- Mohammad Ali Ghahari 810100201

- Mohammad Sadeghi 810100175

- Nima Tajik 810100104

سوال 1) سیستم عامل xv6 رابط های اساسی را تامین می کند که توسط سیستم عامل Unix معرفی شده اند

به جهت داشتن رابطی انعطاف پذیر در معماری Unix که سازوکارهای آن به خوبی باهم ترکیب میشوند، بسیاری از سیستم عامل های معروف همچون Linux , Windows و Mac رابط هایشان مشابه این سیستم عامل است لذا یادگیری سیستم عامل xv6 یادگیری باقی سیستم های عامل را تسهیل میکند

سیستم عامل xv6 دارای فرمی مرسوم از هسته می باشد و یک پردازنده (program) دارد که خدمات لازم برای اجرای برنامه ها را تامین میکند.

برای دفاع ازین موضوع میتوان به هدرهای این سیستم عامل که حاوی دستورات معماری موجود در x86.h است اشاره کرد

سوال 2) هر برنامه در حال اجرا یک "روند" نامیده میشود که خود دارای حافظه ای برای دستورات ، داده ها و پشته است. دستورات در واقع روند محاسبه برنامه را پیاده سازی می کند. داده ها نیز متغیر هایی هستند که محاسبات روی آنها یا توسط آنها در طول روند برنامه اجرا میشود و پشته نیز رویه فراخوانی های انجام شده در برنامه را مدیریت می کند.

هنگام نیاز برنامه به استناد به هسته ، توسط رویه فراخوانی در رابط ، فراخوانی سیستم انجام میشود و به هسته ارسال میشود و هسته سرویس خواسته شده را بر می گرداند و اینگونه پروسه در حال اجرا بین هسته و رابط در تناوب است

هسته همچنین از سازوکار های حفاظتی CPU استفاده میکند تا مطمئن شود هر پروسه در حال اجرا حافظه اختصاصی و مجزای خودش را دارد. در واقع با استفاده از امتیازات مورد نیازدرسخت افزار است که هسته این سیستم حفاظتی را پیاده سازی می کند.

در هسته نیز ما مجموعه ای از فراخوانی ها را توسط برنامه های مختلف داریم که در استک ذخیره شده و به ترتیب هسته سرویس های خواسته شده را به برنامه مخصوص خود بر می گرداند و اینگونه پردازنده را به برنامه های مختلف اختصاص میدهد

Shell نیز چیز پیچیده ای نیست و یک برنامه است که دستورات کاربر را خوانده و اجرا می کند و نکته آن این است که در سطح هسته نیست

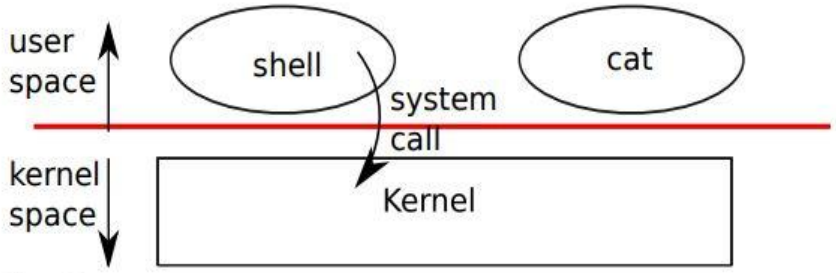


Figure 0-1. A kernel and two user processes.

سوال 4) تابع fork() می تواند درون یک برنامه صدا زده شود که کارش این است که یک برنامه دیگرعین برنامه ای که توسط آن صدا زده شده می سازد و همان مقدار حافظه که برنامه اصلی دارد را به این برنامه اختصاص می دهد که در اصطلاح به این برنامه ساخته شده "برنامه فرزند" می گوییم

اما در واقع این تابع یک عدد صحیح بر میگرداند. اگر در برنامه غیر فرزند باشد، شماره یا مشخصه فرزند ساخته شده توسط آن برنامه برگردانده میشود و اگر در برنامه فرزند باشیم عدد صفر برگردانده میشود

