گزارش‌کار

پویا نقوی - روزبه بستان‌دوست - صادق حایری

گام اول: آشنایی با مجازی‌سازی، نصب و اشکال‌زدایی

برای شروع روی یه گنو/لینوکس توزیع Ubuntu ۱۶.۰۴ LTS کار رو شروع کردیم.
اول سورس لینوکس رو از گیت‌هاب امام تورووالز دانلود می‌کنیم (ورژنی که استفاده کردیم ۴.۹ هست)
https://github.com/torvalds/linux

واسه کامپایل کردن اول نیاز به یه فایل config داریم که واسه اینکه سریع و راحت بسازیمش میریم توی پوشه پروژه و دستور

```
make menuconfig
```

رو اجرا می‌کنیم که واسمون یه برنامه‌ی کوچولو میاره بالا که می‌تونیم چیزایی که نیاز داریم رو توش بگیم می‌خوایم یا نه.
کانفیگ‌های دیفالتش رو می‌ذاریم باشه و همونجوری سیو می‌کنیم و ازش میایم بیرون.
برای اینکه symbolفایل هم بسازه کنار کامپایل کردن، می‌ریم توی کانفیک‌فایل و فلگ‌های

```
CONFIG_DEBUG_INFO_REDUCED
CONFIG_GDB_SCRIPTS
```

رو هم فعال می‌کنیم.
https://01.org/linuxgraphics/gfx-docs/drm/dev-tools/gdb-kernel-debugging.html

قبل از شروع به کار هم باید چندتا کتاب‌خونه نصب کنیم که توی کامپایل ازشون استفاده می‌شه که با دستور زیر می‌شه نصبشون کرد:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

چون لینوکس ورژن ۴ هست از یه کتاب‌خونه‌های اضافی هم نیاز داره که با دستورهای زیر نصب می‌شه:

```
sudo apt-get install python-pip python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev zlib1g-dev
```

برای اینکه کارهامون سریع‌تر پیش بره هم ccache رو هم نصب می‌کنیم:

```
sudo apt-get install ccache
```

برای شروع به کامپایل کردن دستور زیر رو می‌زنیم:

```
ccache Make -j 8 bzImage
```

که فلگ j واسه اینه که تعداد هسته‌هایی که واسه کامپایل کردن درگیر می‌شن رو مشخص کنیم که هر ۸تا هسته رو درگیر می‌کنیم.

ارگیومنت bzImage هم واسه اینه که بگیم می‌خوایم فایل رو کامپایل کنیم فقط.
بعد از چند دقیقه کامپایل تموم می‌شه و می‌تونیم فایل کامپایل شده رو توی پوشه arch/x۸۶/boot/ پیدا کنیم.

برای اینکه کرنل رو اجرا کنیم و دیباگ کنیم اول qemu, gdb رو نصب می‌کنیم.

```
sudo apt-get install qemu gdb
```

برای اجرای کرنل روی qemu نیاز داریم که یه فایل‌سیستم اولیه براش ایجاد کنیم: (نمی‌دونم چیه و چرا و اصن درسته یا نه):

```
mkinitramfs -o FS
```

حالا می‌تونیم با زدن دستور زیر کرنل رو توی qemu بالا بیاریم:

```
qemu-system-x86_64 -S -s -kernel bzImage -initrd FS -m 1GB
```

با فلگ کرنل فایل کرنلی که درست شده رو انتخاب می‌کنیم (توی پوشه arch/x۸۶/boot)
با فلگ initrd همون فایل‌سیستمی که با دستوری قبلی ساختیم رو مشخص می‌کنیم که اگه اینو نزنیم پنیک می‌کنه کرنلمون.
با فلگ m هم مقدار رمی که می‌دیم بهش که می‌تونیم کلا چیزی نگیم.
با فلگ S میایم qemu رو توی حالت stoping بالا میاریم و چیزی انجام نمی‌ده تا براش سیگنال continue رو با gdb بفرستیم.
با فلگ s می‌گیم که telnetاش رو راه بندازه که بتونیم با gdb بهش وصل بشیم.

