثبات ها اول اسم آنها حرف e هست که مخفف extended است. علت این نامگذاری این است که این ثبات ها ۳۲بیتی هستند. وظیفه این مدل ثبات ها نگهداری داده، برخی از پوینترها و برخی عملیات ریاضی است.
در ثبات قطعه آدرس استک، برخی داده ها و کد نگهداری میشود. SS یک ثبات قطعه هست که پوینتر به استک را نگه میدارد.
ثبات وضعیت شامل اطلاعات وضعیت پردازنده ها است. یکی ازین ثبات ها EFLAGS نام دارد که وضعیت پرچم هایی مثل carry و zero را نگه میدارد و هر بیت آن، نشان دهنده وضعیت یک پرچم است.
ثبات های کنترلی مثل CR0 اطلاعات کنترلی مثل CPU را نگهداری میکند. برای مثال CR3 آدرس صفحه جدول پردازه کنونی را در خود نگه میدارد که برای نگاشت آدرس های مجازی به آدرس های فیزیکی استفاده می شود. در کل این ثبات ها مدل آدرس دهی، درخواست های مختلف مثل interrupt را نگه میدارند.

سوال 18.کد معادل entry.s در هسته لینوکس را بیابید.
کد معادل entry.S در هسته لینوکس start_kernel () می باشد .که مسئول اجرای بیشتر setup system مانند (.interrupts, the rest of memory management, device and driver initialization, etc).
می باشد.

اجرای هسته xv6

سوال 19. اگر آدرس مجازی بود، به یک آدرس فیزیکی نیاز داشتیم تا آدرس مجازی را در خود نگه دارد. بنابراین در نهایت به آدرس فیزیکی نیاز بود و استفاده از آدرس مجازی منطقی نبود.

سوال 21. مختصری راجع به محتوای فضای آدرس مجازی هسته توضیح دهید.

در این فضا xv6 از 32 بیت آدرس مجازی استفاده میکند که 10 بیت اول برای ایندکس دادن به دایرکتوری page table است.10 بیت بعدی برای قرار دادن entry جدول با ایندکس دادن به جدول داخلی (PTE) است و این شامل 20 بیت شماره فریم فیزیکی و پرچم ها است.

سوال 22. علاوه بر صفحه بندی در حد ابتدایی از قطعه بندی به منظور حفاظت هسته استفاده خواهد شد. این عملیات توسط seginit() انجام میگردد. همانطور که ذکر شد، ترجمه قطعه تاثیری بر ترجمه آدرس منطقی نمیگذارد. زیرا تمامی قطعه ها اعم از کد و داده روی یکدیگر می افتند. با این حال برای کد و داده های سطح کاربر پرچم USER_SEG تنظیم شده است. چرا؟

این کار برای این است که بتوان تمایز بین ماهیت پردازه های سطح کاربر و پردازه های سطح هسته قائل شد. به عبارتی، درست است که محتوای هر دوی این پردازه ها در یک فضای فیزیکی قرار دارد ولی برای ما مهم است که بدانیم که آن محتوا متعلق به یک پردازه سطح کاربر است یا خیر.

اجرای نخستین برنامه سطح کاربر

سوال 23. جهت نگهداری اطلاعات مدیریتی برنامه های سطح کاربر ساختاری تحت عنوان struct proc ارائه شده است. اجزای آن را توضیح داده و ساختار معادل آن در سیستم عامل لینوکس را بیابید.

uint sz:
سایز حافظه پردازنده به بایت
pde_t* pgdir:
یک پوینتر به آدرس Page table
:char *kstack
یک پوینتر به آدرس پایین استک هسته در پردازه فعلی
:enum procstate state
وضعیت پردازه
:int pid
عدد نظیر شده (آیدی) به پردازه
:struct proc *parent
پوینتر به والد پردازه
:struct trapframe *tf
پوینتر به یک trapframe برای syscall کنونی
:struct context *context
پوینتر به استراکت context که برای context switching در هسته استفاده می شود
:void *chan
اگر صفر نباشد، پردازه به خواب می رود
:int killed
اگر صفر نباشد پردازه از بین رفته است
:struct file *ofile[NOFILE]
پوینتر به فایل های باز این پردازه
:struct inode *cwd
پوینتر به پوشه فعلی
:char name[16]
اسم پردازه (که برای دیباگ کردن کاربرد دارد)

ساختار معادل آن در لینوکس task_struct نام دارد که در این لینک قابل مشاهده است.

سوال 27. کدام بخش از آماده سازی سیستم بین تمامی هسته های پردازنده مشترک و کدام بخش اختصاصی است؟زمان بند روی کدام هسته اجرا میشود؟

کد زیر بین هر دو cpu موجود در xv6 برای setup اجرا می شود.

ابتدا جدول interrupt descriptor table -idt که اطلاعات اینکه برای هر interrupt number چه کاری باید انجام شود (pointer به) interrupt handler را initialize می کند زیرا هر cpu هنگامی که به system callبر می خورد باید بداند چه عملیاتی را اجرا نماسد .سپس به startothers که مخصوص شروع non -boot process است اعلام میکند که اماده به انجام عملیات است چون هر دو می توانند non- boot process را اجرا نمایند سپس Schedular را فعال می کند تا cpu ها بتوانند از زمانبندی استفاده کنند و process های روی خود را مدیریت کند.

```
mpmain(void)
}
;cprintf("cpu%d: starting %d\n", cpuid(), cpuid())
idtinit();      // load idt register
xchg(&(mycpu()->started), 1); // tell startothers() we're up
scheduler();    // start running processes
{
```

