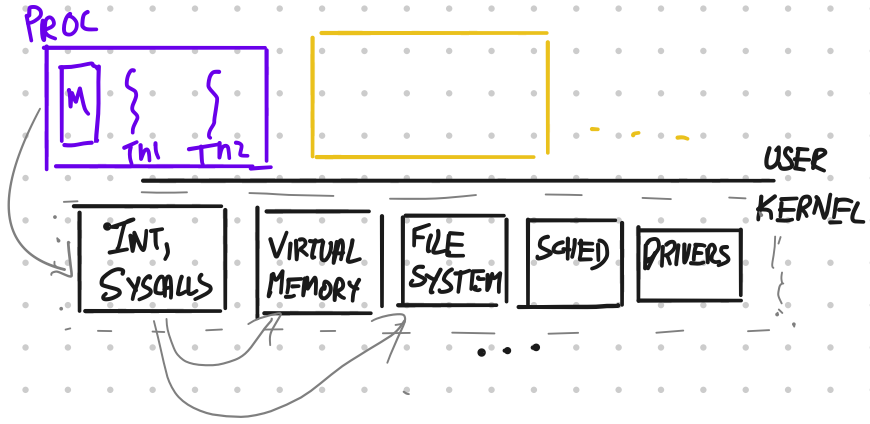
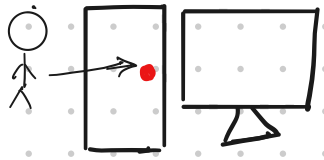
CS202 C-002): Putting "it" all together

Next class: Final exam review — bring questions

Where we are



TODAY - HOW DOES THIS ALL COME TO BE



↳ Hopefully as a reminder of some of the things we covered this semester

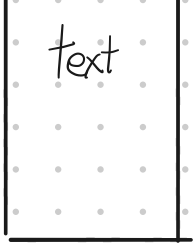
The answer: Many calls to exec..

- What is a program & how is it loaded

PROGRAM

	FIELDS	VALUES	EXPLANATION
1	e_ident		
	EI_MAG	0x7F, "ELF"	CONSTANT SIGNATURE
	EI_CLASS, EI_DATA	1 ELFCLASS32, 1 ELFDATA2LSB	32 BITS, LITTLE-ENDIAN
	EI_VERSION	1 EV_CURRENT	ALWAYS 1
	e_type	2 ET_EXEC	EXECUTABLE
	e_machine	28 EM_ARM	ARM PROCESSOR
	e_version	1 EV_CURRENT	ALWAYS 1
3	e_entry	0x8000060	ADDRESS WHERE EXECUTION STARTS
	e_phoff	0x40	PROGRAM HEADERS' OFFSET
	e_shoff	0xB0	SECTION HEADERS' OFFSET
	e_ehsize	0x34	ELF HEADER'S SIZE
	e_phentsize	0x20	SIZE OF A SINGLE PROGRAM HEADER
	e_phnum	1	COUNT OF PROGRAM HEADERS
	e_shentsize	0x28	SIZE OF A SINGLE SECTION HEADER
	e_shnum	4	COUNT OF SECTION HEADERS
	e_shstrndx	3*	INDEX OF THE NAMES' SECTION IN THE TABLE

ELF Header

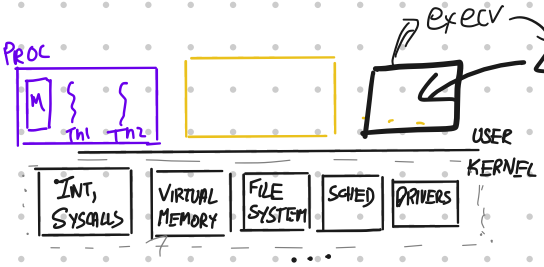


2	p_type	1 PT_LOAD	THE SEGMENT SHOULD BE LOADED IN MEMORY
..	p_offset	0	OFFSET WHERE IT SHOULD BE READ
..	p_vaddr	0x8000000	VIRTUAL ADDRESS WHERE IT SHOULD BE LOADED
..	p_paddr	0x8000000	PHYSICAL ADDRESS WHERE IT SHOULD BE LOADED
	p_filesz	0x90	SIZE ON FILE
	p_memsz	0x90	SIZE IN MEMORY
	p_flags	5 PF_R|PF_X	READABLE AND EXECUTABLE