حالا gdb رو میاریم بالا:

```
gdb ./vmlinux
```

که اون فایل vmlinux همون symbolفایل‌هایی هست که توی کامپایل ساخته واسمون و می‌تونه به کد برسه بهش.

برای وصل شدن به qemu دستور زیر رو توی gdb می‌زنیم:

```
target remote localhost:1234 (gdb)
```

حالا می‌تونیم با زدن دستور win کد رو ببینیم و با دستور b توی هر فایلی خواستیم breakpoint بذاریم و با n خط به خط بریم جلو و با c برسیم به بریک‌پوینت و با زدن دستور show یا watch مقدار متغیرها رو ببنیم.

البته وقتی با این روش وصل بشیم کدها رو نمی‌تونه لود کنه که برای حل کردنش باید از تیکه کد زیر استفاده کنیم!

```
\ gdb
ex "add-auto-load-safe-path $-      حالت امنیت جی‌دی‌بی روغیر فعال می‌کنه برای این که بتونه همه‌ی فایل‌ها رو باز کنه
\ "(pwd)
\ "ex "file vmlinux-      سیمبول‌فایل که توی کامپایل کردن کرنل ساخته شده رو لود می‌کنیم
\ 'ex 'set arch i386:x86-64:intel-      آرچیتکتر رو عوض میکنیم که بتونه بشناسه ولی باگ می‌خوره :/
\ 'ex 'target remote localhost:1234-      وصل می‌شیم
\ 'ex 'break start_kernel-      بریک‌پوینت می‌ذاریم
\ 'ex 'continue-      شروع می‌کنیم کرنل رو ولی باگ داره و ارور می‌ده
\ 'ex 'disconnect-      برای رفع ارور قبلی یبار قطع و وصل باید بشیم
\ 'ex 'set arch i386:x86-64-      آرچیتکتر رو عوض می‌کنیم چون هربار وصل شدن ریسیت می‌شه
'ex 'target remote localhost:1234-      وصل می‌شیم و اینبار درست می‌شه!
```

ولی جدیدترین نسخه clion که هنوز رلیز نهایی نشده هم می‌شه استفاده کرد که آسون‌تره.

گام دوم: آشنایی

Gendisk

این استراکت اطلاعات دیسک دیوایس ها رو برای کرنل نشان می دهد. که نیازی نیست نویسنده های درایورها از آن اطلاعی داشته باشند.

major : شماره درایوری که آن دیوایس روی آن است.

Minor: شماره دیوایس بین تمام دیوایس های روی یک درایور. اگه یه درایور بیش از یک دیوایس داشته باشه minor های اون هم به همون تعداد زیاد میشن.

مثلا به یه درایور هم موس هم کیبورد وصله، این جوری major هر دو ۱ میشه ولی minor یکی ۱ و دیگری ۲ میشه.

Disk_name: اسم اون دیسک دیوایس رو نگه میداره. که اینجا میشه دید /proc/partitions

truct block_device_operations *fops: دیوایس ها عملیات هاشونو با یه فایل معرفی میکنند.

struct request_queue *queue;: برای مدیریت کردن ورودی و خروجی به این دیوایس استفاده میشه.

int flags: استیت اون درایور رو نگه میداره. مثلا اینکه یه چیزی مثل CD که خارج شونده است و یا خیر.

sector_t capacity;: ظرفیت اون درایور رو بیان می کنه تو یه سکتور ۵۱۲ بایتی.

void *private_data;: درایورها از این پوینتر استفاده میکنن برای دسترسی به اطلاعات داخلی خودشون.

unsigned int events; تعداد ایونت هایی که میتونه انجام بده.