هنگامی که یک process می خواهد اجرا شود setup اختصاصی یکcpu شروع می شود اول آنکه برنامه باید در state RUNNBLE قرار گیرد سپس باید cpu بداند که برنامه ای که میخواهد اجرا نماید در کجای actual memory قرار دارد و بتواند mapping بین virtual memory و actual memory را تشخیص دهد که در واقع page table آن initialize شود . همچنین p->tf->eip برابر صفر میشود که این به این معنی است که در زمان برگشت به سطح کاربر از آدرس مجازی 0 که ابتدای کد initcode.S است اجرا می شود .چون در هر system call باید برنامه به مد kernel برود و دوباره به مد user بر گردد لازم است فیلد trapframe که ثبات مربوط به userspace را در خود دارد در جایی ذخیره کند تا اطلاعات مربوط به مد user از دست نرود .سپس DPL که descriptor privilege level هست و از 0 تا 3 مقدار دهی می شود را مقدار دهی کرده تا بتواند عملیات در سطح kernel را انجام دهد. میزانPrivilege بودن با عدد کم تر اضافه می شود . بعد از اتمام کار مد kernel دوباره به مد user بر میگردد.

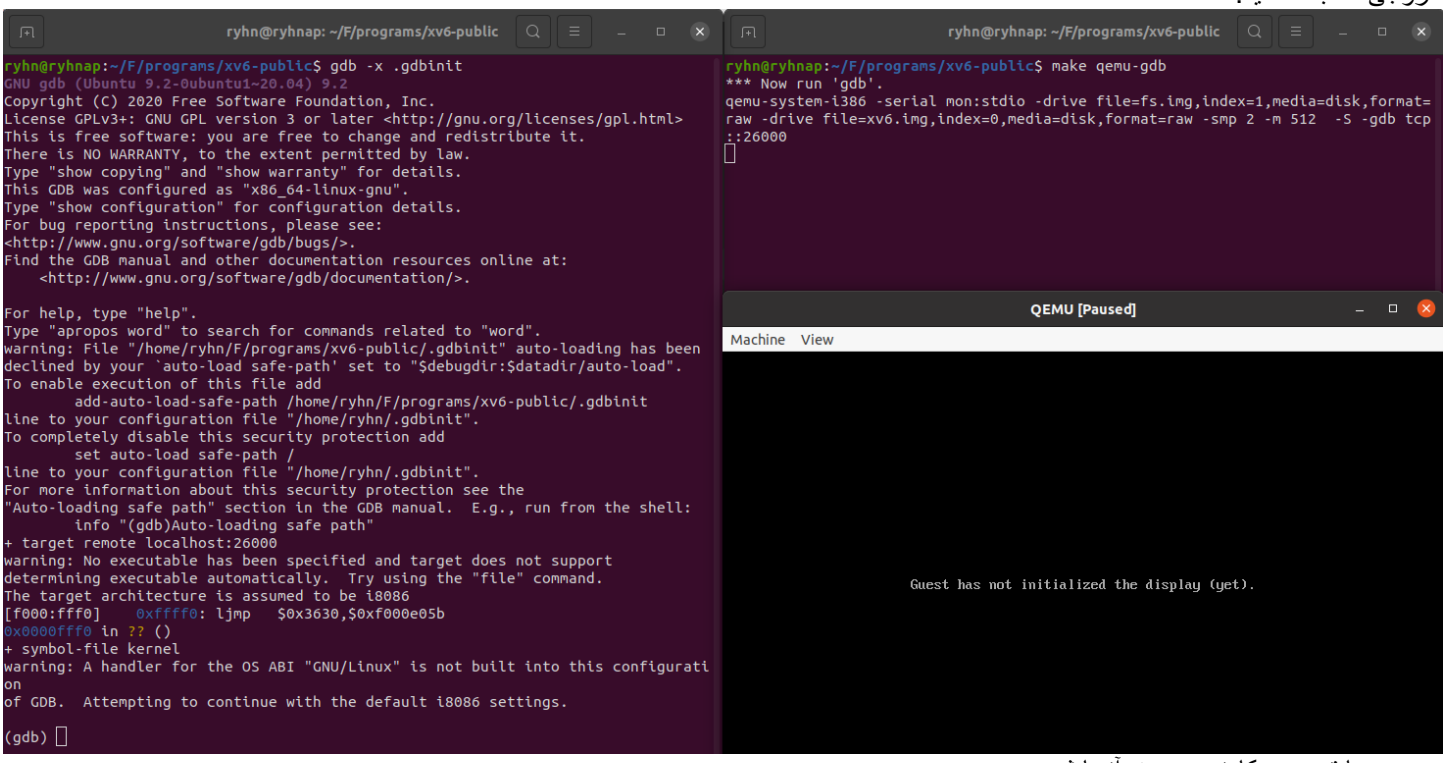
هر cpu تابع scheduling را صدا می کند پس scheduler در هر دو اجرا می شود تا هر cpu بین process هایی که اجرا می کند بتواند زمان بندی را انجام دهد (کامنت فایل proc.c تابع scheduler)

روند اجرای GDB

اشکال‌زدایی:

ابتدا gdb را اجرا می‌کنیم و با توجه به اینکه می‌خواهیم خروجی مانند اسلایدها باشد دو مدل خروجی داریم یکی همان کامند صورت آزمایشگاه است و دیگری کامندای است که به صورت خودکار و دیفالت خود کرنل ارتباط را برقرار می‌کند و خروجی صورت پروژه را دارد.

