- 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 Windows , Linux وMac 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

-------------------------------------------------------------------------------------------------------------------------------------------------

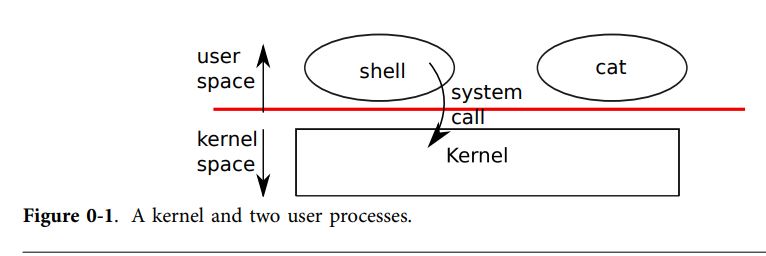
سوال 2) هر برنامه در حال اجرا یک "روند" نامیده میشود که خود دارای حافظه ای برای دستورات ، داده ها و پشته است. دستورات در واقع روند محاسبه برنامه را پیاده سازی می کند. داده ها نیز متغیر هایی هستند که محاسبات روی آنها یا توسط آنها در طول روند برنامه اجرا میشود و پشته نیز رویه فراخوانی های انجام شده در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



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

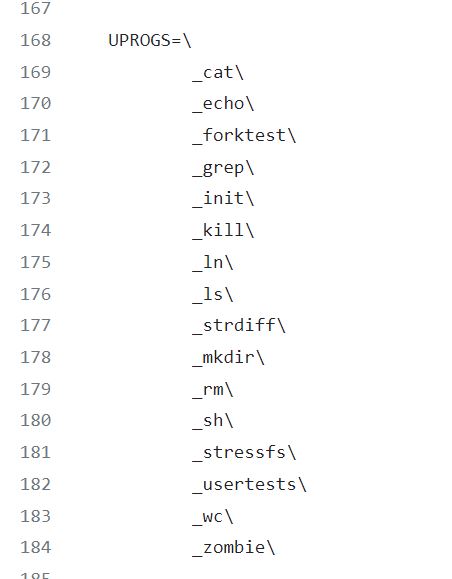
اما مزیت اینکه این دو تابع را مجزا داشته باشیم از منظر ورودی/خروجی قابل بیان است.

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 close() , open() 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 در UPROGS 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



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]](#footnote-1),[[2]](#footnote-2)SS[[3]](#footnote-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 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

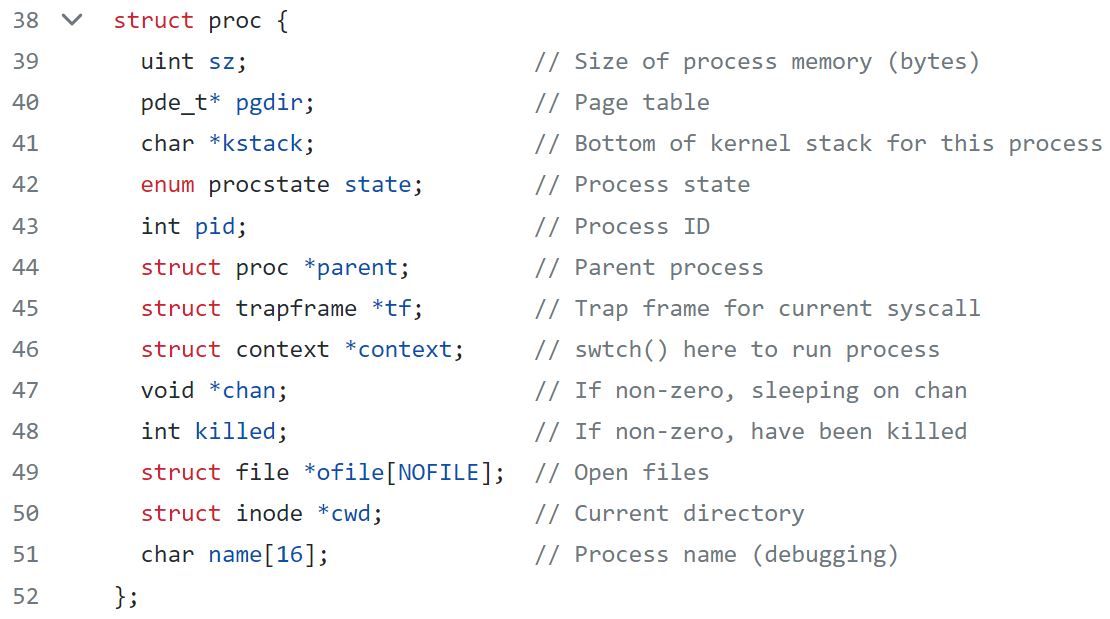
-------------------------------------------------------------------------------------------------------------------------------------------------

