EEC420 - Operating Systems

This is an introductory course to operating systems (OS) - we use x86 systems as reference (ARM may be included in the future).

Practical Exercise

General Introduction

The core of an Operating System (OS) is the kernel.

OS Kernel

- system management software
  - provides hardware access to application software (user programs)
- four primary tasks:
  - process/thread management (multitasking)
  - memory management
  - disk management
  - peripheral management (I/O)

Bootstrap Process

Bootstrapping

- common issue for general purpose computers
  - software usually reside in secondary memory
  - how do we load software to primary memory?
- bootstrap process helps load an OS
  - usually in multiple stages
  - first stage as simple as possible
  - small footprint in system address space?
- processors on powerup (or reset)
  - have specific address to start execution
  - e.g. for IA32 0xFFFFFFF0 (32-bit address)
  - legacy from 8086 0xFFFF0 (20-bit address)
- non-volatile memory placed at that address
  - ROM → EEPROM → NVRAM
  - contains first stage bootstrap code
  - e.g. for IA32 we have BIOS

BIOS (x86) Bootstrap

- BIOS bootloader
  - for cold-boot, perform POST (power-on self-test)
  - iterate boot device list
  - attempt to load first sector (next stage bootstrap)
- load address is at 0x7C00
  - first sector (512 bytes) loaded here
  - must end with boot signature 0x55 0xAA (else HALT)
  - on hard-disks, usually have MBR
    - specifies partitions, so effective code size <512
    - loads first sector of boot partition
  - very limited, but can utilize BIOS functions
- Master Boot Record (MBR) - Read more @ wikipedia
  - first sector of a partitioned storage
  - classic mbr:
    - 446 bytes executable code
    - 64 bytes partition entries (4 primary partitions)
    - 2 bytes boot signature (0x55 0xAA)
  - modern mbr
    - 218 bytes executable code
    - 2 bytes (always 0x00?)
    - 4 bytes disk timestamp
    - 216 bytes executable code
    - 4 bytes disk signature
    - 2 bytes (always 0x00?)
    - 64 bytes partition entries (4 primary partitionss)
    - 2 bytes boot signature (0x55 0xAA)
  - superseeded by GUID partition table (GPT) - Read more @ wikipedia

BIOS (x86) Functions

- invoked using software interrupts
  - as interrupt service (handler) routine
- e.g. int 0x13 is for disk i/o routine
  - register ah used as function select
    - e.g. ah = 0x02 is to read sector from disk
  - register dl should have drive number
    - 0x80 is first hard disk
  - limitations: int 0x13 supports disk size <8GB
- extended bios provide more functions
  - extended disk excess

BIOS (x86) Operating Modes

- IA32 processors support dual operating mode
  - Real mode & protected mode
- Real mode
  - legacy 16-bit operating environment from 8086
  - segmented 20-bit address space
    - maximum 1MB address

    - an address is made up of *segment* and *offset*
        - *segment* - 16-bit segment selector (cs,ds,ss)
        - *offset* - 16-bit offset within a segment
        - address = *segment* « 4 + *offset*
    - segments obviously overlaps by 64k - 16
        - multiple address can refer to same physical address
        - e.g. 0x0000:0x7c00 and 0x07c0:0x0000 refer to same location
    - IA32 provides ways to switch operating mode
- Protected mode
    - 32-bit operating mode
    - segmented OR flat memory address
    - provides virtual memory (e.g. paging)

From BIOS to OS

- BIOS boots to real mode
    - newer ones switch to protected mode to get more features
    - subsequent bootstrap code should check or assume in real mode
    - usually loads bootloaders (e.g. LILO for Linux, NTLDR for Windows)
- BIOS functions no longer available in protected mode
    - only works in real mode
    - some OS temporarily drop to real mode to utilize BIOS functions!
    - so, OS in protected mode drives hardware directly (device drivers!)
    - Some OS (Windows) maintains backward compatibility
        - place WinAPI in place of legacy BIOS
        - older DOS programs can still invoke old interrupts services
    - OS (here, most of the time) means bootloaders
        - bootloaders can be part of an OS
        - bootloaders load actual OS kernel

BIOS to UEFI Transition

- BIOS is being replaced by UEFI
    - Unified Extensible Firmware Interface
- UEFI features
    - modular bootloading
    - boot off really large hard disks (>2TB)
    - hardware drivers in early boot
        - useful ones like graphics and networking
    - built-in shell (diagnostic/maintenance)
- utilize GPT instead of MBR?
- requires a special UEFI partition
    - can have multiple firwares

Hardware (x86) for OS Kernel

- need to separate system (kernel) & application (user) software
- hardware-level privilege

IA32 Privilege Levels

- called protections rings (4)
  - represents privilege levels
  - rings 0 to 3 (0 - highest, 3 - lowest)
  - most only use 2 (0 - kernel, 3 - user)

IA32 Control Registers

- designated crX (e.g. cr0, cr2, cr3)
  - cr0 - enable/disable protected mode
  - cr3 - used for memory paging (virtualization)
  - cr2 - page fault address
- these should only be accessed in ring 0
  - at boot, x86 in real-mode (no privilege levels)
  - modify cr0 - switch to protected mode
  - first code in protected mode is in ring 0
  - setup virtual memory (paging)
  - setup self to run in ring 0, others in ring 3!

IA32 Protected Memory

- memory access in protected mode
  - can access >1MB, use paging
  - needs Global Descriptor Table (GDT)
    - setup memory size and location
    - also defines protection levels (ring)
  - can also setup local descriptor table
    - not used much?
- create 2 main memory segments
  - memory only for ring 0 - kernel space
  - memory allowed for ring 3 (all) - user space
  - each memory space has own code/data segment
  - stack segment defined separately (per process, not system?)

IA32 Task Management

- basic hardware level multitasking support
  - kernels may implement software-level
- defined by Task State Segment (TSS) data structure
  - also contains protection levels