Future topics

- System calls
- VFS
- Scheduling
- Memory management

http://oclug.on.ca/KernelWalk
Booting
Outline

- PC hardware overview
- software components
- boot overview
- follow the code
PC hardware basics
PC hardware basics

- CPU
- memory
- disk controller
- HDD
PC hardware basics

- CPU
- memory
- disk controller
- HDD
PC hardware basics

- CPU
- memory
- disk controller
- HDD
- video
PC hardware basics

- CPU
- cache
- MMU
- bridge
- video
- ATA
- RAM
- RAM
- USB
- BIOS
- HDD
- HDD
- stick

Bart Trojanowski  http://www.jukie.net/~bart/  August 21, 2008
PC hardware basics

CPU

- lots of variants
  - Intel Core2 Duo
  - AMD Opteron
  - VIA Eden

- backward compatible to 1970s
  - emulates 8086
  - 16bit
  - 1M memory
## PC hardware basics

### CPU

<table>
<thead>
<tr>
<th>&quot;real mode&quot;</th>
<th>&quot;protected mode&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>- all BIOS services</td>
<td>- few BIOS services</td>
</tr>
<tr>
<td>- 1M addressable</td>
<td>- 4G memory</td>
</tr>
<tr>
<td></td>
<td>- virtual memory</td>
</tr>
</tbody>
</table>
PC hardware basics

- MMU
  - memory managment unit
  - "north bridge" or in CPU
  - paging
  - virtual memory
PC hardware basics

MMU

virtual memory

logical 1

logical 2

physical RAM
PC hardware basics

MMU

virtual memory

logical 1

logical 2

page

physical RAM

Bart Trojanowski  http://www.jukie.net/~bart/  August 21, 2008
PC hardware basics

virtual memory
- page mappings
PC hardware basics

virtual memory
- addressing

4GB*

MMU

logical 1

physical RAM

logical 2

Bart Trojanowski  http://www.jukie.net/~bart/  August 21, 2008
PC hardware basics

- MMU
- virtual memory
- shared pages

Bart Trojanowski  http://www.jukie.net/~bart/  August 21, 2008
PC hardware basics

- "south bridge"
- local bus
- PCI
- peripheral controllers (ATA, USB, etc)
- memory addressable h/w devices
- routes interrupts
PC hardware basics

memory mapped IO

bridge

logical

physical

PCI device

physical RAM
... for more on MM topics tune in next time.
Software
Software

- firmware
- BIOS
- MBR
- boot loader
- kernel
Software
Firmware
Software
Firmware

- CPU
- microcode
- initialize processor(s)
Software
BIOS
Software
BIOS

- real mode
- 16bit registers
- 1M addressable memory
- limited HW access
- slow
Software

BIOS

Initialization
- CPU init
  - cache init
  - interconnect
- HW enumeration
  - reset / setup
- POST
Software

BIOS

Services

- memory layout
- disk
- PCI
- APM & ACPI
- graphics
Software

BIOS

Disk services

AH = operation
DL = drive, first HDD is 0x80
...

INT 0x13
Software

Master Boot Record
Software

Master Boot Record

- 440 bytes of code
- first disk block
- bootloader unique
- loads rest of the bootloader
Software
Master Boot Record

- aka "1st stage bootloader"
Software
Bootloader
Software

Bootloader

- BIOS calls
- loads kernel and initrd into RAM
- slow
Software

Bootloader

- BIOS services only see 1M of memory
- typical kernel is ~2M
- typical initrd is ~5M
Software

Bootloader

- BIOS services only see 1M of memory
- typical kernel is ~2M
- typical initrd is ~5M

... that's a problem!
Software
Bootloader

"unreal" mode

- switch to protected mode
- enable flat 32bit memory
- return to real mode
Software
Kernel

kernel

user space
Booting
Booting

1. BIOS
2. bootloader
3. kernel
4. user space
Booting
On disk...
Booting
On disk...

1 block \( \square \) 512 bytes
Booting
On disk...

- stage1
- stage2
- config

HDD

Bart Trojanowski       http://www.jukie.net/~bart/       August 21, 2008
Booting

BIOS
Booting BIOS

In memory...

BIOS code/data

IVT / BIOS data

Bart Trojanowski  http://www.jukie.net/~bart/  August 21, 2008
Booting BIOS

In memory...

