MKC - Exercise 2

Nils Asmussen

2021-06-17
Roadmap

• 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 by loading the 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

• Goto user/build and **make** user binary
• 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

- **Flags** is required, all the others are optional
- If flags[3] is set, mods_count and mods_addr is valid
- mods_addr is the physical address to an array of module structs with length mods_count

<table>
<thead>
<tr>
<th>Multi boot info pointer</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem_lower</td>
<td></td>
</tr>
<tr>
<td>mem_upper</td>
<td></td>
</tr>
<tr>
<td>boot_device</td>
<td></td>
</tr>
<tr>
<td>cmdline</td>
<td></td>
</tr>
<tr>
<td>mods_count</td>
<td></td>
</tr>
<tr>
<td>mods_addr</td>
<td></td>
</tr>
<tr>
<td>syms[4]</td>
<td></td>
</tr>
<tr>
<td>mmap_length</td>
<td></td>
</tr>
<tr>
<td>mmap_addr</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
• 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
  
  ```c
  void * Ptab::remap(phys_addr)
  ```
• Replaces previous mapping, thus whenever calling remap, the old pointer is invalid
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

### Content:
- **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**

<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 version osabi</td>
</tr>
<tr>
<td>abi version</td>
<td>padding</td>
</tr>
<tr>
<td>padding</td>
<td>padding</td>
</tr>
<tr>
<td>type</td>
<td>machine</td>
</tr>
<tr>
<td>version</td>
<td>entry</td>
</tr>
<tr>
<td>entry</td>
<td>ph_offset</td>
</tr>
<tr>
<td>ph_offset</td>
<td>sh_offset</td>
</tr>
<tr>
<td>sh_offset</td>
<td>flags</td>
</tr>
<tr>
<td>flags</td>
<td>eh_size ph_size</td>
</tr>
<tr>
<td>eh_size</td>
<td>ph_count sh_size</td>
</tr>
<tr>
<td>ph_count</td>
<td>sh_count strtab</td>
</tr>
<tr>
<td>sh_count</td>
<td>strtab</td>
</tr>
<tr>
<td>ELF Header</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td></td>
</tr>
<tr>
<td>file offset</td>
<td></td>
</tr>
<tr>
<td>virtual address</td>
<td></td>
</tr>
<tr>
<td>physical address</td>
<td></td>
</tr>
<tr>
<td>file size</td>
<td></td>
</tr>
<tr>
<td>mem size</td>
<td></td>
</tr>
<tr>
<td>flags</td>
<td></td>
</tr>
<tr>
<td>alignment</td>
<td></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
• Continue in root_invoke()
  - user binary is still mapped in
• Set `current->regs.eip` to correct entry point
• Remap program header table and iterate over all (two) program headers
• If type != `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
• Some sanity checks:
  – File size and mem size should be equal
  – Virtual address and file offset should be equal (modulo page size)
• \texttt{Ptab::insert_mapping (virt, phys, attr)}
  – Inserts a mapping from virtual address \texttt{virt} to physical address \texttt{phys} with attributes \texttt{attr}
• See class \texttt{Ph} in kernel/include/elf.h
  – If \texttt{flags & Ph::PF_W} \rightarrow page should be mapped writable, thus \texttt{attr = 7}, otherwise \texttt{attr = 5}
• Add mapping for all pages in all segments
• \texttt{ret_user_iret()} to start user program
x86 Page Tables: virt → phys

- **cr3**: Page directory address
- **pdir addr**: Page table address
- **ptab addr**: Page table entry

**Page Table Entry**:
- P – present (1: entry valid)
- R/W – 0: read only, 1: writable
- S/U – 0: kernel only, 1: user

**Page Directory**:
- 20 bit phys addr
- ... S U R W P

**Page Table**:
- Data
- Stack
- Text

**Cr3**: Page directory address

**Pdir addr**: Page table address

**Ptab addr**: Page table entry

**31 12 2 1 0**