Booting Linux

Jack Rosenthal

February 23, 2017

Mines Linux Users Group

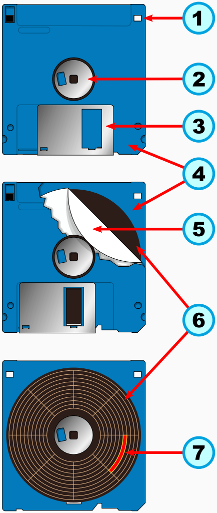
Table of contents

1. Master Boot Record
2. Extensible Firmware Interface
3. Booting Linux
4. Resources

Master Boot Record

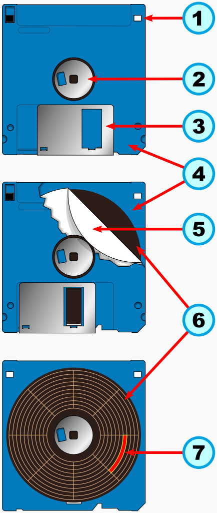
Floppy Disks

- Floppy disks organized into 512-byte sectors
- Intel 8086 originally only allowed booting from floppy
- First sector is the **boot sector**, 512 bytes of executable x86 machine code which runs in **real mode**.



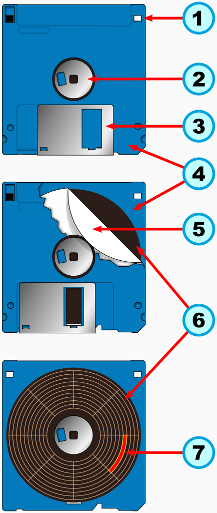
Floppy Disks

- Floppy disks organized into 512-byte sectors
- Intel 8086 originally only allowed booting from floppy
- First sector is the **boot sector**, 512 bytes of executable x86 machine code which runs in **real mode**.



Floppy Disks

- Floppy disks organized into 512-byte sectors
- Intel 8086 originally only allowed booting from floppy
- First sector is the **boot sector**, 512 bytes of executable x86 machine code which runs in **real mode**.



The 10 MB Hard Disk Came

- IBM wanted a way to boot their systems off their new 10 MB hard disk in 1983
- They added a 4-partition table to the end of the 512-byte boot sector
- Boot sectors compatible with older systems because the machine code ends before the partition data
- This is called **Master Boot Record**

XCOMP introduces a complete micro-size disk subsystem with more...

- **MORE STORAGE**
- **MORE SPEED**
- **MORE VALUE**
- **MORE SUPPORT**

S100 users . . . The XCOMP subsystem is now available with 10 megabytes of storage; 5 megabytes also available at $2,898.00. Compare the price and features of any other 5¼-inch — or even 8-inch system, and you'll agree that XCOMP's value is unbeatable.

OUTPERFORMS OTHER HARD DISKS
Floppy disk and larger, more expensive hard disks are no match for this powerful little system. More data is available on every seek: 64K on 10MB and 32K on 5MB. Faster seek time too — an average of 70MS. It provides solid performance anywhere with only 20 watts of power. Data is protected in the sealed enclosure, and the landing zone for heads provides another margin of safety. The optional power board plugs directly into the S100 bus and provides power for the drive.

FAST CONTROLLER
The XCOMP controller is the key to this system's high efficiency operation. Speed-up features include interleave without table lookup, block-deblock with controller buffer, and read lookahead. OEMs worldwide have already proven the outstanding performance of the XCOMP controller.

MORE SOFTWARE
Included with the system is software for testing, formatting, I/O drivers for CP/M®, plus an automatic CP/M driver attach program. Support software and drivers for MP/M® and Oasis® are also available. The sophisticated formatting program assigns alternate sectors for any weak sectors detected during formatting, assuring the lowest possible error rate — at least ten times better than floppies.

WARRANTY
The system has a full one-year warranty on parts and workmanship.

