

گزارش آزمایشگاه سیستم عامل: پروژه شماره ۱

گروه ۲۲

پردیس زندگیری - محمد امانلو - شهرزاد ممیز

۸۱۰۱۰۰۸۴ - ۸۱۰۱۰۰۲۷۲ - ۸۱۰۱۰۰۸۱

Last commit code:

C98954f3eb6cac54ee2bf7528912069b448bd12f

Github repo:

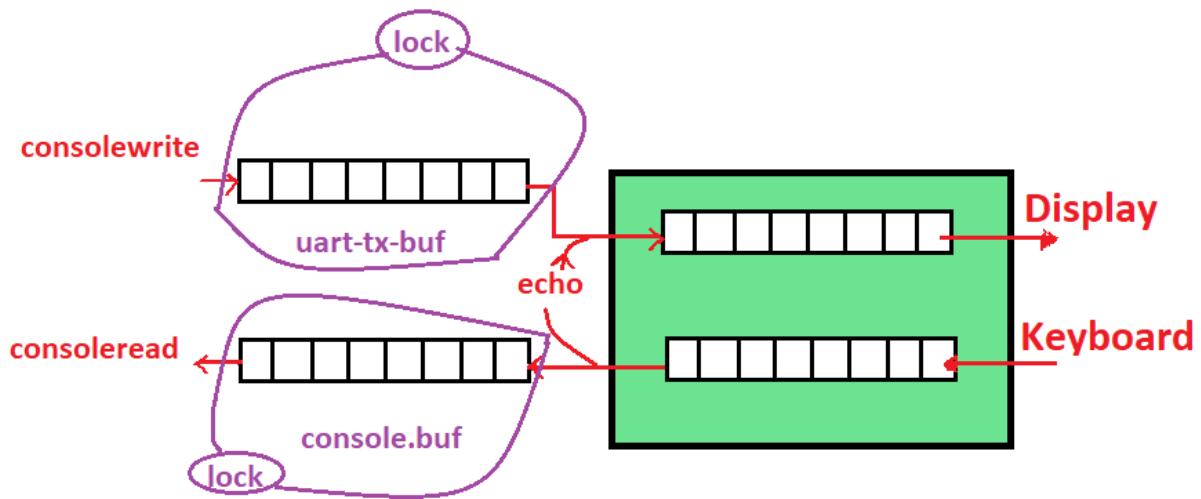
https://github.com/MohammadAmanlou/OS_lab

۱: معماری سیستم عامل xv6 چیست؟ چه دلایلی در دفاع از نظر خود دارد؟

xv6 یک پیاده‌سازی جدید از نسخه ششم یونیکس برای سیستم‌های چند پردازنده x86 و RISC-V است. معماری این سیستم عامل همانطور که در پی دی اف مستندات این سیستم عامل گفته شده است از معماری کلی یونیکس تبعیت می‌کند.

دسته بندی فایل‌های این سیستم عامل نیز مثل یونیکس system calls و user level program و مانند آن است.

این سیستم عامل در واقع شبیه یونیکس (Unix Like) و مشابه v6 Unix نوشته شده است. این سیستم عامل با AMSIC و برای x86 multiprocessor طراحی شده است؛ از دلایل این میتوان به وجود فایل asm.h که در آن استفاده از معماری x86 ذکر شده اشاره کرد؛ همچنین در فایل mmu.h از واحد مدیریت حافظه x86 استفاده شده است و در فایل x86.h دستورات اسمبلی که مخصوص پردازنده‌های مبتنی بر x86 است، بکار گرفته شده است. این موضوع از دسته بندی فایل‌ها که شامل user-level، system calls، file systems می‌شود نیز قابل مشاهده است.



۲: یک **process** در سیستم عامل xv6 از چه بخش هایی تشکیل شده است؟ این سیستم عامل به طور کلی چگونه پردازنده را به **process** های مختلف اختصاص میدهد؟

یک پردازه در سیستم عامل xv6 از بخش های زیر تشکیل شده است:
1. User-space-memory که شامل instructions و data و stack است.

2. State هر پردازنده که به طور خصوصی در اختیار kernel قرار دارد.
یک پردازه در این سیستم عامل از یک بخش که فقط برای هسته قابل دسترسی است و بخش دیگر که شامل دستورات و داده ها و stack و .. است تشکیل شده است.

این سیستم عامل با استفاده از روش time-sharing به صورت transparent پردازنده ها را مدیریت میکند. می دانیم روش multiprogramming برای time-sharing استفاده می شود. در این روش اگر برنامه ای در حال اجرا در صورتی که سهمیه زمانی استفاده اش از CPU به پایان برسد، دسترسی CPU از این process گرفته شده و محتوا رجیستر های آن در memory ذخیره شده و سپس process بعدی اجرا می شود و CPU به آن تخصیص داده می شود. وقتی دوباره نوبت به این process رسید محتوا از memory به رجیستر ها بازیابی می شوند. با این روش همه process ها به صورت هموارند رو به پیشرفت هستند. هسته سیستم عامل برای رهگیری درست هر process به آن یک کد به نام pid اختصاص می دهد.

PCB in xv6: struct proc

Page 23, process structure and process states

```

2334 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
2335
2336 // Per-process state
2337 struct proc {
2338     uint sz;                      // Size of process memory (bytes)
2339     pde_t* pgdir;                 // Page table
2340     char *kstack;                 // Bottom of kernel stack for this process
2341     enum procstate state;        // Process state
2342     int pid;                     // Process ID
2343     struct proc *parent;         // Parent process
2344     struct trapframe *tf;        // Trap frame for current syscall
2345     struct context *context;    // swtch() here to run process
2346     void *chan;                  // If non-zero, sleeping on chan
2347     int killed;                 // If non-zero, have been killed
2348     struct file *ofile[NFILE];   // Open files
2349     struct inode *cwd;          // Current directory
2350     char name[16];              // Process name (debugging)
2351 };
2352

```

۴: فراخوانی های سیستمی **exec** و **fork** چه عملی انجام میدهد؟ از نظر طراحی ادغام نکردن این دو چه مزیتی دارد؟

- **فراخوانی سیستمی exec:** حافظه پردازه فعلی را با یک حافظه جدید که در آن یک برنامه با فایل ELF لود شده است، جایگزین میکند. در واقع (`exec()`) راهی برای اجرای یک برنامه در پردازه فعلی است. این فراخوانی به یک فرآیند اجزای انتقال به یک برنامه دیگر را میدهد. مانند زمانی که یک برنامه نیاز به اجرای برنامه دیگری را دارد. در اثر اجرای این فراخوانی سیستمی کد و داده های برنامه قدیمی از حافظه حذف می شوند و فضای آدرس پاک میشود و کد و داده های برنامه جدید در فضای آدرس **process** بارگذاری میشود.
- بر خلاف تابع **fork** برنامه به **caller** تابع (`exec()`) باز نمیگردد و برنامه جدید اجرا میشود. مگر اینکه در زمان اجرای این تابع یک ارور رخ دهد. برنامه جدید اجرا شده در یک نقطه با استفاده از تابع **exit** اجرای پردازه را خاتمه میدهد

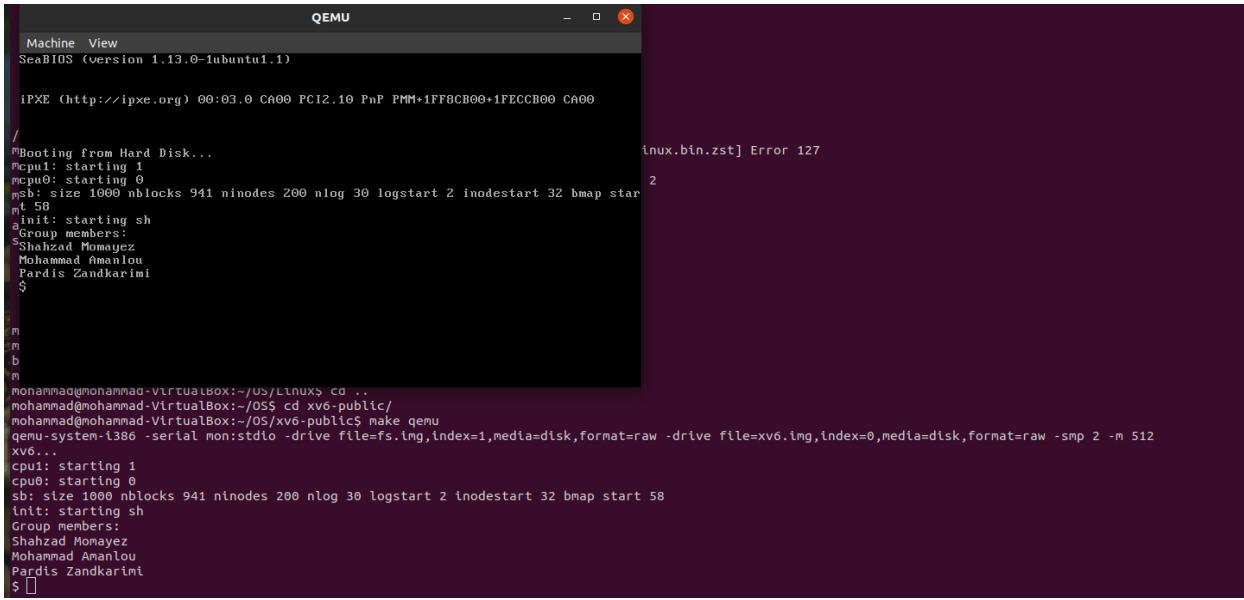
- **فراخوانی سیستمی fork:** به طور کلی پردازشی که امکان ایجاد یک پردازش دیگر را داشته باشد از فراخوانی سیستمی **fork** بهره می برد. پردازشی که بواسطه پردازش اولیه ایجاد شده است را **child** می نامیم. پردازش **child** با همان حافظه **parent** خود کار می کند. در واقع این فراخوانی یک نسخه کپی از پردازه هایی میسازد که این فراخوانی را صدا زده اند. همان