خروجی مشابه اسلاید:



خروجی با توجه به کامند صورت آزمایش:

```
ryhn@ryhnap: ~/F/programs/xv6-public
ryhn@ryhnap:~/F/programs/xv6-public$ gdb kernel
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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:
<http://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"...
Reading symbols from kernel...
warning: File "/home/ryhn/F/programs/xv6-public/.gdbinit" auto-loading has been
declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /home/ryhn/F/programs/xv6-public/.gdbinit
line to your configuration file "/home/ryhn/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ryhn/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
(gdb) target remote tcp::26000
Remote debugging using tcp::26000
0x0000fff0 in ?? ()
(gdb)
```

```
ryhn@ryhnap: ~/F/programs/xv6-public
ryhn@ryhnap:~/F/programs/xv6-public$ make qemu-gdb
*** Now run 'gdb'.
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  -S -gdb tcp
::26000
```

```
QEMU [Paused]
Machine  View

Guest has not initialized the display (yet).
```

```
ryhn@ryhnap: ~/F/programs/xv6-public
ryhn@ryhnap:~/F/programs/xv6-public$ gdb kernel
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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:
<http://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"...
Reading symbols from kernel...
warning: File "/home/ryhn/F/programs/xv6-public/.gdbinit" auto-loading has been
declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /home/ryhn/F/programs/xv6-public/.gdbinit
line to your configuration file "/home/ryhn/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ryhn/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
(gdb) target remote tcp::26000
Remote debugging using tcp::26000
0x80103f75 in mycpu () at proc.c:45
45        apicid = lapicid();
(gdb)
```

```
ryhn@ryhnap: ~/F/programs/xv6-public
ryhn@ryhnap:~/F/programs/xv6-public$ make qemu-gdb
*** Now run 'gdb'.
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  -S -gdb tcp
::26000
xv6...
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 #5:
1. Anahita Moshtagh
2. Mona MohadesMojtahedi
3. Reyhane Ahmadpoor
$
```

```
QEMU [Paused]
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 #5:
1. Anahita Moshtagh
2. Mona MohadesMojtahedi
3. Reyhane Ahmadpoor
$
```

دیدن متغییر:

```
ryhn@ryhnap:~/F/programs/xv6-public$ gdb -x .gdbinit
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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:
<http://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".
warning: File "/home/ryhn/F/programs/xv6-public/.gdbinit" auto-loading has been
declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /home/ryhn/F/programs/xv6-public/.gdbinit
line to your configuration file "/home/ryhn/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ryhn/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i386
=> 0x80103f75:  mov    0x80114240,%esi
0x80103f75 in ?? ()
+ symbol-file kernel
(gdb) watch *0x1234567
Hardware watchpoint 1: *0x1234567
(gdb) b *0x98
Breakpoint 2 at 0x98
(gdb) info b
Num     Type           Disp Enb Address    What
1       hw watchpoint  keep y              *0x1234567
2       breakpoint     keep y   0x00000098
(gdb) del 2
(gdb) info b
Num     Type           Disp Enb Address    What
1       hw watchpoint  keep y              *0x1234567
(gdb)
```

```
ryhn@ryhnap:~/F/programs/xv6-public$ make qemu-gdb
*** Now run 'gdb'.
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  -S -gdb tcp
::26000
xv6...
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 #5:
1. Anahita Moshtagh
2. Mona MohadesMojtahedi
3. Reyhane Ahmadpoor
$
```

```
QEMU [Paused]
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 #5:
1. Anahita Moshtagh
2. Mona MohadesMojtahedi
3. Reyhane Ahmadpoor
$ _
```

۱. برای مشاهده Breakpoint ها از چه دستوری استفاده می شود؟
از دستور info breakpoints استفاده می شود.

۲. برای حذف یک Breakpoint از چه دستوری و چگونه استفاده می شود؟
می توان با استفاده از دستور
<clear <filename>:<line
می توان شماره خط را به gdb داد تا اگر Breakpoint ای که در آن خط وجود داشته باشد، آن را حذف کند.

۳. دستور زیر را اجرا کنید. خروجی آن چه چیزی را نشان میدهد؟
bt $
این دستور مخفف backtrace هست و چیزی که نشان میدهد توابع فراخوانده شده و لود شده در استک میباشد.
خروجی دستور به اینصورت است که در هر خط تابع فراخوانی شده را می نویسد و در خط بعد تابعی که تابع خط فعلی را فراخوانی کرده چاپ میکند.

```
(gdb) b convert_chars_to_int
Breakpoint 1 at 0x3f: convert_chars_to_int. (2 locations)
(gdb) continue
Continuing.
[Switching to Thread 2]

Thread 2 hit Breakpoint 1, main (argc=-2096880564, argv=0x71fff0e4) at factor.c:37
37          int fd, j = 0, n = convert_chars_to_int(argv[1]);
(gdb) bt
#0  main (argc=-2096880564, argv=0x71fff0e4) at factor.c:37
(gdb) n
printint (fd=0, xx=0, base=12264, sgn=0) at printf.c:27
27        i = 0;
(gdb) bt
#0  printint (fd=0, xx=0, base=12264, sgn=0) at printf.c:27
#1  0xffffffff in ?? ()
(gdb) n
34        while(--i >= 0)
(gdb) bt
#0  printint (fd=0, xx=<optimized out>, base=<optimized out>, sgn=1) at printf.c:34
#1  0x00000057 in convert_chars_to_int (number=<error reading variable: Cannot access memory at address 0x3008>)
    at factor.c:28
#2  main (argc=-2096880564, argv=0x71fff0e4) at factor.c:37
(gdb)
```