ALSO AVAILABLE FROM XCOMP

- General Purpose controllers (8 bit interface), with easy interface to microprocessor-based systems.
- GP controller adapter that plugs directly into most Z80 computers.
- ST/R GP controller for the 5MB and 10MB drive above, with ST506 type interface.
- SG/R GP controller for SA1000 interface.
- SM/R GP controller for storage module drives.
- ST/S, SG/S, and SM/S, same as above, for the S100 bus.

Quantity discounts available. Distributor, Dealer, and OEM inquiries invited.

See your local Dealer, or call:

XCOMP, Inc.
7566 Trade Street
San Diego, CA 92121
Tel: (714) 271-8730
Telex: 182786

Circle 405 on inquiry card.

The 10 MB Hard Disk Came

- IBM wanted a way to boot their systems off their new 10 MB hard disk in 1983
- They added a 4-partition table to the end of the 512-byte boot sector
- Boot sectors compatible with older systems because the machine code ends before the partition data
- This is called **Master Boot Record**

XCOMP introduces a complete micro-size disk subsystem with more...

- MORE STORAGE
- MORE SPEED
- MORE VALUE
- MORE SUPPORT

S100 users . . . The XCOMP subsystem is now available with 10 megabytes of storage; 5 megabytes also available at $2,898.00. Compare the price and features of any other 5¼-inch — or even 8-inch system, and you'll agree that XCOMP's value is unbeatable.

OUTPERFORMS OTHER HARD DISKS
Floppy disk and larger, more expensive hard disks are no match for this powerful little system. More data is available on every seek: 64K on 10MB and 32K on 5MB. Faster seek time too — an average of 70MS. It provides solid performance anywhere with only 20 watts of power. Data is protected in the sealed enclosure, and the landing zone for heads provides another margin of safety. The optional power board plugs directly into the S100 bus and provides power for the drive.

FAST CONTROLLER
The XCOMP controller is the key to this system's high efficiency operation. Speed-up features include interleave without table lookup, block-deblock with controller buffer, and read lookahead. OEMs worldwide have already proven the outstanding performance of the XCOMP controller.

MORE SOFTWARE
Included with the system is software for testing, formatting, I/O drivers for CP/M®, plus an automatic CP/M driver attach program. Support software and drivers for MP/M® and Oasis® are also available. The sophisticated formatting program assigns alternate sectors for any weak sectors detected during formatting, assuring the lowest possible error rate — at least ten times better than floppies.

WARRANTY
The system has a full one-year warranty on parts and workmanship.

ALSO AVAILABLE FROM XCOMP

- General Purpose controllers (8 bit interface), with easy interface to microprocessor-based systems.
- GP controller adapter that plugs directly into most Z80 computers.
- ST/R GP controller for the 5MB and 10MB drive above, with ST506 type interface.
- SG/R GP controller for SA1000 interface.
- SM/R GP controller for storage module drives.
- ST/S, SG/S, and SM/S, same as above, for the S100 bus.

Quantity discounts available. Distributor, Dealer, and OEM inquiries invited.

See your local Dealer, or call:

XCOMP, Inc.
7566 Trade Street
San Diego, CA 92121
Tel: (714) 271-8730
Telex: 182786

Circle 405 on inquiry card.

The 10 MB Hard Disk Came

- IBM wanted a way to boot their systems off their new 10 MB hard disk in 1983
- They added a 4-partition table to the end of the 512-byte boot sector
- Boot sectors compatible with older systems because the machine code ends before the partition data
- This is called **Master Boot Record**

XCOMP introduces a complete micro-size disk subsystem with more...

- MORE STORAGE
- MORE SPEED
- MORE VALUE
- MORE SUPPORT

S100 users. . . The XCOMP subsystem is now available with 10 megabytes of storage; 5 megabytes also available at $2,898.00. Compare the price and features of any other 5¼-inch — or even 8-inch system, and you'll agree that XCOMP's value is unbeatable.