ACPI "stuff"

BIOS code/data

IVT / BIOS data
Booting

BIOS

Loads & jumps to MBR
Booting

Bootloader
Booting

Bootloader

MBR loads "step2 block"
Bootloading

Bootloader

MBR loads "step2 block"

step2:          # "disk address packet"
   .word 16, 1
   .word 0x7e00, 0  # destination
   .qword 0x...     # LBA disk location

mov  $0x42, %ah   # extended read
mov  $0x80, %dl   # drive c:
lea  step2, %si   # descriptor in DS:SI
int  $0x13
Booting

Bootloader

MBR loads "step2 block"

```assembly
step2:                      # "disk address packet"
  .word 16, 1              # destination
  .word 0x7e00, 0          # LBA disk location
mov  $0x42, %ah           # extended read
mov  $0x80, %dl           # drive c:
lea  step2, %si           # descriptor in DS:SI
int  $0x13
```

Bart Trojanowski     http://www.jukie.net/~bart/     August 21, 2008
Booting

Bootloader

stage1 loads stage2
Booting

Bootloader

stage2 finds the config
Booting

Bootloader

stage2 loads kernel
Booter

stage2 loads initial ram disk

HDD

RAM

0x100000
Booting

Bootloader

stage2 jumps into the kernel
Booting

Kernel
Booting
Kernel

In the beginning...

initial ram disk
compressed vmlinux
real mode stub

0x100000
RAM
Booting Kernel

In the beginning...

- initial ram disk
- compressed vmlinux
- early stack/heap
- real mode stub

Bart Trojanowski     http://www.jukie.net/~bart/     August 21, 2008
Booting Kernel

In the beginning...

- initial ram disk
- compressed vmlinux
- kernel command line
- early stack/heap
- real mode stub

RAM

0x100000

~64K

Bart Trojanowski
http://www.jukie.net/~bart/
August 21, 2008
Booting
Kernel
real-mode
Booting
Kernel
real-mode

- 8086 asm
- enters at "start_of_setup"
- resets disk controllers
- sets up stack
- calls main()
Booting
Kernel
real-mode

- parse some config options
- probe some HW through BIOS
  - memory
  - APM
- set video

- call go_to_protected_mode()
Booting
Kernel
real-mode

arch/x86/boot/pm.c

- setup for switch
- IDT (null table)
- GDT (flat memory layout)

- switch to protected mode
Booting Kernel

decompressing

arch/x86/boot/compressed/head_32.S

- 386 asm
- enters at "startup_32"
- sets up segments & stack
Booting Kernel

decompressing

arch/x86/boot/compressed/head_32.S

- 386 asm
- enters at "startup_32"
- sets up segments & stack
- moves things around
Booting Kernel

decompressing

arch/x86/boot/compressed/head_32.S

- 386 asm
- enters at "startup_32"
- sets up segments & stack
- moves things around
Bootstrapping Kernel

decompressing

arch/x86/boot/compressed/head_32.S

- 386 asm
- enters at "startup_32"
- sets up segments & stack
- moves things around
- call decompress_kernel()
Booting Kernel

decompressing

arch/x86/boot/compressed/misc.c

- mostly C
- uses gunzip
Booting Kernel

decompressing

arch/x86/boot/compressed/misc.c

- mostly C
- uses gunzip
Booting Kernel

decompressing

arch/x86/boot/compressed/misc.c

- mostly C
- uses gunzip
- jumps to decompressed kernel
Booting Kernel

start real kernel

arch/x86/kernel/head_32.S

- 386 asm
- enters at "startup_32"
- setup segments & stack
Booting Kernel

start real kernel

- arch/x86/kernel/head_32.S
  - 386 asm
  - enters at "startup_32"
  - setup segments & stack
  - copy cmdline
Booting Kernel

start real kernel

- arch/x86/kernel/head_32.S
- 386 asm
- enters at "startup_32"
- setup segments & stack
- copy cmdline
- setup virtual memory
- setup_idt
- call start_kernel()
Booting
Kernel
initialize subsystems

init/main.c

start_kernel()
- init all subsystems
- call rest_init()
Booting
Kernel

kick off scheduler

init/main.c

rest_init()

- start PID 1 task

- PID 0 calls cpu_idle()
Kernel

Booting

kick off scheduler

PID 0

start_kernel

rest_init
Booting

Kernel

click off scheduler

PID 0

start_kernel

rest_init

PID 1
Booting
Kernel
kick off scheduler

PID 0

- start_kernel
- rest_init
- cpu_idle

PID 1
Booting
Kernel
kick off scheduler

PID 0
start_kernel
rest_init
cpu_idle

PID 1
kernel_init
Booting

Kernel

finish off init stuff

init/main.c

kernel_init()

- start SMP
- init more subsystems
- call init_post()
Booting Kernel

finish off init stuff

init/main.c

init_post()

- finds "init" executable
- run via kernel_execve()
- else panic
Any questions?