تابع exec() اما وظیفه دیگری دارد. این تابع در واقع حافظه اصلی برنامه در حال اجرا را با فایل دیگری که در سیستم واقع است جایگزین می کند

این فایل در سیستم عامل xv6 دارای فرمت ELF است و باید شامل هر سه بخش اصلی پروسه یعنی دستورات و داده ها و پشته باشد. بعد از اجرای این تابع برعکس تابع fork() به برنامه بر نمی گردیم بلکه از فایل جایگزین که از نقطه شروع (که در فایل اظهار شده) در حال اجراست مشغول به بارگذاری دستورات میشویم.

این تابع دو ورودی دارد اولی فایل جایگزین شده با برنامه و دومی ورودی که باید برنامه داخل فایل روی آن انجام شود

اما مزیت اینکه این دو تابع را مجزا داشته باشیم از منظر ورودی/خروجی قابل بیان است.

با جدا بودن این دو تابع میتوان با استفاده از fork() یک پردازه فرزند ایجاد کرد و با استفاده از برخی توابع فراخوانی سیستم مانند open() , close() تغییراتی در ورودی و خروجی بوجود آورد و سپس با دستور exec() آن را اجرا کرد بدون آنکه تغییری در برنامه در حال اجرا(که fork() در آن صدا زده شده است) ایجاد شود

اما اگر یکی باشند باید تغییرات ورودی یا خروجی بعنوان پارامتر پاس داده شوند یا اینکه shell قبل اجرای برنامه فایل دیسکریپتورها را تغییر داده و پس از اتمام آن آنها را به حالت اولیه برگرداند که در هر دو حالت هندل کردن این موضوع مشکلات فراوانی خواهد داشت

سوال8) با توجه به تصویر درUPROGS فرمان های محیط کاربر یا بعبارتی رشته هایی که می خواهیم دستور شوند را درین بخش اضافه میکنیم

متغیرULIB نیز طبق جستجوهای انجام شده ارجاع دهنده به کتابخانه سطح کاربرست که شامل پوشه های توابع فراخوانی سیستم و توابع کاربردی دیگری است. در واقع وظیفه آن ساختن و لینک کردن کتابخانه سطح کاربرست

```
167
168     UPROGS=\
169             _cat\
170             _echo\
171             _forktest\
172             _grep\
173             _init\
174             _kill\
175             _ln\
176             _ls\
177             _strdiff\
178             _mkdir\
179             _rm\
180             _sh\
181             _stressfs\
182             _usertests\
183             _wc\
184             _zombie\
185
```

سوال11) فایل مورد نظر برای بوت هسته سیستم عامل bootblack است که فرمت آن raw binary است که فایل از نوع تکست است

تفاوت این نوع فایل دودویی با بقیه آنها در فرمت است که ELF نیست

همچنین بدلیل نشناخته شدن elf توسط cpu این کد فقط شامل بخش .text است که حاوی دستورات اصلی برنامه است و باعث کم شدن حجم فایل بوت میشود

سورس کد اسمبلی شده این فایل دودویی در پوشه مربوط به گزارش آورده شده است(out_assembly.txt)

سوال12) علت استفاده ازین دستور این است که امکان کپی کردن فایل ها و تبدیل آن ها به فرمت های دیگر را داشته باشیم مثلا تبدیل یک فایل ELF به فایل باینری یا یک فایل Hex یا ...

سوال14)

عام منظوره ----> EAX : برای عملیات های ریاضی و IO استفاده میشود / EBX : بعنوان اشاره گر اصلی برای دسترسی به حافظه مورد استفاده است

قطعه ---> DS(data segment) : اشاره به بخشی از دادگان دارد که دادگان برنامه در حال اجرا در آن ذخیره شده است(دیگر ثبات ها:CS,ES[1],SS[2]،[3])

کنترلی ---> CR0 : مدیریت حافظه و عملیات های اصلی را کنترل میکند / CR2 : نگهدارنده آدرس خطی خطا(ها) ی صفحه

وضعیت ---> EFLAGS : وضعیت پروسه ها و همچنین نشانک هایی مربوط به خروجی عملیات های ریاضی و منطقی را در خود دارد بعلاوه جریان کنترل برنامه. مثال هایی ازین نشانک ها zero flag , sign flag , carry flag , interrupt flag, parity flag و غیره اشاره کرد.