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]](#footnote-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

****

سوال 18) معادل کد entry.S که برای ورود هسته xv6 به حافظه است در سیستم عامل لینوکس و در سورس کد معماری آن فایلی با اسم entry\_32.S

می باشد که سورس آن در پوشه مربوط به گزارش آورده شده است

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

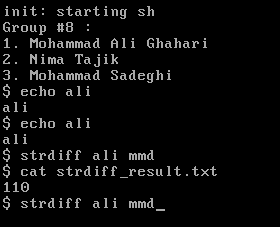
برخی توابع نیز بصورت اختصاصی در هسته اول اجرا میشوند مثل kinit1 , tvinit,kinit2,fileinit , …

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

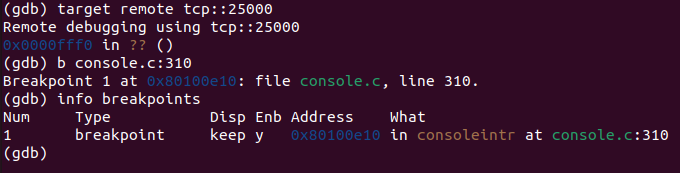
در ضمن همه پردازنده ها باید آدرس فیزیکی page table که توسط هسته اول تولید میشود را داشته باشند برای همین تابع switchkvm مشترک است

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

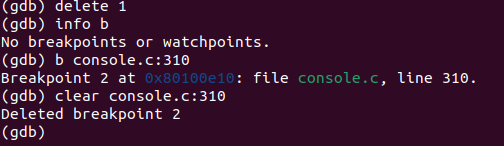
برنامه سطح کاربر:



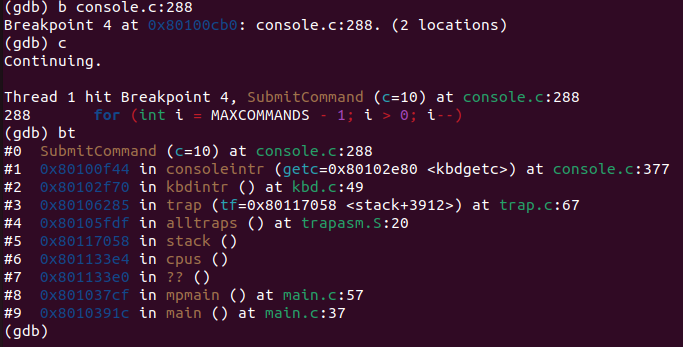
پاسخ سوالات اشکال زدایی  
سوال 1) برای مشاهده breakpoint ها از دستور info breakpoints استفاده می شود.



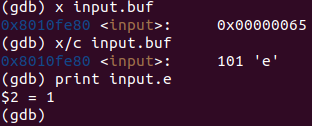
سوال 2) برای حذف کردن breakpoint ها می توان از دستور clear ومحل آنها استفاده کرد. اگر ورودی به آن داده نشود تمام breakpoint ها حذف میشود اما میتوان مانند ست کردن breakpoint ها با همان فرمت breakpoint هارا با دستور clear حذف کرد. همچنین می توان با استفاده از دستور delete با استفاده از شماره breakpoint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



