به نام خدا

گزارش کار پروژه اول آزمایشگاه سیستم عامل

" آشنایی با هسته سیستم عامل xv6 "

گروه 5 :

- سینا طبسی
810199554

- سید حامد میرامیرخانی
810199500

- فاطمه محمدی
810199489

Repository Link: https://github.com/HamedMiramirkhani/OS_Lab_CA1

last commit: c550039d0f1b77469e537603f40d71c2d23edbdb

❖ آشنایی با سیستم عامل xv6

1.معماری سیستم عامل xv6 چیست؟

UNIX-LIKE (شبه یونیکس) و مشابه Unix v6 نوشته شده است.

چه دلایلی در دفاع از نظر خود دارید؟

چرا که پردازنده xv6 پیاده سازی مجدد نسخه Dennis Ritchie's and Ken Thompson's Unix version 6 میباشد؛ و از همان ساختار و سبک unix v6 پیروی میکند.

2.یک پردازنده در سیستم عامل xv6 از چه بخش هایی تشکیل شده است؟

- User-space memory (instructions, data, and stack)
  حافظه فضای کاربری(شامل دستورات – داده ها – استک)
- Per-process state private to the kernel
  تنها برای هسته قابل رویت است.

این سیستم عامل به طور کلی چگونه پردازنده را به پردازه های مختلف اختصاص میدهد؟

پردازنده xv6 به صورت transparency ، CPU های در دسترس در میان مجموعه ای از فرآیندهایی که منتظر اجرا هستند، سوییچ میکند.
زمانی که یک فرآیند اجرا نمیشود رجیسترهای CPU را ذخیره میکند و هنگام اجرای بعدی فرآیند آنها را بازیابی میکند.
** هسته سیستم عامل (kernel) یک شناسه فرآیند یا pid به هر فرآیند نسبت میدهد.

4. فراخوانی های سیستمی fork , exec چه عملی انجام میدهند؟

- فراخوانی fork: ساختن یک فرایند(process)

فراخوانی fork یک فرآیند که فرآیند فرزند خوانده میشود با دقیقا محتویات حافظه فرایند فراخوانی کننده (فرایند والد) میسازد.
فراخوانی fork هر دو فرزند و والد را برمیگرداند.

- فراخوانی exec: بارگذاری (load) یک فایل و اجرا کردن آن.

فراخوانی exec حافظه فرایند فراخوانی کننده(فرایند فعلی)را با یک حافظه جدید که در ان برنامه (با فایل ELF) بارگیری شده جایگزین میکند. در واقع از این فراخوانی برای اجرای یک برنامه در فرایند فعلی استفاده میشود.

از نظر طراحی، ادغام نکردن این دو چه مزیتی دارد؟

در زمان I/O redirection ابتدا فراخوانی fork اجرا شود و یک فرایند جدید ایجاد شود و پس از ایجاد فرایند فرزند با استفاده از فراخوانی exec برنامه در خواست شده توسط کاربر جایگزین فرایند فرزند شود و ادغام نکردن این دو فراخوانی این امکان را میدهد که تغییرات لازم را در صورت لزوم پس از فراخوانی fork در file descriptor انجام دهد و سپس فراخوانی exec انجام شود .

❖ کامپایل سیستم عامل xv6

8. در makefile متغیر هایی به نام های ULIB و UPROGS تعریف شده است. کاربرد انها چیست؟

- **متغیرهای UPROGS**

این متغیر ها مربوط به برنامه های سطح کاربر میشود و شامل لیستی از این برنامه ها میشود.
در هنگام کامپایل سیستم عامل xv6، این برنامه ها نیز کامپایل میشوند و به فایلهای قابل اجرا تبدیل میشوند. و میتوان آنها را در shell فراخوانی کرد.

- **متغیرهای ULIB**

همانطور که از اسم این متغیر ها میتوان فهمید این متغیر ها شامل یک سری کتابخانه(کتابخانه های سطح کاربر) میشوند.
این کتابخانه ها در بسیاری از کدهای xv6 استفاده شده اند و به همین دلیل به کامپایل این فایل ها نیاز داریم.

❖ اجرا بر روی شبیه ساز QEMU

11.برنامه‌های کامپایل شده در قالب فایل‌های دودویی نگه‌داری می‌شوند. فایل مربوط به بوت نیز دودویی است.

نوع این فایل دودویی چیست؟

دو فایل bootasm.S و bootmain.c که به ترتیب به زبان های assembly و C هستند، کامپایل شده و Object File های این دو فایل که فایل extention آنها .o است و فایل باینری به شمار می روند ساخته میشوند.

```
hamed@Hamed:~/Desktop/OS-Lab-CA1/codes$ objdump -D -b binary -mi386 -Maddr16,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 fc 00                call   0x147
  4b:   00 00                   add    %al,(%bx,%si)
  4d:   66 b8 00 8a 66 89       mov    $0x89668a00,%eax
  53:   c2 66 ef                ret    $0xef66
  56:   66 b8 e0 8a 66 ef       mov    $0xef668ae0,%eax
  5c:   eb fe                   jmp    0x5c
  5e:   66 90                   xchg   %eax,%eax
        ...
```

تفاوت این نوع فایل دودویی با دیگر فایل‌های دودویی کد xv6 چیست؟چرا از این نوع فایل دودویی استفاده شده است؟

تفاوت فایل های boot با دیگر فایل ها در قرارگیری first block on first است که همیشه در sector اول است اما برای سایر فایل ها چنین نیست.

این فایل را به زبان قابل فهم انسان (اسمبلی) تبدیل نمایید.(راهنمایی: از ابزار objdump استفاده کنید. باید بخشی از آن مشابه فایل bootasm.S باشد.)

با دستور زیر فایل به زبان اسمبلی تبدیل میشود:

Objdump -D bootmain.o

12. علت استفاده از دستور objcopy در حین اجرای عملیات makefile چیست؟

با استفاده از این دستور می توان محتویات یک فایل object را در یک فایل object دیگر کپی کرد و در کنار مزیتهایی که دارد یک سری امکانات اضافی نیز ارائه میشود.

مزیت ها:

** برای کپی کردن نیازی به یکسان بودن فرمت فایل ورودی و مقصد نمی باشد.

** تمامی فرمت های موجود در کتابخانه BFD پشتیبانی میشوند.

14. یک ثبات عام منظوره، یک ثبات قطعه، یک ثبات وضعیت و یک ثبات کنترلی در معماری را نام برده و وظیفه هر یک را به طور مختصر توضیح دهید

رجیستر عام منظوره: نگهداری متغیرها حین محاسبات مانند EBX , EDX , ESI , EDI ,..

رجیستر قطعه: اشاره گر به قطعات مختلف مثل استک، داده و کد مانند SS:اشاره گر به استک, CS اشاره گر به کد

رجیستر وضعیت: اطلاعات وضعیت کنونی پردازنده را نگه میدارد مانند EFLAGS که اطلاعاتی درباره فلگ های zero و overflow و ... را نگه میدارد.

رجیستر کنترلی: مسئول تغییر یا کنترل پردازنده را دارد مانند CR0 که وظیفه ی activate protected mode و یا switch بین task ها را دارد.

18. کد معادل entry.s را در هسته لینوکس بیابید.

معادل entry.s در هسته لینوکس در این لینک است.

❖ اجرای هسته xv6

19. چرا این آدرس فیزیکی است؟

فرض کنیم آدرس آن مجازی بود، باز هم باید یک بخش فیزیکی در نظر میگرفتیم تا آدرس این مکان مجازی را تبدیل به فیزیکی کند. پس عملا کار بیهوده ای بود.

22. علاوه بر صفحه بندی در حد ابتدایی از قطعه بندی به منظور حفاظت از هسته استفاده خواهد شد. این عملیات توسط seginit انجام می گردد. همان طور که ذکر شد، ترجمه قطعه تاثیری بر ترجمه آدرس منطقی نمیگذارد زیرا تمامی قطعات روی یکدیگر می افتد. با این حال برای کد و داده های سطح کاربر پرچم SEG_USER تنظیم شده است. چرا؟

از آنجایی که محتوای هر دوی این پردازنده ها در یک فضای فیزیکی قرار گرفته اند، باید بتوانیم تمایزی بین پردازه های سطح کاربر با سطح کرنل ایجاد کنیم. چنین کاری به ما نشان میدهد قطعات، مربوط به سطح کاربر هستند و اجازه دسترسی به هسته را ندارند.

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.

sz: سایز موری پردازه بر حسب byte

Tf: فریم trap برای syscall فعلی

pgdir: پوینتر به page table

Kstack: تشخیص پایین stack کرنل پردازه

contex: نگهداری context switching

state: تشخیص وضعیت پردازه

pid: تشخیص آیدی مختص پردازه

parent: تشخیص سازنده پردازه

ofile: تشخیص فایل های باز شده

cwd: تشخیص پوشه فعلی

name: تشخیص نام پردازه

killed: تشخیص kill شدن: صفر نباشد پردازه kill شده است.

chan: تشخیص حالت خواب پردازه: صفر نباشد پردازه در حالت خواب است

معادل آن(task_struct) در لینوکس در این لینک است.

