MKC – Exercise 2

Nils Asmussen

2022-05-19
Roadmap

• User start.S and linker script
• Multiboot Header
• Map physical memory
• ELF

• Hands-on
  – Parse Multiboot Info and ELF Header
  – Load and execute user binary
$ git clone https://os.inf.tu-dresden.de/repo/git/mkc.git
$ git checkout exercise2

# build it
$ make

# run it
$ make run
• Open user/src/start.S
  – In the .text segment
  – Global symbol __start:
  – Setup a stack: load address of stack_top into esp
  – Call main_func()

• Open user/src/linker.ld
  – Program entry point at symbol __start
  – Two segments: data (rw) and text (rx)
  – Put section .text in segment text and sections .data and .bss and in data
  – ALIGN stack and text to page boundary (0x1000)
Building and Loading the User Program

- **make** user binary and go to user/build
- Inspect binary by `nm user.nova.debug`
  
  ```
  00002000 T __start  
  0000200c T main_func  
  00002000 D stack_top  
  ```

- There are two symbols in the text segment and one in data

- Next: pass binary to the boot loader and load it as boot module after the kernel
  - `cat Makefile`
### Multiboot Information

#### `multiboot.h`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>flags</strong></td>
<td>Flags is required, all the others are optional</td>
</tr>
<tr>
<td><strong>mem_lower</strong></td>
<td></td>
</tr>
<tr>
<td><strong>mem_upper</strong></td>
<td></td>
</tr>
<tr>
<td><strong>boot_device</strong></td>
<td></td>
</tr>
<tr>
<td><strong>cmdline</strong></td>
<td></td>
</tr>
<tr>
<td><strong>mods_count</strong></td>
<td>If <code>flags[3]</code> is set, <code>mods_count</code> and <code>mods_addr</code> is valid</td>
</tr>
<tr>
<td><strong>mods_addr</strong></td>
<td><code>mods_addr</code> is the physical address to an array of module structs with length <code>mods_count</code></td>
</tr>
<tr>
<td><strong>syms[4]</strong></td>
<td></td>
</tr>
<tr>
<td><strong>mmap_length</strong></td>
<td></td>
</tr>
<tr>
<td><strong>mmap_addr</strong></td>
<td></td>
</tr>
<tr>
<td><strong>...</strong></td>
<td></td>
</tr>
</tbody>
</table>

- `mod_start` string reserved (0)
- `mod_end` string reserved (0)
Remap

- But: multiboot info addr and mods_addr are **physical** addresses
- Need to (temporarily) add a mapping into the virtual address space → kernel's remap area **void**
  *
  \texttt{Ptab::remap(phys_addr)}
- Replaces previous mapping, thus whenever calling remap, the old pointer is invalid

Multi boot info

Kernel physical memory virtual memory

remap ▼ area
Task 1 – Find and Map Binary

- Open kern/src/ec.cc : root_invoke()
- `Ec::current->regs.eax` contains mbi pointer
- remap Multiboot Info, check flags:3, get mods_addr and count
- remap Multiboot module structure, print start and end address of user binary
- remap user binary (it's an ELF object)
- see kern/include/multiboot.h and elf.h
Executable and Linkable Format (ELF)

- ELF Header contains offset where to find PH table (ph_offset)
- Program header table describes the segments to be used at runtime
### ELF Header Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>magic</td>
<td>7f 'E' 'L' 'F'</td>
</tr>
<tr>
<td>class</td>
<td>Data class</td>
</tr>
<tr>
<td>data version</td>
<td>Data version</td>
</tr>
<tr>
<td>osabi</td>
<td>OS ABI version</td>
</tr>
<tr>
<td>abi version</td>
<td>ABI version</td>
</tr>
<tr>
<td>padding</td>
<td>Padding</td>
</tr>
<tr>
<td>padding</td>
<td>Padding</td>
</tr>
<tr>
<td>type</td>
<td>Type</td>
</tr>
<tr>
<td>machine</td>
<td>Machine</td>
</tr>
<tr>
<td>version</td>
<td>Version</td>
</tr>
<tr>
<td>entry</td>
<td>Entry – user EIP</td>
</tr>
<tr>
<td>ph_offset</td>
<td>Offset of program header table in file</td>
</tr>
<tr>
<td>sh_offset</td>
<td>Offset of section header table in file</td>
</tr>
<tr>
<td>flags</td>
<td>Flags</td>
</tr>
<tr>
<td>eh_size</td>
<td>ELF header size</td>
</tr>
<tr>
<td>ph_size</td>
<td>Program header size</td>
</tr>
<tr>
<td>ph_count</td>
<td>Number of program headers</td>
</tr>
<tr>
<td>sh_count</td>
<td>Number of section headers</td>
</tr>
<tr>
<td>sh_size</td>
<td>Section header size</td>
</tr>
<tr>
<td>strtab</td>
<td>String table</td>
</tr>
</tbody>
</table>

- Check magic, data (1) and type (2)
- entry – user EIP
- ph_count : number of program headers
- ph_offset : where within the file the program header table starts
Program Header Table

<table>
<thead>
<tr>
<th>ELF Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>file offset</td>
</tr>
<tr>
<td>virtual address</td>
</tr>
<tr>
<td>physical address</td>
</tr>
<tr>
<td>file size</td>
</tr>
<tr>
<td>mem size</td>
</tr>
<tr>
<td>flags</td>
</tr>
<tr>
<td>alignment</td>
</tr>
</tbody>
</table>

- If type = PT_LOAD(1), load this segment
- Flags: 2 writable?
- Offset: where this segment starts relative to the beginning of the file
- Virtual address: where to map this segment to
- File/Mem size: segment size in file and memory
Task 2 – Decode ELF and Start Program

- Continue in root_invoke()
  - user binary is still mapped in
- Set \texttt{current->regs.eip} to correct entry point
- Remap program header table and iterate over all (two) program headers
- If type \texttt{!= PT_LOAD}, ignore this segment
- Align them properly to 4k page boundaries
  - phys/virt addresses : align down
  - mem size : align up
- Print all virt/phys addresses and mem sizes
Add Page Table Entries

• Some sanity checks:
  – File size and mem size should be equal
  – Virtual address and file offset should be equal (modulo page size)

• `Ptab::insert_mapping (virt, phys, attr)`
  – Inserts a mapping from virtual address `virt` to physical address `phys` with attributes `attr`

• See class `Ph` in `kernel/include/elf.h`
  – If `flags & Ph::PF_W` → page should be mapped writable, thus `attr = 7`, otherwise `attr = 5`

• Add mapping for all pages in all segments

• `ret_user_iret()` to start user program
x86 Page Tables: virt → phys

- cr3
- pdir addr
- ptab addr
- page directory
- page table
- 20 bit phys addr
- ... S U R W P
  - P – present (1: entry valid)
  - R/W – 0: read only, 1: writable
  - S/U – 0: kernel only, 1: user

0x1000 .data
0x2000 .stack
0x1000 .text