TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for System Architecture, Operating Systems Group

MKC – Exercise 2

Benjamin Engel

2014-07-10

Overview

- Brief look into assembly and linker script
- Parse Multiboot Header
- Map module(s), decode ELF binary
- Load program segments and jump to user binary entry point

- Hands-on
  - Tiny assembly snippet to start user code
  - Multiboot Info and ELF Header
  - Load and execute user binary

Linker and Assembly

- 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

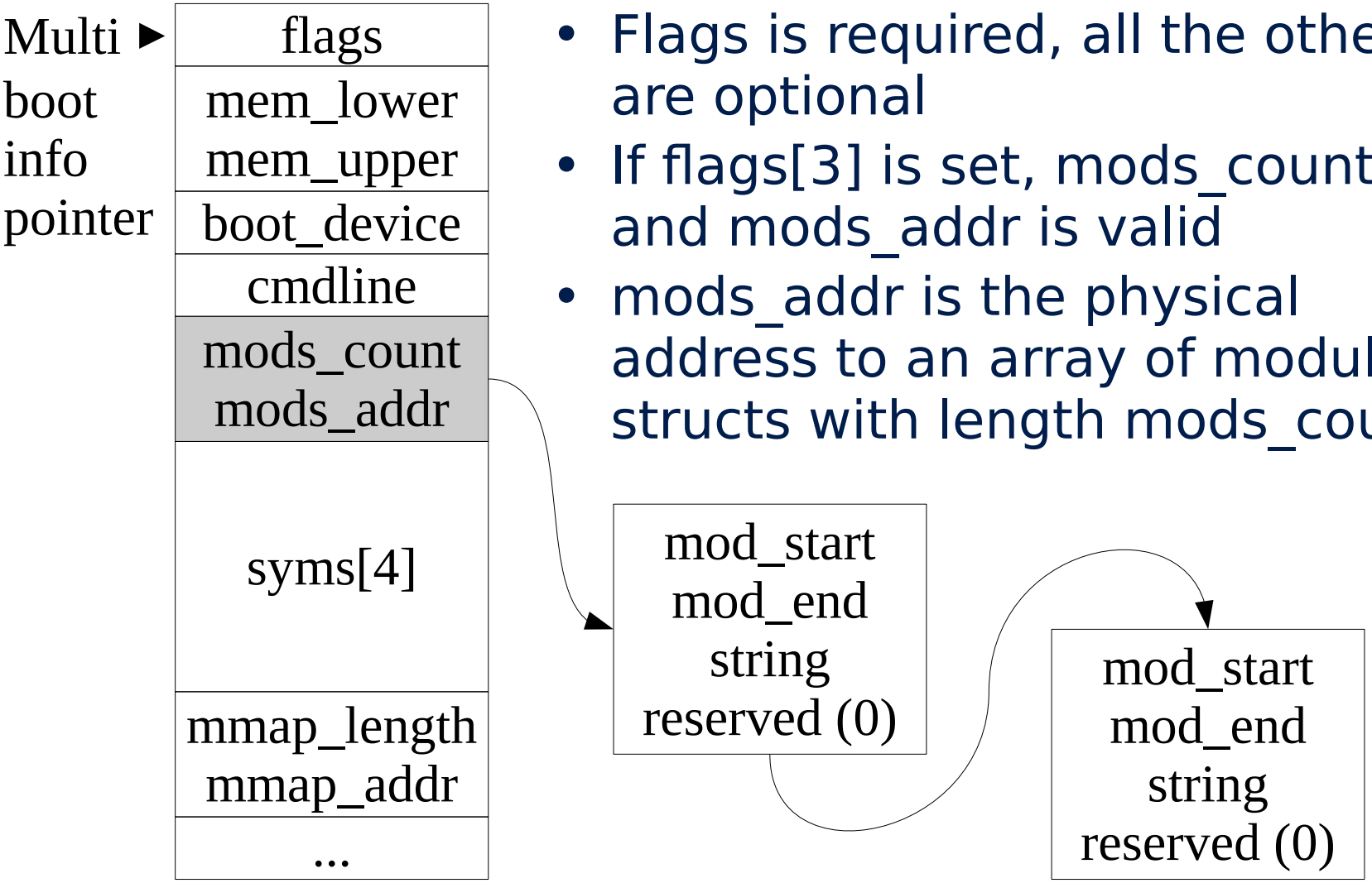
- Goto user/build and **`make`** user binary
- Inspect binary by **`nm user.nova.debug`**

```
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 user binary to the boot loader and load it as second boot module after the kernel
  - **`ls boot`** and **`cat boot/menu.lst`**

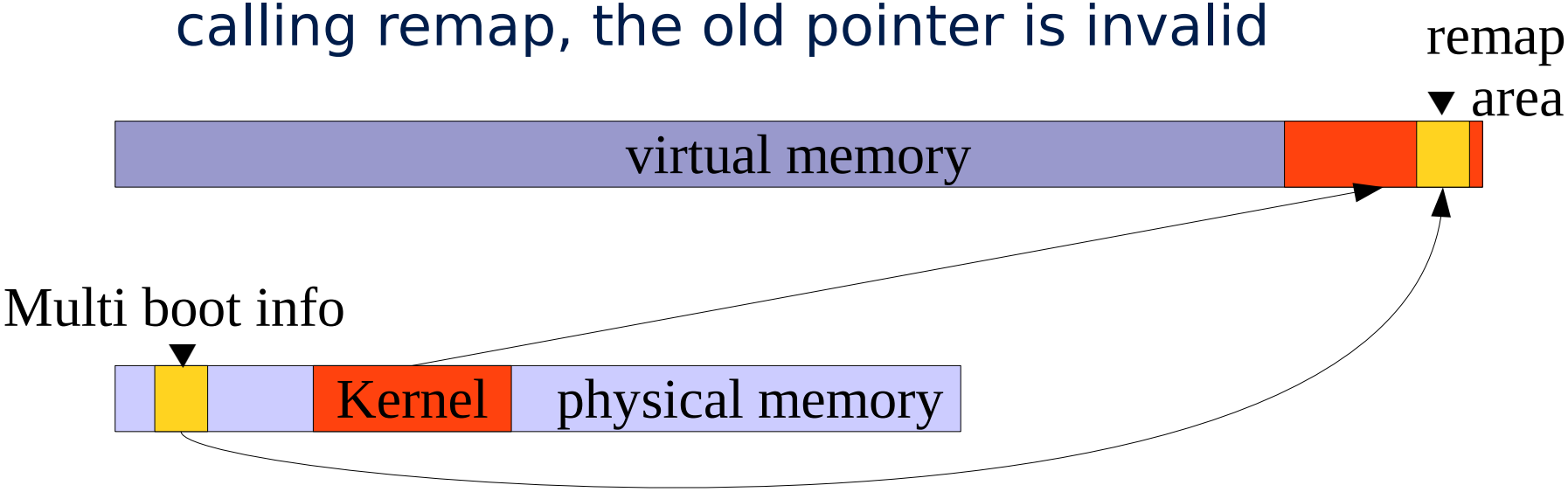
Multiboot Information



- 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

Remap

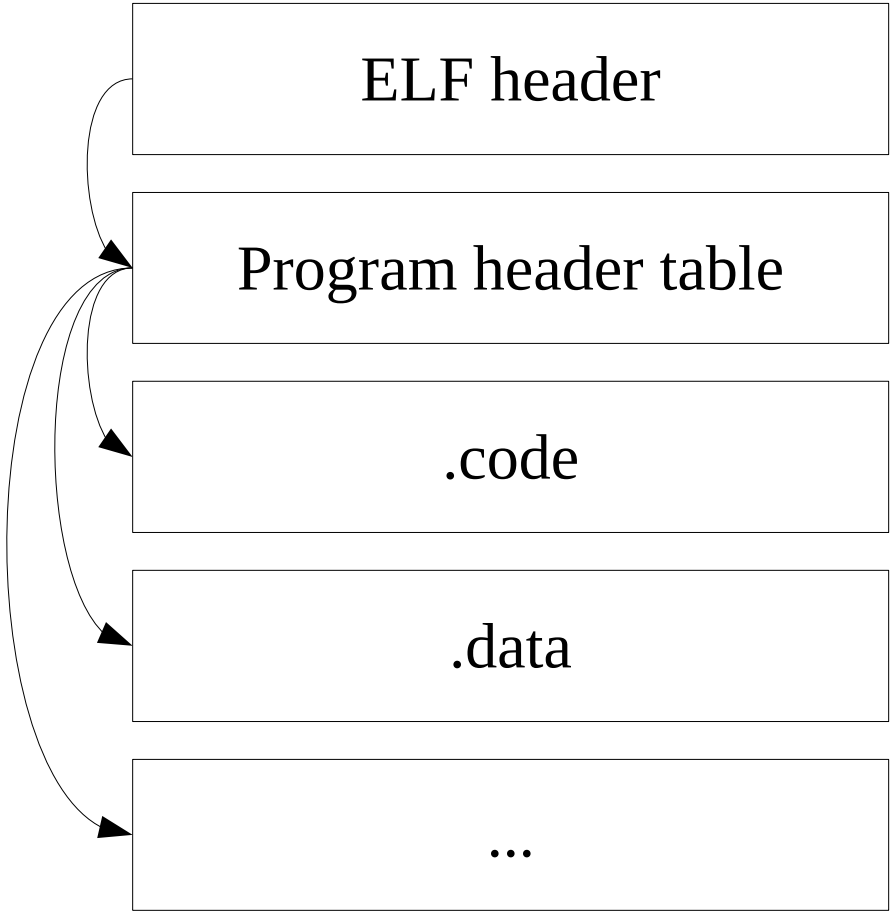
- But: multiboot info addr and mods_addr are **physical** addresses