OUTPERFORMS OTHER HARD DISKS
Floppy disk and larger, more expensive hard disks are no match for this powerful little system. More data is available on every seek: 64K on 10MB and 32K on 5MB. Faster seek time too — an average of 70MS. It provides solid performance anywhere with only 20 watts of power. Data is protected in the sealed enclosure, and the landing zone for heads provides another margin of safety. The optional power board plugs directly into the S100 bus and provides power for the drive.

FAST CONTROLLER
The XCOMP controller is the key to this system's high efficiency operation. Speed-up features include interleave without table lookup, block-deblock with controller buffer, and read lookahead. OEMs worldwide have already proven the outstanding performance of the XCOMP controller.

MORE SOFTWARE
Included with the system is software for testing, formatting, I/O drivers for CP/M®, plus an automatic CP/M driver attach program. Support software and drivers for MP/M® and Oasis® are also available. The sophisticated formatting program assigns alternate sectors for any weak sectors detected during formatting, assuring the lowest possible error rate — at least ten times better than floppies.

WARRANTY
The system has a full one-year warranty on parts and workmanship.

ALSO AVAILABLE FROM XCOMP

- General Purpose controllers (8 bit interface), with easy interface to microprocessor-based systems.
- GP controller adapter that plugs directly into most Z80 computers.
- ST/R GP controller for the 5MB and 10MB drive above, with ST506 type interface.
- SG/R GP controller for SA1000 interface.
- SM/R GP controller for storage module drives.
- ST/S, SG/S, and SM/S, same as above, for the S100 bus.

Quantity discounts available. Distributor, Dealer, and OEM inquiries invited.

See your local Dealer, or call:

XCOMP, Inc.
7566 Trade Street
San Diego, CA 92121
Tel: (714) 271-8730
Telex: 182786

Circle 405 on inquiry card.

The 10 MB Hard Disk Came

- IBM wanted a way to boot their systems off their new 10 MB hard disk in 1983
- They added a 4-partition table to the end of the 512-byte boot sector
- Boot sectors compatible with older systems because the machine code ends before the partition data
- This is called **Master Boot Record**

XCOMP introduces a complete micro-size disk subsystem with more...

- MORE STORAGE
- MORE SPEED
- MORE VALUE
- MORE SUPPORT

S100 users . . . The XCOMP subsystem is now available with 10 megabytes of storage; 5 megabytes also available at $2,898.00. Compare the price and features of any other 5¼-inch — or even 8-inch system, and you'll agree that XCOMP's value is unbeatable.

OUTPERFORMS OTHER HARD DISKS
Floppy disk and larger, more expensive hard disks are no match for this powerful little system. More data is available on every seek: 64K on 10MB and 32K on 5MB. Faster seek time too — an average of 70MS. It provides solid performance anywhere with only 20 watts of power. Data is protected in the sealed enclosure, and the landing zone for heads provides another margin of safety. The optional power board plugs directly into the S100 bus and provides power for the drive.

FAST CONTROLLER
The XCOMP controller is the key to this system's high efficiency operation. Speed-up features include interleave without table lookup, block-deblock with controller buffer, and read lookahead. OEMs world-wide have already proven the outstanding performance of the XCOMP controller.

MORE SOFTWARE
Included with the system is software for testing, formatting, I/O drivers for CP/M®, plus an automatic CP/M driver attach program. Support software and drivers for MP/M® and Oasis® are also available. The sophisticated formatting program assigns alternate sectors for any weak sectors detected during formatting, assuring the lowest possible error rate — at least ten times better than floppies.

WARRANTY
The system has a full one-year warranty on parts and workmanship.

ALSO AVAILABLE FROM XCOMP

- General Purpose controllers (8 bit interface), with easy interface to microprocessor-based systems.
- GP controller adapter that plugs directly into most Z80 computers.
- ST/R GP controller for the 5MB and 10MB drive above, with ST506 type interface.
- SG/R GP controller for SA1000 interface.
- SM/R GP controller for storage module drives.
- ST/S, SG/S, and SM/S, same as above, for the S100 bus.