گام سوم: بررسی کد منبع

مرحله نخست: بوت شدن سیستم عامل

Bios.۱: وقتی سیستم را روشن می کنیم،CPU در ROM دنبال دستور بعدی می گردد و در ROM تابع JUMP ی وجود دارد(در قالب یکسری از دستورات) که به CPU می گوید که BIOS را بالا بیاورد.BIOS لیست تمامی دستگاه هایی که قابلیت Boot شدن دارند را مشخص می کند و در یکی از آنها مانند HardDisk یا USB Flash، Boot Loader را پیدا کرده و در حافظه بارگذاری می کند و هنگامی که همه این مراحل انجام شد و BootLoader بارگذاری شد،BIOS کنترل را به دست آن(MBR BootLoader) می دهد.

MBR.۲:در اولین سکتور Bootable Disk قرار دارد(معمولا dev/hda/ یا dev/sda/) و اندازه آن کمتر از ۵۱۲ بایت است و دارای ۳ بخش است:۱)اطلاعات اصلی boot loader که در ۴۴۶ بایت اول قرار دارد. ۲)۶۴ بایت بعدی که به partition table اختصاص دارد. ۳) و در ۲ بایت بعدی mbr validation check قرار دارد. و در کل MBR اطلاعاتی از GRUB را دارد و به زبان ساده MBR بارگذاری می شود و GRUB BootLoader را بارگذاری می کند.

GRUB.۳:در این مرحله kernel در ۳ مرحله بارگذاری می شود:

BootLoader:۱ Grub Stage مقدماتی که در MBR هست فضایی کمتر از ۵۱۲ Byte را روی دیسک می گیرد بنابراین فضای کوچکی برای شامل شدن دستورات پیچیده سیستم عامل است. در عوض BootLoader مقدماتی توابعی را اجرا می کند که مرحله ۱.۵ یا ۲ BootLoader را بارگذاری می کند.

Grub Stage ۱.۵:مرحله ۱ Grub Stage به صورت مستقیم می تواند ۲ Grub Stage را بارگذاری کند ولی به صورت عادی ۱.۵ Grub Stage بارگذاری میکند.این زمانی اتفاق می افتد که قسمت /boot فراتر از ۱۰۲۴ cylender head در هارد درایو قرار داشته باشد.۱.۵ GrubStage در اولین ۳۰ کیلو بایت بعد از هارد دیسک و قبل از اولین پارتیشن قرار دارد و این حافظه برای ذخیره file system drivers و modules درنظر گرفته شده است و مرحله ۱.۵،مرحله ۲ را از هر جایی در file system مانند //boot grub بارگذاری کند.

Grub Stage ۲:در این مرحله Kernel و هر ماژول مورد نیاز از /boot/grub/grub.conf بارگذاری می شود و یک رابط گرافیکی به مانندSplash Image واقع در /grub/splash.xpm.gz با لیستی از kernal های در دسترس که میتوانیم انتخاب کنیم.

۴.Kernel:کرنل هسته اصلی سیستم عامل است که تمام پردازش های سیستم را در دست دارد و طی مراحل زیر بارگذاری می شود.

۱)kernel به محض اینکه بارگذاری شد،سخت افزار و حافظه ای که به سیستم اختصاص داده شده است را تنظیم(config) می کند.
۲)سپس initrd را از حالت فشرده در می آورد و آن را بارگذاری می کند و همچنین تمام درایور های مورد نیاز را نیز بارگذاری می کند.
۳)ماژول های kernel با کمک برنامه هایی همچون insmod و rmmod که در initrdimage قرار دارند بارگذاری و unload می شود.
۴)توجه می کند که نوع هارددیسک LVM یا RAID باشد.
۵)initrdimage را unmount می کند و قسمت های حافظه را که توسط disk image اشغال شده است را آزاد می کند.
۶)kernel قسمت root را که در فایل grub.conf به صورت read-only مشخص شده است را mount می کند.
۷)init process را اجرا می کند.