طور که گفته شد سیستم عامل برای هر process یک pid در نظر می‌گیرد در صورت استفاده از fork پس از اجرای parent، وقتی نوبت به اجرای child می‌رسد pid متناسب با child را برمی‌گرداند و پس از اجرای child به نشانه پایان عملیات 0 pid= را برمی‌گرداند. چون این دستور برای ایجاد برنامه جدید که فرزند برنامه فعلی به حساب می‌آید تعییه شده است. اگر برنامه فرزند برنامه به دلیلی متوقف شود و یا اجرا به طول بینجامد برای برنامه پدر نیز همین انفاق می‌افتد. پس از اجرای برنامه فرزند به برنامه پدر بازمی‌گردیم. تفاوت این دو دستور در اینجا مشخص می‌شود. در فراخوانی سیستمی exec از برنامه جدید لزوماً به برنامه قبلی برنامی گردیم و همین موضوع باعث ایجاد کاربرد های متفاوت و در نتیجه عدم ادغام این دو دستور شده است.

ادغام نکردن این دو فراخوانی این امکان را میدهد که تغییرات لازم را در صورت لزوم پس از فراخوانی file descriptor fork در exec انجام دهد و سپس فراخوانی exec انجام شود.

در واقع به طور دقیق تر مزیت اصلی ادغام نکردن این دو فراخوانی در این است که به هنگام IO وقایتی کاربر در shell برنامه ای را اجرا می‌کند. Shell به صورت اتوماتیک دستور را از ترمینال می‌خواند، و با استفاده از فراخوانی fork یک پردازه جدید می‌سازد. در پردازه فرزند با استفاده از تابع exec برنامه درخواست شده توسط کاربر را جایگزین پردازه فعلی (فرزنده) می‌کند. پس از اتمام پردازه فرزند به main باز می‌گردد و منتظر دستور جدید می‌شود. در صورتی که این دو تابع ادغام شوند، یا باید حالت‌های redirection به عنوان پارامتر به تابع redirect پاس داده شوند که هندل کردن این حالت دردرس‌های خودش را دارد و یا اینکه shell پیش از اجرای این تابع، file descriptor های خود را تغییر دهد و بعد از اتمام کار این تابع نیز به حالت قبل برگرداند. در واقع اگر فراخوانی های سیستمی exec و fork ادغام بود، یک طرح پیچیده‌تر برای redirect کردن standard I/O توسط shell نیاز بود یا خود برنامه باید می‌فهمید که چگونه standard I/O را redirect کند.

اضافه کردن یک متن به Boot massage



برای پیاده سازی این دستور در فایل `init.c` یک دستور `printf` در `boot message` می کنیم. که در شکل زیر قابل مشاهده است.

```

10 int
11 main(void)
12 {
13     int pid, wpid;
14
15     if(open("console", O_RDWR) < 0){
16         mknod("console", 1, 1);
17         open("console", O_RDWR);
18     }
19     dup(0); // stdout
20     dup(0); // stderr
21
22     for(;){
23         printf(1, "init: starting sh\n");
24         printf(1, "Group members:\nShahzad Momayez\nMohammad Amanlou\nPardis Zandkarimi\n");
25         pid = fork();
26         if(pid < 0){
27             printf(1, "init: fork failed\n");
28             exit();
29         }
30         if(pid == 0){
31             exec("sh", argv);
32             printf(1, "init: exec sh failed\n");
33             exit();
34         }
35         while((wpid=wait()) >= 0 && wpid != pid)

```

اضافه کردن چند قابلیت به کنسول xv6
:ctrl + B دستور

در ابتدا در `qemu` مشابه زیر سیستم عامل اجرا می شود.

The screenshot shows a terminal window titled "QEMU" with the following content:

```

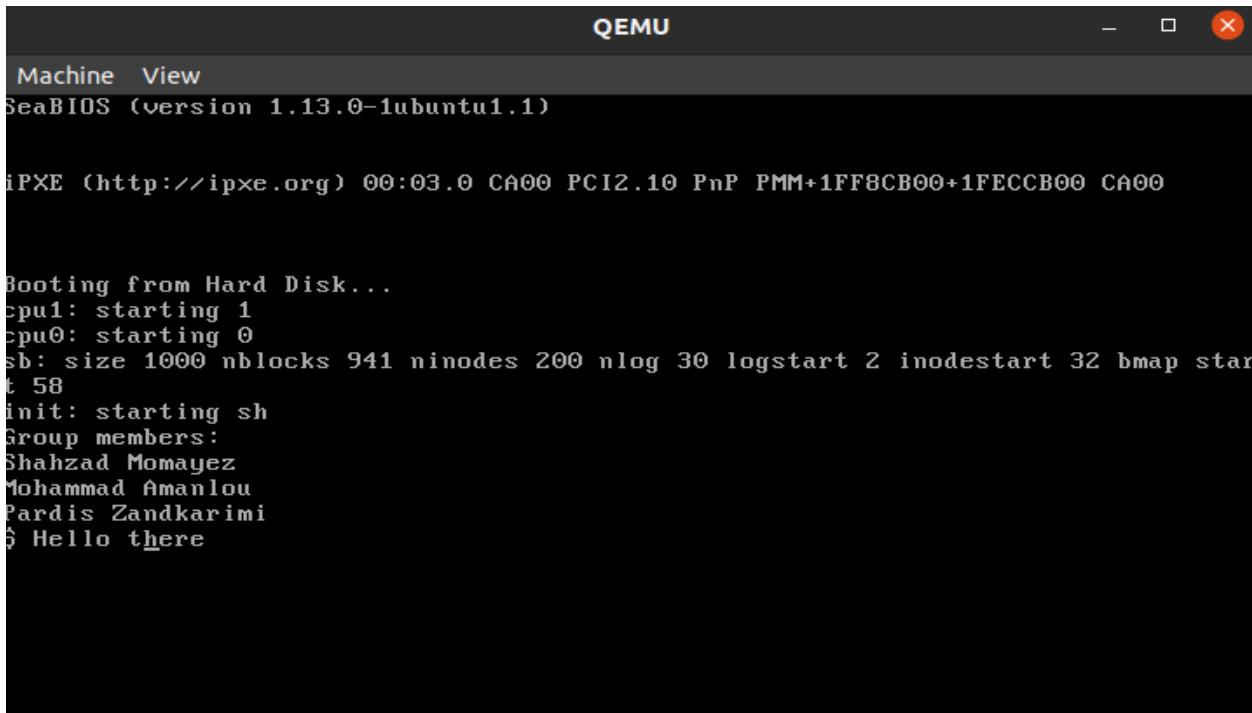
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCIZ.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 start 58
init: starting sh
Group members:
Shahzad Momagez
Mohammad Amanlou
Pardis Zandkarimi
$ Hello there_

```

پس از فشردن هر بار **ctrl+B** طبق دستورالعمل، **cursor** یک خانه به عقب منتقل می شود و دستورات از این به بعد در این نقطه اعمال می شود. نکته مهم این است که وقتی **cursor** عقبگرد میکند پس از هر بار نوشتن در محلی بین چند کاراکتر باید کاراکترهای بعد **cursor** یک واحد شیفت به راست و با هر بار استفاده از **BackSpace** کاراکترهای بعد از آن یک واحد شیفت به چپ باید داشته باشند. همچنین باید توجه کرد که پس از اعمال این دستور باید دستورهای دیگر و همچنین پس از اعمال دستورات دیگر این دستور بتواند به درستی اجرا شود.