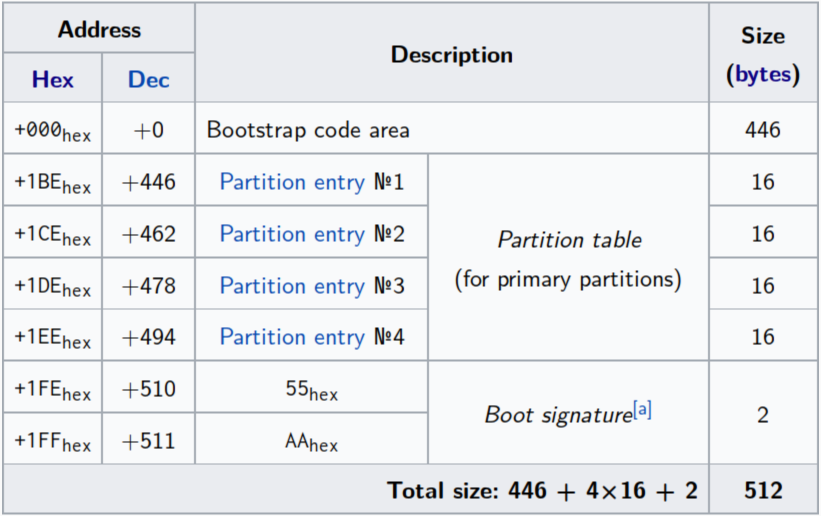
Quantity discounts available. Distributor, Dealer, and OEM inquiries invited.

See your local Dealer, or call:

XCOMP
XCOMP, Inc.
7566 Trade Street
San Diego, CA 92121
Tel: (714) 271-8730
Telex: 182786

Circle 405 on inquiry card.

Master Boot Record



Address		Description		Size (bytes)
Hex	Dec			
$+000_{\text{hex}}$	+0	Bootstrap code area		446
$+1\text{BE}_{\text{hex}}$	+446	Partition entry №1	*Partition table* (for primary partitions)	16
$+1\text{CE}_{\text{hex}}$	+462	Partition entry №2		16
$+1\text{DE}_{\text{hex}}$	+478	Partition entry №3		16
$+1\text{EE}_{\text{hex}}$	+494	Partition entry №4		16
$+1\text{FE}_{\text{hex}}$	+510	55_{hex}	*Boot signature*[a]	2
$+1\text{FF}_{\text{hex}}$	+511	AA_{hex}		
			Total size: 446 + 4×16 + 2	**512**

What does a MBR bootloader do?

1. Determine the partition to boot from
2. Determine where your kernel image is on the partition
3. Load the kernel into memory
4. Enable protected mode
5. Set up the environment for the kernel (stack space, etc.)
6. Call your kernel's `main` function

You will probably agree, that's a lot to do in 446 bytes of machine code.

What does a MBR bootloader do?

1. Determine the partition to boot from
2. Determine where your kernel image is on the partition
3. Load the kernel into memory
4. Enable protected mode
5. Set up the environment for the kernel (stack space, etc.)
6. Call your kernel's `main` function

You will probably agree, that's a lot to do in 446 bytes of machine code.

What does a MBR bootloader do?

1. Determine the partition to boot from
2. Determine where your kernel image is on the partition
3. Load the kernel into memory
4. Enable protected mode
5. Set up the environment for the kernel (stack space, etc.)
6. Call your kernel's `main` function

You will probably agree, that's a lot to do in 446 bytes of machine code.

What does a MBR bootloader do?

1. Determine the partition to boot from
2. Determine where your kernel image is on the partition
3. Load the kernel into memory
4. Enable protected mode
5. Set up the environment for the kernel (stack space, etc.)
6. Call your kernel's `main` function

You will probably agree, that's a lot to do in 446 bytes of machine code.

What does a MBR bootloader do?

