گزارش کار ساخت سیستم‌عامل از ابتدا

ارائه‌دهنده: سینا محمودی خورندی

دانشجوی دکتری مهندسی نرم‌افزار – سیستم‌های توزیعی

توسعه سیستم‌عامل تمامی مراحل مدیریت یک کامپیوتر از ابتدای بوت تا انتهای قطع برق را شامل می‌شود. از این‌رو آشنایی کامل با مراحل بوت شدن یک کامپیوتر با معماری خاص، دستورات ممتاز از یک سو و آشنایی با سرویس‌های که سیستم‌عامل به کاربردها مقیم در آن می‌دهد، برای توسعه دهنده ضروری می‌باشد. آگاهی از واسط‌های مدیریتی ایجاد شده توسط سخت‌افزار برای سیستم‌عامل نیز از برای توسعه‌دهنده ضروری می‌باشد. در ادامه برای بوت یک سیستم و اشکالزدایی آن از ابزارهای شبیه‌سازی و اشکال‌زدایی استافده می‌گردد. از eclipse و GDB برای اشکالزدایی و از qemu و bochs برای شبیه‌سازی استفاده می‌شود. البته برای ماشین‌های با معماری متفاوت نیز باید شبیه‌سازهای دیگری استفاده نمود.

برای کامپایل سیستم‌عامل از مرحله بوت تا مرحله مدیریت پیچیده کارها، شبکه‌بندی و کار با ورودی خروجی و فایل‌های مورد استفاده قرار می‌گیرد. ترتیب کامپایل شدن، لینک شدن بسیار مهم است. از طرفی ساخت فایل اجرایی مورد نظر و فرمت فایل اجرایی خروجی نیز بسته به سیستم و نحوه بوت متفاوت است. برای ساخت سیستم‌عامل در مرحله ابتدایی برای یک سیستم ۳۲ بیتی باید ابزار nasm، gcc، ld برای کامپایل فایل‌ها در اختیار داشت. البته می‌توان از gas به جای nasm نیز استافده نمود که در این صورت برخی مراحل نیاز به تبدیل فرمت فایل‌ها وجود دارد. مراحل طی شده در این مستند، بر مبنای اطلاعات موجود در سایت http://wiki.osdev.org/ تدوین شده است. البته برخی از این مراحل نیز دارای اصلاحات و ساختاردهی مجددی است که مولف در آن ایجاد کرده است.

ساخت یک برنامه بوت ساده

در ابتدا مقداری درباره نحوه بوت یک سیستم x86 بحث می‌نماییم. این معماری به دلیل رایج بودن آن انتخاب شده است و روال کلی توسعه می‌تواند به معماری‌های مختلف تعمیم داده شود. در یک سیستم x86 پس از روشن شدن و راه افتادن قطعاتی مانند CPU، RAM، IO-devices و Mainboard-devices، قسمتی سخت‌افزاری به نام BIOS به دنبال رسانه‌ای که سیستم‌عامل را در خود جا داده است می‌گردد. در BIOS

می‌توان ترتیب رسانه‌های بوت را بسته به پشتیبانی سخت‌افزاری آن رسانه‌ها و Mainboard تعیین نمود. در صورتیکه هیچ رسانه‌ای به این منظور وجود نداشته باشد، تمامی عملکرد ماشین متوقف می‌گردد. رسانه‌هایی مانند cd-rom، hard-disk، network-card و portable-device از جمله این ابزارهای می‌باشند. در سیستم x86 به صورت سنتی می‌توان فلاپی را توسط BIOS تقلید (emulate) نمود. پس از شناسایی رسانه بوت توسط BIOS، ۵۱۲ بایت اول آن در مکانی از حافظه با آدرس 0000:7C00 hex قرار می‌گیرد. و آغاز کار سیستم‌عامل رسما از طرف BIOS شروع می‌شود. در ادامه بخشی از کار به عهده برنامه است و بخشی دیگر به عهده BIOS است. برنامه موجود در حافظه (که نمی‌توان به آن سیستم‌عامل گفت) وظیفه اجرای دستورات مورد نیاز برای آوردن دیگر بخش‌های سیستم‌عامل را به درون حافظه دارد. در صورت وقوع وقفه نیز BIOS آن را مدیریت می‌نماید. نکته‌ای که در این بخش حایز اهمیت است، حضور سیستم در وضعیت Real Mode می‌باشد. بدان معنا که آدرس‌دهی حافظه به صورت Real Mode صورت می‌پذیرد. در این حالت تنها 1MB ابتدایی حافظه توسط برنامه قابل دسترسی است و هیچگونه حفاظتی از دیگر قسمت‌ها نیز صورت نمی‌پذیرد. برای دسترسی به دیگر قسمت‌های حافظه باید یک دستور ممتاز برای تغییر حالت اجرایی در Real Mode به Protected Mode اجرا نمود. در ماشین‌های x86 در Protected Mode حافظه‌ای به اندازه 4GB در اختیار برنامه‌ها قرار دارد. نحوه آدرس‌دهی عناصر در حالت Real Mode به صورت seg*16+off می‌باشد. که seg نشان‌دهنده عدد قطعه و off نشان‌دهنده فاصله از ابتدای قطعه می‌باشد. می‌توان اطلاعات بیشتر راجع به Real Mode را در http://www.osdev.org/wiki/Real_Mode مشاهده نمود. در جدول زیر می‌توان خلاصه‌ای از این مکان‌های حافظه در Real Mode مشاهده نمود.

