COS318 Precept 2
Bootup Mechanism

Wei Dong
Sept. 27, 2006

Outline

- More about X86 assembly language
- bootblock.s
- createimage.c
DF and String instructions

- **lodsb**: %al ← %ds:(%si), update %si
- **stosb**: %es:(%di) ← %al, update %di
- **movsb**: %es:(%di) ← %ds:(%si), update %si and %di

All string instructions update indices by:
- If DF == 1 then index ← index - 1
- If DF == 0 then index ← index + 1

Repeat String Operations

- **rep** only work with string instructions
  - Repeat while %cx != 0
  - Decrease %cx by 1 each time

Usage
- Setup %ds:%si and/or %es:%di
- cld/std
- %cx ← number of bytes
- rep lodsb/stosb/movsb
Calling convention

- How to pass parameters & receive result
- cdecl, pascal,fastcall, stdcall, ...
- Use same convention for both calling & called functions
- cdecl:
  - Parameters pushed to stack from right to left
  - Stack cleanup performed by the caller
  - Return value in %eax
  - %eax, %ecx, %edx are available for function

Example (32-bit code)

```
int function(int a, int b);
int a, b, x;
...
x = function(a,b);
```

```assembly
pushl b
pushl a
call function
add $12, %esp
movl %eax, x
```
Example (cont.)

```c
int function (int a, int b)
{
    int c = a + b;
    return c;
}
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x9FFE8</td>
<td>c</td>
</tr>
<tr>
<td>0x9FFEC</td>
<td>old %ebp</td>
</tr>
<tr>
<td>0x9FF0</td>
<td>return addr</td>
</tr>
<tr>
<td>0x8FFF4</td>
<td>a</td>
</tr>
<tr>
<td>0x8FFF8</td>
<td>b</td>
</tr>
<tr>
<td>0x8FFFC</td>
<td>...</td>
</tr>
</tbody>
</table>

Stack layout in 16-bit code

<table>
<thead>
<tr>
<th>Short call</th>
<th>Long call</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2(%bp)</td>
<td>-2(%bp)</td>
</tr>
<tr>
<td>(%bp)</td>
<td>(%bp)</td>
</tr>
<tr>
<td>2(%bp)</td>
<td>2(%bp)</td>
</tr>
<tr>
<td>4(%bp)</td>
<td>4(%bp)</td>
</tr>
<tr>
<td>6(%bp)</td>
<td>6(%bp)</td>
</tr>
<tr>
<td>local var</td>
<td>local var</td>
</tr>
<tr>
<td>old %bp</td>
<td>old %bp</td>
</tr>
<tr>
<td>ret %ip</td>
<td>ret %ip</td>
</tr>
<tr>
<td>param 1</td>
<td>ret %cs</td>
</tr>
<tr>
<td>param 2</td>
<td>param 1</td>
</tr>
</tbody>
</table>
bootblock.s: 16-bit or 32-bit?

- Concepts
  - 16/32-bit code, real/protected mode
  - X86: start at real mode, later switch to protected mode
- The clean approach:
  - Bootloader does the switch
  - Kernel is pure 32-bit protected mode code
  - See bootblock.s in future projects if interested
- This project:
  - Only work with real mode

bootblock.s: Common Errors

- Not setting up %ds, %ss, %sp
- No “$” for constants
- Wrong offset from %bp for parameters
- Insert instructions before os_size
Moving code!

- Code can be moved as data
- Using label to figure out where to jump to

```c
start:
  jmp over
  ...
  rep movsb
  ljmp ?

next:
```

Jump to where?
Assume code moved to %es:%di, then
%cs ← %es
%ip ← %di + $(next – start)
How to do that?

Disc geometry

```
cylinder 0, head 0, sector 1
...
cylinder 0, head 0, sector MAX_SEC
cylinder 0, head 1, sector 1
...
cylinder 0, head MAX_HEAD, sector MAX_HEAD
cylinder 1, head 0, sector 1
...
...
Use INT 0x13 Function 8 for MAX_SEC and MAX_HEAD
```
Offset address overflow

- INT 0x13 use %es:%bx as buffer
- %bx is 16-bit, range from 0x0000-0xFFFF
- Example
  - Assume: %bx = 0xFE0A, then after reading a sector
  - %bx + 0x200 = 0x00A0, lose 0x10000
  - Solution: update %es to reflect the overflow
  - Problem: how to detect overflow, how to adjust %es?

Get ELF segment information

- How to avoid dealing with the file format?
  - Write the output of `readelf` to a text file
  - Read from that text file
  - Or use `popen` to avoid a temporary file
- But that’s not interesting
createimage.c

- Segments in program header table is not ordered!
- Actually no need to pad after internal segments
  - fseek beyond the end of the file will automatically cause the next file writing operation to fill the gap with 0s.
- Pad at the end so the whole file is divisible by 0x200