سوال19) با ذخیره کردن آدرس فیزیکی صفحه در CR3 پردازنده قادر خواهد بود ترجمه آدرس های مجازی به فیزیکی را به شکل کارآمد تری انجام دهد

وقتی توسط یک آدرس مجازی دسترسی به حافظه درخواست شود، پردازنده با استفاده از اطلاعات رجیستر CR3 و مراجعه به lookup table آدرس درست را correspond میکند. در واقع علت ذخیره کردن فیزیکی آدرس

```
145
146     ULIB = ulib.o usys.o printf.o umalloc.o
```

[1] Extra segment
[2] Stack segment
[3] Code segment

بجهت تسهیل ترجمه بین آدرس های مجازی به فیزیکی است و در صورت مجازی بودن این آدرس مجدد نیاز به یک جدول دیگر برای نگاشت خواهیم داشت که این یک حلقه بی نهایت بوجود خواهد آورد لذا آدرس page table باید فیزیکی باشد

سوال22) قطعه بندی هسته تنها ازلایه فضای حافظه هسته محافظت می کند و تضمینی برای حفاظت از لایه سطح کاربر نمی دهد.

با نگهداری seg-user سیستم عامل میتواند اطمینان حاصل کند که برنامه سطح کاربر فقط به قطعه حافظه خودش دسترسی دارد و کاملا از سطح هسته مجزاست و نمیتواند مستقیما تغییری در حافظه هسته بوجود آورد و بالعکس و این بدین معنا خواهد بود که هر دو بخش از منظر دسترسی به حافظه در حفاظت کامل هستند.

سوال23)اجزای ساختار پروسس:

1. **Unit sz** : مقدار حافظه مورد نیاز برنامه را به بایت نگهداری میکند
2. **Pde_t* pgdir** : اشاره گری است که به ابتدای آرایه **PDE**[4] اشاره میکند
3. **Char *kstack** : اشاره گر به ابتدای پشته مورد استفاده در هسته
4. **Enum procstate state** : مراحل یا حالات مختلف برنامه را ذخیره میکند
5. **Int pid** : مشخصه منحصر بفرد برنامه را ذخیره میکند
6. **Struct proc *parent** : به برنامه ی والد اشاره می کند
7. **Struct trapframe *tf** : اشاره گر به جایی است که خطا ها و interrupt ها را ذخیره میکنیم
8. **Struct context *context** : هنگام سوییچ برنامه ها مقادیررجیسترهای عام منظوره پروسه اجرا شده را در خود ذخیره می کند
9. **Void *chan** : برنامه ممکن است بدلایل مختلفی مثل انتظار برای دریافت ورودی لازم باشد متوقف شود و این اشاره گر مشخص میکند که برنامه باید توقف کند یا خیر.
10. **Int killed** : مشابه اشاره گر قبلی عمل میکند ولی بجای توقف برنامه آن را همانجایی که هست پایان میدهد
11. **Struct file *ofile[NOFILE]** : اشاره گر به جایی که فایل و پوشه های بازشده در برنامه را ذخیره میکند
12. **Struct inode *cwd** : نگهدارنده دایرکتوری از برنامه است که برنامه در آن دارد انجام میشود(current working directory)
13. **Char name[16]** : آرایه ای که نام برنامه در حال اجرا را در خود ذخیره دارد

[4] Page directory entry

سوال 18) معادل برای ورود هسته است در سیستم در سورس کد با اسم می باشد که پوشه مربوط به شده است