۵.Init Process:به فایل /etc/inittab می رود تا درباره ی Linux Run Level تصمیم گیری کند و موارد زیر Linux Run Level های در دسترس هستند:

۰-halt
۱-Single User Mode
۲-Multi User,without NTFS
۳-Full Multiuser Mode
۴-Unused
۵-X11
۶-Reboot

۶.Runlevel Programs:هنگامی که Linux System در حال بالا آمدن است ما سرویس های منتوعی را می بینیم که در حال اجرا شدن است و این ها Runlevel Program هایی هستند که از Run level directory با توجه به Run Level ی که مشخص کرده ایم اجرا می شود.دایرکتوری هایی که از آن اجرا می کند با توجه به Run Level متفاوت است و در زیر لیست آن ها آمده است:

```
/Run level 0 — /etc/rc.d/rc0.d
/Run level 1 — /etc/rc.d/rc1.d
/Run level 2 — /etc/rc.d/rc2.d
/Run level 3 — /etc/rc.d/rc3.d
/Run level 4 — /etc/rc.d/rc4.d
/Run level 5 — /etc/rc.d/rc5.d
/Run level 6 — /etc/rc.d/rc6.d
```

-۱

دلیل اول: چون در شروع کار نیاز هست که ریجسترهای cpu مقداردهی شوند ولی با زبان c به آن‌ها دسترسی نداریم و نیاز هست که به صورت دستی مقداردهی شوند.
دلیل دوم: این قسمت از کد باید بدون نیاز به boot loader اجرا شود (البته در bootهای جدید دیگر این فایل استفاده نمی‌شود)
دلیل سوم: به دلیل اینکه به زبان اسمبلی نوشته شده است خیلی بهینه‌تر و سریع‌تر از کدی که با c نوشته شده باشد می‌تواند عمل کنم.
در انتهای کد به حالت protected mode می‌رود و به تابع main اصلی c می‌رود.

-۲

در ابتدا تابع main ما در حالت real mode قرار داریم. با تعدادی دستور مقداردهی های اولی را انجام می دهیم. مثلا با دستور console_init کنسول را مقداردهی اولیه می کنیم. یا توسط دستور detect_memory مموری را شناسایی می کنیم. یا توسط دستور validate_cpu اطمینان پیدا می کنیم که CPU را به درستی در اختیار داریم یا خیر. سپس در انتهای تابع main از حالت real mode خارج شده و وارد protected mode می شویم. در حالت real mode دسترسی به تمام اجزا بدون محافظت انجام پذیر است اما با رفتن به مد protected دسترسی ها به حافظه و... را محدود می کنیم.

-۳

چون این تابع در فولدر arch می باشد بنابراین وابسته به معماری سیستم می باشد(مانند x۸۶) و طبعا دستورات آن در معماری های مختلف،متفاوت می باشد در نتیجه این تابع در معماری های مختلف،متفاوت می باشد.
در این آدرس‌ها تابع صدا زده می‌شود.

linux/arch/sparc/kernel/setup_32.c
linux/arch/um/kernel/skas/process.c
linux/arch/parisc/kernel/setup.c
linux/arch/x86/kernel/head32.c
linux/arch/x86/kernel/head64.c

این تابع مقدار بازگشتی ندارد (void)

وظایف start_kernel:

۱) فعال کردن lock validator می باشد که وظیفه lock validator این است که به عنوان مثال در Symmetric Multiprocessing Systems
وقتی یک منبع در چند جا باید استفاده شود یا اینکه ۲ منبع(که ترتیب استفاده آنها مهم است) باید در چند جا استفاده شوند،یک مدیریت باید انجام شود که این مدیریت با فعال شدن lock validator تحقق می یابد.

۲) شناسه پردازنده را مقداردهی اولیه می کند.
۳) زیرسیستم های control groups(cgroups) اولیه را فعال می کند.control groups ها قابلیتی از linux kernel هستند که استفاده از منابعی همچون CPU,Memory,Network را مدیریت و مجزاسازی می کند.
۴) مقدار دهی اولیه cache های مختلف در لایه vfs(Virtual File System) که یک interface یکتا برای kernel فراهم می آورد تا با درخواست های مختلف I/O دست و پنجه نرم کند.
۵) مقدار دهی اولیه Memory Manager
و وظایف مختلف دیگر همچون مقدار دهی اولیه RCU,vmalloc,scheduler,IRQs,ACPI و ... می توان اشاره کرد.