27. کدام بخش آماده سازی سیستم بین تمامی هسته های پردازنده مشترک و کدام بخش اختصاصی است؟(از هر یک مثال و توضیح دهید) زمان بند روی کدام هسته اجرا میشود؟

همانطور که در main.c معلوم است یک سری دستورات مثل allocate کردن physical page و ساختن trap vector ها و ساختن لینک لیستی از بافر ها به عهده Bootstrap processor است. ولی کارهایی مانند مپ کردن آدرس مجازی که توسط تابع seginit انجام میشود بین آنها مشترک است.

```
17  int
18  main(void)
19  {
20    kinit1(end, P2V(4*1024*1024)); // phys page allocator
21    kvmalloc();      // kernel page table
22    mpinit();        // detect other processors
23    lapicinit();     // interrupt controller
24    seginit();       // segment descriptors
25    picinit();       // disable pic
26    ioapicinit();    // another interrupt controller
27    consoleinit();   // console hardware
28    uartinit();      // serial port
29    pinit();         // process table
30    tvinit();        // trap vectors
31    binit();         // buffer cache
32    fileinit();      // file table
33    ideinit();       // disk
34    startothers();   // start other processors
35    kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()
36    userinit();      // first user process
37    mpmain();        // finish this processor's setup
38  }
39
40  // Other CPUs jump here from entryother.S.
41  static void
42  mpenter(void)…
49
50  // Common CPU setup code.
51  static void
52  mpmain(void)…
59
```

چاپ نام اعضای گروه

مطابق شکل زیر در فایل init.c خط ۲۳ را اضافه میکنیم:

```c
// init: The initial user-level program

#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

char *argv[] = { "sh", 0 };

int
main(void)
{
  int pid, wpid;

  if(open("console", O_RDWR) < 0){
    mknod("console", 1, 1);
    open("console", O_RDWR);
  }
  dup(0);  // stdout
  dup(0);  // stderr

  for(;;){
    printf(1, "Group #5 Members:\n1- Fatemeh Mohammadi\n2- Sina Tabasi\n3- Hamed Miramirkhani\n");
    pid = fork();
    if(pid < 0){
      printf(1, "init: fork failed\n");
      exit();
    }
    if(pid == 0){
      exec("sh", argv);
      printf(1, "init: exec sh failed\n");
      exit();
    }
    while((wpid=wait()) >= 0 && wpid != pid)
      printf(1, "zombie!\n");
  }
```

اضافه کردن ۳ قابلیت به کنسول

این موارد در فایل console.c اعمال می شوند.

قبل از پیاده سازی ۳ قابلیت جدید، بعضی از قسمت های کد را تغییر میدهیم. هدف از این تغییرات، امکان اضافه شدن قابلیت های جدید و همینطور افزایش خوانایی کد است.

```c
#define INPUT_BUF 128
struct {
  char buf[INPUT_BUF];
  uint r;  // Read index
  uint w;  // Write index
  uint e;  // Edit index
  uint last; //Last Character Index
} input;
```

یک فیلد جدید به استراکت input اضافه میکنیم. با استفاده از این این فیلد آخرین کاراکتر بافر مشخص میشود.

```
190   int get_pos() {
191     outb(CRTPORT, 14);
192     int pos = inb(CRTPORT+1) << 8;
193     outb(CRTPORT, 15);
194     pos |= inb(CRTPORT+1);
195     return pos;
196   }
```

چند خط کد بالا را که در کد وجود داشت برای خوانایی بیشتر داخل یک تابع تعریف میکنیم.با استفاده از این تابع مکان نشانگر در هر لحظه مشخص میشود.

```
198   static void change_pos(int pos) {
199     outb(CRTPORT, 14);
200     outb(CRTPORT+1, pos>>8);
201     outb(CRTPORT, 15);
202     outb(CRTPORT+1, pos);
203   }
```

مشابه get_pos() تابع بالا را تعریف میکنیم. با استفاده از این تابع می توانیم مکان نشانگر را تغییر دهیم.

```
292       case C('H'): case '\x7f':  // Backspace
293         if(input.e != input.w){
294           consputc(BACKSPACE);
295           shift_left_input();
296           input.e--;
297         }
298         break;
```

```
320     default:
321       if(c != 0 && input.e-input.r < INPUT_BUF){
322         c = (c == '\r') ? '\n' : c;
323         if(c == '\n' || c == C('D'))
324           input.buf[input.last++ % INPUT_BUF] = c;
325         else {
326           shift_right_input();
327           input.buf[input.e++ % INPUT_BUF] = c;
328         }
329         consputc(c);
330         if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
331           input.e = input.last;
332           input.w = input.e;
333           wakeup(&input.r);
334         }
335       }
336       break;
```

کیس ctrl+H و همینطور قسمت default در تابع consoleintr را refactor میکنیم. تابع مورد استفاده در خط ۲۹۵ و ۳۲۶ در ادامه بیان میشوند.

همچنین توابعی به کد اضافه شدند:

```
205   void shift_right_input() {
206     int index, next_char, pos;
207     pos = get_pos();
208     change_pos(pos + 1);
209     index = input.e;
210     next_char = input.buf[index % INPUT_BUF];
211     input.buf[index % INPUT_BUF] = ' ';
212     while(index < input.last) {
213       int temp = next_char;
214       next_char = input.buf[(index + 1) % INPUT_BUF];
215       input.buf[(index + 1) % INPUT_BUF] = temp;
216       consputc(input.buf[(index + 1) % INPUT_BUF]);
217       index++;
218     }
219     input.last++;
220     change_pos(pos);
221   }
```

```
223   void shift_left_input() {
224     int index, pos;
225     pos = get_pos();
226     index = input.e - 1;
227     while(index < input.last) {
228       input.buf[index % INPUT_BUF] = input.buf[(index + 1) % INPUT_BUF];
229       consputc(input.buf[index % INPUT_BUF]);
230       index++;
231     }
232     consputc(' ');
233     input.last--;
234     change_pos(pos);
235   }
```

برای پیاده کردن قابلیت های مورد نظر باید بتوانیم طبق شرایطی کاراکتر های داخل بافر را به چپ یا راست شیفت بدهیم، دو تابع بالا چنین خاصیتی دارند.

اضافه کردن ۳ قابلیت:

کلید میانبر shift+x مطابق زیر تعریف میشود. دقت کنید کاراکتر space با استفاده از کد ascii بدست آمده است(شیفت به همراه یک کاراکتر ، کاراکتر جدیدی میسازد که اختلاف این دو کاراکتر ۳۲ واحد است و ۳۲ کد اسکی space است.)

```
187   #define C(x)  ((x)-'@')  // Control-x
188   #define S(x)  ((x)+' ')  // Shift-x
```

a) shift + [

```
237   void go_first_line() {
238     int pos = get_pos();
239     int delta_pos = pos%80 - 2;
240     input.e -= delta_pos;
241     change_pos(pos - delta_pos);
242   }
```

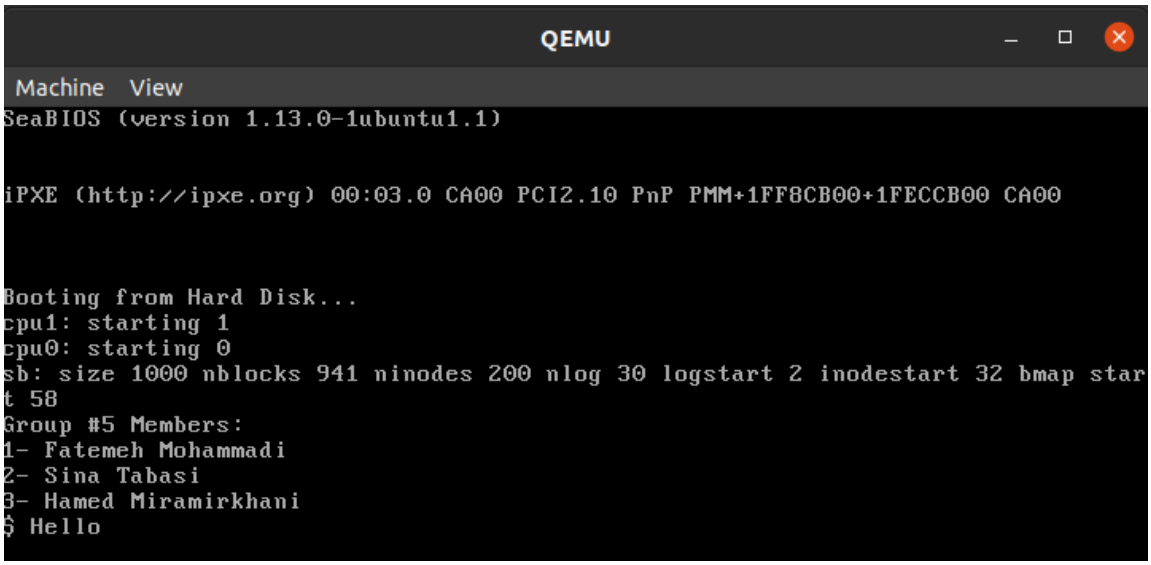
```
302       case S('['):  // Move to First of Line
303         if(get_pos()%80 != 2) // is cursor at first of line?
304         {
305           go_first_line();
306           change_pos(get_pos()-1);
307           shift_right_input();
308           change_pos(get_pos()+1);
309         }
310         break;
```

این دستور وظیفه دارد نشانگر را به ابتدای خط ببرد.