1. Determine the partition to boot from
2. Determine where your kernel image is on the partition
3. Load the kernel into memory
4. Enable protected mode
5. Set up the environment for the kernel (stack space, etc.)
6. Call your kernel’s `main` function

You will probably agree, that’s a lot to do in 446 bytes of machine code.

What does a MBR bootloader do?

1. Determine the partition to boot from
2. Determine where your kernel image is on the partition
3. Load the kernel into memory
4. Enable protected mode
5. Set up the environment for the kernel (stack space, etc.)
6. Call your kernel's `main` function

You will probably agree, that's a lot to do in 446 bytes of machine code.

What does a MBR bootloader do?

1. Determine the partition to boot from
2. Determine where your kernel image is on the partition
3. Load the kernel into memory
4. Enable protected mode
5. Set up the environment for the kernel (stack space, etc.)
6. Call your kernel's `main` function

You will probably agree, that's a lot to do in 446 bytes of machine code.

Some *Real* Challenges

- Most C compilers won’t compile to real mode code, so booting is on the *list of things you can’t even do in C*
- Real mode uses 16 memory segments of 64K each
- To switch segments, you must issue special instructions to the processor
- This gives you a total of 1 MiB of memory to use for booting
- *Does your kernel fit in 1 MiB? Minus the memory you are using for your program to boot?*

Some *Real* Challenges

- Most C compilers won't compile to real mode code, so booting is on the *list of things you can't even do in C*
- Real mode uses 16 memory segments of 64K each
- To switch segments, you must issue special instructions to the processor
- This gives you a total of 1 MiB of memory to use for booting
- *Does your kernel fit in 1 MiB? Minus the memory you are using for your program to boot?*

Some *Real* Challenges

- Most C compilers won’t compile to real mode code, so booting is on the *list of things you can’t even do in C*
- Real mode uses 16 memory segments of 64K each
- To switch segments, you must issue special instructions to the processor
- This gives you a total of 1 MiB of memory to use for booting
- *Does your kernel fit in 1 MiB? Minus the memory you are using for your program to boot?*

Some *Real* Challenges

- Most C compilers won’t compile to real mode code, so booting is on the *list of things you can’t even do in C*
- Real mode uses 16 memory segments of 64K each
- To switch segments, you must issue special instructions to the processor
- This gives you a total of 1 MiB of memory to use for booting
- *Does your kernel fit in 1 MiB? Minus the memory you are using for your program to boot?*

Some *Real* Challenges

- Most C compilers won't compile to real mode code, so booting is on the *list of things you can't even do in C*
- Real mode uses 16 memory segments of 64K each
- To switch segments, you must issue special instructions to the processor
- This gives you a total of 1 MiB of memory to use for booting
- *Does your kernel fit in 1 MiB? Minus the memory you are using for your program to boot?*

Approaches to Solving Booting Challenges

- **Geek Booting:** Do everything your kernel needs to boot in the 512-byte boot sector. You will need your kernel to fit in 1 MiB as well. This is hard.
- **One-Stage Booting:** Write your bootloader in the first 1 MiB of your kernel image, then write a 512-byte program that loads that program. The 1 MiB program is responsible for loading the rest of your kernel and booting it.
- **Two-Stage Booting:** Write a separate kernel that fits in 1 MiB called a bootloader. This program is responsible for providing a high level interface to boot other kernels. GRUB is an example.

Approaches to Solving Booting Challenges

- **Geek Booting:** Do everything your kernel needs to boot in the 512-byte boot sector. You will need your kernel to fit in 1 MiB as well. This is hard.
- **One-Stage Booting:** Write your bootloader in the first 1 MiB of your kernel image, then write a 512-byte program that loads that program. The 1 MiB program is responsible for loading the rest of your kernel and booting it.
- **Two-Stage Booting:** Write a separate kernel that fits in 1 MiB called a bootloader. This program is responsible for providing a high level interface to boot other kernels. GRUB is an example.