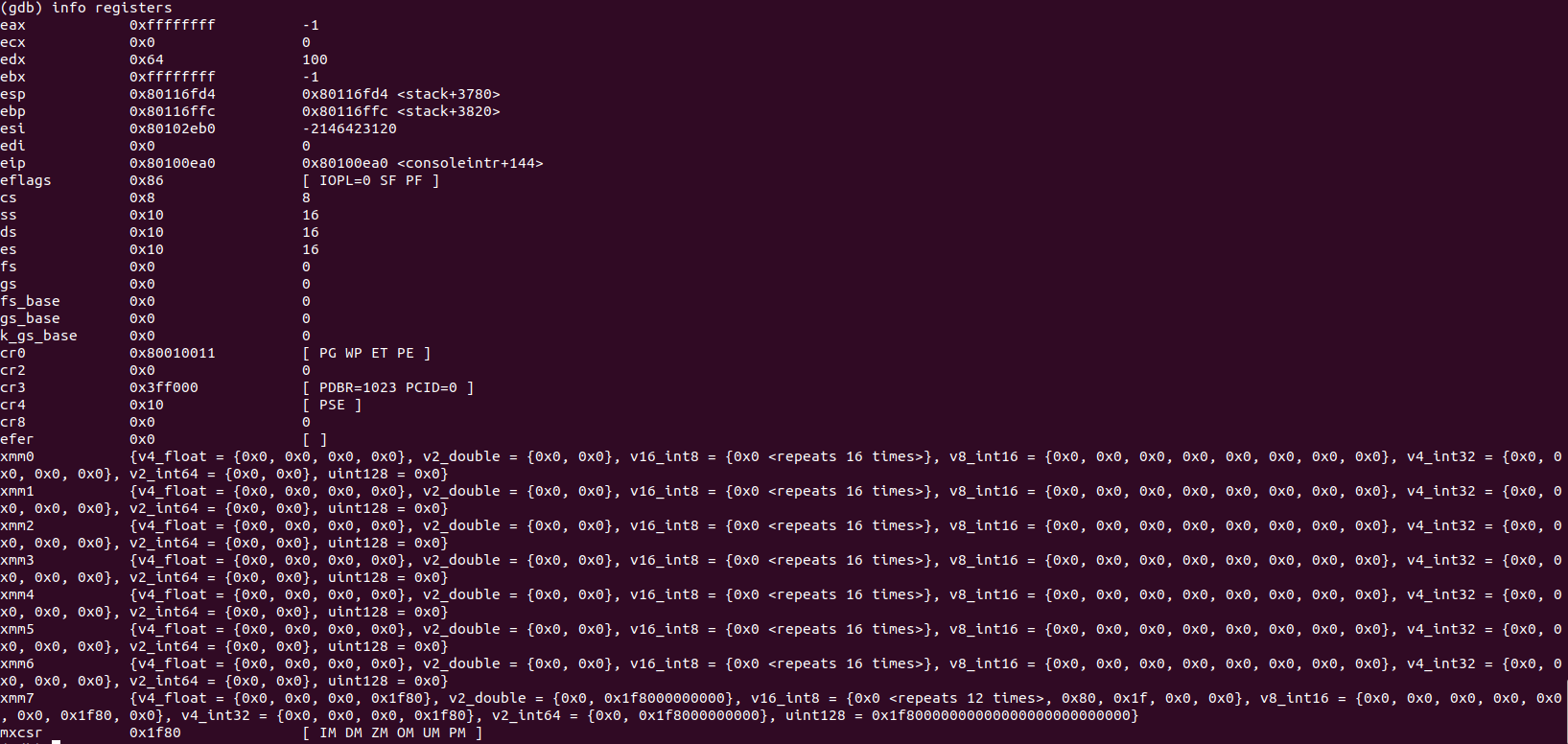
سوال 3) دستور bt به ما stack trace را نشان می دهد. یعنی در تابعی که قرار داریم چهfunction call هایی انجام شده تا به این تابع رسیده ایم.