start	end	size	type	description
Low Memory (the first MiB)				
0x00000000	0x000003FF	1 KiB	RAM - partially unusable (see above)	Real Mode IVT (Interrupt Vector Table)
0x00000400	0x000004FF	256 bytes	RAM - partially unusable (see above)	BDA (BIOS data area)
0x00000500	0x00007BFF	almost 30 KiB	RAM (guaranteed free for use)	Conventional memory
0x00007C00 (typical location)	0x00007DFF	512 bytes	RAM - partially unusable (see above)	Your OS BootSector

0x00007E00	0x0007FFFF	480.5 KiB	RAM (guaranteed free for use)	Conventional memory
0x00080000	0x0009FBFF	approximately 120 KiB, depending on EBDA size	RAM (free for use, if it exists)	Conventional memory
0x0009FC00 (typical location)	0x0009FFFF	1 KiB	RAM (unusable)	EBDA (Extended BIOS Data Area)
0x000A0000	0x000FFFFF	384 KiB	various (unusable)	Video memory, ROM Area

از آنجا که خواندن و فهمیدن کد زبان اسمبلی gas مشکل‌تر از nasm می‌باشد، در ابتدا کدها با nasm بیان می‌شوند و سپس برخی از آن‌ها را برای gas نیز بیان می‌کنیم. در ادامه یک کد مربوط به نوشتن یک عبارت در خروجی نشان داده شده است. در این کد سیستم پس از بوت شدن متنی را در خروجی با کمک وقفه BIOS می‌نویسد. روال این کد پس از شکل تشریح می‌گردد. اگرچه این متن برای آموزش زبان اسمبلی نیست، با این حال برخی از ناگفته‌های این زبان بیان می‌گردند.

```
%macro BiosPrint 1
        mov si, word %1
ch_loop:lodsb
  or al, al
  jz done
  mov ah, 0x0E
  int 0x10
  jmp ch_loop
done:
%endmacro

[ORG 0x7c00]
  xor ax, ax
  mov ds, ax

  BiosPrint msg
 cli
hang:
  jmp hang

msg   db 'Welcome to Your OS', 13, 10, 0

  times 510-($-$$) db 0
  db 0x55
  db 0xAA
```

خط اول اعلان یک ماکرو می‌باشد که دارای یک پارامتر 1 می‌باشد. شماره جلوی نام ماکرو نشان‌دهنده تعداد پارامترهای ماکرو می‌باشد (http://www.nasm.us/doc). علامت 1% که در سطر دوم به کار رفته است ارجاع به اولین پارامتر ماکرو می‌باشد. دستور lodsb آدرس قرار گرفته در si در AL بار می‌کند و سپس مقادیر