QEMU

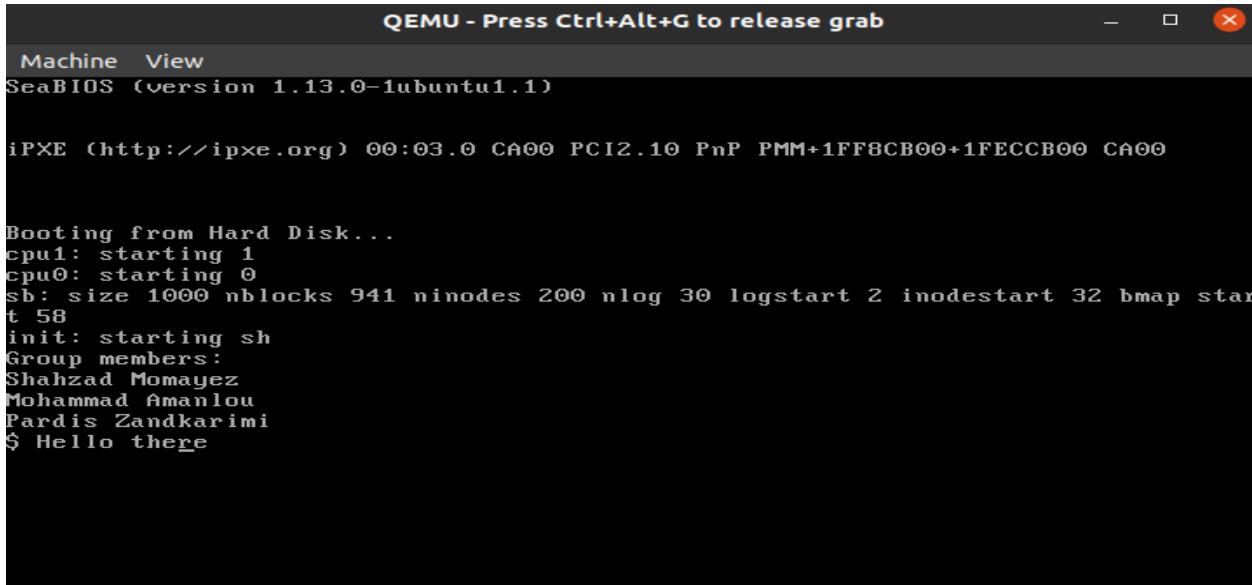
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

```
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00

Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Group members:
Shahzad Momayez
Mohammad Amanlou
Pardis Zandkarimi
$ Hello there
```

شکل فوق نتیجه حاصل از 3 بار فشردن کلید میانبر **ctrl+B** را نمایش می دهد.

:ctrl + F



QEMU - Press Ctrl+Alt+G to release grab

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 start 58
init: starting sh
Group members:
Shahzad Momayez
Mohammad Amanlou
Pardis Zandkarimi
$ Hello there
```

تصویر فوق نیز جایگاه cursor پس از یک بار فشردن کلید میانبر **ctrl+F** را نمایش می دهد.
منطق پیاده سازی این دو دستور و بخشی از کدهای نوشته شده برای این دو دستور به نحو زیر است:

```
23
24     static int backs = 0;
25
```

مطابق فوق یک متغیر به نام `backs` تعریف شده که تعداد بازگشت به عقب ها تا آن لحظه را در خود نگه میدارد. با هر بار اجرای دستور یا پاک کردن صفحه و ... که باعث به ابتدای خط رفتن می شود. تعداد `backs` مجدداً صفر می شود. سپس هر عملیات از جمله نوشتن، حذف کردن، عقب یا جلو کردن و ... با توجه به مقدار `backs` بازنویسی می شود.
دو عملیات اصلی برای این کار تعریف شده است که به نام های توابع `pull_back` و `push_forward` در فایل `console.c` قابل مشاهده است.

```

static void pull_back(int pos ){
    for (int i = pos - 1; i < pos + backs; i++)
        crt[i] = crt[i + 1];
}

static void push_forward(int pos) {
    for (int i = pos + backs; i > pos; i--)
        crt[i] = crt[i - 1];
}

```

در نهایت نیز با اضافه کردن یک `case` به تابع `consoleintr` دستور `ctrl+B` اجرا میکنیم. (تابع `consoleintr` به تابع `ctrl+B` دستور `case` نیز با گرفتن یک `enum` `curs_action` که نشانگر جهت حرکت `cursor` است مقدار `pos` را به `update` می کند).

```

286 static void curs_action(enum curs_act act)
287 {
288     int pos;
289
290     // get cursor position
291     outb(CRTPORT, 14);
292     pos = inb(CRTPORT + 1) << 8;
293     outb(CRTPORT, 15);
294     pos |= inb(CRTPORT + 1);
295
296     //if
297     switch (act)
298     {
299         case BACK:
300             pos--;
301             break;
302         case FORWARD:
303             pos++;
304             break;
305         default:
306             break;
307     }
308
309     // reset cursor
310     outb(CRTPORT, 14);
311     outb(CRTPORT + 1, pos >> 8);
312     outb(CRTPORT, 15);

```

:دستور L

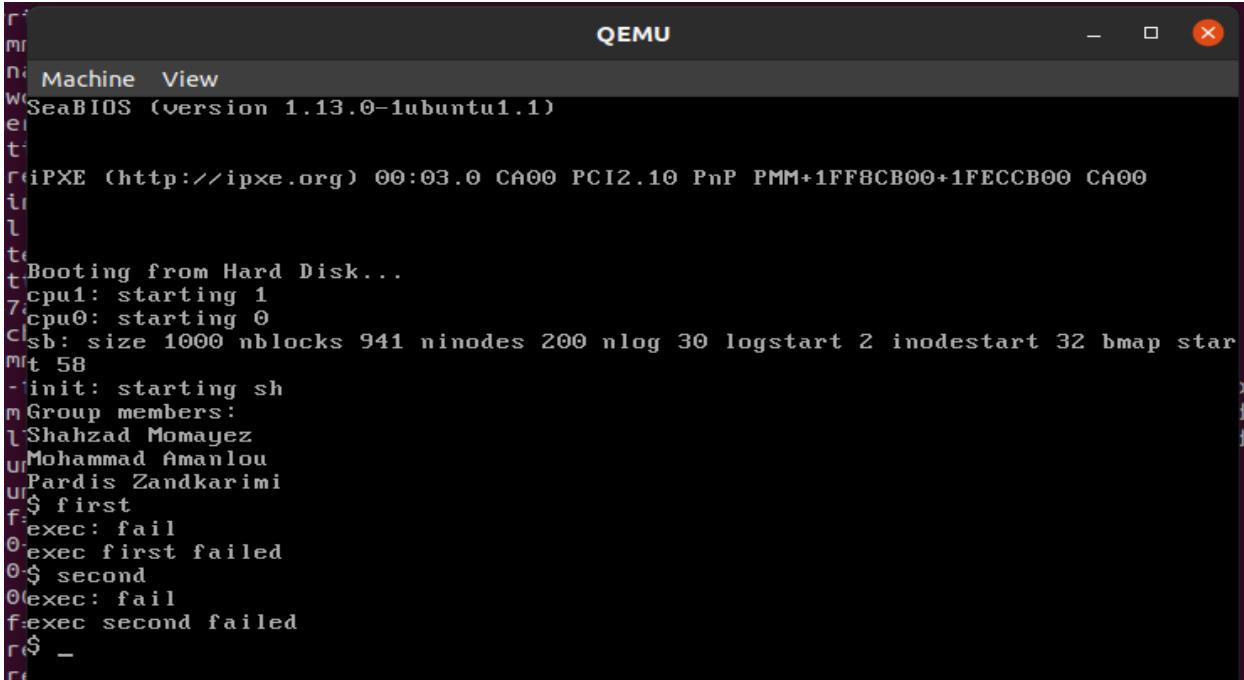
پس از هر بار فشرده شدن این کلید میانبر تمامی محتوای داخل صفحه پاک شده و به ابتدای سطر می رویم. و آماده دریافت دستور جدید می شویم. باید توجه کرد ابتدای خط با علامت \$ مشخص شده است. همچنین با هر بار فشرده شدن این کلید میانبر متغیر backs صفر می شود.