- Need to (temporarily) add a mapping into the virtual address space → kernel's remap area
  `void * Ptab::remap(phys_addr)`
- Replaces previous mapping, thus whenever calling remap, the old pointer is invalid



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

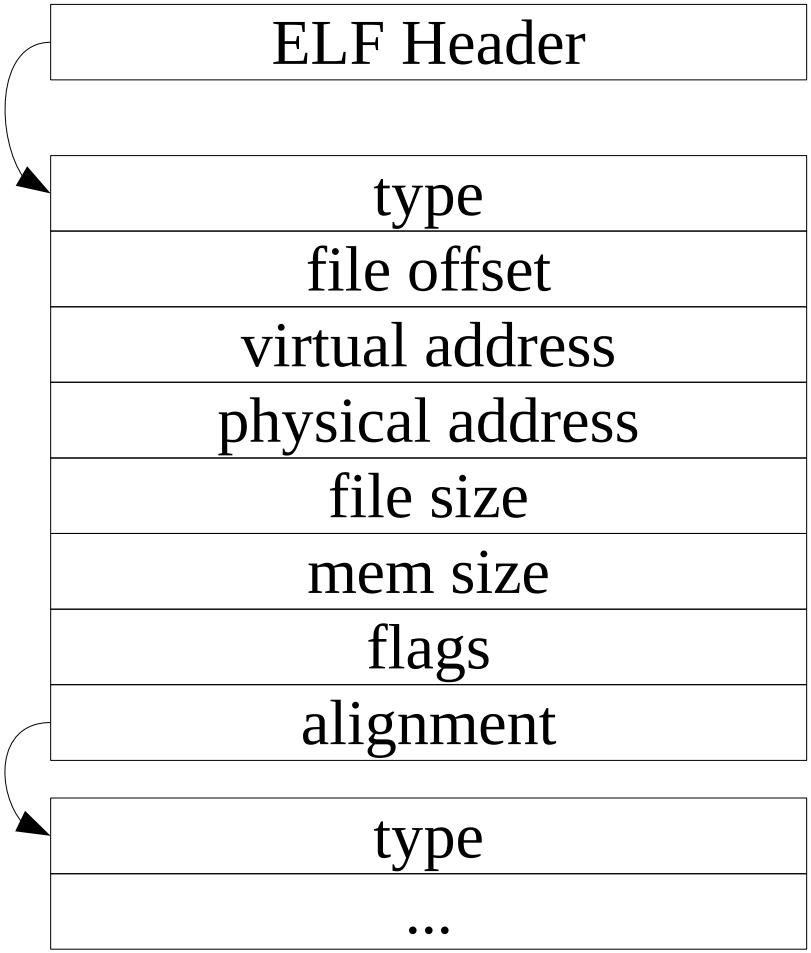
ELF Header Format

elf.h

magic : 7f 'E' 'L' 'F'			
class	data	version	osabi
abi version			
padding			
padding			
type		machine	
version			
entry			
ph_offset			
sh_offset			
flags			