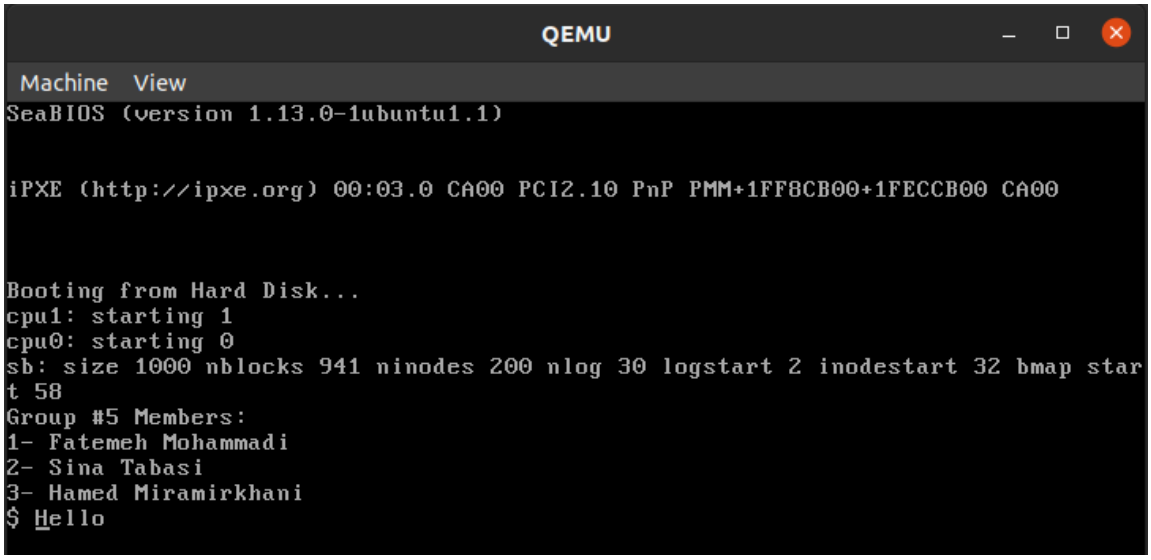
پس از انتقال نشانگر به ابتدای خط باید بافر یک واحد به چپ شیفت پیدا کند و همینطور یک space اضافه کند، خطوط ۳۰۶ تا ۳۰۸ این کار را میکنند.

مشاهده عملکرد:

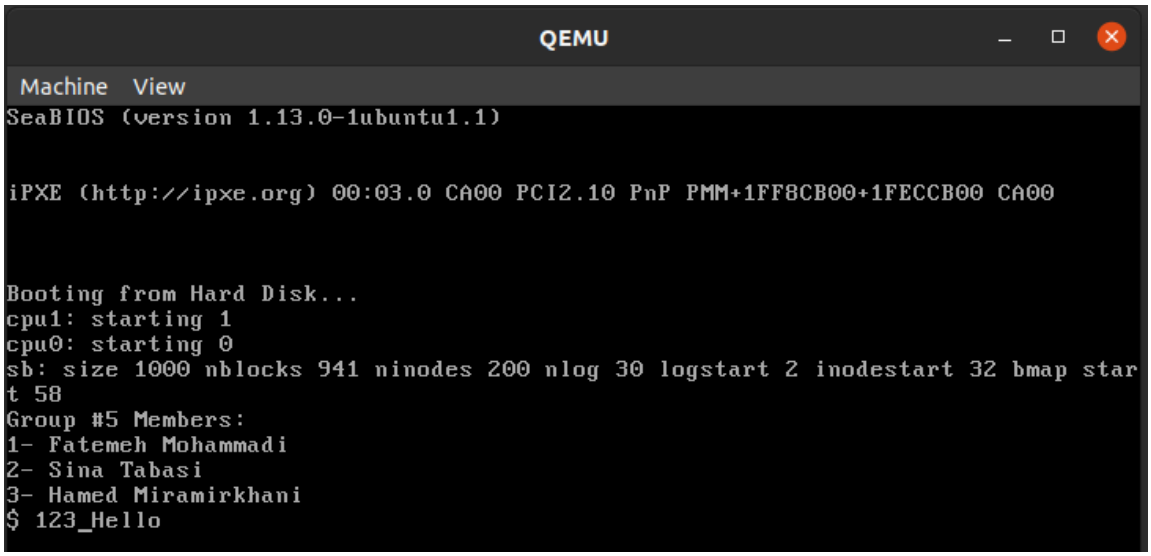
print “Hello”



press shift + [



print “123”



b) shift +]

```
244   void go_end_line() {
245     int pos = get_pos();
246     int delta_pos = input.last - input.e;
247     input.e += delta_pos;
248     change_pos(pos + delta_pos);
249   }
```

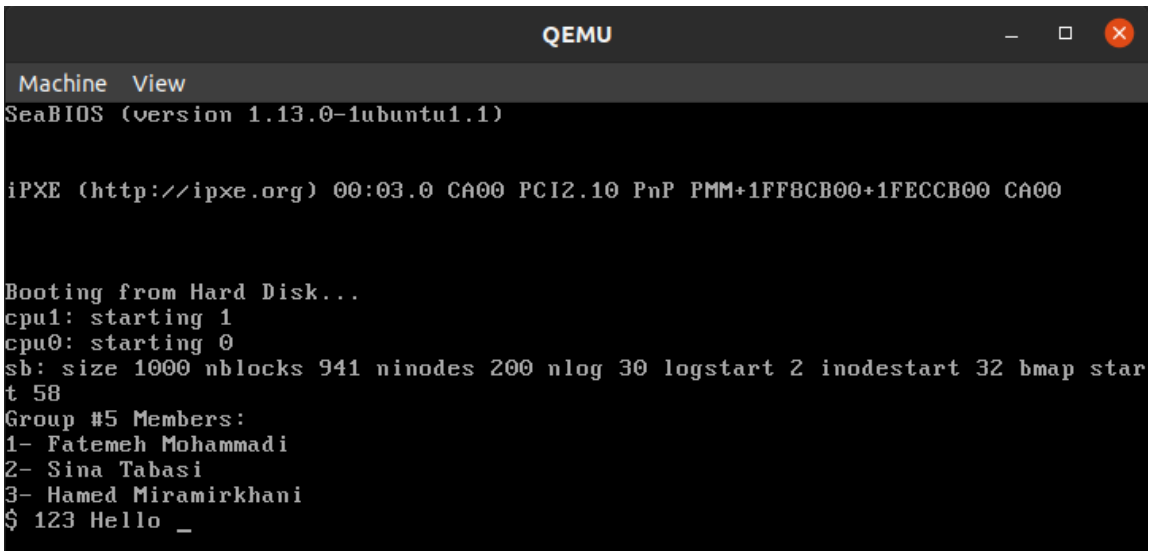
```
311     case S(']'):  // Move to End of Line
312       if(input.last != input.e) // is cursor at end of line?
313       {
314         go_end_line();
315         shift_right_input();
316         input.buf[input.e++ % INPUT_BUF] = ' ';
317         change_pos(get_pos()+1);
318       }
319       break;
```

این دستور باید نشانگر را به انتهای خط ببرد.

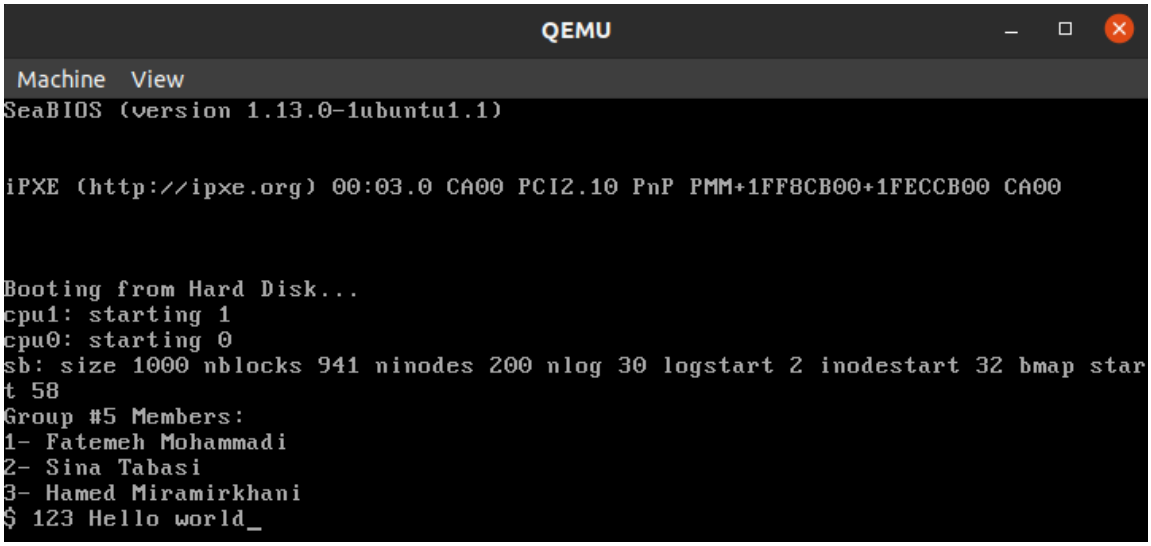
پس از انتقال نشانگر به انتهای خط باید یک اسپیس هم در بافر به عنوان آخرین کاراکتر قرار بگیرد، خطوط ۳۱۵ تا ۳۱۷ این کار را میکنند.