```
246     static void clear_terminal()
247     {
248         int pos;
249
250         // get cursor position
251         outb(CRTPORT, 14);
252         pos = inb(CRTPORT + 1) << 8;
253         outb(CRTPORT, 15);
254         pos |= inb(CRTPORT + 1);
255
256
257
258         for (int pos = 0; pos < SCREEN_SIZE ; pos++){
259             consputc(BACKSPACE);
260         }
261         backs = 0;
262         input.e = input.w = input.r = 0;
263         consputc('$');
264         consputc(' ');
265         pos = 2;
266
267         // reset cursor
268         outb(CRTPORT, 14);
269         outb(CRTPORT + 1, pos >> 8);
270         outb(CRTPORT, 15);
271         outb(CRTPORT + 1, pos);
272     }
```

دستور :Arrow Up

با زدن این کلید میانبر می بایست تا ده دستور قبلی نمایش داده شود. برای این کار یک struct به نام history در فایل `console.c` ایجاد کردیم که در یک آرایه در آن مقادیری که در `input` داریم را ذخیره میکنیم. سپس با هر بار دستور بالا یا پایین این مقادیر را بازنشانی میکنیم. همچنین با بازنشانی این مقادیر باید مطمئن باشیم که سایر دستورات روی این مقادیر به نحو صحیحی کار میکند.

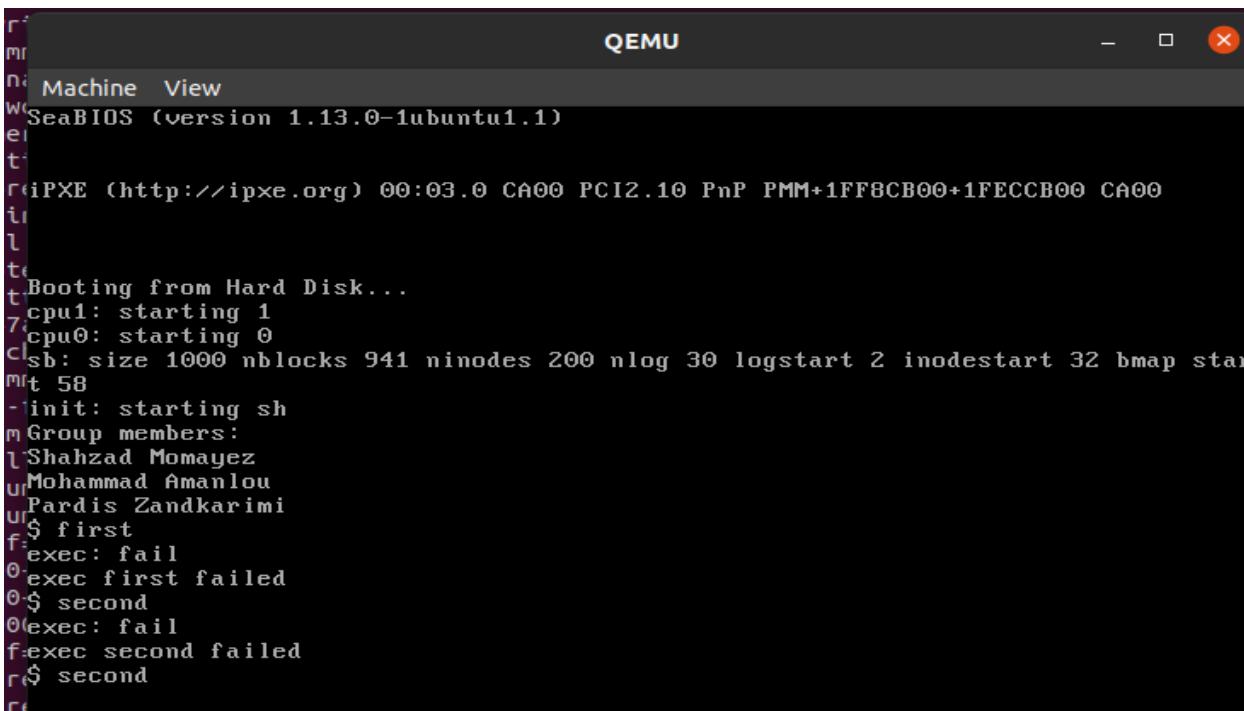


```

r
MI Machine View
W SeaBIOS (version 1.13.0-1ubuntu1.1)
e
t
r(iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00
i
l
t
t Booting from Hard Disk...
t cpu1: starting 1
7 cpu0: starting 0
cl sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
mt 58
-t init: starting sh
m Group members:
l Shahzad Momayez
ur Mohammad Amanlou
ur Pardis Zandkarimi
ur $ first
f exec: fail
0 exec first failed
0 $ second
0 exec: fail
f exec second failed
r $ -
r

```

پس از یک بار دستور بالا می بینیم که مقدار `second` در خط فرمان قرار می گیرد.



```

r
MI Machine View
W SeaBIOS (version 1.13.0-1ubuntu1.1)
e
t
r(iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00
i
l
t
t Booting from Hard Disk...
t cpu1: starting 1
7 cpu0: starting 0
cl sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
mt 58
-t init: starting sh
m Group members:
l Shahzad Momayez
ur Mohammad Amanlou
ur Pardis Zandkarimi
ur $ first
f exec: fail
0 exec first failed
0 $ second
0 exec: fail
f exec second failed
r $ second
r

```

دستور :Arrow Down

پس از چند بار استفاده از دستور Arrow up با استفاده از دستور Arrow down می توانیم به دستور بعدی برویم.

```

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

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
mt 58
init: starting sh
Group members:
Shahzad Momayez
Mohammad Amanlou
Pardis Zandkarimi
$ first
exec: fail
exec first failed
$ second
exec: fail
exec second failed
$ first
$ second

```

پس از دو بار استفاده از Arrow up به مقادیر فوق می رسیم. حال با یک بار فشردن کلید down مجدداً مقدار second در خط فرمان قرار می گیرد.

```

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

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
mt 58
init: starting sh
Group members:
Shahzad Momayez
Mohammad Amanlou
Pardis Zandkarimi
$ first
exec: fail
exec first failed
$ second
exec: fail
exec second failed
$ second

```

اجرا و پیاده سازی برنامه سطح کاربر:

```

1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4 #include "fcntl.h"
5
6 int main(int argc, char *argv[])
7 {
8     if(argc != 3){
9         printf(2, "Usage: strdiff <string1> <string2>\n");
10        exit();
11    }
12    char *str1 = argv[1];
13    char *str2 = argv[2];
14
15    int i = 0;
16    unlink("strdiff_result.txt");
17    int fd = open("strdiff_result.txt", O_CREATE | O_RDWR);
18    if(fd < 0){
19        printf(2, "strdiff: cannot open/create strdiff_result.txt\n");
20        exit();
21    }
22    while(str1[i] != '\0' && str2[i] != '\0'){
23        if(str1[i] != str2[i]){
24            //printf(1, "%d", ((int)str1[i]) < ((int)str2[i]));
25            char diff[2];
26            if(((int)str1[i]) < ((int)str2[i])){
27                diff[0] = '1';
28            }
29            else{
30                diff[0] = '0';
31            }
32            diff[1] = '\0';
33            write(fd, diff, 1);
34        }
35        i++;
36    }
37    if(str2[i] != '\0'){
38        while(str2[i] != '\0'){
39            //printf(1, "%d" , 1);
40            char diff[2];
41            diff[0] = '1';
42            diff[1] = '\0';
43            write(fd, diff, 1);
44            i++;
45        }
46    }
47    else if(str1[i] != '\0'){
48        while(str1[i] != '\0'){
49            //printf(1, "%d" , 0);
50            char diff[2];
51            diff[0] = '0';
52            diff[1] = '\0';
53            write(fd, diff, 1);
54            i++;
55        }
56    }
57    close(fd);
58    exit();
59 }
60 }
```

```
$ strdiff apple banana
$ cat strdiff_result.txt
: 100011
$ -
```

۸: در **Makefile** متغیر هایی به نام های **UPROGS** و **ULIB** تعریف شده است. کاربرد آن ها چیست؟

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

ULIB معادل User Libraries و **UPROGS** معادل Programs User های کاربر و کتابخانه های کاربر محسوب می شوند.

UPROGS : برای برنامه های سطح کاربر مورد استفاده قرار میگیرد که قابلیت اجرا در xv6 را دارند. این متغیر لیستی از این برنامه ها را شامل میشود.

برنامه یا دستور هایی که کاربر میتوانند اجرا کنند شامل

```
cat\ _echo\ _forktest\ _grep\ _init\ _kill\ _ln\ _ls\ _mkdir\ _rm\ _sh\_
\stressfs
\usertests\ _wc\ _zombie_
```

هستند که همه ی آن ها در UPROG 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 -
- ...

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

۱۱: برنامه های کامپایل شده در فایل های دودویی نگهداری میشوند. فایل های 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 اسمبلى تبدیل کنید (ابزار objdump استفاده شود. بخشی از آن مشابه فایل bootasm.S باشد)

این فایل از نوع ELF میباشد.

در ابتدای بوت سیستم، فایل S.bootasm که یک کد به زبان اسمبلى است، اجرا میشود و پردازنده را در حالت 32 بیت mode-protected قرار داده که این حالت باعث میشود رجیسترها، آدرسهاي مجازی و اکثر محاسبات عددی به جای 16 بیت با 32 بیت هنگل شوند. برای این که این تغییر مدام انجام شود نیاز به زبان اسمبلى داریم زیرا processor در ابتدا تنها زبان اسمبلى را میفهمد.

تفاوت این فایل با دیگر فایل های دودویی در نداشتن هدر(header) میباشد. چون cpu وظیفه اجرای instruction را بر عهده دارد و هدر های فایل دودویی اینستراکشن نیستند و cpu توانایی فهمیدن آنها را ندارد. به همین علت از این نوع فایل دودویی استفاده میشود.

تبدیل فایل دودویی به اسمبلى:

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