۴. دو تفاوت دستور **x** و **print** را توضیح دهید. چگونه میتوان محتوای یک ثبات خاص را چاپ کرد؟
یکی از تفاوت های این دو دستور نحوه نمایش اطلاعات است.
تفاوت دیگری که دارند این است که دستور print با دریافت یک expression مقدارش را نشان میدهد اما دستور x مقدار ذخیره شده در یک آدرس را نمایش میدهد.

```
(gdb) print buf
$1 = '\000' <repeats 15 times>
(gdb) print convert_chars_to_int
$2 = {int (char *)} 0x160 <convert_chars_to_int>
(gdb) x 0x00000057
0x57 <main+87>: 0x33685a58
(gdb)
```

برای دریافت محتوای یک ثبات خاص میتوان از دستور< info register < register name استفاده کرد.
۵. برای نمایش وضعیت ثباتها از چه دستوری استفاده میشود؟ متغیرهای محلی چطور؟ نتیجه این دستور را در گزارشکار خود بیاورید. همچنین در گزارش خود توضیح دهید که در معماری **x86** رجیسترهای **edi** و **esi** نشانگر چه چیزی هستند؟
برای وضعیت ثبات ها از info registers استفاده میشود.
برای متغیرهای محلی از info locals استفاده میشود.

```
(gdb) info registers
ah         bx         dl         edi        fctrl      fp         mm1        mmx        si         st3        xmm0       xmm7
al         ch         ds         edx        fioff      fs         mm2        mxcsr      sp         st4        xmm1
all        cl         dx         eflags     fiseg      fstat      mm3        orig_eax   ss         st5        xmm2
ax         cs         eax        eip        float      ftag       mm4        pc         sse        st6        xmm3
bh         cx         ebp        es         fooff      general    mm5        ps         st0        st7        xmm4
bl         dh         ebx        esi        fop        gs         mm6        restore    st1        system     xmm5
bp         di         ecx        esp        foseg      mm0        mm7        save       st2        vector     xmm6
(gdb) info registers esi
esi            0x821    2081
(gdb) info locals
Align                  digits                 int                    printf                 uchar
Header                 dup                    kill                   printint               uint
__bss_start            exec                   link                   read                   unlink
_edata                 exit                   long int               sbrk                   unsigned char
_end                   fd                     main                   sgn                    unsigned int
atoi                   fork                   malloc                 short int              uptime
base                   free                   memmove                short unsigned int     wait
buf                    freep                  memset                 sleep                  write
char                   fstat                  mkdir                  stat                   x
chdir                  getpid                 mknod                  strchr                 xx
close                  gets                   neg                    strcmp
convert_chars_to_int   header                 open                   strcpy
digit                  i                      pipe                   strlen
(gdb) info locals digit
digits = "null)\000\000\000\060\061\062\063\064\065\066\067\070"
buf = '\000' <repeats 15 times>
i = <optimized out>
neg = 0
x = <optimized out>
(gdb)
```

دو رجیستر edi و esi درواقع به ترتیب به عنوان destination و source برای انجام یکسری از عملیات استفاده میشوند. معمولا به عنوان counter استفاده می شوند چون این دو ثبات با اجرای کد C تغییر نمیکنند.

۶. به کمک استفاده از **GDB**، درباره ساختار **input struct** موارد زیر را توضیح دهید: توضیح کلی این **struct** و متغیرهای درونی آن و نقش آنها، نحوه و زمان تغییر مقدار متغیرهای درونی(برای مثال، **input.e** در چه حالتی تغییر میکند و چه مقداری میگیرد)

این استراکت دارای سه متغیر r,w,e میباشد.
r اندیسی از بافر را نشان میدهد که تا آنجا محتوای بافر خوانده شده و توسط سیستم عامل به آن محتوا رسیدگی شده. w اندیسی از بافر را نشان میدهد که تا آنجا در بافر نوشته شده و پس از اتمام یک خط در کنسول، این مقدار توسط e آپدیت میشود. e هم اندیسی از بافر را نشان میدهد که در حال حاضر میتوان مقدارش را با keyboard input عوض کرد.
با فشردن کلمات کیبورد و یا با کمک قابلیت های پیاده شده میتوان مقدار e را تغییر داد.
هنگامی که بافر پر میشود (128تا کاراکتر روی بافر قرار داشته باشد) و یا enter فشرده میشود مقادیر w و r آپدیت میشوند و برابر e میشود.البته در کد ما این مقادیر با rightend عوض میشوند که بخاطر اضافه شدن دستورات جدید به ساختار اضافه شده است. همچنین این استراکت یک آرای 128تایی از کاراکتر نیز به نام buf دارد که همان بافری است که در خط فعلی ذخیره کرده ایم.

۷. خروجی دستورهای layout src و layout asm در TUI چیست؟
در layout src کد سورس برنامه مشاهده میشود.
در layout sam کد برنامه به فرم اسمبلی نمایش داده میشود.