مشاهده عملکرد:

press shift +]



print “world”



c) ctrl + W

```
299     case C('W'): // remove pre-cursor word
300       remove_last_word();
301       break;
```

```
251   void remove_char()
252   {
253     shift_left_input();
254     consputc(BACKSPACE);
255     input.e--;
256   }
```

```
258   void remove_last_word() {
259     while(input.e != input.w) {
260       if(get_pos()%80 == 0) // is cursor at first of line?
261         break;
262       if(input.buf[(input.e % INPUT_BUF)-1] != 32) // 32 = ascii SPACE
263         break;
264       remove_char();
265     }
266     while(input.e != input.w) {
267       if(get_pos()%80 == 0) // is cursor at first of line?
268         break;
269
270       if(input.buf[(input.e % INPUT_BUF)-1] == 32) // 32 = ascii SPACE
271         break;
272       remove_char();
273     }
274   }
```

وظیفه ی این دستور این است که آخرین کلمه را حذف کند.

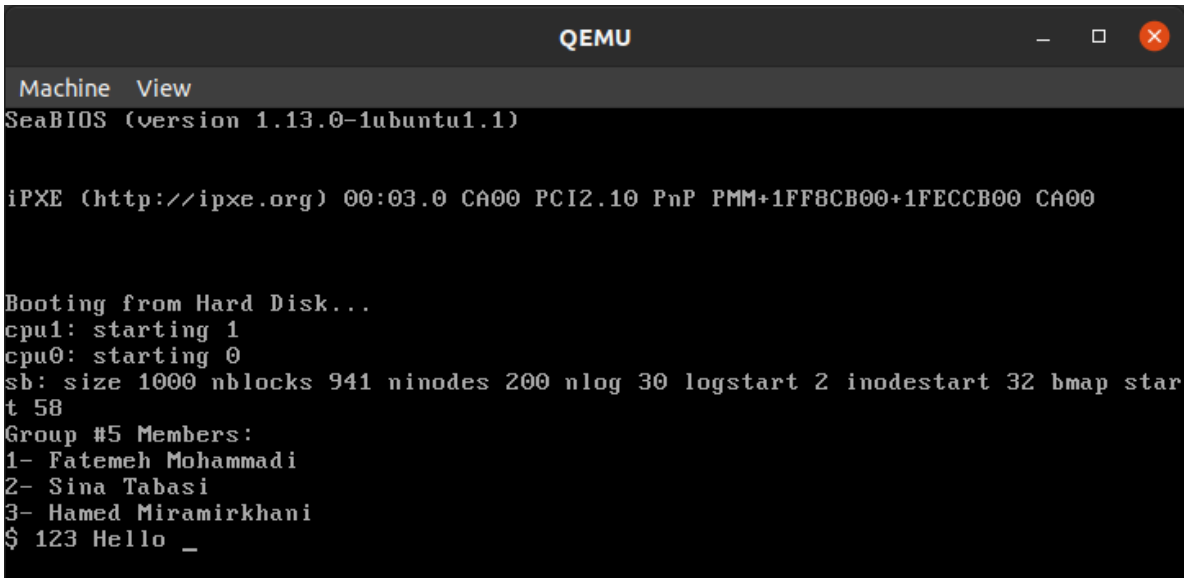
خطوط ۲۵۹ تا ۲۶۵ هر spaceی که آخر ورودی وجود دارد را پاک میکنند، مثلا حالت زیر را در نظر بگیرید:

```
Group #5 Members:
1- Fatemeh Mohammadi
2- Sina Tabasi
3- Hamed Miramirkhani
$ hello
```

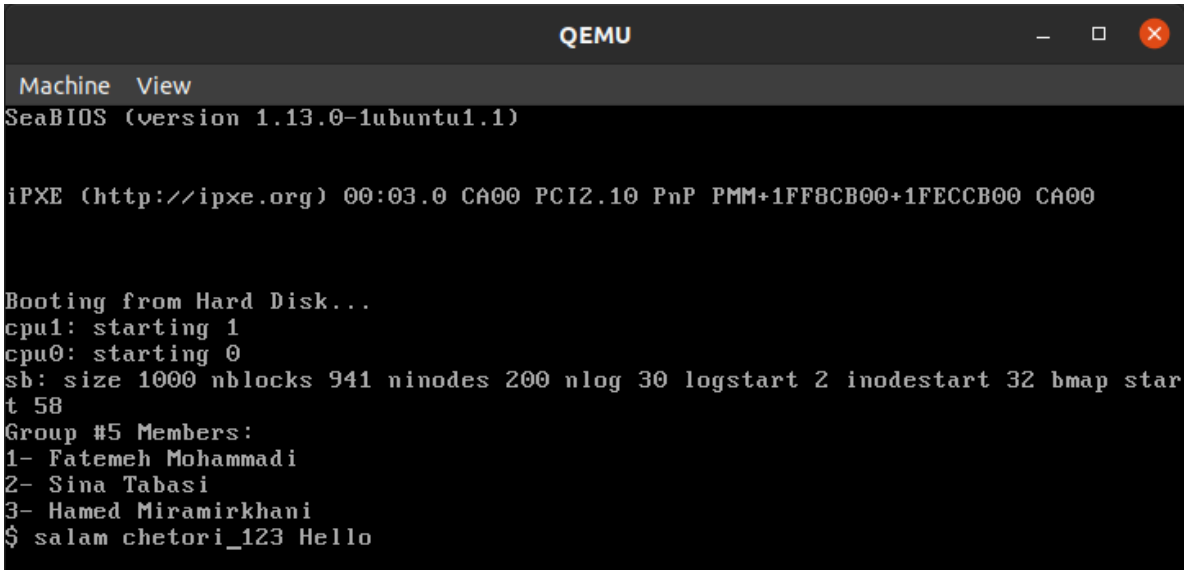
خطوط ۲۶۶ تا ۲۷۳ هم آخرین کلمه را حذف میکنند.

مشاهده عملکرد:

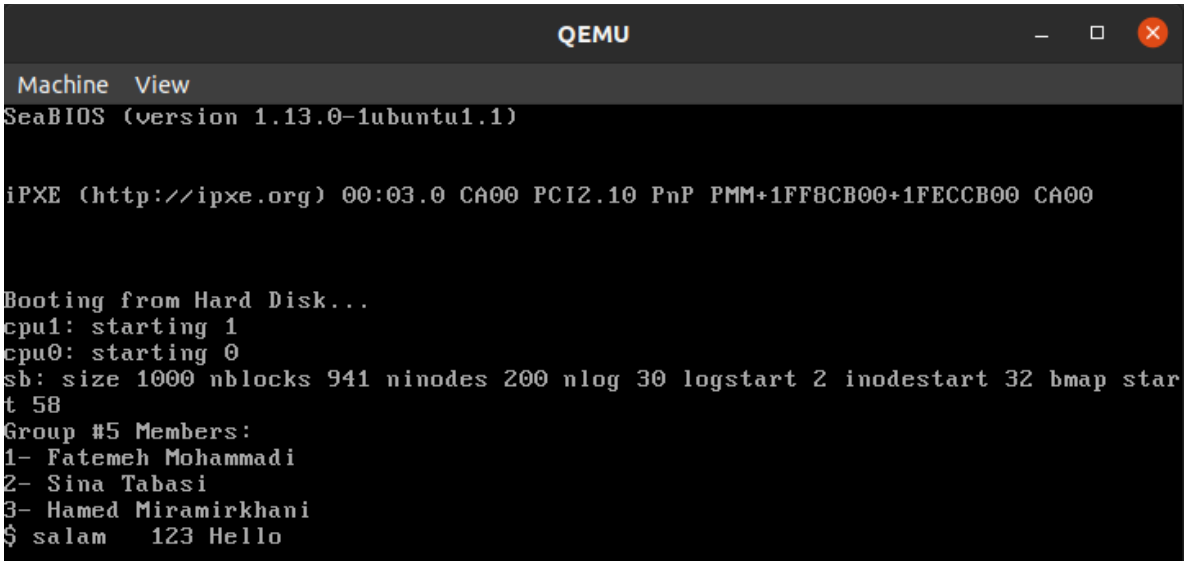
press ctrl + W



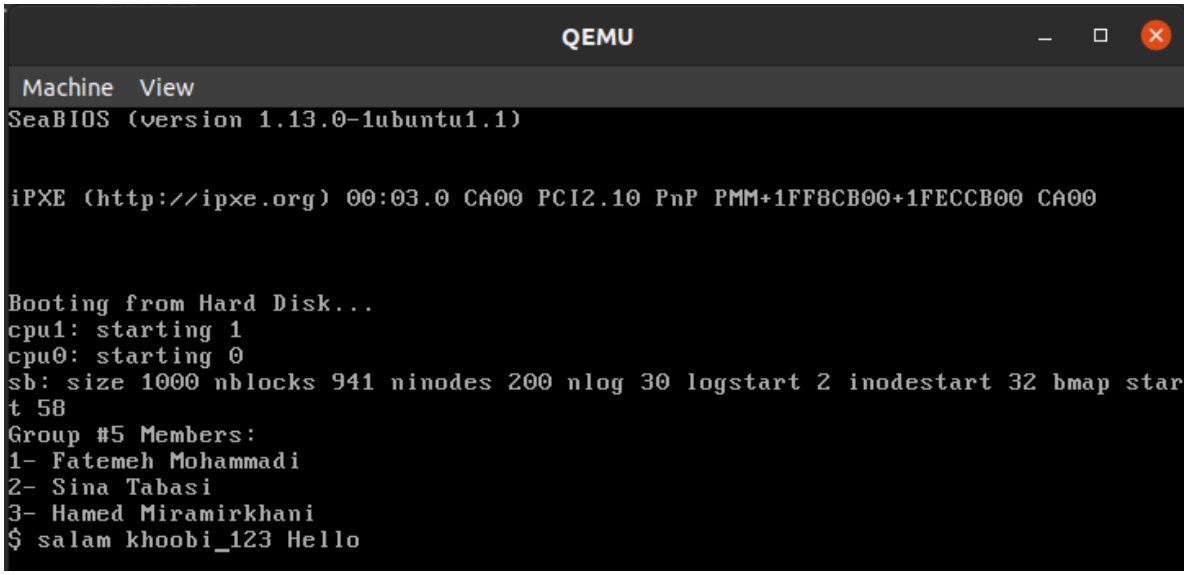
press shift +] and print “salam chetori”



press ctrl + W



print “khoobi”



❖ اجرا و پیاده سازی یک برنامه سطح کاربر

این برنامه سطح کاربر mmm.c می باشد که تا 7 عدد را از کاربر گرفته و به ترتیب میانگین ، میانه و مد این اعداد را محاسبه کرده و در فایل mmm_result.txt ذخیره می نماید.

بخش های مختلف کد به صورت زیر نمایش داده شده است:

1-تابع tostring:

این تابع تبدیل int به string برای ما انجام می دهد تا بتوان با استفاده از آن ، نتیجه میانگین یا میانه یا مد را در فایل mmm_result.txt بنویسیم:

```
5   void tostring(char str[], int num)
6   {
7       int i, rem, len = 0, n;
8
9       n = num;
10      while (n != 0)
11      {
12          len++;
13          n /= 10;
14      }
15      for (i = 0; i < len; i++)
16      {
17          rem = num % 10;
18          num = num / 10;
19          str[len - (i + 1)] = rem + '0';
20      }
21      str[len] = '\0';
22  }
```

2- تابع average:

این تابع میانگین اعدادی که در arr ذخیره شده اند را حساب کرده و با دستور write آن را در فایل mmm_result.txt می نویسد:

```
80  void avrage(int arr[],int i,int fd) {
81      int avg = 0;
82      char buf[512];
83      for (int j = 0; j < i; j++) {
84          avg += arr[j];
85      }
86      tostring(buf,avg/ i);
87      write(fd, buf,strlen(buf));
88      write(fd," ",1);
89  }
```

2- تابع median:

این تابع ابتدا اعداد را sort کرده و سپس با توجه به فرد یا زوج بودن تعداد اعداد ، میانه را تعیین کرده و نتیجه را در فایل txt مورد نظر می نویسد:

```
50  void median(int arr[],int i,int fd){
51      char buf[512];
52      int t;
53      int result;
54
55      for (int f = 0; f < i - 1; f++)
56      {
57          for (int j = 0; j < i - f - 1; j++)
58          {
59              if(arr[j] > arr[j + 1]){
60                  t = arr[j];
61                  arr[j] = arr[j+1];
62                  arr[j+1] = t;
63              }
64          }
65      }
66
67      if (i % 2  == 0)
68      {
69          result = (arr[(i/2)-1] + arr[i/2])/2;
70      }
71      else
72      {
73          result = arr[(i-1)/2];
74      }
75      tostring(buf,result);
76      write(fd, buf,strlen(buf));
77      write(fd," ",1);
78  }
```

3- تابع mode :

این تابع مد اعداد موجود در arr را به دست می آورد و سپس آن را در فایل txt مورد نظر می نویسد:

```c
void mode(int arr[],int n,int fd) {
    int minValue = 0, maxCount = 0, i, j;
    char buf[512];

    for (i = 0; i < n; ++i) {
        int count = 0;

        for (j = 0; j < n; ++j) {
            if (arr[j] == arr[i])
                ++count;
        }

        if (count > maxCount) {
            maxCount = count;
            minValue = arr[i];
        }
        if (count == maxCount && arr[i] < minValue){
            minValue = arr[i];
        }
    }
    tostring(buf,minValue);
    write(fd, buf,strlen(buf));
    write(fd,"\n",1);

}
```

4-تابع main:

تابع اصلی برنامه است که ابتدا تعداد آرگومان و خود آرگومان های وارد شده از ترمینال را بررسی می کند و سه تابع average و median و mode در این تابع فراخوانی می شوند:

```c
int main(int argc, char *argv[]){

    unlink("mmm_result.txt");
    if(argc > 8)
    {
        printf(1,"%s \n","The limit on length of numbers has exceeded");
        exit();
    }
    int buf[argc-1];
    for(int i=0; i<argc-1; i++)
    {
        buf[i] = atoi(argv[i+1]);
    }
    int fd = open("mmm_result.txt",O_CREATE | O_RDWR);
    avrage(buf,argc-1,fd);
    median(buf,argc-1,fd);
    mode(buf,argc-1,fd);
    close(fd);
    exit();
}
```

5- تغییرات makefile:

برای اضافه کردن برنامه سطح کاربر mmm به سیستم عامل xv6 ، باید تغییراتی در makefile آن به وجود آورد که به شرح زیر می باشد:

ابتدا باید فایلی برای run کردن برنامه mmm ایجاد کرد. بنابراین باید یک executable file به نام _mmm ایجاد نمود. بنابراین با اضافه کردن _mmm به UPROGS در makefile سیستم عامل ، این executable file ایجاد می شود:

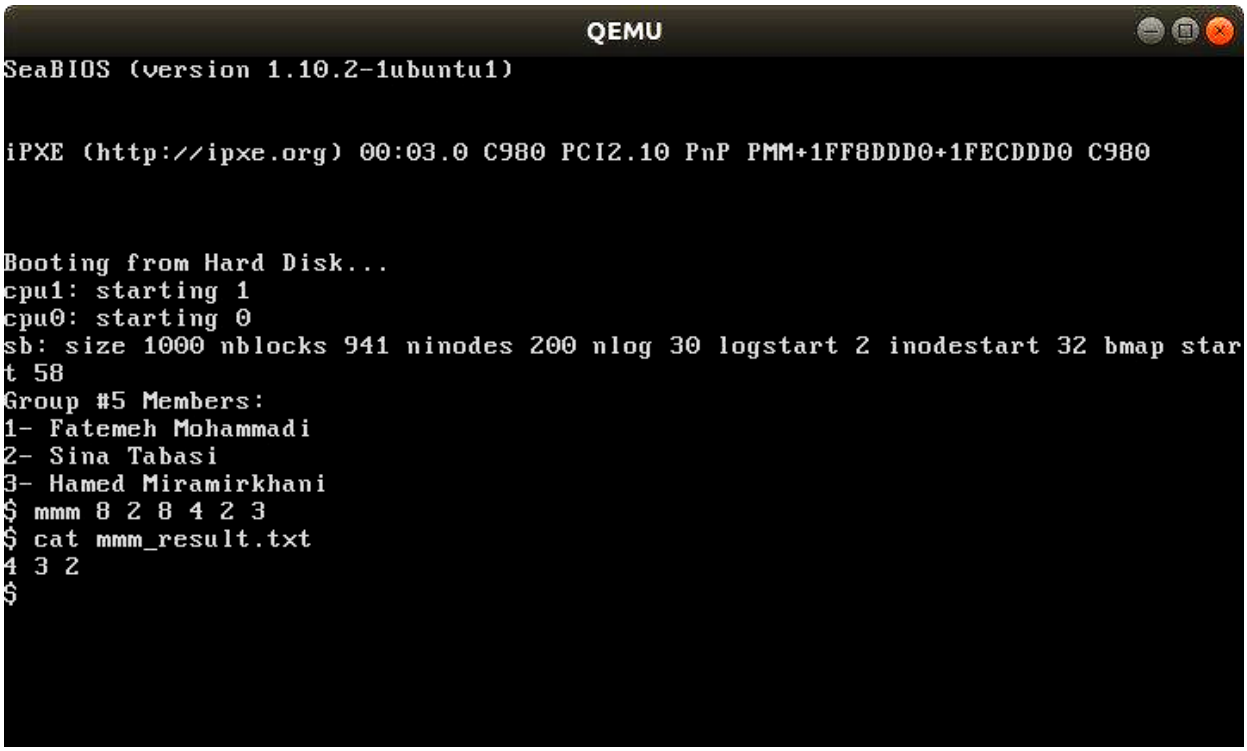
```
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mmm\
178     _mkdir\
179     _rm\
180     _sh\
181     _stressfs\
182     _usertests\
183     _wc\
184     _zombie\
```