```
objdump: warning: No such file
pardis@pardis-VirtualBox:~/Downloads/OS/xv6-public$ objdump -S bootblock.o > bootblock.asm
pardis@pardis-VirtualBox:~/Downloads/OS/xv6-public$ 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
```

۱۲: 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 کپی کند و **bootblock** که یک فایل دودویی خام (raw binary file) میباشد را تولید کند. بخش **text** فایل همان اینسٹراکشن‌ها هستند که برای **cpu** قابل اجرا میباشند.

این دستور محتوای یک **file object** را در یک فایل دیگر کپی میکند و برای خواندن و نوشتن **file** از کتابخانه **BFD GNU object** استفاده میکند و میتواند **file object** مقصد را با فرمتی متفاوت از **file object** مبدا بنویسد. فایلهای **temporary** درست میکند تا ترجمه هایش را انجام دهد و سپس آنها را حذف میکند؛ به تمامی فرمتهای توصیف شده در **BFD** دسترسی دارد و به همین دلیل میتواند بیشتر فرمتهای را بدون اینکه صریحاً به آن اعلام شود، تشخیص دهد.

۱۴: یک segment register یک general purpose register و یک control register status register هریک را به صورت مختصر توضیح دهید.

ثبات قطعه:

آدرس کد و استک و داده در آن نگهداری میشود.

مانند:

ss پوینتر به استک

ds پوینتر به استک

cs پوینتر به کد

ثبات عام منظوره:

این سیستم عامل دارای ۸ ثبات عام منظوره میباشد که عبارتند از :

%eax, %ebx, %ecx, %edx, %edi, %esi, %ebp, %esp

همچنین **xv6** دارای پروگرم کاونتری به نام **eip** نیز میباشد.

از این نوع ثبات برای نگهداری عملیات‌های ریاضی و بعضی اشاره‌گرهای داده‌ها استفاده میشود.

ثبات وضعیت:

این نوع ثبات اطلاعاتی راجع به وضعیت **cpu** را در خود دارد. **EFLAGS** هایی مانند **zero**, **sign**, **..** در این بخش مشخص میشوند.

carry

ثبات کنترلی:

کنترل **cpu** در این بخش انجام میشود و وظیفه تغییر مدل آدرس دهی و کنترل تداخل (interrupt) و **paging** است که شامل `%cr0`, `%cr2`, `%cr3`, `%cr4` و ... میباشد.

۱۸: کد معادل **entry.S** در هسته لینوکس را بیابید

<https://github.com/torvalds/linux/blob/master/arch/x86/entry/entry.S>

۱۹: چرا این آدرس فیزیکی است؟

در هر صورت ما نیاز به آدرسی فیزیکی جهت لینک کردن این آدرس های مجازی به حافظه سیستم داریم و در صورت مجازی مشخص کردن این بخش نیز نیاز به بخشی فیزیکی جهت مشخص کردن آن بود.

به بیانی دیگر برای تبدیل آدرس مجازی به آدرس فیزیکی نیازمند **Table** ذکر شده هستیم و برای دسترسی به این جدول نیاز به آدرس آن داریم. در صورتی که آدرس این جدول به صورت مجازی ذخیره شود، برای پیدا کردن آدرس فیزیکی اش به خودش نیاز خواهیم داشت.

۲۲: 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 نمی گذارد. زیرا تمامی قطعه ها اعم از کد و داده روی یکدیگر می افتد. با این حال برای کد و داده های سطح کاربر پرچم **SEG_USER** تنظیم شده است. چرا؟ (راهنمایی: علت مربوط به ماهیت دستورالعمل ها و نه آدرس است.)

جهت ایجاد تمایز میان پردازنده های سطح کاربر و پردازنده های سطح هسته از فلگ **SEG_USER** استفاده میشود.

۲۳: جهت نگهداری اطلاعات مدیریتی برنامه های سطح کاربر ساختاری تحت عنوان **struct proc** خط ۲۳۶ ارائه شده است. اجزای آن را توضیح داده و ساختار معادل آن در سیستم عامل لینوکس را بیابید.

اجزای ساختار :**proc struct**

• **Pgdir** : پوینتر متعلق به **page table**

- Name : نام پردازنده مورد استفاده
- SZ : سایز حافظه پردازنده
- Kstack : پایین استک کرنل که در پردازنده وجود دارد.
- State : وضعیت پردازنده
- Parent : سازنده پردازنده
- Pid : عدد اختصاص داده شده به این پردازنده
- Tf : چارچوب trap interrupt برای فراخوانی سیستمی فعلی
- Context : برای نگهداری شده است context switching
- Chan : در صورت صفر بودن به این معناست که پردازنده موقتاً غیر فعال است.
- Killed : اگر صفر نباشد به معنای kill شدن پردازه های پردازنده است
- Ofile : فایل های باز شده
- Cwd : نمایانگر پوشه کنونی

ساختار معادل این استراکت در کرنل لینوکس به نام `task_struct` آمده است.
[linux/sched.h at master · torvalds/linux \(github.com\)](https://github.com/torvalds/linux/blob/master/include/linux/sched.h)

```

42  /* task_struct member predeclarations (sorted alphabetically): */
43  struct audit_context;
44  struct bio_list;
45  struct blk_plug;
46  struct bpf_local_storage;
47  struct bpf_run_ctx;
48  struct capture_control;
49  struct cfs_rq;
50  struct fs_struct;
51  struct futex_pi_state;
52  struct io_context;
53  struct io_uring_task;
54  struct mempolicy;
55  struct nameidata;
56  struct nsproxy;
57  struct perf_event_context;
58  struct pid_namespace;
59  struct pipe_inode_info;
60  struct rcu_node;
61  struct reclaim_state;
62  struct robust_list_head;
63  struct root_domain;
64  struct rq;
65  struct sched_attr;
66  struct sched_param;
67  struct seq_file;
68  struct sighand_struct;
69  struct signal_struct;
70  struct task_delay_info;
71  struct task_group;
72  struct user_event_mm;

```

۲۷: کدام بخش از آماده سازی سیستم، بین تمامی هسته های پردازنده مشترک و کدام بخش اختصاصی است؟
 از هر کدام یک مورد را با ذکر دلیل توضیح دهید. زمان بند روی کدام هسته اجرا می شود؟

برای جواب دادن به پرسش فوق، می بایست به فایل main.c در کد xv6 مراجعه کنیم.

با اجرا شدن تابع `main`، تمامی توابع موجود در آن تابع توسط اولین هسته اجرا میشود، سپس هنگام رسیدن به تابع `start_others`، سایر هسته ها شروع به کار میکنند. توابعی که در تابع `main` وجود دارند به صورت اختصاصی توسط اولین هسته استفاده میشوند، اما توابعی که در تابع `npmain()` استفاده شده اند به صورت مشترک توسط تمامی هسته ها استفاده میشوند، مانند تابع `(scheduler)` در تابع `mpmain` صدا زده میشود که این تابع بین تمامی هسته ها مشترک است. کد های مربوط به مدیریت هسته ها در زیر آمده است:

```

main.c
~/Downloads/OS/xv6-public

4 seginit();           // segment descriptors
5 picinit();           // disable pic
6 ioapicinit();        // another interrupt controller
7 consoleinit();       // console hardware
8 uartinit();          // serial port
9 pinit();              // process table
0 tvinit();             // trap vectors
1 binit();              // buffer cache
2 fileinit();           // file table
3 ideinit();            // disk
4 startothers();        // start other processors
5 kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()
6 userinit();           // first user process
7 mpmain();              // finish this processor's setup
8 }
9
0 // Other CPUs jump here from entryother.S.
1 static void
2 mpenter(void)
3 {
4     switchkvm();
5     seginit();
6     lapicinit();
7     mpmain();
8 }
9
0 // Common CPU setup code.
1 static void
2 mpmain(void)
3 {
4     sprintf("cpu%d: starting %d\n", cpuid(), cpuid());
5     idtinit();           // load idt register
6     xchg(&(mycpu()->started), 1); // tell startothers() we're up
7     scheduler();         // start running processes
8 }

```

هسته اول : boot می کند. دیدیم که هسته اول با استفاده از کد `entry.S` وارد تابع `main` در فایل `main.c` شد. در واقع همه توابع آماده سازی سیستم که در این تابع `call` شده اند توسط هسته اول اجرا میشوند. هسته های دیگر نیز وارد تابع `mpenter` میشوند.

تابع mpenter: در این تابع 4 تابع برای آماده سازی call میشوند. پس نتیجه میگیریم که این 4 تابع بین تمامی هسته ها مشترک خواهد بود.

بخشهایی که اختصاصا و تنها در هسته اول اجرا میشوند:

kinit1 - kvmalloc)setupkvm) - mpinit - picinit- ioapicinit-consoleinit-uartinit-pinit- tvinit-binit-fileinit-ideinit-startothers-kinit2- userinit

بخشهایی از آماده سازی سیستم که در تمام هسته ها مشترک هستند:

switchkvm - seginit -lapicinit- mppmain

۱: برای مشاهده **Breakpoint** ها از چه دستوری استفاده میشود؟

از دستور info breakpoints یا maint info breakpoints استفاده میشود.

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

هم پس از استفاده از info و دیدن شماره breakpoint با استفاده از del breakpointnum امکانپذیر است و هم از طریق دستور clear filename:linenum

۳: دستور **bt** را اجرا کنید. خروجی آن چه چیزی را نشان میدهد؟

این دستور که در واقع مخفف backtrace است، توابع فراخوانده شده و لود شده در استک را نمایش میدهد. ترتیب نمایش هم از سر به ته استک است (یعنی اول توابع عمیق تر نمایش داده میشوند)

۴: دو تفاوت دستور های **x** و **print** را توضیح دهید. چگونه میتوان محتوای یک ثبات خاص را چاپ کرد؟

دستور print یه عنوان ورودی یک expression 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 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 میشود؟ متغیر های محلی چطور؟ نتیجه این دو دستور را در گزارش خود بیاورید. در معماری **x86** رجیستر های edi و esi نشانگر چه چیزی هستند؟

برای نمایش وضعیت ثبات ها میتوان از info register استفاده کرد. استفاده کرد. علاوه بر آن از مخفف این دستور یعنی **locals** نیز میتوان استفاده کرد. برای نمایش متغیر های محلی از طریق locals اقدام میشود.

در ابتدای اسمی این ثباتها به معنی Extended بوده و در حالت 32 بیت به کار میرود.
Extended Destination Index است [ED] و برای اشاره به یک مقصد در عملیات stream به کار میرود.
DI به عنوان نشانگر داده و مقصد برخی عملیات مربوط به string استفاده می‌شود.

ESI: مخفف Extended source index است و برای اشاره به مبدا در عملیات استریم بکار میرود(SI) به عنوان نشانگر داده و به عنوان مبدا در برخی عملیات مربوط به رشته ها استفاده می‌شود.)

```
(gdb) info registers
eax    0x00000000      0
ecx    0xb0      176
edx    0x1      1
ebx    0x200      512
esp    0x0003beef70  0x0003beef70
ebp    0x0003beef78  0x0003beef78
esi    0x001113a70   2146354576
edi    0x001113a74   -2146354572
eip    0x000103e25  0x000103e25
erflags 0x0      [ IOPL=0 ]
cs     0x0      0
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    0x0000010011  [ PG CD NW WP ET PE ]
cr2    0x0      0
cr3    0x3ff000  [ PDBR=0 PCID=0 ]
cr4    0x10      [ PSE ]
cr5    0x0      0
cr6    0x0      0
cr7er  0x0      [ ]
xmm0   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm1   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm2   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm3   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm4   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm5   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm6   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm7   {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 12 times>, 0x00, 0x1f, 0x00, 0x0}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0, 0x1f800000000000000000000000000000}, uint128 = 0x1f800000000000000000000000000000}
r8b   0x0      0
v4_int32 0x0      0

```

```
(gdb) info locals
No symbol table info available.
(gdb) info variables
All defined variables:

File umalloc.c:
21: static Header base;
22: static Header *freep;

Non-debugging symbols:
0x00000a84  digits
0x00000da4  __bss_start
0x00000da4  _edata
0x00000db0  _end
```

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

- توضیح کلی این Struct و متغیر های درونی آن
- نحوه و زمان تغییر مقدار متغیر های درونی

این استراکت دارای یک آرایه ۲۸ اتایی از کاراکتر (باfr) و ۳ متغیر است: w, r, e

هر ۳ این متغیرها نشانده‌اندیس در بافر هستند. `W` اندیسی است که تا آنجا دستورات اجرا شده و توسط سیستم عامل مدیریت شده‌اند. `W` اندیسی است که تا آنجا در بافر ذخیره شده است (یعنی مثلاً در هنگام وارد کردن دستور جدید، اندیس اول خط نمایش داده می‌شود) و پس از وارد کردن کامند جدید و فشردن اینتر، `W` آپدیت می‌شوند و مقدار `e` هم اندیسی که در حال تایپ در آن هستیم را نمایش میدهد (یعنی مثلاً با وارد کردن یک کاراکتر جدید در کنسول، یکی زیاد می‌شود). در بافر هم که کامند وارد شده ذخیره می‌شود.

۷: خروجی دستورهای `layout asm` و `layout src` در TUI چیست؟

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

از این نما می‌توانید بریک پوینت‌ها را تعیین کنید، مقادیر متغیرها را بررسی کنید و اجرای برنامه را کنترل کنید.

نمای منبع طرح به ویژه برای درک منطق سطح بالای برنامه شما در حین اشکال زدایی مفید است.

در نمای "layout asm"، کد اسمبلی جدا شده برنامه خود را مشاهده می‌کنید.

این نما دستورالعمل‌های واقعی ماشین را نشان می‌دهد که با کد شما مطابقت دارد.

این به شما امکان می‌دهد نحوه ترجمه کد خود را به دستورالعمل‌های اسمبلی سطح پایین بررسی کنید.

در این نما می‌توانید بریک پوینت‌ها را تعیین کنید، رجیسترها را بررسی کنید و کد را جدا کنید.

نمای `layout asm` 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 می‌شوند.

۸: برای جابجایی میان توابع زنجیره فرآخوانی جاری (نقطه توقف) از چه دستوراتی استفاده می‌شود؟

از دو دستور `up` و `down` می‌توان استفاده کرد. هر دوی این دستورها می‌توانند یک عدد به عنوان ورودی بگیرند (که به صورت پیشفرض ۱ است) که می‌توان با استفاده از آن چندین لایه حرکت کرد.

*بخش اختیاری (نمره اضافه)

در این بخش ابتدا با استفاده از دستور `uname -a` ورژن لینوکس نصب شده خود را یافته و سپس از لینک داخل دستور آزمایش، هسته لینوکس نزدیکترین به هسته خودمان را دانلود می‌کنیم.

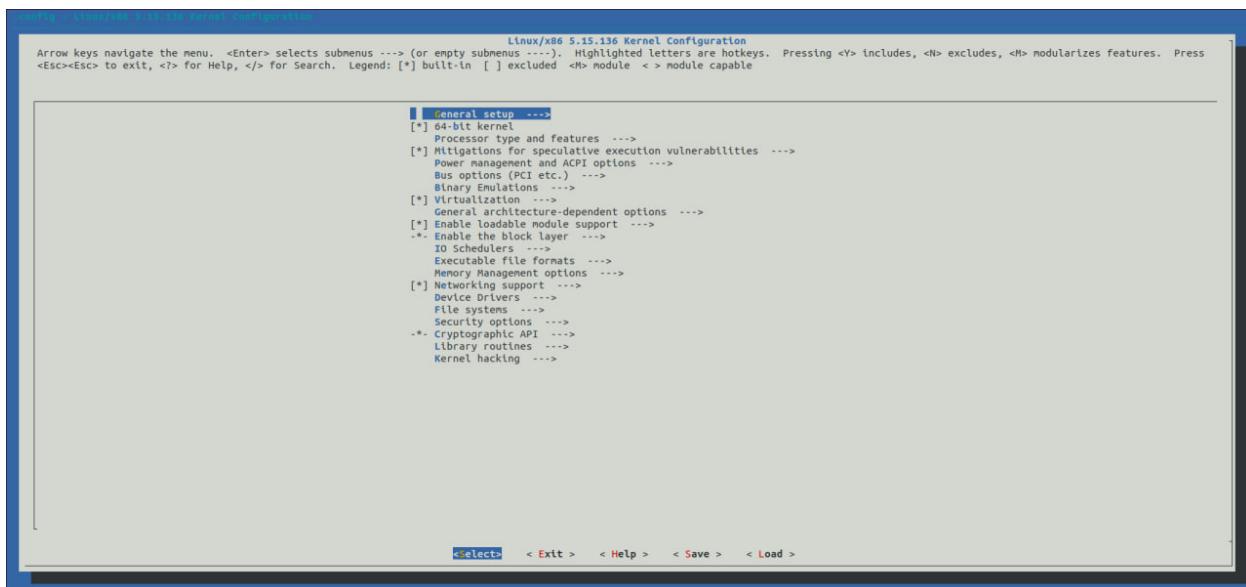
```
mohammad@mohammad-VirtualBox:~$ uname -a
Linux mohammad-VirtualBox 5.15.0-86-generic #96~20.04.1-Ubuntu SMP Thu Sep 21 13:23:37 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
mohammad@mohammad-VirtualBox:~$
```

در مرحله بعدی و پس از `extract` کردن فایل‌ها موجود در یک پوشه در یک `FileSystem` که حجم کافی داشته باشد، می‌توانیم با یکی از ۳ روش گفته شده در صورت پروژه این هسته را کانفیگ کنیم. در ذیل هر ۳ روش انجام شده است و در نهایت از روش اول استفاده شده است.

روش اول) استفاده از کانفیگ پیش فرض:

```
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ ls
arch certs Documentation fs init ipc Kconfig lib MAINTAINERS mm README scripts sound usr
block crypto drivers include io_uring Kbuild kernel LICENSES Makefile net samples security tools virt
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ make defconfig
HOSTCC scripts/basic/fixedep
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/confdato.o
HOSTCC scripts/kconfig/expr.o
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
#
```

روش دوم) استفاده از :menuconfig



روش سوم) استفاده از کانفیگ موجود در سیستم عامل فعلی

```
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ cp -v /boot/config-$(uname -r) .config
'/boot/config-5.15.0-86-generic' -> '.config'
```

سپس در صورت وجود مشکل در کامپایل تغییرات زیر را در کانفیگ ایجاد میکنیم:

```
# configuration written to .config
#
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ scripts/config --disable SYSTEM_TRUSTED_KEYS
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ scripts/config --disable SYSTEM_REVOCATION_KEYS
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ scripts/config --set-str CONFIG_SYSTEM_TRUSTED_KEYS ""
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ scripts/config --set-str CONFIG_SYSTEM_REVOCATION_KEYS ""
```

در نهایت با استفاده از روش اول کانفیگ ایجاد شده و سپس با یکی از دو دستور زیر و با صرف زمان زیادی هسته با این کانفیگ کامپایل می شود.

(روش اول)

```
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ make -j8
  SYNC  include/config/auto.conf.cmd
.config:8870:warning: symbol value 'm' invalid for ASHMEM
.config:9951:warning: symbol value 'm' invalid for ANDROID_BINDER_IPC
.config:9952:warning: symbol value 'm' invalid for ANDROID_BINDERFS
*
* Restart config...
*
*
* Android
*
Enable the Anonymous Shared Memory Subsystem (ASHMEM) [N/y/?] (NEW) y
*
* X86 Platform Specific Device Drivers
*
X86 Platform Specific Device Drivers (X86_PLATFORM_DEVICES) [Y/?] y
  WMI (ACPI_WMI) [M/y/?] m
    WMI embedded Binary MOF driver (WMI_BMOF) [M/n/?] m
      Huawei WMI laptop extras driver (HUAWEI_WMI) [M/n/?] m
    Sysfs structure for UV systems (UV_SYSFS) [M/n/y/?] m
    WMI support for MXM Laptop Graphics (MXM_WMI) [M/?] m
    PEAQ 2-in-1 WMI hotkey driver (PEAQ_WMI) [M/n/?] m
    Xiaomi WMI key driver (XIAOMI_WMI) [M/n/?] m
    Gigabyte WMI temperature driver (GIGABYTE_WMI) [M/n/?] m
    Acer Aspire One temperature and fan driver (ACERHDF) [M/n/y/?] m
    Acer Wireless Radio Control Driver (ACER_WIRELESS) [M/n/y/?] m
    Acer WMI Laptop Extras (ACER_WMI) [M/n/?] m
    AMD SoC PMC driver (AMD_PMC) [M/n/y/?] m
    Advantech ACPI Software Button Driver (ADV_SWBUTTON) [M/n/y/?] m
    Apple Gmux Driver (APPLE_GMUX) [M/n/?] m
    Asus Laptop Extras (ASUS_LAPTOP) [M/n/?] m
    Asus Wireless Radio Control Driver (ASUS_WIRELESS) [M/n/y/?] m
    ASUS WMI Driver (ASUS_WMI) [M/n/?] m
      Asus Notebook WMI Driver (ASUS_NB_WMI) [M/n/?] m
    Cisco Meraki MX100 Platform Driver (MERAKI_MX100) [M/n/?] m
    Eee PC Hotkey Driver (EEEPC_LAPTOP) [M/n/?] m
    Eee PC WMI Driver (EEEPC_WMI) [M/n/?] m
    Fujitsu-Siemens Amilo rfkill support (AMILO_RFKILL) [M/n/y/?] m
    Fujitsu Laptop Extras (FUJITSU_LAPTOP) [M/n/?] m
    Fujitsu Tablet Extras (FUJITSU_TABLET) [M/n/y/?] m
    GPD Pocket Fan Controller support (GPD_POCKET_FAN) [M/n/y/?] m
    Wireless hotkey button (WIRELESS_HOTKEY) [M/n/y/?] m
    Device driver to enable PRTL support (IBM_RTL) [M/n/y/?] m
    Lenovo IdeaPad Laptop Extras (IDEAPAD_LAPTOP) [M/n/?] m
    Thinkpad Hard Drive Active Protection System (hdaps) (SENSORS_HDAPS) [M/n/y/?] m
    Thinkpad ACPT Laptop Extras (THINKPAD_ACPT) [M/n/?] m
```

(روش دوم)

```
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ make -j8
SYNC      include/config/auto.conf.cmd
.config:8870:warning: symbol value 'm' invalid for ASHMEM
.config:9951:warning: symbol value 'm' invalid for ANDROID_BINDER_IPC
.config:9952:warning: symbol value 'm' invalid for ANDROID_BINDERFS
*
* Restart config...
*
*
* Android
*
Enable the Anonymous Shared Memory Subsystem (ASHMEM) [N/y/?] (NEW) y
*
* X86 Platform Specific Device Drivers
*
X86 Platform Specific Device Drivers (X86_PLATFORM_DEVICES) [Y/?] y
  WMI (ACPI_WMI) [M/y/?] m
    WMI embedded Binary MOF driver (WMI_BMOF) [M/n/?] m
    Huawei WMI laptop extras driver (HUAWEI_WMI) [M/n/?] m
  Sysfs structure for UV systems (UV_SYSFS) [M/n/y/?] m
  WMI support for MXM Laptop Graphics (MXM_WMI) [M/?] m
  PEAQ 2-in-1 WMI hotkey driver (PEAQ_WMI) [M/n/?] m
  Xiaomi WMI key driver (XIAOMI_WMI) [M/n/?] m
  Gigabyte WMI temperature driver (GIGABYTE_WMI) [M/n/?] m
  Acer Aspire One temperature and fan driver (ACERHDF) [M/n/y/?] m
  Acer Wireless Radio Control Driver (ACER_WIRELESS) [M/n/y/?] m
  Acer WMI Laptop Extras (ACER_WMI) [M/n/?] m
  AMD Soc PMC driver (AMD_PMC) [M/n/y/?] m
  Advantech ACPI Software Button Driver (ADV_SWBUTTON) [M/n/y/?] m
  Apple Gmux Driver (APPLE_GMUX) [M/n/?] m
  Asus Laptop Extras (ASUS_LAPTOP) [M/n/?] m
  Asus Wireless Radio Control Driver (ASUS_WIRELESS) [M/n/y/?] m
  ASUS WMI Driver (ASUS_WMI) [M/n/?] m
    Asus Notebook WMI Driver (ASUS_NB_WMI) [M/n/?] m
  Cisco Meraki MX100 Platform Driver (MERAKI_MX100) [M/n/?] m
  Eee PC Hotkey Driver (EEEPC_LAPTOP) [M/n/?] m
  Eee PC WMI Driver (EEEPC_WMI) [M/n/?] m
  Fujitsu-Siemens Amilo rfkill support (AMILO_RFKILL) [M/n/y/?] m
  Fujitsu Laptop Extras (FUJITSU_LAPTOP) [M/n/?] m
  Fujitsu Tablet Extras (FUJITSU_TABLET) [M/n/y/?] m
  GPD Pocket Fan Controller support (GPD_POCKET_FAN) [M/n/y/?] m
  Wireless hotkey button (WIRELESS_HOTKEY) [M/n/y/?] m
  Device driver to enable PRTL support (IBM_RTL) [M/n/y/?] m
  Lenovo IdeaPad Laptop Extras (IDEAPAD_LAPTOP) [M/n/?] m
  Thinkpad Hard Drive Active Protection System (hdaps) (SENSORS_HDAPS) [M/n/y/?] m
  Thinkpad ACPI Laptop Extras (THINKPAD_ACPI_LAPTOP) [M/n/?] m
```

تعداد **thread** ها در این جا 8 در نظر گرفته شده است. تا با سرعت بهتری فرآیند انجام شود.

همچنین به جهت افزایش سرعت می توان بخش هایی از کانفیگ که از قبل موجود هست یا به طور پیش فرض در هسته وجود دارد را از کامپایل کردن حذف کرد که به نحو زیر قابل انجام است:

```

bash: cd: ./config: No such file or directory
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ make localmodconfig
using config: '.config'
System keyring enabled but keys "debian/canonical-certs.pem" not found. Resetting keys to default value.
*
* Restart config...
*
*
* PCI GPIO expanders
*
AMD 8111 GPIO driver (GPIO_AMD8111) [N/m/y/?] n
BT8XX GPIO abuser (GPIO_BT8XX) [N/m/y/?] (NEW) y
OKI SEMICONDUCTOR ML7213 IOH GPIO support (GPIO_ML_IOH) [N/m/y/?] n
ACCES PCI-IDIO-16 GPIO support (GPIO_PCI_IDIO_16) [N/m/y/?] n
ACCES PCIe-IDIO-24 GPIO support (GPIO_PCIE_IDIO_24) [N/m/y/?] n
RDC R-321x GPIO support (GPIO_RDC321X) [N/m/y/?] n
*
* PCI sound devices
*
PCI sound devices (SND_PCI) [Y/n/?] y
  Analog Devices AD1889 (SND_AD1889) [N/m/?] n
  Avance Logic ALS300/ALS300+ (SND_ALS300) [N/m/?] n
  Avance Logic ALS4000 (SND_ALS4000) [N/m/?] n
  ALi M5451 PCI Audio Controller (SND_ALI5451) [N/m/?] n
  AudioScience ASIxxxx (SND_ASIPCI) [N/m/?] n
  ATI IXP AC97 Controller (SND_ATIXP) [N/m/?] n
  ATI IXP Modem (SND_ATIXP_MODEM) [N/m/?] n
  Aureal Advantage (SND_AU8810) [N/m/?] n
  Aureal Vortex (SND_AU8820) [N/m/?] n
  Aureal Vortex 2 (SND_AU8830) [N/m/?] n
  Emagic Audiowerk 2 (SND_AW2) [N/m/?] n
  Aztech AZF3328 / PCI168 (SND_AZT3328) [N/m/?] n
  BT87x Audio Capture (SND_BT87X) [N/m/?] n
  SB Audigy LS / Live 24bit (SND_CA0106) [N/m/?] n
  C-Media 8338, 8738, 8768, 8770 (SND_CMIPCI) [N/m/?] n
  C-Media 8786, 8787, 8788 (Oxygen) (SND_OXYGEN) [N/m/?] n
  Cirrus Logic (Sound Fusion) CS4281 (SND_CS4281) [N/m/?] n
  Cirrus Logic (Sound Fusion) CS4280/CS461x/CS462x/CS463x (SND_CS46XX) [N/m/?] n
  Creative Sound Blaster X-Fi (SND_CTXFI) [N/m/?] n
  (Echoaudio) Darla20 (SND_DARLA20) [N/m/?] n
  (Echoaudio) Gina20 (SND_GINA20) [N/m/?] n
  (Echoaudio) Layla20 (SND_LAYLA20) [N/m/?] n
  (Echoaudio) Darla24 (SND_DARLA24) [N/m/?] n
  (Echoaudio) Gina24 (SND_GINA24) [N/m/?] n
  (Echoaudio) Layla24 (SND_LAYLA24) [N/m/?] n
  (Echoaudio) Mona (SND_MONA) [N/m/?] n
  (Echoaudio) Mia (SND_MIA) [N/m/?] n
  (Echoaudio) 3G cards (SND_ECHO3G) [N/m/?] n
  (Echoaudio) Tadpole (SND_TADPOLE) [N/m/?] n

```

This command looks at the loaded kernel modules of your system and modifies the .config file such that only these modules are included in the build. This should work perfectly fine as long as you don't plan to use the generated build on another machine than the one you compile the source on.

پس از انجام شدن کامپایل جهت بررسی اینکه کامپایل به درسی و کامل انجام شده است، دستور زیر را در ترمینال میزنیم:

\$echo \$?

در صورتی که این دستور مقدار 0 را برگرداند یعنی کامپایل بدون خطا انجام شده است. در غیر این صورت یعنی خطایی در کامپایل وجود دارد.

```

mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ echo $?
0

```

بعد از اینکه کامپایل بدون خطا انجام شد، ابتدا **module** ها و سپس هسته جدید را بر روی سیستم نصب میکنیم.

```
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ sudo make modules_install
[sudo] password for mohammad:
INSTALL /lib/modules/5.15.136/kernel/drivers/thermal/intel/x86_pkg_temp_thermal.ko
INSTALL /lib/modules/5.15.136/kernel/fs/efivarfs/efivarfs.ko
INSTALL /lib/modules/5.15.136/kernel/net/ipv4/netfilter/iptable_nat.ko
INSTALL /lib/modules/5.15.136/kernel/net/netfilter/nf_log_syslog.ko
INSTALL /lib/modules/5.15.136/kernel/net/netfilter/xt_LOG.ko
INSTALL /lib/modules/5.15.136/kernel/net/netfilter/xt_MASQUERADE.ko
INSTALL /lib/modules/5.15.136/kernel/net/netfilter/xt_addrtype.ko
INSTALL /lib/modules/5.15.136/kernel/net/netfilter/xt_mark.ko
INSTALL /lib/modules/5.15.136/kernel/net/netfilter/xt_nat.ko
DEPMOD /lib/modules/5.15.136
mohammad@mohammad-VirtualBox:~/OS/Linux/linux-5.15.136$ sudo make install
sh ./arch/x86/boot/install.sh 5.15.136 \
    arch/x86/boot/bzImage System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.15.136 /boot/vmlinuz-5.15.136
update-initramfs: Generating /boot/initrd.img-5.15.136
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.15.136 /boot/vmlinuz-5.15.136
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.15.136 /boot/vmlinuz-5.15.136
run-parts: executing /etc/kernel/postinst.d/vboxadd 5.15.136 /boot/vmlinuz-5.15.136
VirtualBox Guest Additions: Building the modules for kernel 5.15.136.

VirtualBox Guest Additions: Look at /var/log/vboxadd-setup.log to find out what
went wrong
run-parts: executing /etc/kernel/postinst.d/xx-update-initrd-links 5.15.136 /boot/vmlinuz-5.15.136
I: /boot/initrd.img.old is now a symlink to initrd.img-5.15.0-86-generic
I: /boot/initrd.img is now a symlink to initrd.img-5.15.136
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.15.136 /boot/vmlinuz-5.15.136
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.15.136
Found initrd image: /boot/initrd.img-5.15.136
Found linux image: /boot/vmlinuz-5.15.0-86-generic
Found initrd image: /boot/initrd.img-5.15.0-86-generic
Found linux image: /boot/vmlinuz-5.15.0-83-generic
Found initrd image: /boot/initrd.img-5.15.0-83-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

سپس بعد از اینکه هسته جدید ساخته شد یک فایل C مطابق تصویر زیر ایجاد میکنیم. که در این فایل طبق دستور آزمایش نام اعضای گروه باید نمایش داده شود. این فایل در هر دایرکتوری دلخواهی میتواند قرار داشته باشد.

```
home > mohammad > OS > P1 > optional > C names.c > ...
1 #include <linux/module.h>
2 #include <linux/kernel.h>
3
4 MODULE_LICENSE("GPL");
5
6 int init_module(void)
7 {
8     printk(KERN_INFO "Group 22:\n- Mohammad Amanlou : 810100084\n- Shahzad Momayez : 810100272\n- Pardis ZandKarimi : 810101081\n");
9     return 0;
10 }
11
12 void cleanup_module(void) {};
```

سپس یک **makefile** مطابق تصویر زیر برای آن ایجاد می کنیم. به طور کلی این **makefile** با استفاده از **makefile** اصلی هسته نوشته شده است و در آن از **object** های موجود در **makefile** استفاده شده است.

```
home > mohammad > OS > Proj1 > M Makefile
1   obj-m += names.o
2
3 all:
4   make -C /lib/modules/$$(uname -r)/build M=$(PWD) modules
```

سپس فایل names.c را make میکنیم. و می بینیم در کنار فایل های دیگر یک فایل به نام kernel object names.ko است ایجاد شده است. سپس دستور \$sudo insmod names.ko را اجرا می کنیم.

در نهایت با استفاده از دستور dmseg اسامی اعضای گروه نمایش داده می شود.

```
[26492.918016] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[26747.700559] Group 22:
    - Mohammad Amanlou : 810100084
    - Shahzad Momayez : 810100272
    - Pardis ZandKarimi : 810101081
mohammad@mohammad-VirtualBox:~/OS/Proj1$
```