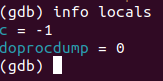


سوال 4) با استفاده از دستور x می توانیم محتوای جایی که پوینتر به آن اشاره می کند را ببینیم مثلا با دستور x/2c POINTER محتوای دو خانه ابتدایی آن را ببینیم اما با دستور print میتوانیم مقادیر متغیر ها یا رجیستر هارا ببینیم.



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.





سوال 6) این struct در فایل console.c و در خط 182 قرار دارد. اینstruct 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

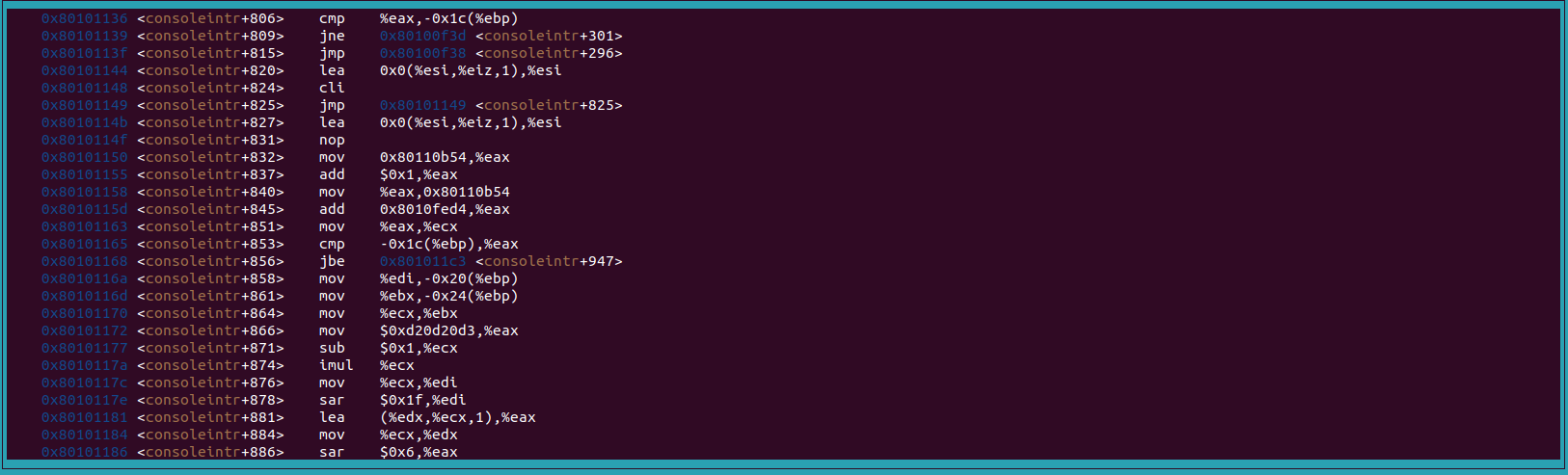
uint e; // Edit index