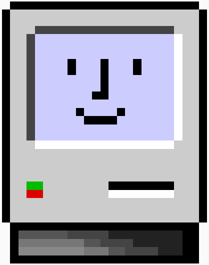
Approaches to Solving Booting Challenges

- **Geek Booting:** Do everything your kernel needs to boot in the 512-byte boot sector. You will need your kernel to fit in 1 MiB as well. This is hard.
- **One-Stage Booting:** Write your bootloader in the first 1 MiB of your kernel image, then write a 512-byte program that loads that program. The 1 MiB program is responsible for loading the rest of your kernel and booting it.
- **Two-Stage Booting:** Write a separate kernel that fits in 1 MiB called a bootloader. This program is responsible for providing a high level interface to boot other kernels. GRUB is an example.

Extensible Firmware Interface

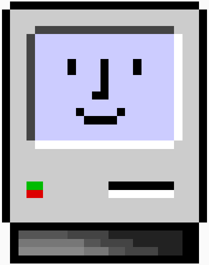
Apple

- Historically, Macs have booted using a hardware chip on the board called the **Macintosh ROM**
- The Mac ROM provided a miniature operating system (with a mouse cursor and all) capable of booting Mac OS
- With the switch to PowerPC from 68K, Apple modified the ROM to include an Open Firmware Interface capable of extending booting capabilities beyond just classical Mac OS



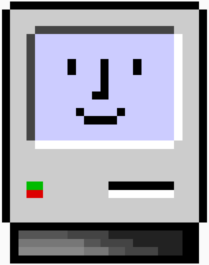
Apple

- Historically, Macs have booted using a hardware chip on the board called the **Macintosh ROM**
- The Mac ROM provided a miniature operating system (with a mouse cursor and all) capable of booting Mac OS
- With the switch to PowerPC from 68K, Apple modified the ROM to include an Open Firmware Interface capable of extending booting capabilities beyond just classical Mac OS



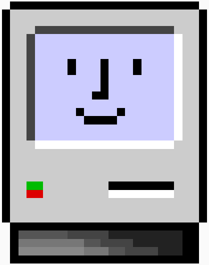
Apple

- Historically, Macs have booted using a hardware chip on the board called the **Macintosh ROM**
- The Mac ROM provided a miniature operating system (with a mouse cursor and all) capable of booting Mac OS
- With the switch to PowerPC from 68K, Apple modified the ROM to include an Open Firmware Interface capable of extending booting capabilities beyond just classical Mac OS



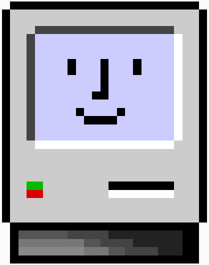
Apple

- With the switch to Intel x86 from PowerPC, Apple looked for a solution to boot Mac OS X from something that didn't suck as much as MBR
- Apple looked at Intel's long forgotten **Extensible Firmware Interface** (EFI)
- EFI was similar to Apple's OFI, but it worked on Intel processors and had plenty of more features
- Thanks Apple! You popularized EFI and made booting x86 suck less!



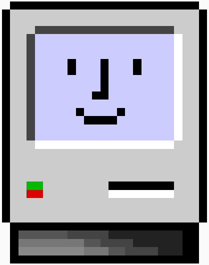
Apple

- With the switch to Intel x86 from PowerPC, Apple looked for a solution to boot Mac OS X from something that didn't suck as much as MBR
- Apple looked at Intel's long forgotten **Extensible Firmware Interface** (EFI)
- EFI was similar to Apple's OFI, but it worked on Intel processors and had plenty of more features
- Thanks Apple! You popularized EFI and made booting x86 suck less!