دسترسی به صورت پروتکتد مود است(چون گفتیم که در انتهای تابع main به حالت پروتکتد میرویم و تا انتها در این مود میمانیم)

-۴

اگه اول این کار را نکنیم موقع بوت شدن نمی‌تونیم لاگ سیستم را بدهیم و همچنین اگه مشکلی پیش بیاید نمی‌توانیم کاری بکنیم.

مرحله دوم: فراخوان سیستمی

ابتدا برنامه سیستم‌کالی که نیاز دارد اجرا شود را در ریجستر eax ذخیره می‌کند تا سیستم‌عامل بتواند آن را پیدا کند، حال برنامه اینتراپت می‌دهد.
سیستم‌عامل مدیریت سیستم را به دست می‌گیرد، در ابتدا اینتراپت را خاموش می‌کند، با دستور save_all تمام ریسجترهای موجود را در استک سیستم‌عامل ذخیره می‌کند که بتواند به جای قبلی برگردد، سپس ریجستر eax را می‌خواند تا بفهمد کدام سیستم‌کال درخواست شده است، سپس در جدول مربوط به سیستم‌کال‌ها آن را پیدا می‌کند و اگر نبود اکسپشن syscall_badsys می‌دهد (با اجرای قطعه کد آن) و اگر توانست دستور را پیدا کند آن‌را اجرا می‌کند و سپس جواب را در ریجسترهای مشخص (نمی‌دانم کجا؟) ذخیره می‌کند و اینتراپت را دوباره روشن می‌کند و بعد ریجسترهایی که در استک ریخته بود به جای خود باز می‌گرداند و برنامه می‌تواند خروجی را ببیند و به کار خود ادامه دهد.

مرحله سوم: فایل desctiptors

سوال ۱: در حالت کلی در پروسس‌های ما جدولی وجود دارد که هر خانه از آن را file descriptor می‌نامند، که هر خانه از آن به یک فایل باز شده اشاره می‌کند و این به این معنی است که ما با هر فایلی کار داریم، به سراغ این جدول می رویم و با پیدا کردن آن فایل با آن کار می کنیم. مثلا فایل read کردن از دیوایس کیبورد (دیوایس‌های ورودی و خروجی مثل کیبورد یا سوکت نتورک هم یک فایل به حساب می‌آیند). فایل‌ها می توانند به یک inode یا یک pipe یا یک فایل خالی(null) اشاره کنن.

سوال ۲: پیاده کردن فرایند پایپ، به این صورت است در خانه دوم برنامه اول (ورودی پایپ) بجای stdout به یک حافظه بافر اشاره می‌کند و در خانه اول برنامه دوم (ورودی پایپ) بجای stdin به همان حافظه بافر اشاره می‌کند، حال وقتی برنامه اول اجرا شود به خانه شماره ۱ می‌رود و دستور write می‌دهد که بجای نوشته شدن در خروجی در بافر ذخیره می‌شود و برنامه دوم دستور read را می‌دهد که بجای خواندن از ورودی stdin، به بافر می‌رود و دیتای آن را می‌خواند.
به این صورت دیتا از خروجی برنامه اول به ورودی برنامه دوم انتقال داده می‌شود.

سوال۳: ایجاد سوکت برای استفاده در جای مورد نیاز، یا ایجاد فضای خالی مثل dev/null برای دور ریختن خروجی.