را برای بار گذاری مقداری بعدی آماده می‌کند. در ادامه وقفه BIOS برای چاپ کاراکتر در خروجی فراخوانی می‌شود. پس از اتمام ارسال تمامی کاراکتر ماکرو پایان می‌یابد. شروع برنامه اصلی در [ORG 0x7C00] می‌باشد. این راهنما دو دستور به اسمبلر می‌دهد. اول اینکه ابتدای برنامه با این دستور است و ابتدای آن را در این خانه حافظه قرار بدهد. دوم اینکه آدرس تمامی متغیرهای از این آدرس به بعد محاسبه گردد. پس از صفر کردن AX، مقدار سگمنت داده به صفر تنظیم می‌گردد. پس از اتمام چاپ تنها برنامه در یک حلقه می‌ماند. دستور cli برای جلوگیری از خاموش شدن موتور رسانه بوت استفاده شده است. شبه دستور times برای پر کردن بخشی از حافظه با مقادیر مورد و یا توافق شده برنامه بوت استفاده می‌شود. بدان معنا که به اندازه باقیمانده از ۵۱۰ بایت مجاز از ابتدای 0000:7C00 بایت را به صفر مقدار دهی کن. معمولا برنامه بوت در انتهای ۵۱۲ بایتش، دو بایت برای مقادیر 0x55 و 0xAA قرار می‌دهد.

حال مرحله اساسی مربوط کامپایل و اجرای را نشان می‌دهیم. از آنجاکه در مراحل پیشرفته‌تر از grub به عنوان برنامه بوت استفاده می‌نماییم، شروع کار با این این مراحل می‌تواند دید مناسبی را در توسعه دهنده ایجاد نماید. برای کامپایل ین کد از دستور زیر استفاده می‌نماییم.

```
nasm boot.asm -f bin -o boot.bin
```

اسمبلر nasm می‌تواند خروجی‌های متنوعی با فرمت‌های متنوع ایجاد نماید. گزینه f– فرمت فایل خروجی را تعیین می‌نماید. در اینجا یک فایل باینری معمولی انتخاب شده است زیرا این کدها به عنوان برنامه بوت مستقیما در آدرس مناسب بار می‌شوند. در ادامه ساخت یک تصویر فلاپی نشان داده می‌شود. از آنجاییکه در برخی سیستم‌های بوت از فلاپی تقلید شده استفاده می‌شود، نحوه ساخت فلاپی بوت مهم می‌باشد.

برای این مهم ابتدا یک فایل با نام flopp.img می‌سازیم. سپس مراحل زیر به ترتیب اجرا می‌نماییم.

```
# dd if=/dev/zero ibs=1024 count=1440 of flopp.img
# mke2fs flopp.img
# losetup /dev/loop0 floppy.img
```

در این مرحله فلاپی به عنوان یک ابزار loopback شناخته شده است. سپس باید تمامی فایل boot.bin را د ابتدای این loopback کپی زیرا BIOS این قسمت در حافظه بار می‌کند. از این رو از دستور dd استفاده می‌نماییم.