کد entry.S که xv6 به حافظه عامل لینوکس و معماری آن فایلی entry_32.S سورس آن در گزارش آورده

```
38  struct proc {
39    uint sz;                     // Size of process memory (bytes)
40    pde_t* pgdir;                // Page table
41    char *kstack;                // Bottom of kernel stack for this process
42    enum procstate state;        // Process state
43    int pid;                     // Process ID
44    struct proc *parent;         // Parent process
45    struct trapframe *tf;        // Trap frame for current syscall
46    struct context *context;     // swtch() here to run process
47    void *chan;                  // If non-zero, sleeping on chan
48    int killed;                  // If non-zero, have been killed
49    struct file *ofile[NOFILE];  // Open files
50    struct inode *cwd;           // Current directory
51    char name[16];               // Process name (debugging)
52  };
```

سوال27) هسته entry.s و باقی entryother.s میشوند که درین اول از طریق هسته ها از طریق وارد تابع اصلی تابع 4 تابع برای آماده سازی سیستم فراخوانده میشود و لذا این 4 تابع بین تمامی هسته ها مشترک هستند(switchkvm,seginit,lapicinit,mpmain)

برخی توابع نیز بصورت اختصاصی در هسته اول اجرا میشوند مثل ... , kinit1 , tvinit,kinit2,fileinit

هر پردازنده زمان بند مربوط به خودش را دارد و در نتیجه این تابع نیز بین هسته ها مشترک خواهد بود

در ضمن همه پردازنده ها باید آدرس فیزیکی page table که توسط هسته اول تولید میشود را داشته باشند برای همین تابع switchkvm مشترک است

و در آخر همه پردازنده ها باید کار خود را شروع کنند و آماده اجرای برنامه ها شوند که این مورد توسط mpmain که مشترک است انجام می گیرد

برنامه سطح کاربر:

```
init: starting sh
Group #8 :
1. Mohammad Ali Ghahari
2. Nima Tajik
3. Mohammad Sadeghi
$ echo ali
ali
$ echo ali
ali
$ strdiff ali mmd
$ cat strdiff_result.txt
110
$ strdiff ali mmd_
```

پاسخ سوالات اشکال زدایی

سوال 1) برای مشاهده breakpoint ها از دستور info breakpoints استفاده می شود.

```
(gdb) target remote tcp::25000
Remote debugging using tcp::25000
0x0000fff0 in ?? ()
(gdb) b console.c:310
Breakpoint 1 at 0x80100e10: file console.c, line 310.
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x80100e10 in consoleintr at console.c:310
(gdb)
```

سوال 2) برای حذف کردن breakpoint ها می توان از دستور clear ومحل آنها استفاده کرد. اگر ورودی به آن داده نشود تمام breakpoint ها حذف میشود اما میتوان مانند ست کردن breakpoint ها با همان فرمت breakpoint هارا با دستور clear حذف کرد. همچنین می توان با استفاده از دستور delete با استفاده از شماره breakpoint ها آن هارا حذف کرد.

```
(gdb) delete 1
(gdb) info b
No breakpoints or watchpoints.
(gdb) b console.c:310
Breakpoint 2 at 0x80100e10: file console.c, line 310.
(gdb) clear console.c:310
Deleted breakpoint 2
(gdb)
```

سوال 3) دستور bt به ما stack trace را نشان می دهد. یعنی در تابعی که قرار داریم چهfunction call هایی انجام شده تا به این تابع رسیده ایم.

```
(gdb) b console.c:288
Breakpoint 4 at 0x80100cb0: console.c:288. (2 locations)
(gdb) c
Continuing.

Thread 1 hit Breakpoint 4, SubmitCommand (c=10) at console.c:288
288         for (int i = MAXCOMMANDS - 1; i > 0; i--)
(gdb) bt
#0  SubmitCommand (c=10) at console.c:288
#1  0x80100f44 in consoleintr (getc=0x80102e80 <kbdgetc>) at console.c:377
#2  0x80102f70 in kbdintr () at kbd.c:49
#3  0x80106285 in trap (tf=0x80117058 <stack+3912>) at trap.c:67
#4  0x80105fdf in alltraps () at trapasm.S:20
#5  0x80117058 in stack ()
#6  0x801133e4 in cpus ()
#7  0x801133e0 in ?? ()
#8  0x801037cf in mpmain () at main.c:57
#9  0x8010391c in main () at main.c:37
(gdb)
```