حال باید فایل اصلی برنامه را به قسمت EXTRA در makefile اضافه نمود:

```
251 EXTRA=\
252     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
253     ln.c ls.c mmm.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
254     printf.c umalloc.c\
255     README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
256     .gdbinit.tmpl gdbutil\
```

6- نتیجه کد:

در این قسمت برنامه را با test case گفته شده در صورت پروژه run کردیم و نتیجه به صورت زیر می باشد:



- اشکال زدائی

روند اجرای GDB

1- دستور مشاهده breakpoint ها

می توان برای دیدن breakpoint از دستور info breakpoints استفاده کرد:

```
(gdb) break cat.c:12
Breakpoint 1 at 0x98: file cat.c, line 12.
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x00000098 in cat at cat.c:12
```

2- دستور حذف یک breakpoint:

برای حذف یک breakpoint از دستور del n استفاده کرد که n در اینجا شماره breakpoint در لیست breakpoint ها می باشد.
در مثال زیر ابتدا ما دو breakpoint در خطوط 12 و 15 در فایل cat.c قرار دادیم سپس با دستور del 2 دومین breakpoint که در خط 15 قرار داشت را حذف کردیم:

```
(gdb) break cat.c:12
Breakpoint 1 at 0x98: file cat.c, line 12.
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x00000098 in cat at cat.c:12
(gdb) break cat.c:15
Breakpoint 2 at 0xec: file cat.c, line 15.
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x00000098 in cat at cat.c:12
2       breakpoint     keep y   0x000000ec in cat at cat.c:15
(gdb) del 2
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x00000098 in cat at cat.c:12
```

کنترل روند اجرا و دسترسی به حالت سیستم

3- خروجی bt:

دستور bt که مخفف backtrace است call stack برنامه در لحظه کنونی در حین متوقف بودن روند اجرای برنامه را نشان می دهد . هر تابع که صدا زده میشود یک stack frame مخصوص به خودش را میگیرد که متغیرهای محلی و آدرس بازگشت و غیره در آن قرار دارند. خروجی این دستور در هر خط یک stack frame را نشان میدهد که به ترتیب از درون ترین frame که در آن قرار دارید شروع میشود .دستور where نیز همانند bt عمل می کند که در بخش های آینده از آن استفاده می کنیم.

در مثال زیر ابتدا ما در خط 12 فایل cat.c یک breakpoint قرار داده تا روند اجرای برنامه در آن نقطه متوقف شود. سپس با دستور continue و سپس دستور cat README در ترمینال دیگر، برنامه اجرا شده و در خط 12 (محل breakpoint) متوقف می شود (خط 12 در ترمینال نمایش داده شده است). حال با اجرای دستور bt می توان call stack را مشاهده کرد. همانطور که در call stack می بینیم خط 12 کد در تابع cat که خود آن در تابع main برنامه cat.c است ، قرار دارد:

```
(gdb) file _cat
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from _cat...done.
(gdb) break 12
Breakpoint 1 at 0x98: file cat.c, line 12.
(gdb) continue
Continuing.
[Switching to Thread 2]

Thread 2 hit Breakpoint 1, cat (fd=3) at cat.c:12
12          while((n = read(fd, buf, sizeof(buf))) > 0) {
(gdb) bt
#0  cat (fd=3) at cat.c:12
#1  0x00000054 in main (argc=<optimized out>, argv=<optimized out>) at cat.c:39
```

4- تفاوت دستور x و print:

با استفاده از دستور print می توان مقدار یک متغیر را چاپ کرد که آرگومان ورودی این دستور، نام متغیر می باشد.

با استفاده از دستور x میتوان محتویات یک خانه حافظه را چاپ کرد. بنابراین آرگومان ورودی این دستور،آدرس خانه حافظه است.

در مثال زیر برای در خط 12 برنامه cat.c یک breakpoint قرار داده و سپس با دستور print fd مقدار متغیر fd را چاپ کردیم .
حال برای استفاده از دستور x ، باید آدرس خانه متغیر fd را بدانیم ، بنابراین با اجرای دستور print &fd آدرس خانه حافظه ای که fd در آن قرار دارد چاپ می شود. حال با اجرا دستور x adr مقدار آن خانه چاپ می شود. همچنین می توان با دستور print/fmt و x/fmt فرمت چاپ را تعیین کرد. بنابراین در انتها با اجرا دستور x/d adr فرمت decimal متغیر چاپ می شود:

```
Breakpoint 1 at 0x98: file cat.c, line 12.
(gdb) continue
Continuing.
[Switching to Thread 2]

Thread 2 hit Breakpoint 1, cat (fd=3) at cat.c:12
12          while((n = read(fd, buf, sizeof(buf))) > 0) {
(gdb) print fd
$1 = 3
(gdb) print &fd
$2 = (int *) 0x2f90
(gdb) x 0x2f90
0x2f90: 0x00000003
(gdb) x/d 0x2f90
0x2f90: 3
```

5- نمایش وضعیت ثباتها و متغیرهای محلی؛ رجیسترهای edi و esi:

با استفاده از دستور info registers می توان وضعیت ثباتها را مشاهده کرد.

برای مشاهده متغیرهای محلی نیز میتوان از دستور info locals استفاده کرد.

دستور info registers و info locals بر روی برنامه cat.c انجام گرفته شده است (خط 12 breakpoint قرار دارد):

```
(gdb) info registers
eax            0x3      3
ecx            0x2fe0   12256
edx            0xbfac   49068
ebx            0x2ff0   12272
esp            0x2f80   0x2f80
ebp            0x2f88   0x2f88
esi            0x3      3
edi            0x3      3
eip            0x98     0x98 <cat+8>
eflags         0x216    [ PF AF IF ]
cs             0x1b     27
ss             0x23     35
ds             0x23     35
es             0x23     35
fs             0x0      0
gs             0x0      0
(gdb) info locals
n = <optimized out>
```

E در ابتدای اسامی این ثباتها به معنی Extended بوده و در حالت 32 بیت به کار میرود. بنابراین به آن ها SI و DI می گوییم.

ثبات SI مخفف Source Index بوده و برای اشاره به یک مبدا در عملیات stream به کار میرود. SI به عنوان نشانگر داده و به عنوان مبدا در برخی عملیات مربوط به رشته ها استفاده میشود.

DI نیز مخفف Destination Index بوده و برای اشاره به یک مقصد در عملیات stream به کار میرود. DI به عنوان نشانگر داده و مقصد برخی عملیات مربوط به رشته ها استفاده می شود .

6-ساختار struct inp:

این struct در فایل console.c تعریف شده است و برای خط ورودی کنسول سیستم عامل استفاده می شود:

```
181  #define INPUT_BUF 128
182  struct {
183    char buf[INPUT_BUF];
184    uint r;  // Read index
185    uint w;  // Write index
186    uint e;  // Edit index
187  } input;
```

آرایه buf بافر و محل ذخیره خط ورودی است که اندازه آن حداکثر 128 کاراکتر است .

متغیرهای دیگر عدد هستند و هر کدام ایندکس ای را برای buf را مشخص می کنند .

متغیر w محل شروع نوشتن خط ورودی کنونی در buf است.

متغیر e محل کنونی کرسر در خط ورودی است .

متغیر r برای خواندن buf استفاده می شود(از w قبلی شروع میکند).

در مثالی می توان نحوه عملکرد این متغیر ها را دید:

در ابتدای کار مقادیر اولیه متغیرها را پرینت می کنیم و یک breakpoint در تابع consoleintr در انتهای

بخش default(جایی که اینتر یا ctrl+d زده می شود یا کرسر از buf فراتر می رود) می گذاریم:

```
(gdb) print input
$1 = {buf = '\000' <repeats 127 times>, r = 0, w = 0, e = 0, last = 0}
(gdb) break console.c:340
Breakpoint 1 at 0x80100e9d: file console.c, line 340.
(gdb)
```

حال دستور continue را اجرا کرده و کرسر راست را وارد می کنیم. توجه کنید که هر حرکت کرسر خود 3 کاراکتر در این بافر می ریزد و مقدار e را افزایش دهد.:

```
(gdb) print input
$1 = {buf = "\033[C", '\000' <repeats 124 times>, r = 0, w = 0, e = 3,
  last = 3}
(gdb) continue
Continuing.
```

حال طبق breakpoint ای که گذاشتیم اجرای برنامه متوقف می شود. می بینیم که ورودی در buf قرار گرفته و متغیر e به 3 تغییر یافته که مکان بعد از آخرین حرف buf است .

این بار عبارت hello everyone را وارد می کنیم:

```
(gdb) print input
$2 = {buf = "hello everyone\n", '\000' <repeats 112 times>, r = 0, w = 15,
  e = 15, last = 15}
(gdb) continue
Continuing.
```

w باز هم به آخر buf رفته و e هم در ابتدای خط ورودی جدید است پس با w برابر است.