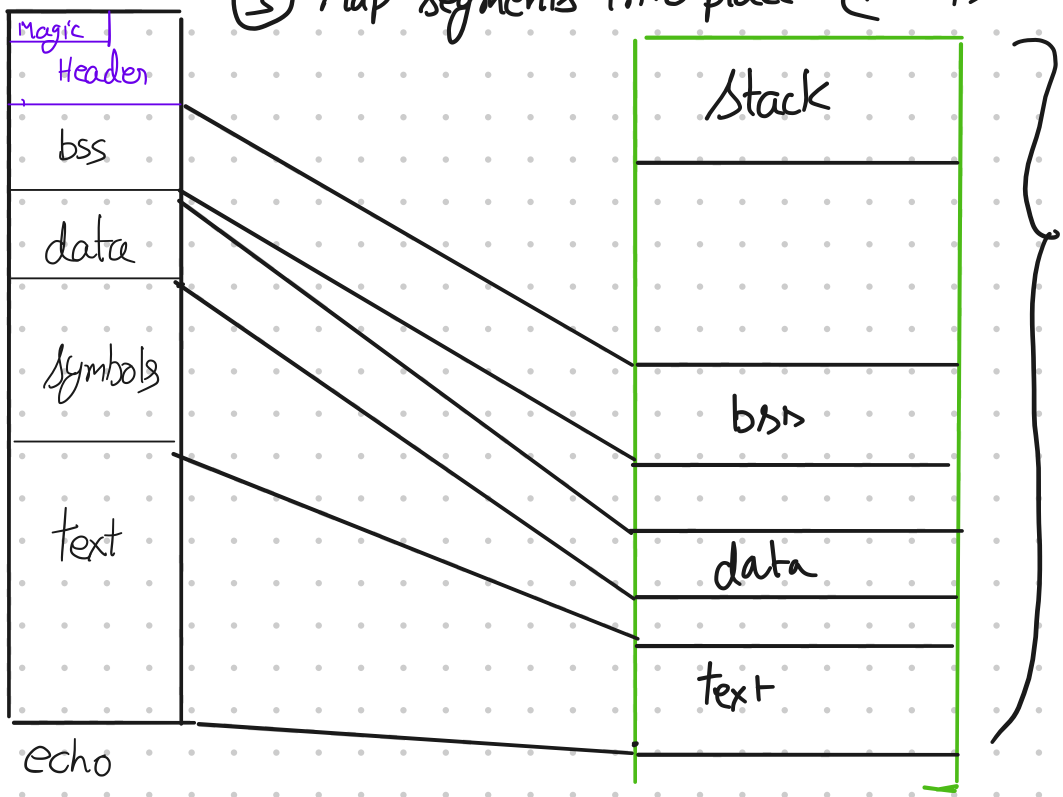
} For each segment

Similar for other executables, e.g., PE

execve → Calls a loader



① Validate the program ② Unmap memory (munmap)
③ Map segments into place (mmap)



Where to map is provided by Segment information!

Note: some segments may not be mapped

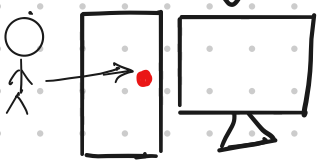
Some segments (stack, heap) are not present in the file

④ Call the executable's entry point
↳ From the header

⑤ Profit

Missing bits - Dynamic libraries
- PIE
} Relocation

Back to



Built by
CPU vendor + Motherboard vendor + ...

Also features in TV season HCF season]

Generally requires proprietary information

Many firmwares exist — we will look at UEFI
(Unified Extensible Firmware Interface]

What UEFI does

① [x86/AMD64 only] Switch from REAL to LONG mode

Backward compat/ anachronism
- 16bit addresses
- No paging
- ...

What we looked at this semester

② Initialize things

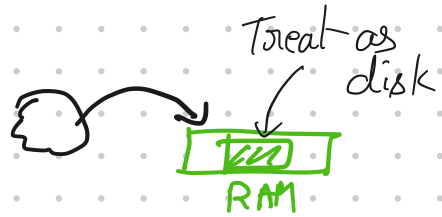
- Identity map memory [Lab 4]
- Set up Interrupt table, etc.
  - ↳ Maps them to firmware
- ENUMERATES DEVICES (see below)
- Load device drivers for some devices
  - → Display
  - → Disk
  - → USB Hub → Keyboard
    - → Mouse
    - → Storage
  - → Network
  - → Sound
  - → ...

Calls for
ⓐ Printing
ⓑ Reading or writing from ...

③ Mount a VFAT partition that contains OS to boot

↳ /EFI or /boot or ...

- From disk
- For network boot



④ Run (exec) OS

↳ Configured path or /EFI/boot/bootx64.efi

OS requirements

ⓐ PE executable

ⓑ Static

ⓒ Entry point

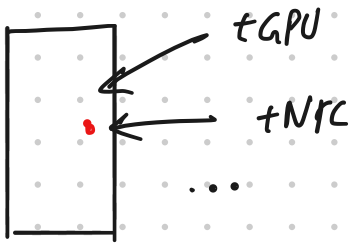
void EntryPoint(EFI_HANDLE,
EFI_SYSTEM_TABLE*)

↳ Starts in sup. mode

→ Must make a call to tell UEFI if does not need it anymore.