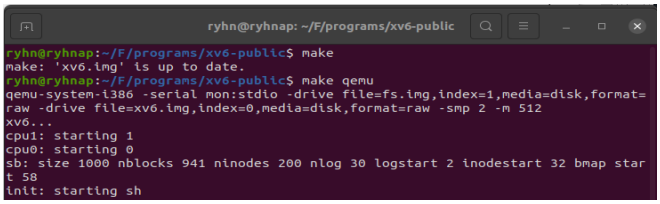
```
gdb kernel
File  Edit  View  Search  Terminal  Help
B+> 0x80100e70 <consoleintr+448>    sub    $0xc,%esp
    0x80100e73 <consoleintr+451>    push   $0x8010a520
    0x80100e78 <consoleintr+456>    call   0x80104a80 <release>
    0x80100e7d <consoleintr+461>    add    $0x10,%esp
    0x80100e80 <consoleintr+464>    test   %edi,%edi
    0x80100e82 <consoleintr+466>    jne    0x80100f40 <consoleintr+656>
    0x80100e88 <consoleintr+472>    lea    -0xc(%ebp),%esp
    0x80100e8b <consoleintr+475>    pop    %ebx
    0x80100e8c <consoleintr+476>    pop    %esi
    0x80100e8d <consoleintr+477>    pop    %edi
    0x80100e8e <consoleintr+478>    pop    %ebp
    0x80100e8f <consoleintr+479>    ret
    0x80100e90 <consoleintr+480>    mov    $0x1,%edi
    0x80100e95 <consoleintr+485>    jmp    0x80100cd0 <consoleintr+32>
    0x80100e9a <consoleintr+490>    lea    0x0(%esi),%esi
    0x80100ea0 <consoleintr+496>    call   0x801008e0 <uppercase_letters_to_end>
    0x80100ea5 <consoleintr+501>    jmp    0x80100cd0 <consoleintr+32>
    0x80100eaa <consoleintr+506>    lea    0x0(%esi),%esi
    0x80100eb0 <consoleintr+512>    mov    0x8010ffa8,%eax
    0x80100eb5 <consoleintr+517>    cmp    0x8010ffa4,%eax
    0x80100ebb <consoleintr+523>    je     0x80100cd0 <consoleintr+32>
    0x80100ec1 <consoleintr+529>    sub    $0x1,%eax
    0x80100ec4 <consoleintr+532>    subl   $0x1,0x8010ffac
    0x80100ecb <consoleintr+539>    mov    %eax,0x8010ffa8
    0x80100ed0 <consoleintr+544>    mov    $0x100,%eax
    0x80100ed5 <consoleintr+549>    call   0x80100410 <consputc>
    0x80100eda <consoleintr+554>    jmp    0x80100cd0 <consoleintr+32>
    0x80100edf <consoleintr+559>    nop
    0x80100ee0 <consoleintr+560>    call   0x80100a90 <reset_cursor>
    0x80100ee5 <consoleintr+565>    jmp    0x80100cd0 <consoleintr+32>
    0x80100eea <consoleintr+570>    lea    0x0(%esi),%esi
    0x80100ef0 <consoleintr+576>    mov    0x8010ffa8,%eax
    0x80100ef5 <consoleintr+581>    cmp    0x8010ffa4,%eax
remote Thread 1 In: consoleintr
(gdb)
```

۸. برای جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستورهایی استفاده میشود؟
up <number of jumps>:default = 1
down <number of jumps>:default = 1

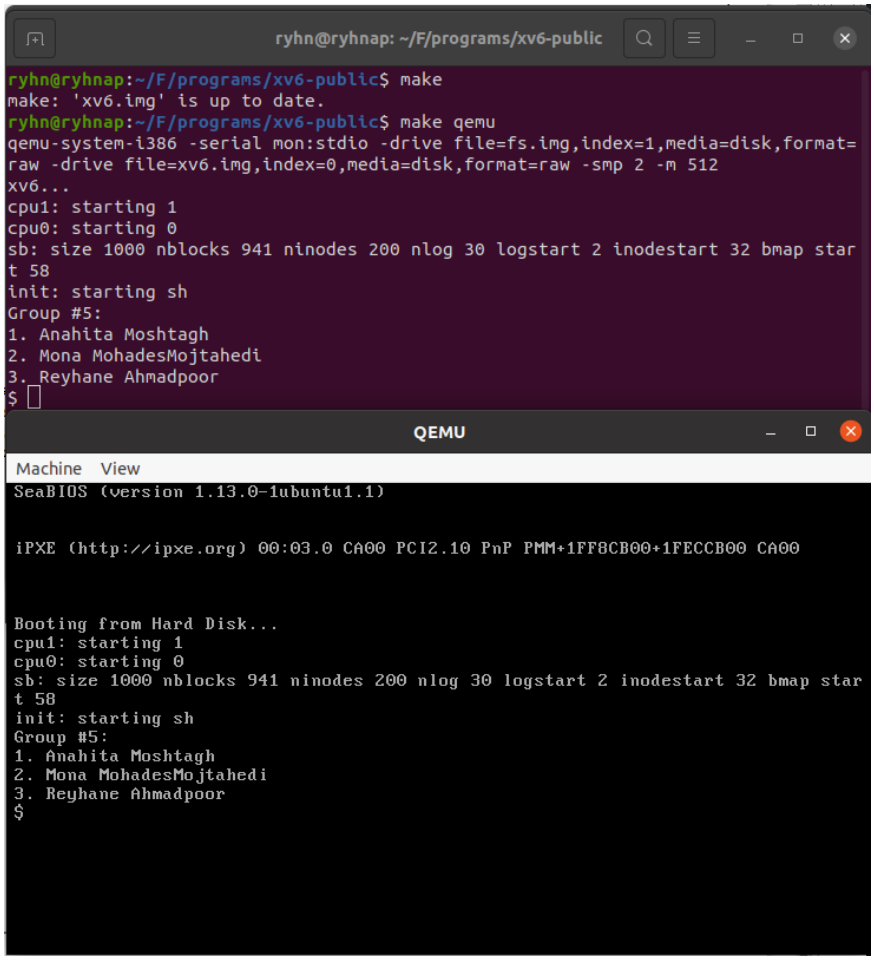
```
(gdb) down
Bottom (innermost) frame selected; you cannot go down.
(gdb) up
#1  0x80106180 in uartintr () at uart.c:76
76          consoleintr(uartgetc);
(gdb) up 2
#3  0x80105cbf in alltraps () at trapasm.S:20
20          call trap
(gdb) down 2
#1  0x80106180 in uartintr () at uart.c:76
76          consoleintr(uartgetc);
(gdb) up 4
#5  0x80112804 in cpus ()
(gdb) up 10
#8  0x8010363f in main () at main.c:37
37          mpmain();         // finish this processor's setup
(gdb) up
Initial frame selected; you cannot go up.
(gdb)
```