eh_size		ph_size	
ph_count		sh_size	
sh_count		strtab	

- 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



- 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

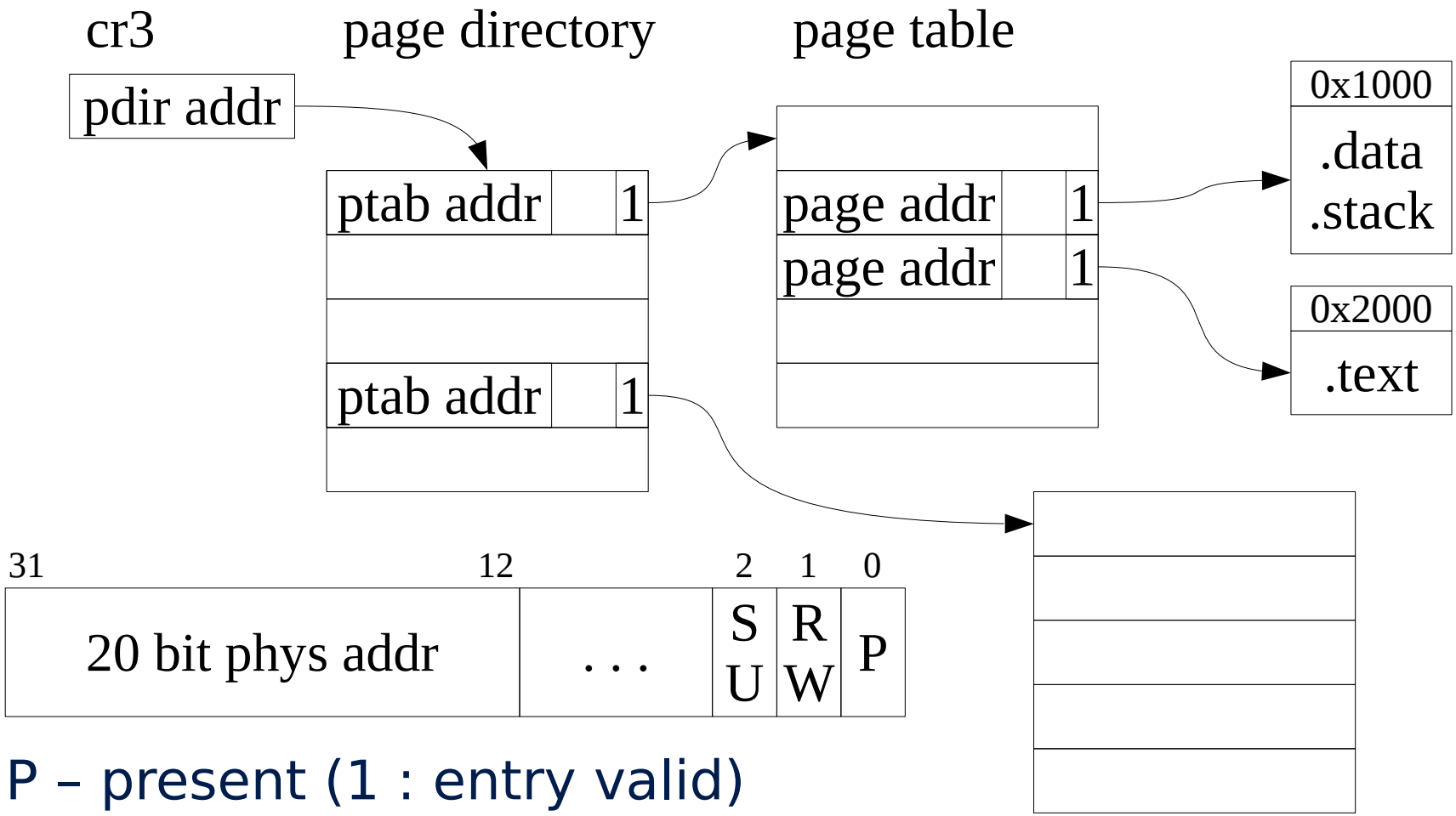
Task 2 – Decode ELF and Start Program

- Continue in root_invoke()
  - user binary is still mapped in
- Set **`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

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



P – present (1 : entry valid)

R/W – 0 : read only, 1 : writable

S/U – 0 : kernel only, 1 : user