حال برنامه را continue کرده و سپس متوقف کنیم(ctrl + c):

```
(gdb) continue
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
mycpu () at proc.c:48
48          for (i = 0; i < ncpu; ++i) {
(gdb) print input
$4 = {buf = "hello everyone\n", '\000' <repeats 112 times>, r = 15, w = 15,
  e = 15, last = 15}
(gdb)
```

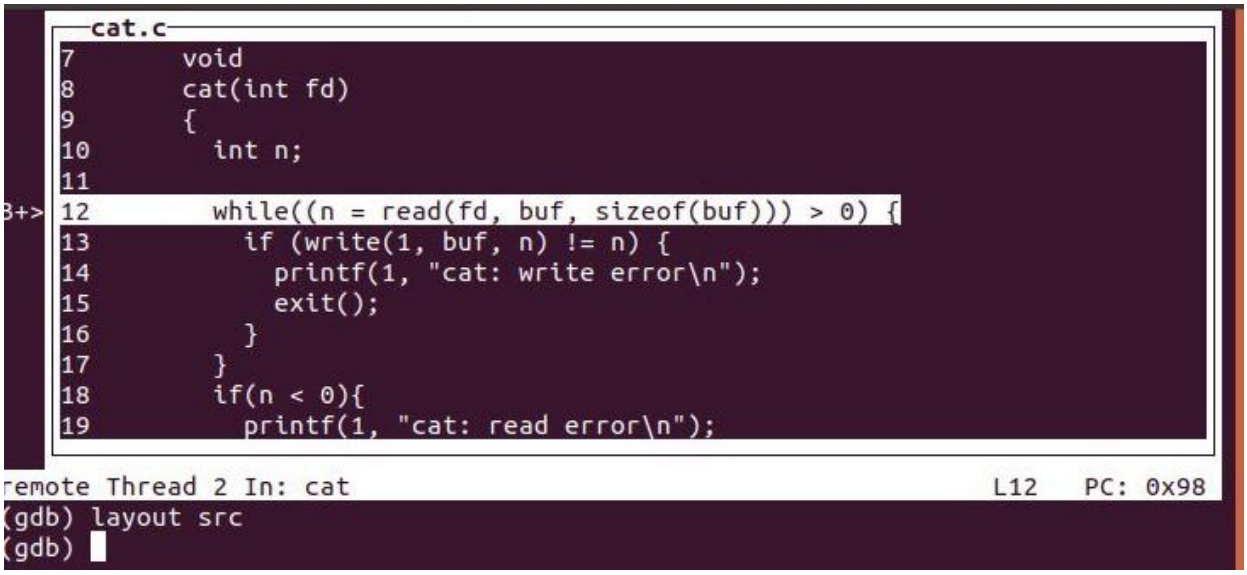
می بینیم که مقدار r به همان مقدار w رسیده است. یعنی از w قبلی (که 0 بود) شروع کرده و به w کنونی می رسد تا کل خط را بخواند(با گذاشتن یک watchpoint می توان دقیق تر بررسی کرد که r یکی یکی جلو می رود).

اشکال زدایی در سطح کد اسمبلی

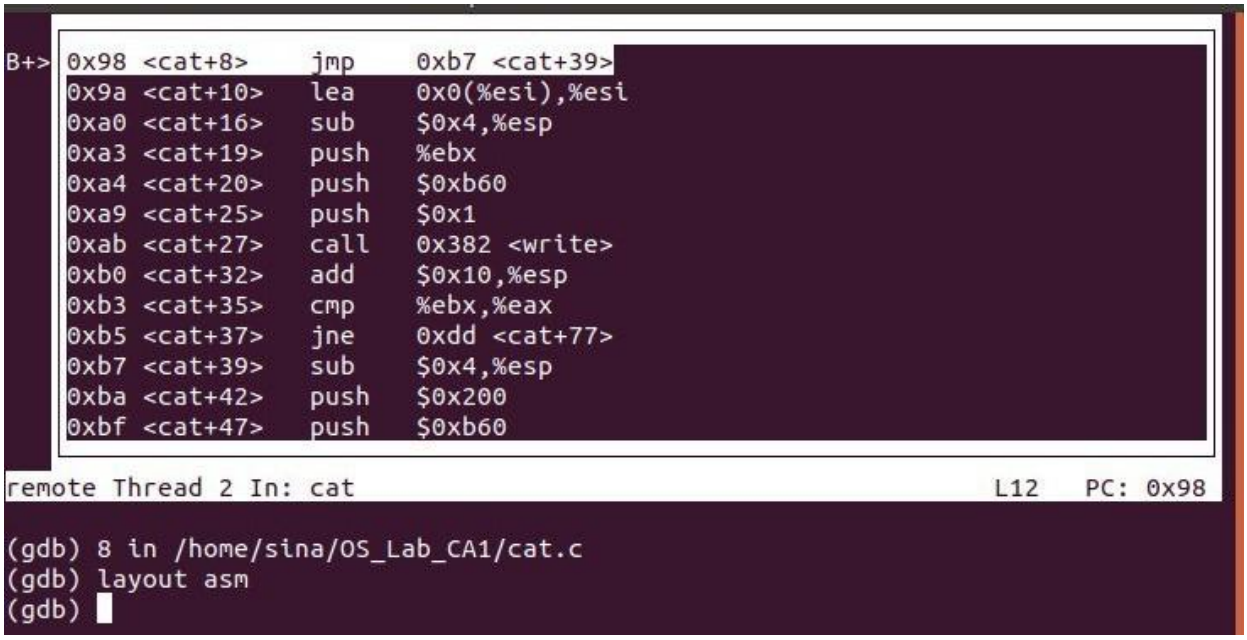
7-خروجی دستورهای layout src و layout asm در TUI:

برنامه مورد استفاده در این بخش cat.c (با قرار دادن breakpoint در خط 12 برنامه) است.

در TUI با استفاده از دستور layout src میتوان کد سورس در حال دیباگ را نمایش داد :



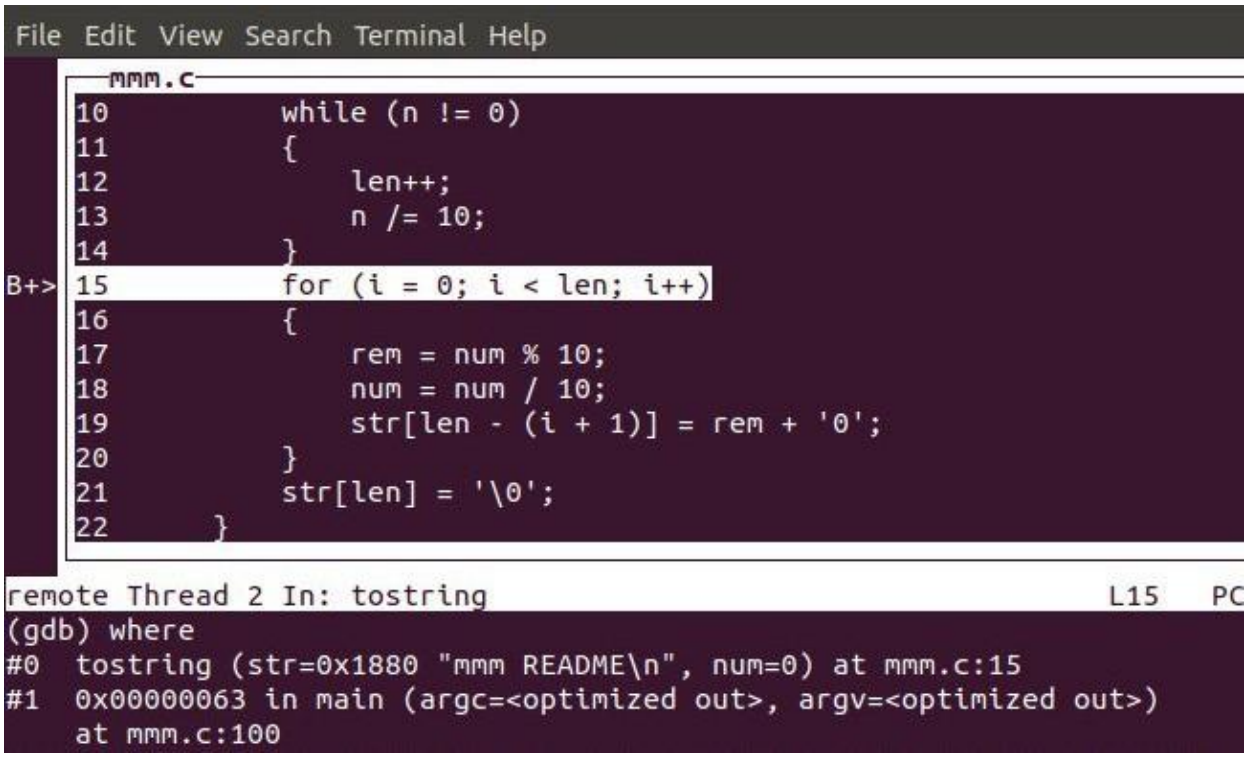
همچنین با استفاده از دستور layout asm می توانیم کد اسمبلی در حال دیباگ را مشاهده کنیم :



حال با استفاده از دستور layout split می توانیم کد سورس و اسمبلی را در کنار هم مشاهده کنیم :

8-دستورهای جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف):

در این بخش ابتدا دستور جابه جایی را برای فایل mmm.c بررسی می کنیم. بنابراین breakpoint را در خط 15 این فایل قرار می دهیم. با دستور where ، می توان call stack آن را مشاهده کرد:



برای حرکت در پشته فراخوانی می توان از دستورات up و down استفاده کرد. اگر دستور up و down به تنهایی مورد استفاده قرار بگیرند ، به اندازه یک واحد در stack call حرکت خواهیم کرد. برای دستور up خواهیم داشت:

```
mmm.c
95          {
96              printf(1,"%s \n","The limit on length of numbers has exceed
97              exit();
98          }
99          int buf[argc-1];
>100        for(int i=0; i<argc-1; i++)
101         {
102             buf[i] = atoi(argv[i+1]);
103         }
104         int fd = open("mmm_result.txt",O_CREATE | O_RDWR);
105         avrage(buf,argc-1,fd);
106         median(buf,argc-1,fd);
107         mode(buf,argc-1,fd);
remote Thread 2 In: main                                    L100  PC: 0x63
#0  tostring (str=0x1880 "mmm README\n", num=0) at mmm.c:15
#1  0x00000063 in main (argc=<optimized out>, argv=<optimized out>)
    at mmm.c:100
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) up
#1  0x00000063 in main (argc=<optimized out>, argv=<optimized out>)
    at mmm.c:100
(gdb)
```

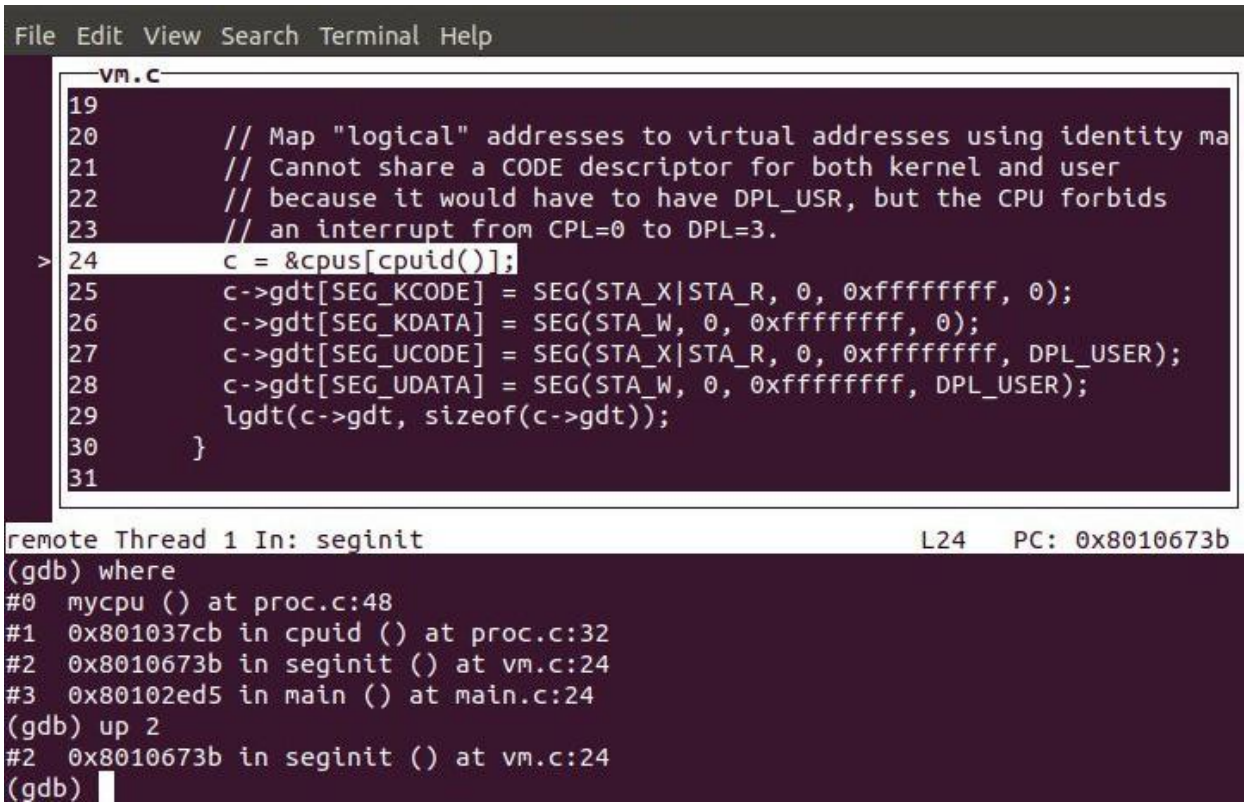
و برای دستور down خواهیم داشت:

```
File  Edit  View  Search  Terminal  Help
mmm.c
10          while (n != 0)
11          {
12              len++;
13              n /= 10;
14          }
B+>15       for (i = 0; i < len; i++)
16          {
17              rem = num % 10;
18              num = num / 10;
19              str[len - (i + 1)] = rem + '0';
20          }
21          str[len] = '\0';
22      }
remote Thread 2 In: tostring                                L15   PC: 0x134
    at mmm.c:100
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) up
#1  0x00000063 in main (argc=<optimized out>, argv=<optimized out>)
    at mmm.c:100
(gdb) down
#0  tostring (str=0x1880 "mmm README\n", num=0) at mmm.c:15
(gdb)
```

حال اگر بخواهیم به صورت چندتایی در stack call حرکت کنیم ، می توان از دستور up n و down n استفاده کرد (در اینجا n برابر تعداد گام هایی است که می خواهیم در call stack حرکت کنیم). برای بررسی این دستور ، به فایلی با فراخوانی توابع بیشتر احتیاج داریم. بنابراین فایل proc.c را مورد بررسی قرار داده و در خط 48 آن breakpoint قرار می دهیم . با دستور where ، می توان stack call این برنامه را مشاهده کرد:

```
File  Edit  View  Search  Terminal  Help
                proc.c
43            panic("mycpu called with interrupts enabled\n");
44
45          apicid = lapicid();
46          // APIC IDs are not guaranteed to be contiguous. Maybe we should
47          // a reverse map, or reserve a register to store &cpus[i].
B+> 48          for (i = 0; i < ncpu; ++i) {
49            if (cpus[i].apicid == apicid)
50              return &cpus[i];
51          }
52          panic("unknown apicid\n");
53        }
54
55        // Disable interrupts so that we are not rescheduled
remote Thread 1 In: mycpu                                  L48   PC: 0x80103751
(gdb) where
#0  mycpu () at proc.c:48
#1  0x801037cb in cpuid () at proc.c:32
#2  0x8010673b in seginit () at vm.c:24
#3  0x80102ed5 in main () at main.c:24
(gdb)
```

حال دستور up 2 را اجرا خواهیم کرد:



می توان دید که در call stack دو واحد به بالا حرکت کرده و در حال حاضر در seginit call قرار داریم.

❖ پیکر بندی و ساختن هسته لینوکس (امتیازی)

- در مرحله اول برای نوشتن نام اعضا در دستور dmesg یک فایل c ایجاد میکنیم (فایل group5-init.c) :

```c
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");

static int __init print_group_info(void) {
    printk(KERN_INFO "Group #5 :\n-Hamed Miramirkhani\n-Sina Tabsi\n-Fatemeh Mohammadi\n");
    return 0;
}
static void __exit outro(void) {
    printk(KERN_INFO "Goodbye :)\n");
}
module_init(print_group_info);
module_exit(outro);
```

- برای کامپایل فایل نوشته شده، Makefile زیر را ایجاد میکنیم:

```
obj-m += group5-init.o

all:
	make -C /lib/modules/$(shell uname -r)/build M="$(PWD)" modules
```

- دستورات زیر را به ترتیب اجرا میکنیم:

1. make

پس از اجرای این دستور فایل group5-init.ko ایجاد میشود.

2. sudo insmod group5-init.ko

2.5 lsmod

با استفاده از این دستور تنها می خواهیم مطمئن شویم که ماژول مورد نظرمان به هسته اضافه شده است یا خیر، همانطور که مشاهده می شود ماژول مورد نظر اضافه شده است:

```
f102m08@8:~$ lsmod
Module                  Size  Used by
btrfs                1630208  0
blake2b_generic        20480  0
xor                    24576  1 btrfs
raid6_pq              122880  1 btrfs
ufs                   110592  0
qnx4                   16384  0
hfsplus               118784  0
hfs                    65536  0
minix                  49152  0
ntfs                  126976  0
msdos                  20480  0
jfs                   233472  0
xfs                  1847296  0
libcrc32c              16384  2 btrfs,xfs
cpuid                  16384  0
group5_init            16384  0
usbhid                 65536  0
rfcomm                 86016  4
cmac                   16384  3
algif_hash             16384  1
algif_skcipher         16384  1
af_alg                 32768  6 algif_hash,algif_skcipher
bnep                   28672  2
nvidia_uvm           1400832  0
binfmt_misc            24576  1
nvidia_drm             73728  2
nvidia_modeset       1212416  2 nvidia_drm
nls_iso8859_1          16384  1
nvidia              56356864  82 nvidia_uvm,nvidia_modeset
```

3. sudo dmesg

در این قسمت خروجی مورد نظر را در ترمینال مشاهده میکنیم :

```
[ 1809.881951] group5_init: module verification failed: signature and/or required key missing - tainting kernel
[ 1809.882328] Group #5 :
               -Hamed Miramirkhani
               -Sina Tabsi
               -Fatemeh Mohammadi
f102m08@8:~/OS_LAB/OS_Lab_CA1/LUNIX-Kernel$ sudo dmesg >> out.txt
```

* خروجی در فایل out.txt ذخیره و نشان داده شده است.