سوال ۴: زمانی که می‌خواهیم یک فایل را باز کنیم، یک پروسس از برنامه ما اجرا می‌شود، که آن پروسس یک(یا چند) جدول از file descriptorsها دارد، که برای باز کردن آن فایل یک ردیف به جدول فایل‌های خود اضافه می‌کند که اشاره‌گر آن ردیف به یک فایل (file struct) اشاره می‌کند، که آن فایل یک اشاره‌گر به inode دارد که همان فایل ما است.

گام چهارم: حالتهای مختلف سیستم

Kernel Mode:در حالت kernel،کد در حال اجرا دسترسی نامحدود و کامل به سخت افزار را دارد و می تواند هر CPU Instruction را اجرا کند و به هر فضایی از حافظه دسترسی دارد.kernel mode به طور معمول برای lowest-level و مورد اعتماد ترین توابع سیستم عامل رزرو شده است.لازم به ذکر است که خرابی ها در kernel mode فاجعه بار هستند و باعث توقف کل سیستم می شوند.

User Mode:در User Mode،کد در حال اجرا دسترسی مستقیم به سخت افزار و حافظه را ندارد و کد های در حال اجرا برای دسترسی به سخت افزار یا حافظه باید به System API ها درخواست بدهند و به خاطر این فرآیند ایزوله سازی،خرابی های که در User Mode به وجود می آیند همیشه قابل بازیابی هستند.

Protected Mode:یک نوع از program operation درکامپیوتر های با ریزپردازنده Intel می باشد که برنامه فقط می تواند در یک فضای پیوسته ۶۴۰ کیلوبایتی ادرس دهی کند. به عنوان مثال در ریز پردازنده اصلی intel(۸۰۸۸)،مطمئن می شود که برنامه های که در حال اجرا در حالت protected می باشد توانایی ادرس دادن و دسترسی به غیر از همان فضای پیوسته ۶۴۰ کیلوبایتی را ندارد.به طور معمول اکثر کد های سیستم عامل و همه ی application program ها در حالت protected اجرا می شوند تا سیستم عامل مطمئن شود که داده ضروری سهوا overwrite نشده باشد.

Real Mode:یک نوع از program operation می باشد که یک instruction میتواند هر فضایی را در حدود ۱MB در RAM را ادرس دهی کند.معمولا برنامه ای در Real Mode اجرا می شود که نیاز به استفاده با به روز رسانی اطلاعات سیستم دارد و ازلحاظ اینکه چگونه این کار را بلد است انجام بدهد،قابل اعتماد می باشد و چنین برنامه ای معمولا جزئی از سیستم عامل یا یک application subsystem به خصوص است.

Driver:درایور یک رابط یک شکل برای دستگاه به سایر قسمت های سیستم عامل می باشد.درایور یک قابلیت نرم افزاری است که به سیستم عامل این قابلیت را می دهد که با یک سخت افزار مخصوص رابطه برقرار کند.بخش های مختلف کامپیوتر نیاز به درایور دارند چرا که از command های استاندارد استفاده نمی کنند و درایور ها به سیستم عامل وسایر برنامه های کامپیوتری این قابلیت را می دهند که به توابع سخت افزاری بدون اینکه لازم باشد جزئیات دقیق سخت افزار مورد استفاده را بدانند،دسترسی داشته باشند.به طور مثال برای آغاز عملیات I/O،درایور،رجیستر های مناسبی را در کنترلر دستگاه load می کند.کنترلر نیز به نوبت این رجیستر ها را برای تعیین نوع عملی که باید انجام بدهد بررسی می کند و انتقال داده ها را از دستگاه به بافر محلی خود شروع می کند و هنگامی که انتقال تمام شد،کنترلر یک وقفه به درایور ارسال می کند و درایور دستگاه با برگرداندن داده ها یا اشاره گر،کنترل را به سیستم عامل ارسال میکند. و به همین علت که درایور یک قابلیتی است که بر روی سخت افزار قرار میگیرد و به آن کمک می کند که با سیستم عامل و سایر سخت افزار ها تعامل داشته باشد باید در kernel mode کار کند که تمام دسترسی ها برای آن باز باشد.