MEMOS-2

CS552 Operating Systems

11/10/2023

Anton Njavro

Overview

- In MEMOS-1 we wrote our own MBR, now we utilize GRUB.
- What GRUB does for us:
  - Enumerates resources.
  - Switch to 32-bit Protected Mode.
  - Finds kernel executable (ELF).
  - Loads it at 0x100000 (1MB), passes information to kernel according to Multiboot and jumps to 0x100000.

Starting Point

- Need to know machine state when GRUB calls into kernel.
    - 32-bit protected mode.
    - Segmentation.
    - Can access data/code anywhere between 0x0 and 4GB barrier.
- Environment:
    - Ring-0
    - No BIOS
- Program:
    - GRUB expects kernel as an ELF binary with multiboot header.

X86 Protected-Mode

- Access to 32-bit instructions and registers.
  - Still can access smaller parts of the register.
- 4GB of memory is addressable.
  - Segmentation provided by GDT.
  - Virtual memory/Paging (not needed yet).
  - Certain privilege levels assigned to each segment.
- Several virtual address spaces, each has maximum 4 GB of addressable memory.
- Entering protected mode:
  - Disable interrupts.
  - Enable A20 Line.
  - Load GDT

```
Physical address = Segment Base (Found from the descriptor GDT[A]) + B
```

How does GRUB help?

- Performs a switch from real to protected mode.
    - There’s 4GB of linear memory space available to kernel.
- Finds kernel as ELF file and loads it at 0x100000.
- Checks for Multiboot header and runs according to information gathered there.
- Starts kernel execution and passes the data in accordance with Multiboot specification.

Multiboot

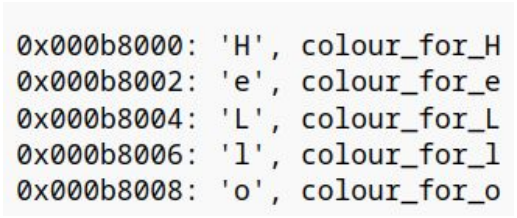
- *“Basically, it specifies an interface between a boot loader and a operating system, such that any complying boot loader should be able to load any complying operating system.”*
- Kernel must define **header** early in its binary file:
  - Specify information bootloader must pass.
  - Verify that binary file is Multiboot-compliant kernel.
- Multiboot defines desired state before kernel invocation.
- Defines boot information format:
  - Data structure passed to OS by bootloader.
  - Address in %EBX
  - Of advisory nature only.

Now you C me!

- We finally get to use C, however still no fancy external/GCC built-in libraries.
    - Compiler flags: **-fno-builtin -nostdinc**
- Calling C funCtion:
    - Caller rules:
        - Save contents of *caller-saved* registers (EAX,ECX,EDX). Push their values onto stack.
        - Pass parameters to subroutine by pushing them onto a stack (inverted order).
        - **call** instruction places return address on top of the stack and branches to subroutine code.
        - Restore previous values/stack, and return value is expected in EAX.
    - Callee rules:
        - Push value of EBP and copy ESP into EBP. Base pointer used for callee reference to arguments.
        - Allocate local variables on stack at known distance from EBP.
        - Save values of *calee-saved* registers used by subroutine (EBX,EDI,ESI)

Video RAM

- No more BIOS!
- We can ask GRUB for specific video mode.
- Text-based VGA buffer is mapped to memory 0xB8000 in main memory.
- We manipulate display by changing each word (16 bits).
- ASCII code byte and attribute code byte.
- Consult OS-Dev for printing tutorial.



```
0x000b8000: 'H', colour_for_H
0x000b8002: 'e', colour_for_e
0x000b8004: 'L', colour_for_L
0x000b8006: 'l', colour_for_l
0x000b8008: 'o', colour_for_o
```