سوال 4) با استفاده از دستور x می توانیم محتوای جایی که پوینتر به آن اشاره می کند را ببینیم مثلا با دستور x/2c POINTER محتوای دو خانه ابتدایی آن را ببینیم اما با دستور print میتوانیم مقادیر متغیر ها یا رجیستر هارا ببینیم.

```
(gdb) x input.buf
0x8010fe80 <input>:     0x00000065
(gdb) x/c input.buf
0x8010fe80 <input>:     101 'e'
(gdb) print input.e
$2 = 1
(gdb)
```

سوال 5) با استفاده از دستور info registers می توان محتوای هر رجیستر را دید و با دستور info all-registers میتوان رجیستر های floating-point و vector registers را هم دید.همچنین با دستور info locals میتوان لیست متغیر های محلی را مشاهده کرد. ثبات SI مخفف Source Index بوده و برای اشاره به یک مبدا در عملیات stream به کار می‌رود. DI نیز مخفف Destination Index بوده و برای اشاره به یک مقصد در عملیات stream به کار می‌رود. E در ابتدای اسامی این ثبات‌ها به معنی Extended بوده و در حالت 32 بیت به کار می‌رود. SI به عنوان نشانگر داده و به عنوان مبدا در برخی عملیات مربوط به رشته‌ها استفاده می‌شود. DI نیز به عنوان نشانگر داده و مقصد برخی عملیات مربوط به رشته‌ها استفاده می‌شود.

```
(gdb) info registers
eax            0xffffffff       -1
ecx            0x0              0
edx            0x64             100
ebx            0xffffffff       -1
esp            0x80116fd4       0x80116fd4 <stack+3780>
ebp            0x80116ffc       0x80116ffc <stack+3820>
esi            0x80102eb0       -2146423120
edi            0x0              0
eip            0x80100ea0       0x80100ea0 <consoleintr+144>
eflags         0x86             [ IOPL=0 SF PF ]
cs             0x8              8
ss             0x10             16
ds             0x10             16
es             0x10             16
fs             0x0              0
gs             0x0              0
fs_base        0x0              0
gs_base        0x0              0
k_gs_base      0x0              0
cr0            0x80010011       [ PG WP ET PE ]
cr2            0x0              0
cr3            0x3ff000         [ PDBR=1023 PCID=0 ]
cr4            0x10             [ PSE ]
cr8            0x0              0
efer           0x0              [ ]
xmm0           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0
x0, 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, 0
x0, 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, 0
x0, 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, 0
x0, 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, 0
x0, 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, 0
x0, 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, 0
x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm7           {v4_float = {0x0, 0x0, 0x0, 0x1f80}, v2_double = {0x0, 0x1f8000000000}, v16_int8 = {0x0 <repeats 12 times>, 0x80, 0x1f, 0x0, 0x0}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0
, 0x0, 0x1f80, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x1f80}, v2_int64 = {0x0, 0x1f8000000000}, uint128 = 0x1f80000000000000000000000000}
mxcsr          0x1f80           [ IM DM ZM OM UM PM ]
```

```
(gdb) info locals
c = -1
doprocdump = 0
(gdb)
```

سوال 6) این struct در فایل console.c و در خط 182 قرار دارد. این struct برای خواندن نوشتن و ادیت کردن ورودی کاربر است.

```
#define INPUT_BUF 128

struct}
  char buf[INPUT_BUF];
  uint r;  // Read index
  uint w;  // Write index
  uint e;  // Edit index
 {input;
```