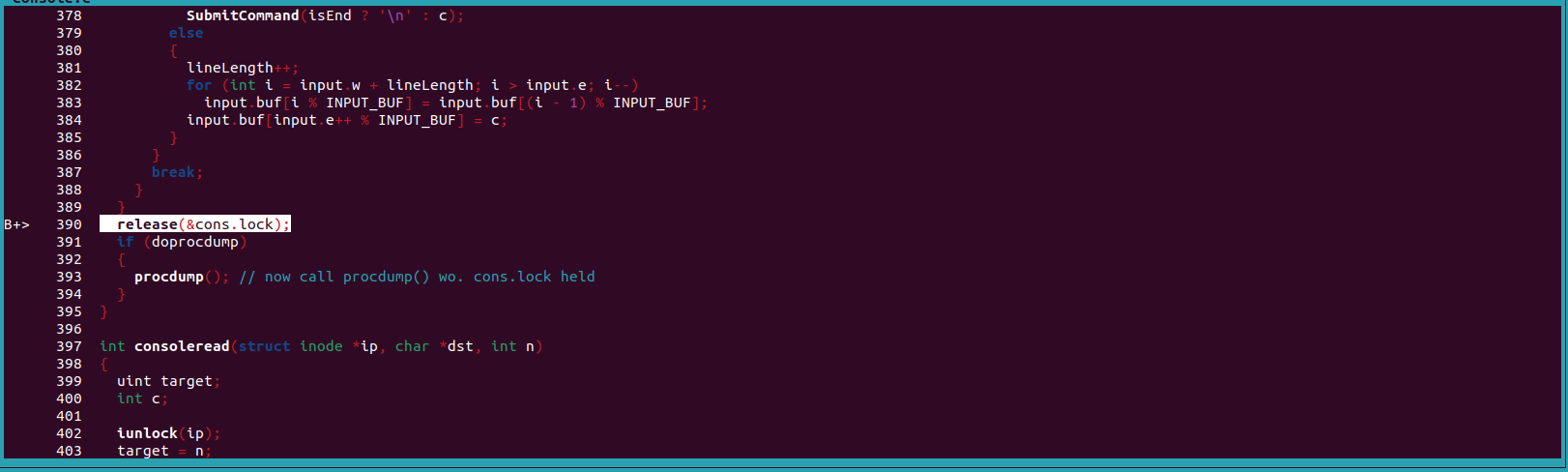
} input;

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

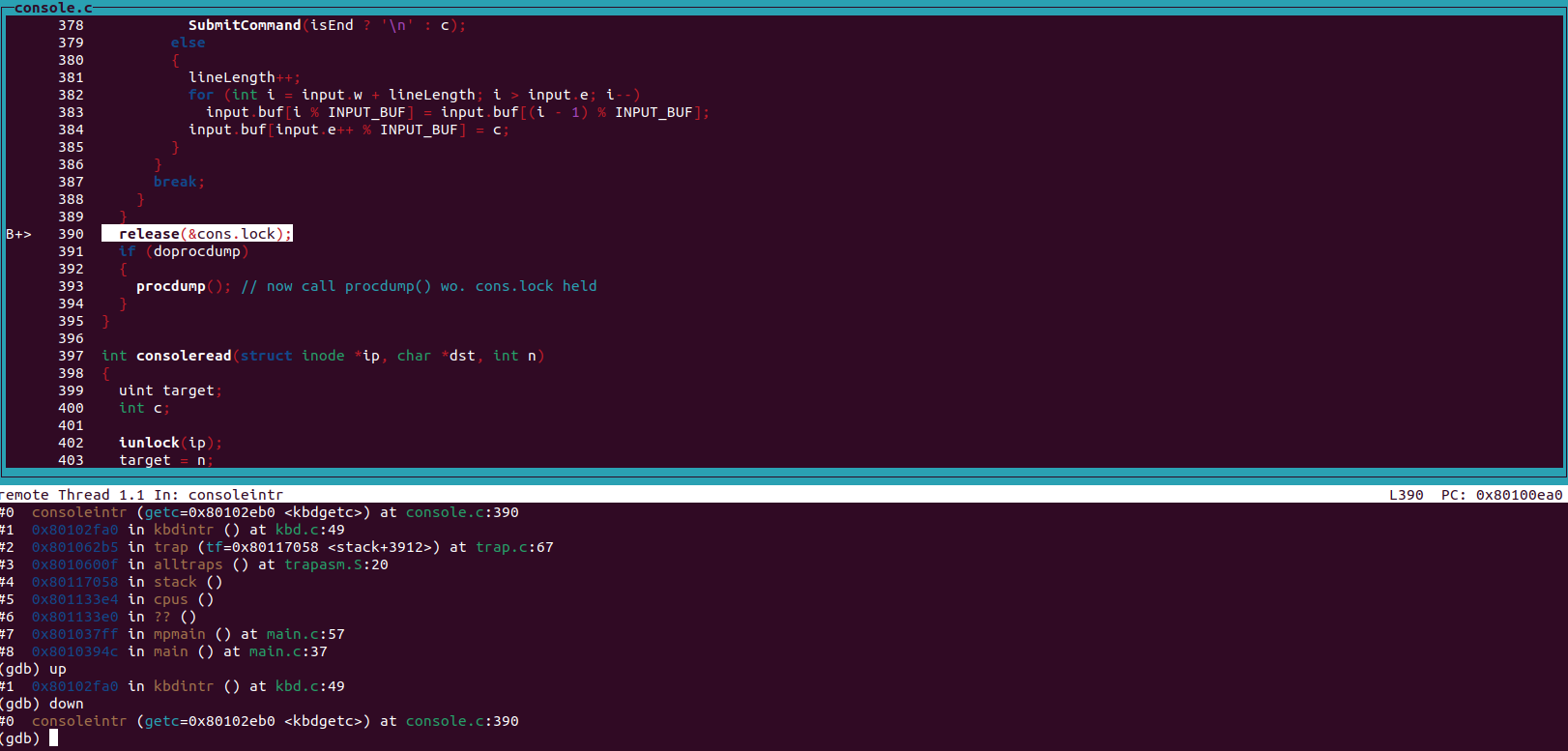
ابتدای command وارد شده توسط کاربر اشاره میکند و تا رسیدن به w از بافر میخواند . W نشان دهنده ایندکسی است که در حین تایپ کردن و ادیت کردن کاربر همراه با r به ابتدای ورودی اشاره میکند و پس از submit کردن به e می رود. E نشان دهنده ایندکسی در بافر است که قرار است ادیت شود یا تغییر داده شود که پس از سابمیت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