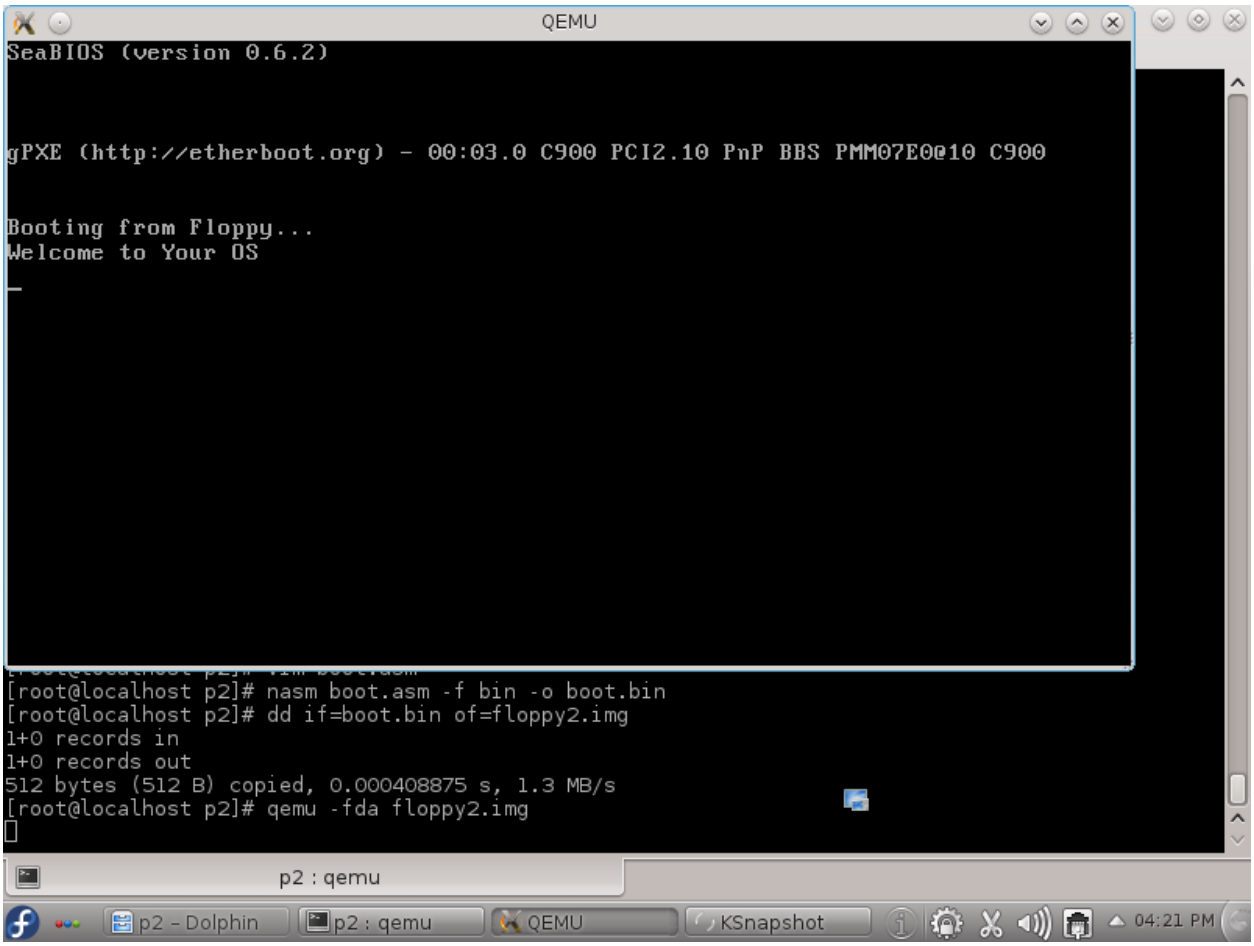
```
# dd if=boot.bin of=/dev/loop0
```

می‌توان بدون قرار دادن floppy.img در loop0 نیز این کار را انجام داد. سپس loop0 را به سیستم بر می‌گردانیم.

```
# losetup -d /dev/loop0
```

در این مرحله اولین تصویر بوت آماده شده است. مرحله بعد استفاده از bochs و یا qemu برای تست بوت می‌باشد. نصب این دو ابزار در حوزه این متن نمی‌باشد از این رو توسعه‌دهندگان می‌تواند به سایت‌های آن‌ها جهت آگاهی از نحوه نصب مراجعه نمایند. به عنوان مثال با استفاده از qemu بر روی Fedora 16 اینگونه خروجی را مشاهده نمود.

```
# qemu -fda floppy2.img
```



برای ساخت cdrom که قابل بوت باشد چندین راه در پیش است. یک راه ساخت cdrom از روی فلاپی ساخته شده می‌باشد.