آرایه BUF مربوط به بافر است که ورودی کاربر(کاراکتر های ورودی) در آن ذخیره می شود و برای ران شدن لازم است در انتهای ورودی، کاراکتر newline(\n) اضافه شود تا توسط تابعwakeup اجرا شود.. R که یک متغیر int بدون علامت است نشان دهنده ایندکسی است که قرار است از آرایه بافر خوانده شود که به

ابتدای command وارد شده توسط کاربر اشاره میکند و تا رسیدن به w از بافر میخواند . W نشان دهنده ایندکسی است که در حین تایپ کردن و ادیت کردن کاربر همراه با r به ابتدای ورودی اشاره میکند و پس از submit کردن به e می رود. E نشان دهنده ایندکسی در بافر است که قرار است ادیت شود یا تغییر داده شود که پس از سابمیت کردن کاربر e را به انتهای ورودی کاربر میبریم و مقدار w را برابر e قرار می دهیم

سوال 7) دو دستور layout src, layout asm برای تغییر نمایش کد استفاده می شود به این صورت که دستور layout src کد برنامه را نمایش می دهد و layour asm کد زبان اسمبلی آن را نمایش میدهد و از آنجا امکان دیباگ کردن فراهم می شود.

```
0x80101136 <consoleintr+806>    cmp    %eax,-0x1c(%ebp)
0x80101139 <consoleintr+809>    jne    0x80100f3d <consoleintr+301>
0x8010113f <consoleintr+815>    jmp    0x80100f38 <consoleintr+296>
0x80101144 <consoleintr+820>    lea    0x0(%esi,%eiz,1),%esi
0x80101148 <consoleintr+824>    cli
0x80101149 <consoleintr+825>    jmp    0x80101149 <consoleintr+825>
0x8010114b <consoleintr+827>    lea    0x0(%esi,%eiz,1),%esi
0x8010114f <consoleintr+831>    nop
0x80101150 <consoleintr+832>    mov    0x80110b54,%eax
0x80101155 <consoleintr+837>    add    $0x1,%eax
0x80101158 <consoleintr+840>    mov    %eax,0x80110b54
0x8010115d <consoleintr+845>    add    0x8010fed4,%eax
0x80101163 <consoleintr+851>    mov    %eax,%ecx
0x80101165 <consoleintr+853>    cmp    -0x1c(%ebp),%eax
0x80101168 <consoleintr+856>    jbe    0x801011c3 <consoleintr+947>
0x8010116a <consoleintr+858>    mov    %edi,-0x20(%ebp)
0x8010116d <consoleintr+861>    mov    %ebx,-0x24(%ebp)
0x80101170 <consoleintr+864>    mov    %ecx,%ebx
0x80101172 <consoleintr+866>    mov    $0xd20d20d3,%eax
0x80101177 <consoleintr+871>    sub    $0x1,%ecx
0x8010117a <consoleintr+874>    imul   %ecx
0x8010117c <consoleintr+876>    mov    %ecx,%edi
0x8010117e <consoleintr+878>    sar    $0x1f,%edi
0x80101181 <consoleintr+881>    lea    (%edx,%ecx,1),%eax
0x80101184 <consoleintr+884>    mov    %ecx,%edx
0x80101186 <consoleintr+886>    sar    $0x6,%eax
```

```
378             SubmitCommand(isEnd ? '\n' : c);
379           else
380           {
381             lineLength++;
382             for (int i = input.w + lineLength; i > input.e; i--)
383               input.buf[i % INPUT_BUF] = input.buf[(i - 1) % INPUT_BUF];
384             input.buf[input.e++ % INPUT_BUF] = c;
385           }
386         }
387         break;
388       }
389     }
B+>  390   release(&cons.lock);
391   if (doprocdump)
392   {
393     procdump(); // now call procdump() wo. cons.lock held
394   }
395 }
396
397 int consoleread(struct inode *ip, char *dst, int n)
398 {
399   uint target;
400   int c;
401
402   iunlock(ip);
403   target = n;
```