اجرا و اشکال زدایی:

به کمک لینکی در استک می‌توانیم qemu را اجرا کنیم و فقط پیش از آن طبق اسلاید qemu را نصب و کدهای xv6 را کلون می‌کنیم.

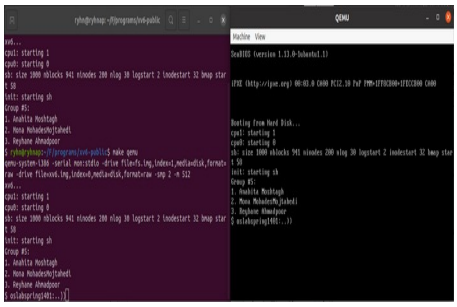


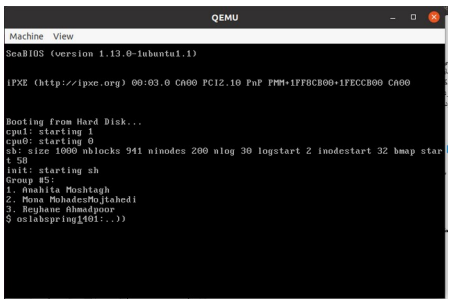
اضافه کردن یک متن به بوت مسیج:



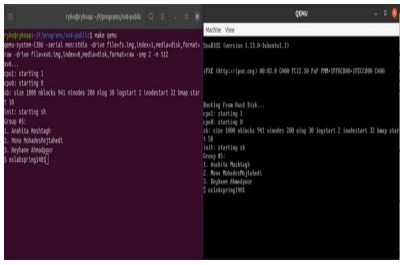
اضافه کردن چند قابلیت به کنسول:

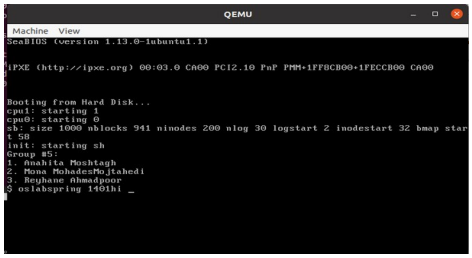
۱.چپ روی:



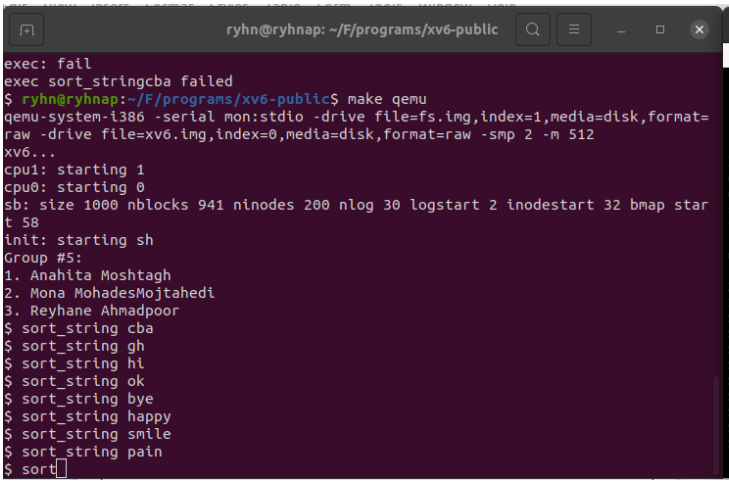


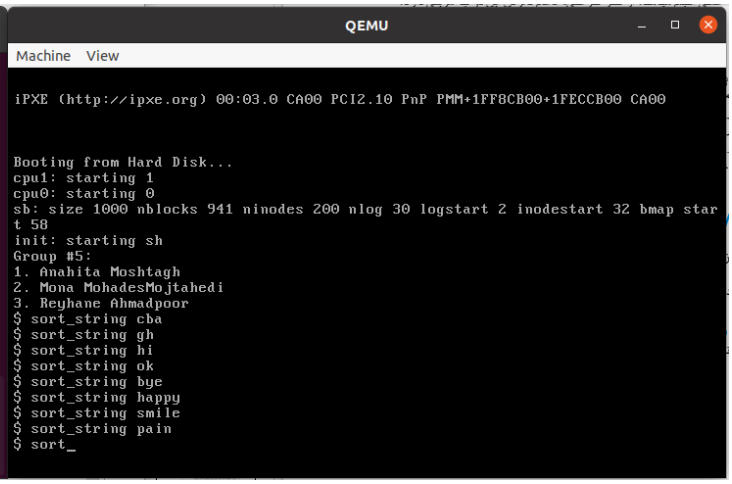
۲.راست روی:

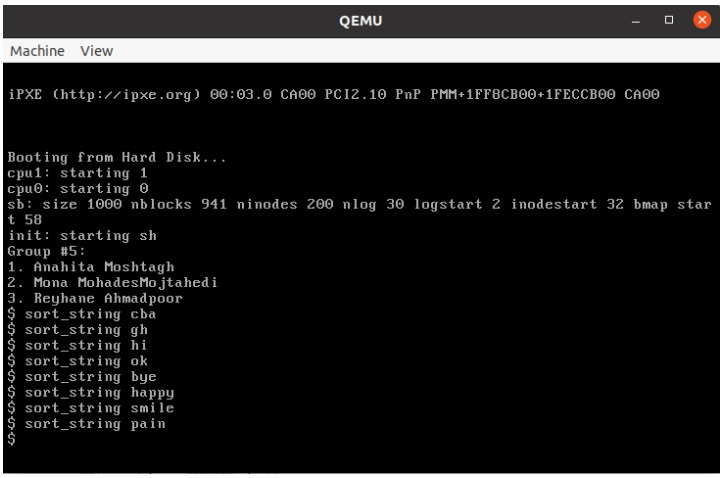


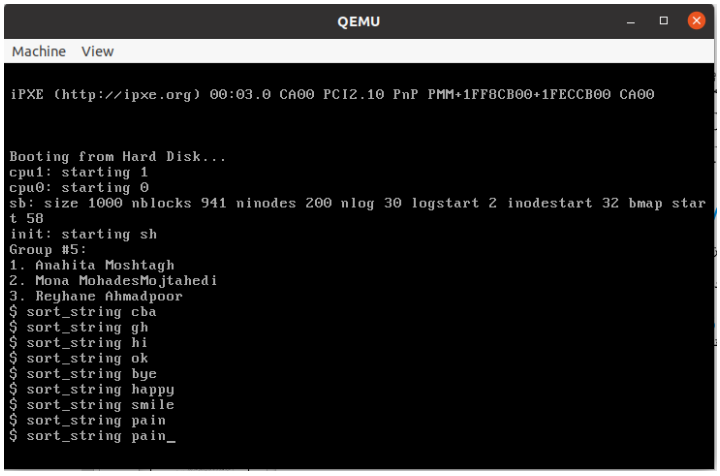


۳.بالا روی:

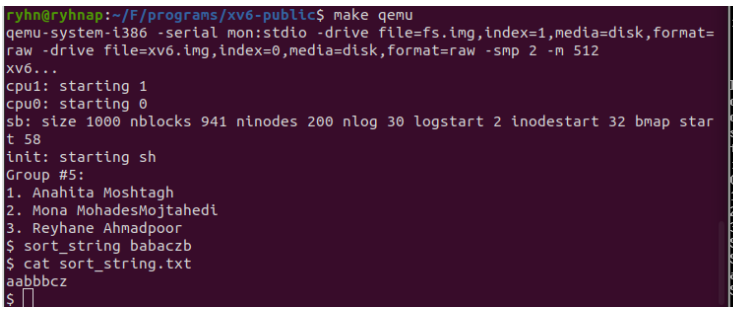


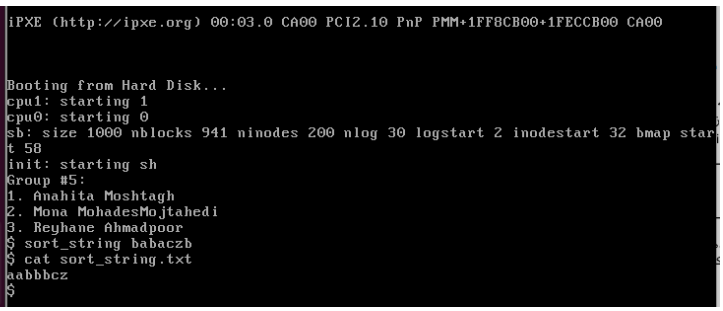






اجرا و پیاده‌سازی یک برنامه سطح کاربر:





امتیازی:

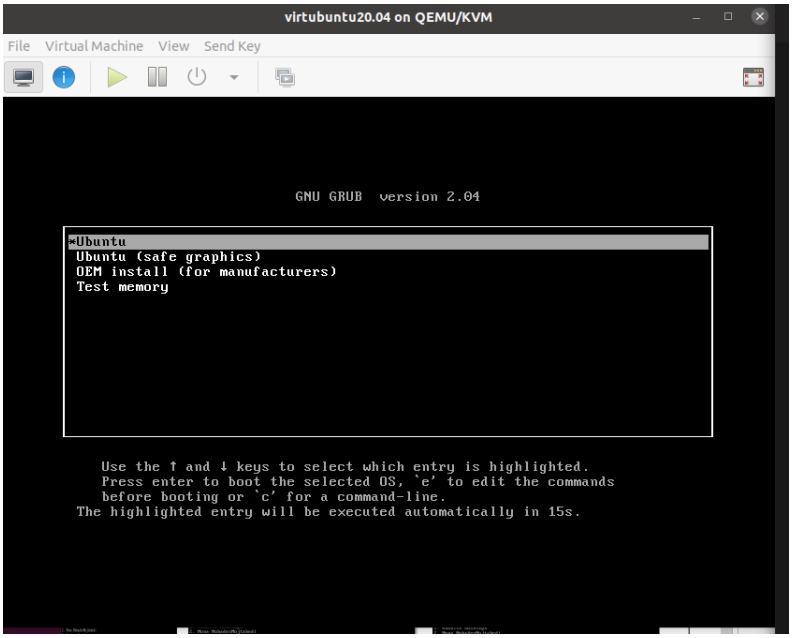
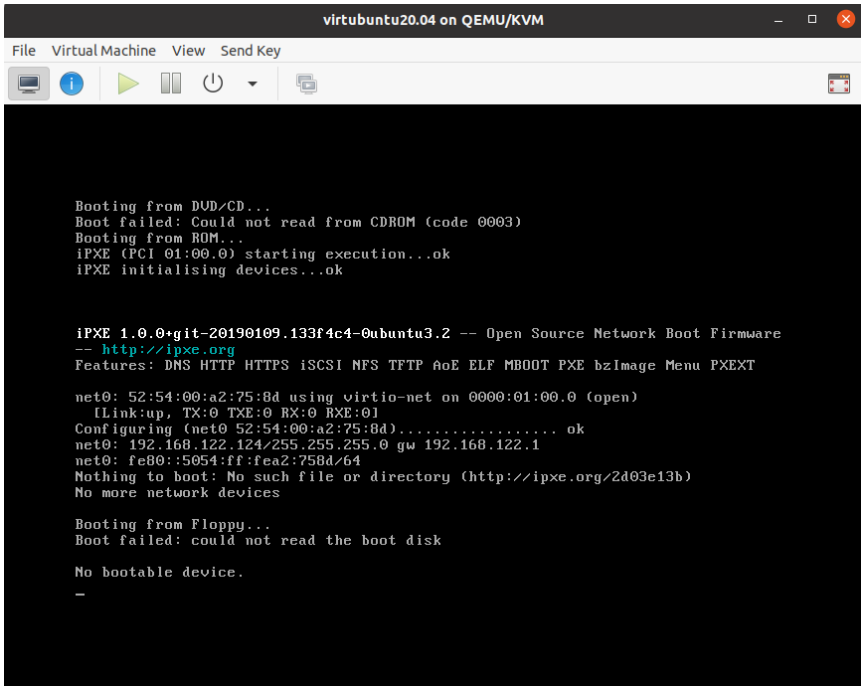
```
ryhn@ryhnap: ~/F/programs/linuxx/linux-5.16.14
ryhn@ryhnap:~/F/programs/linuxx/linux-5.16.14$ make defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/confdata.o
  HOSTCC  scripts/kconfig/expr.o
  LEX     scripts/kconfig/lexer.lex.c
/bin/sh: 1: flex: not found
make[1]: *** [scripts/Makefile.host:9: scripts/kconfig/lexer.lex.c] Error 127
make: *** [Makefile:619: defconfig] Error 2
ryhn@ryhnap:~/F/programs/linuxx/linux-5.16.14$ sudo apt-get install libncurses-d
ev flex bison openssl libssl-dev dkms libelf-dev libudev-dev libpci-dev libibert
y-dev autoconf
[sudo] password for ryhn:
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version (2.69-11.1).
autoconf set to manually installed.
libudev-dev is already the newest version (245.4-4ubuntu3.15).
libudev-dev set to manually installed.
The following additional packages will be installed:
  dctrl-tools libfl-dev libfl2 libssl1.1
Suggested packages:
  bison-doc debtags menu flex-doc ncurses-doc libssl-doc
```

```
ryhn@ryhnap: ~/F/programs/linuxx/linux-5.16.14
n auto mode
Setting up openssl (1.1.1f-1ubuntu2.11) ...
Setting up dctrl-tools (2.24-3) ...
Setting up libpci-dev:amd64 (1:3.6.4-1ubuntu0.20.04.1) ...
Setting up libfl-dev:amd64 (2.6.4-6.2) ...
Setting up dkms (2.8.1-5ubuntu2) ...
Processing triggers for libc-bin (2.31-0ubuntu9.7) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for install-info (6.7.0.dfsg.2-5) ...
ryhn@ryhnap:~/F/programs/linuxx/linux-5.16.14$ make defconfig
  LEX     scripts/kconfig/lexer.lex.c
  YACC    scripts/kconfig/parser.tab.[ch]
  HOSTCC  scripts/kconfig/lexer.lex.o
  HOSTCC  scripts/kconfig/menu.o
  HOSTCC  scripts/kconfig/parser.tab.o
  HOSTCC  scripts/kconfig/preprocess.o
  HOSTCC  scripts/kconfig/symbol.o
  HOSTCC  scripts/kconfig/util.o
  HOSTLD  scripts/kconfig/conf
*** Default configuration is based on 'x86_64_defconfig'
#
# configuration written to .config
#
ryhn@ryhnap:~/F/programs/linuxx/linux-5.16.14$
```

```
ryhn@ryhnap: ~/F/programs/linuxx/linux-5.16.14
.config - Linux/x86 5.16.14 Kernel Configuration

                Linux/x86 5.16.14 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

      General setup  --->
      [*] 64-bit kernel
          Processor type and features  --->
          Power management and ACPI options  --->
          Bus options (PCI etc.)  --->
          Binary Emulations  --->
      [*] Virtualization  --->
          General architecture-dependent options  --->
      [*] Enable loadable module support  --->
      -*- Enable the block layer  --->
      v(+)

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

```
ryhn@ryhnap:~/F/programs/linuxx/linux-5.16.14$ make menuconfig
  UPD     scripts/kconfig/mconf-cfg
  HOSTCC  scripts/kconfig/mconf.o
  HOSTCC  scripts/kconfig/lxdialog/checklist.o
  HOSTCC  scripts/kconfig/lxdialog/inputbox.o
  HOSTCC  scripts/kconfig/lxdialog/menubox.o
  HOSTCC  scripts/kconfig/lxdialog/textbox.o
  HOSTCC  scripts/kconfig/lxdialog/util.o
  HOSTCC  scripts/kconfig/lxdialog/yesno.o
  HOSTLD  scripts/kconfig/mconf


*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

ryhn@ryhnap:~/F/programs/linuxx/linux-5.16.14$
```