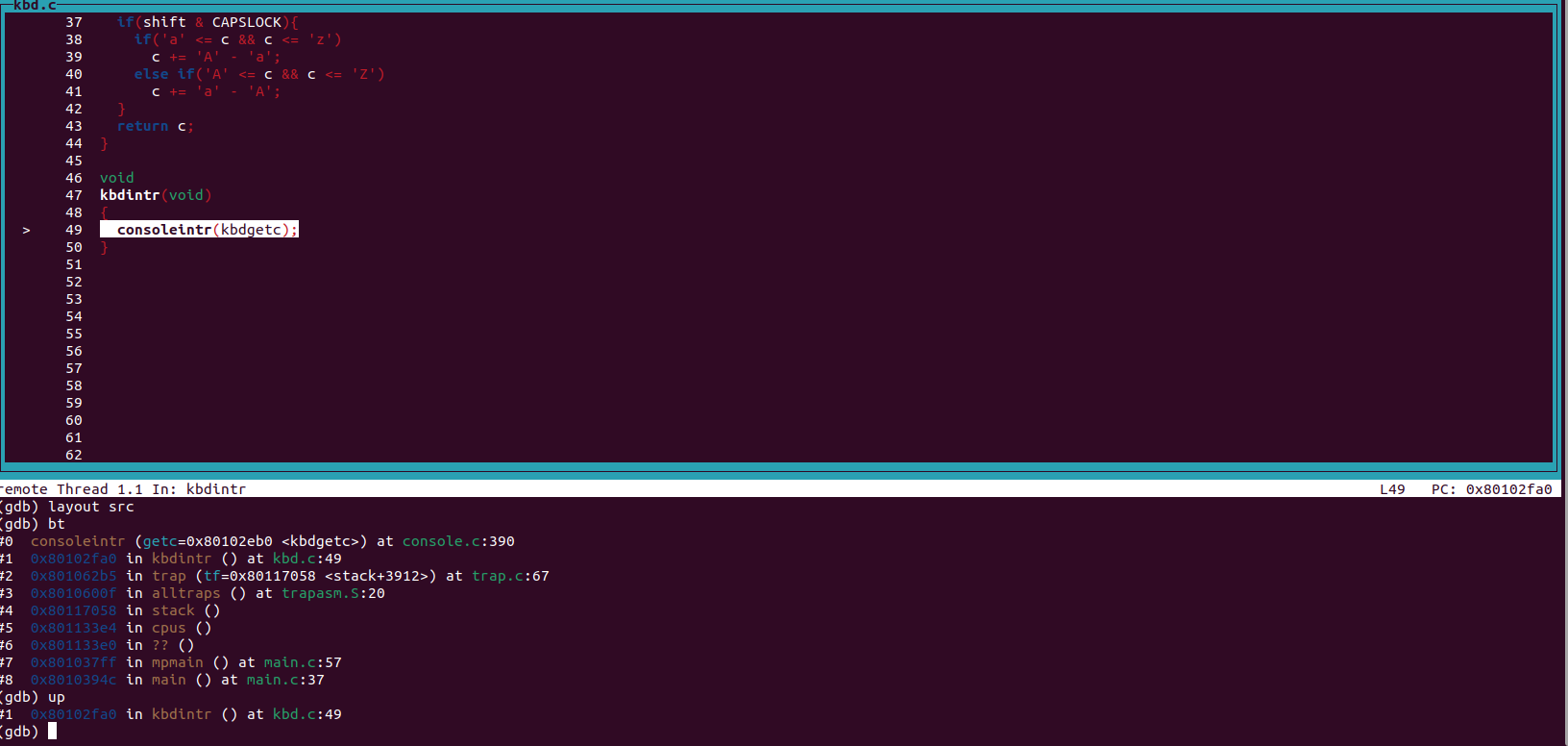
سوال 7) دو دستور layout src, layout asm برای تغییر نمایش کد استفاده می شود به این صورت که دستور layout src کد برنامه را نمایش می دهد و layour asm کد زبان اسمبلی آن را نمایش میدهد و از آنجا امکان دیباگ کردن فراهم می شود.