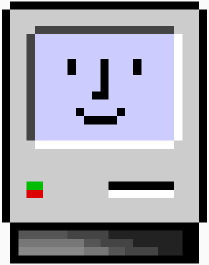
Apple

- With the switch to Intel x86 from PowerPC, Apple looked for a solution to boot Mac OS X from something that didn't suck as much as MBR
- Apple looked at Intel's long forgotten **Extensible Firmware Interface** (EFI)
- EFI was similar to Apple's OFI, but it worked on Intel processors and had plenty of more features
- Thanks Apple! You popularized EFI and made booting x86 suck less!



Apple

- With the switch to Intel x86 from PowerPC, Apple looked for a solution to boot Mac OS X from something that didn't suck as much as MBR
- Apple looked at Intel's long forgotten **Extensible Firmware Interface** (EFI)
- EFI was similar to Apple's OFI, but it worked on Intel processors and had plenty of more features
- Thanks Apple! You popularized EFI and made booting x86 suck less!



UEFI in a Nutshell

- Simply write your bootloader in C and leave a `.efi` binary on the FAT32 formatted **EFI System Partition**, the system's UEFI firmware takes care of running your program for you
- Provides high level interfaces to the graphical console, hardware, disks, memory, and even network
- Capable of doing hash checks on your bootloader to ensure it was not tampered with by a computer virus

UEFI in a Nutshell

- Simply write your bootloader in C and leave a `.efi` binary on the FAT32 formatted **EFI System Partition**, the system’s UEFI firmware takes care of running your program for you
- Provides high level interfaces to the graphical console, hardware, disks, memory, and even network
- Capable of doing hash checks on your bootloader to ensure it was not tampered with by a computer virus

UEFI in a Nutshell

- Simply write your bootloader in C and leave a `.efi` binary on the FAT32 formatted **EFI System Partition**, the system’s UEFI firmware takes care of running your program for you
- Provides high level interfaces to the graphical console, hardware, disks, memory, and even network
- Capable of doing hash checks on your bootloader to ensure it was not tampered with by a computer virus

Hello World EFI-Style

```c
#include <efi.h>
#include <efilib.h>

EFI_STATUS
EFIAPI
efi_main (EFI_HANDLE Handle, EFI_SYSTEM_TABLE *Table)
{
    InitializeLib(Handle, Table);
    Print(L"Hello, world!\n");
    return EFI_SUCCESS;
}
```

Booting Linux

So this is all great, how does Linux boot?

1. First, the compressed Linux kernel (`vmlinuz`) is loaded by the bootloader and started
2. The Linux kernel then loads a file system called `initrd` into memory which contains just enough programs to mount your disk and load drivers
3. The kernel flag `root` specifies where your root partition is located to be mounted
4. Once the root partition is mounted, `/etc/fstab` is read to determine any other partitions to be mounted
5. `/bin/init` is called

So this is all great, how does Linux boot?

1. First, the compressed Linux kernel (`vmlinuz`) is loaded by the bootloader and started
2. The Linux kernel then loads a file system called `initrd` into memory which contains just enough programs to mount your disk and load drivers
3. The kernel flag `root` specifies where your root partition is located to be mounted
4. Once the root partition is mounted, `/etc/fstab` is read to determine any other partitions to be mounted
5. `/bin/init` is called

So this is all great, how does Linux boot?

1. First, the compressed Linux kernel (`vmlinuz`) is loaded by the bootloader and started
2. The Linux kernel then loads a file system called `initrd` into memory which contains just enough programs to mount your disk and load drivers
3. The kernel flag `root` specifies where your root partition is located to be mounted
4. Once the root partition is mounted, `/etc/fstab` is read to determine any other partitions to be mounted
5. `/bin/init` is called

So this is all great, how does Linux boot?

1. First, the compressed Linux kernel (`vmlinuz`) is loaded by the bootloader and started
2. The Linux kernel then loads a file system called `initrd` into memory which contains just enough programs to mount your disk and load drivers
3. The kernel flag `root` specifies where your root partition is located to be mounted
4. Once the root partition is mounted, `/etc/fstab` is read to determine any other partitions to be mounted
5. `/bin/init` is called

So this is all great, how does Linux boot?