سوال8) برای جابجایی بین زنجیره توابع فراخوانی کافی است از up , down استفاده کنیم. میتوان با استفاده از bt زنجیره توابع فراخوانی را لیست کرد و با این دو بین آنها جابجا شد.(چیزی را اجرا نمیکند)

console.c

```
      378           SubmitCommand(isEnd ? '\n' : c);
      379         else
      380         {
      381           lineLength++;
      382           for (int i = input.w + lineLength; i > input.e; i--)
      383             input.buf[i % INPUT_BUF] = input.buf[(i - 1) % INPUT_BUF];
      384           input.buf[input.e++ % INPUT_BUF] = c;
      385         }
      386       }
      387       break;
      388     }
      389   }
B+>   390   release(&cons.lock);
      391   if (doprocdump)
      392   {
      393     procdump(); // now call procdump() wo. cons.lock held
      394   }
      395 }
      396
      397 int consoleread(struct inode *ip, char *dst, int n)
      398 {
      399   uint target;
      400   int c;
      401
      402   iunlock(ip);
      403   target = n;
```

```
remote Thread 1.1 In: consoleintr                                          L390   PC: 0x80100ea0
#0  consoleintr (getc=0x80102eb0 <kbdgetc>) at console.c:390
#1  0x80102fa0 in kbdintr () at kbd.c:49
#2  0x801062b5 in trap (tf=0x80117058 <stack+3912>) at trap.c:67
#3  0x8010600f in alltraps () at trapasm.S:20
#4  0x80117058 in stack ()
#5  0x801133e4 in cpus ()
#6  0x801133e0 in ?? ()
#7  0x801037ff in mpmain () at main.c:57
#8  0x8010394c in main () at main.c:37
(gdb) up
#1  0x80102fa0 in kbdintr () at kbd.c:49
(gdb) down
#0  consoleintr (getc=0x80102eb0 <kbdgetc>) at console.c:390
(gdb)
```

kbd.c

```
       37   if(shift & CAPSLOCK){
       38     if('a' <= c && c <= 'z')
       39       c += 'A' - 'a';
       40     else if('A' <= c && c <= 'Z')
       41       c += 'a' - 'A';
       42   }
       43   return c;
       44 }
       45
       46 void
       47 kbdintr(void)
       48 {
  >    49   consoleintr(kbdgetc);
       50 }
       51
       52
       53
       54
       55
       56
       57
       58
       59
       60
       61
       62
```

```
remote Thread 1.1 In: kbdintr                                              L49    PC: 0x80102fa0
(gdb) layout src
(gdb) bt
#0  consoleintr (getc=0x80102eb0 <kbdgetc>) at console.c:390
#1  0x80102fa0 in kbdintr () at kbd.c:49
#2  0x801062b5 in trap (tf=0x80117058 <stack+3912>) at trap.c:67
#3  0x8010600f in alltraps () at trapasm.S:20
#4  0x80117058 in stack ()
#5  0x801133e4 in cpus ()
#6  0x801133e0 in ?? ()
#7  0x801037ff in mpmain () at main.c:57
#8  0x8010394c in main () at main.c:37
(gdb) up
#1  0x80102fa0 in kbdintr () at kbd.c:49
(gdb)
```

√

آخرین نسخه هسته لینوکس 6.5.7 دانلود شد. با توجه به سرچ های انجام شده برای نمایش نام اعضای گروه با استفاده از دستور printk در هنگام بوت سیستم کافی بود در مسیر /init و در فایل main.c در تابع do_initcalls دستور مد نظر اضافه گردد.