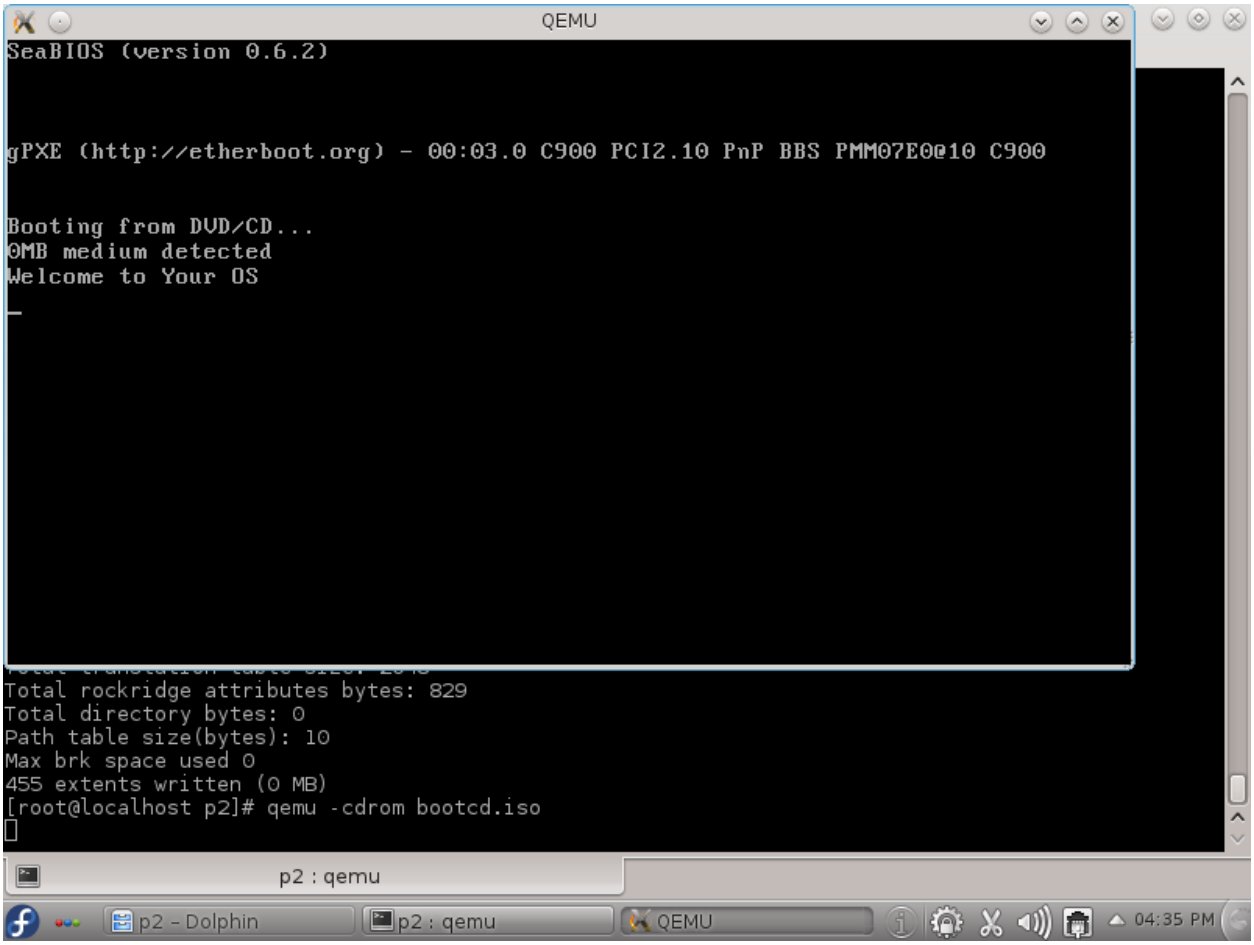
```
# mkisofs -r -b floppy2.img -o bootcd.iso.
```

و یا ساخت آن از روی فایل boot.bin می‌باشد.

```
# mkisofs -r -b boot.bin -no-emul-boot -boot-load-seg 0 -o bootcd.iso.
```

مطالعه گزینه‌های دستور mkisofs به عهده توسعه‌دهندگان می‌باشد. سپس با دستور زیر بار دیگر تصویر بوت بالا می‌آید. در حالت اول BIOS یک فلاپی را برای سیستم تقلید می‌نماید و سیستم را بر اساس آن بوت می‌نماید. در این حالت اندازه فلاپی دارای استانداردی است که باید حفظ گردد. در حالت دوم BIOS مستقیما ۵۱۲ بایت اول cdrom را در حافظه بار می‌کند.

```
# qemu -cdrom bootcd.iso
```



اگرچه در بسیاری از مراحل ساخت نسخه کامپایلر، اسمبلر و سیستم‌عامل مهم است، در اینجا تنها معماری ۳۲ بیتی و یا ۶۴ بیتی مهم می‌باشد. این برنامه بوت روی هر دو نوع معماری قابل اجراست. بخشی از کار نیز می‌تواند ساخت یک Cross Compiler مناسب برای ساخت سیستم‌عامل می‌باشد. در این مقال تنها این نکته را متذکر می‌شوم که برای این کامپایلر وجود binutils یکی از ملزومات است.