1. First, the compressed Linux kernel (`vmlinuz`) is loaded by the bootloader and started
2. The Linux kernel then loads a file system called `initrd` into memory which contains just enough programs to mount your disk and load drivers
3. The kernel flag `root` specifies where your root partition is located to be mounted
4. Once the root partition is mounted, `/etc/fstab` is read to determine any other partitions to be mounted
5. `/bin/init` is called

So what is `/bin/init`?

- `init` is the process with PID 1; it is the super-parent process of every process started on your system
- If `init` were to die, the kernel would panic
- Historically, System V style `init` programs would start a shell script located at `/etc/rc` that then loads your programs and desktop environment
- Most `/etc/rc` files use modularized shell scripts under `/etc/rc.d` or `/etc/init.d` to start services
- Shell scripts are *slow*, and all sorts of standards exist for how to write these shell scripts

So what is /bin/init?

- `init` is the process with PID 1; it is the super-parent process of every process started on your system
- If `init` were to die, the kernel would panic
- Historically, System V style `init` programs would start a shell script located at `/etc/rc` that then loads your programs and desktop environment
- Most `/etc/rc` files use modularized shell scripts under `/etc/rc.d` or `/etc/init.d` to start services
- Shell scripts are *slow*, and all sorts of standards exist for how to write these shell scripts

So what is `/bin/init`?

- `init` is the process with PID 1; it is the super-parent process of every process started on your system
- If `init` were to die, the kernel would panic
- Historically, System V style `init` programs would start a shell script located at `/etc/rc` that then loads your programs and desktop environment
- Most `/etc/rc` files use modularized shell scripts under `/etc/rc.d` or `/etc/init.d` to start services
- Shell scripts are *slow*, and all sorts of standards exist for how to write these shell scripts

So what is /bin/init?

- `init` is the process with PID 1; it is the super-parent process of every process started on your system
- If `init` were to die, the kernel would panic
- Historically, System V style `init` programs would start a shell script located at `/etc/rc` that then loads your programs and desktop environment
- Most `/etc/rc` files use modularized shell scripts under `/etc/rc.d` or `/etc/init.d` to start services
- Shell scripts are *slow*, and all sorts of standards exist for how to write these shell scripts

So what is `/bin/init`?

- `init` is the process with PID 1; it is the super-parent process of every process started on your system
- If `init` were to die, the kernel would panic
- Historically, System V style `init` programs would start a shell script located at `/etc/rc` that then loads your programs and desktop environment
- Most `/etc/rc` files use modularized shell scripts under `/etc/rc.d` or `/etc/init.d` to start services
- Shell scripts are *slow*, and all sorts of standards exist for how to write these shell scripts

systemd: An alternative `init`

- Theory: Shell scripts as a configuration file is clunky and provides scattered interfaces
- Acts as a replacement `/bin/init` but uses configuration files rather than shell scripts
- This topic kind of deserves a talk of it's own? Anyone want to do it?

systemd: An alternative `init`

- Theory: Shell scripts as a configuration file is clunky and provides scattered interfaces
- Acts as a replacement `/bin/init` but uses configuration files rather than shell scripts
- This topic kind of deserves a talk of it's own? Anyone want to do it?

systemd: An alternative `init`

- Theory: Shell scripts as a configuration file is clunky and provides scattered interfaces
- Acts as a replacement `/bin/init` but uses configuration files rather than shell scripts
- This topic kind of deserves a talk of it's own? Anyone want to do it?

Resources

Resources

- **OSDev Wiki:** Great resource on developing your own OS, including writing bootloaders. `http://osdev.org`
- There's nothing else. That wiki has about everyting you need.

Resources

- OSDev Wiki: Great resource on developing your own OS, including writing bootloaders. `http://osdev.org`
- There's nothing else. That wiki has about everyting you need.

Questions?

Copyright Notice

Colorado School of Mines
Linux Users Group