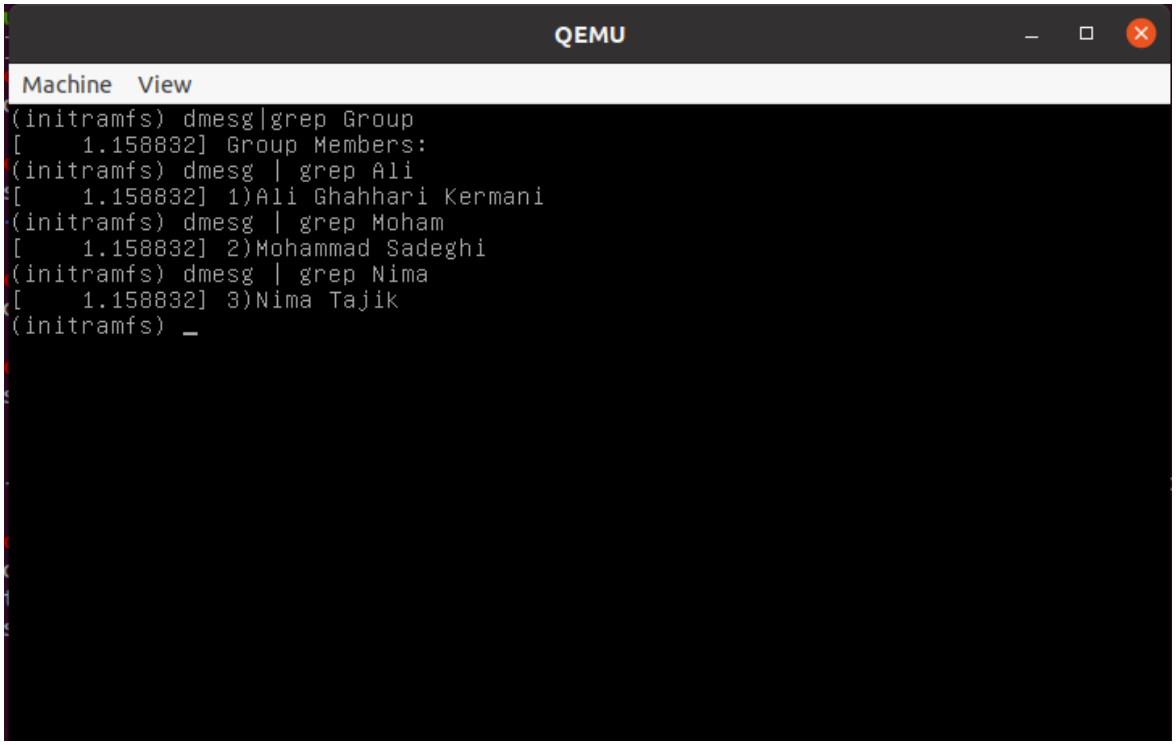
```
1296  static void __init do_initcalls(void)
1297  {
1298      int level;
1299      size_t len = saved_command_line_len + 1;
1300      char *command_line;
1301
1302      command_line = kzalloc(len, GFP_KERNEL);
1303      if (!command_line)
1304          panic("%s: Failed to allocate %zu bytes\n", __func__, len);
1305
1306      printk("Group Members:\n1)Ali Ghahhari Kermani\n2)Mohammad Sadeghi\n3)Nima Tajik");
1307
1308      for (level = 0; level < ARRAY_SIZE(initcall_levels) - 1; level++) {
1309          /* Parser modifies command_line, restore it each time */
1310          strcpy(command_line, saved_command_line);
1311          do_initcall_level(level, command_line);
1312      }
1313
1314      kfree(command_line);
1315  }
1316
```

همانطور که دیده می شود این دستور در فایل و تابع اشاره شده در بالا اضافه شد.

سپس هسته را make می کنیم و با دستور های زیر ان را در محیط qemu اجرا می کنیم.

```
make -j `nproc` 4 bzImage
cd arch/x86/boot/
mkinitramfs -o initrd.img-6.5.7
qemu-system-x86_64 -kernel bzImage -initrd initrd.img-6.5.7 -m 1G
```

نام اعضای گروه از طریق دستور dmesg قابل مشاهده است


QEMU
Machine View
(initramfs) dmesg|grep Group
[1.158832] Group Members:
(initramfs) dmesg | grep Ali
[1.158832] 1)Ali Ghahhari Kermani
(initramfs) dmesg | grep Moham
[1.158832] 2)Mohammad Sadeghi
(initramfs) dmesg | grep Nima
[1.158832] 3)Nima Tajik
(initramfs)