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)

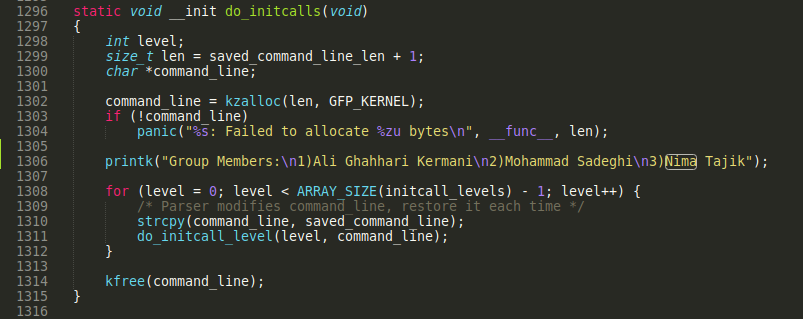


√

---------------------------------------------------------------------------------------------------------------------------------------------------------------

بخش optional

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.



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 محیطqemu اجرا می کنیم.

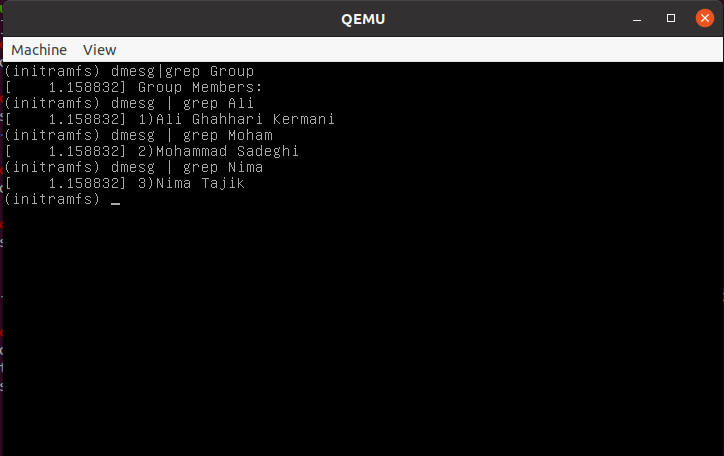
make -j `nproc` 4 bzImage

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 dmesgقابل مشاهده است



1. Extra segment [↑](#footnote-ref-1)
2. Stack segment [↑](#footnote-ref-2)
3. Code segment [↑](#footnote-ref-3)
4. Page directory entry [↑](#footnote-ref-4)