ENUMERATE DEV. TREE

- Want to be able to add/change peripherals



- Want to build Operating Systems that work across different computers
(Painful when this is not the case)

- How

ⓐ Tree to describe devices
(+ how they are connected "devices")

ⓑ Passed in as input from firmware to kernel

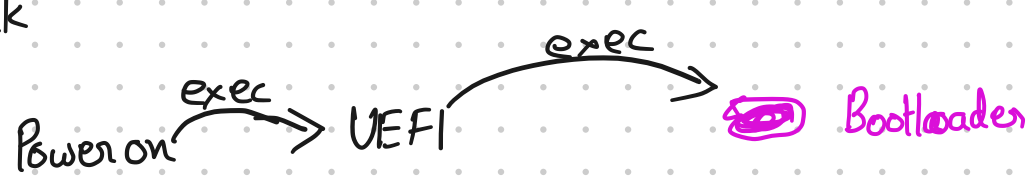
- How created – "Enumeration" Don't worry about this

void EntryPoint(EFI_HANDLE,
EFI_SYSTEM_TABLE*)

- → Physical memory layout
- → Dev. tree
- → Available EFI services

...

Going back

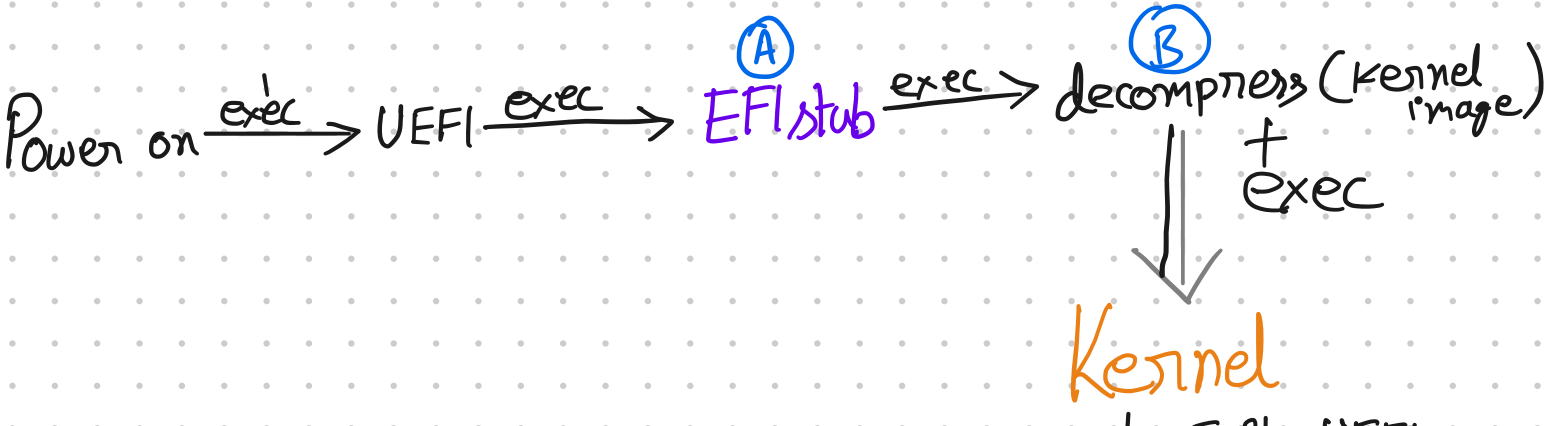


On recent Linux: Vmlinuz

(A) → PE header + EFI stub – logic to load ELF binary

(B) → ELF binary + bz2 or gzip decompressor

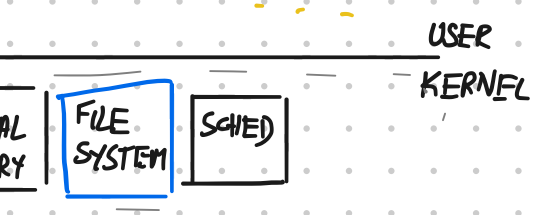
(C) → Compressed kernel image



kernel

① Initializes system
↳ int, syscall, etc.
→ Move away from identity page table

② Use device tree to load device drivers



③ Mount root file system
/
↳ Loads FS, etc.

PO Exec → UEFI exec → Bootloader exec → kernel exec → Init

Init (PID=1)

↳ Forked & execed by kernel — UID = ROOT

① Finish initializing system
→ Display res.
→ Network
→ Services
→ ...

② Fork+exec Session Manager

PO $\xrightarrow{\text{Exec}}$ UEFI $\xrightarrow{\text{exec}}$ Bootloader $\xrightarrow{\text{exec}}$ kernel $\xrightarrow[\text{exec}]{\text{fork}}$ init $\xrightarrow[\text{exec}]{\text{fork}}$ Session Manager

For simplicity

→ login(1) = Run as root (UID=1).

① Check credentials
↓ If correct

② Fork ↳ Call setuid
setsid

→ Exec. user's shell

Now you can do stuff.

Unrolling - shell exit ⟶ login(1)
↳ Cleans up session
→ Exits

login(1) exit ⟶ init restarts login(1)

Shutdown ⟶ Tells init to exit
↳ Kills all

child processes.

Monolithic vs Micro Kernel