Lab1 系统引导

徐圣斌

kingxu@smail.nju.edu.cn

2019-3-5

实验目的

- 学习在 Linux 环境下编写、调试程序，初步掌握 Shell、 Vim、GCC、Binutils、Make、QEMU、GDB 的使用
- 学习 AT&T 汇编程序的特点
- **理解系统引导程序的含义，理解系统引导的启动过程**

实验内容

- 在实模式下实现一个 Hello World 程序
  - 在实模式下在终端中打印 Hello, World!
- 在保护模式下实现一个 Hello World 程序
  - 在保护模式下在终端中打印 Hello, World!
- **在保护模式下加载磁盘中的 Hello World 程序并运行**
  - 从实模式切换至保护模式，在保护模式下读取磁盘 1 号扇区中的 Hello World 程序至内存中的相应位置，跳转执行该 Hello World 程序，并在终端中打印 Hello, World!

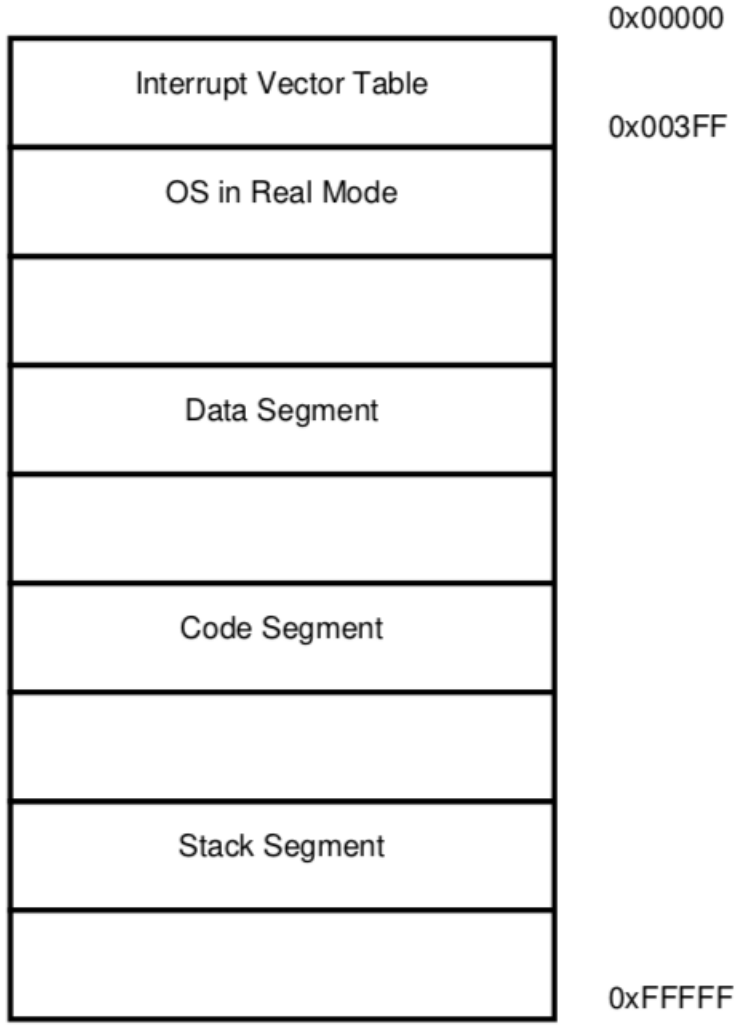
8086简介

- 8086 的寄存器集合
  - 通用寄存器(16 位):AX，BX，CX，DX，SP，BP，DI，SI
  - 段寄存器(16 位):CS，DS，SS，ES
  - 状态和控制寄存器(16 位):FLAGS，IP
- 寻址空间与寻址方式
  - 采用实地址空间进行访存，寻址空间为 2^{20}
  - 物理地址 = 段寄存器 << 4 + 偏移地址
  - CS=0x0000:IP=0x7C00 和 CS=0x0700:IP=0x0C00 以及 CS=0x7C0:IP=0x0000 所寻地址是完全一致的

8086简介

- 8086的中断
  - 中断向量表存放在物理内存的开始位置(0x0000至0x03FF)
  - 最多可以有 256 个中断向量
  - 0x00 至 0x07 号中断为系统专用
  - 0x08 至 0x0F，0x70 至 0x77 号硬件中断为 8259A 使用

8086简介



- 一个实模式下用户程序的例子
  - 各个段在物理上必须是连续的
  - 装载程序在装入程序时需要按照具体的装载位置设置CS，DS，SS

8086简介

- 安全性问题
  - 程序采用物理地址来实现访存，无法实现对程序的代码和数据的保护
  - 一个程序可以通过改变段寄存器和偏移寄存器访问并修改不属于自己的代码和数据
- 分段机制本身的问题
  - 段必须是连续的，从而无法利用零碎的空间
  - 段的大小有限制(最大为 64KB)，从而限制了代码的规模

80386保护模式

- 保护模式带来的变化
  - 通用寄存器(从 16 位扩展为 32 位):EAX，EBX，ECX，EDX，ESI，EDI，EBP，ESP
  - 段寄存器(维持 16 位):CS，DS，SS，ES，FS，GS 状态和控制寄存器(32/64 位):EFLAGS，EIP，CR0，CR1， CR2，CR3
  - 系统地址寄存器:GDTR，IDTR，TR，LDTR
  - 调试与测试用寄存器:DR0，···，DR7，TR0，···，TR7

80386保护模式

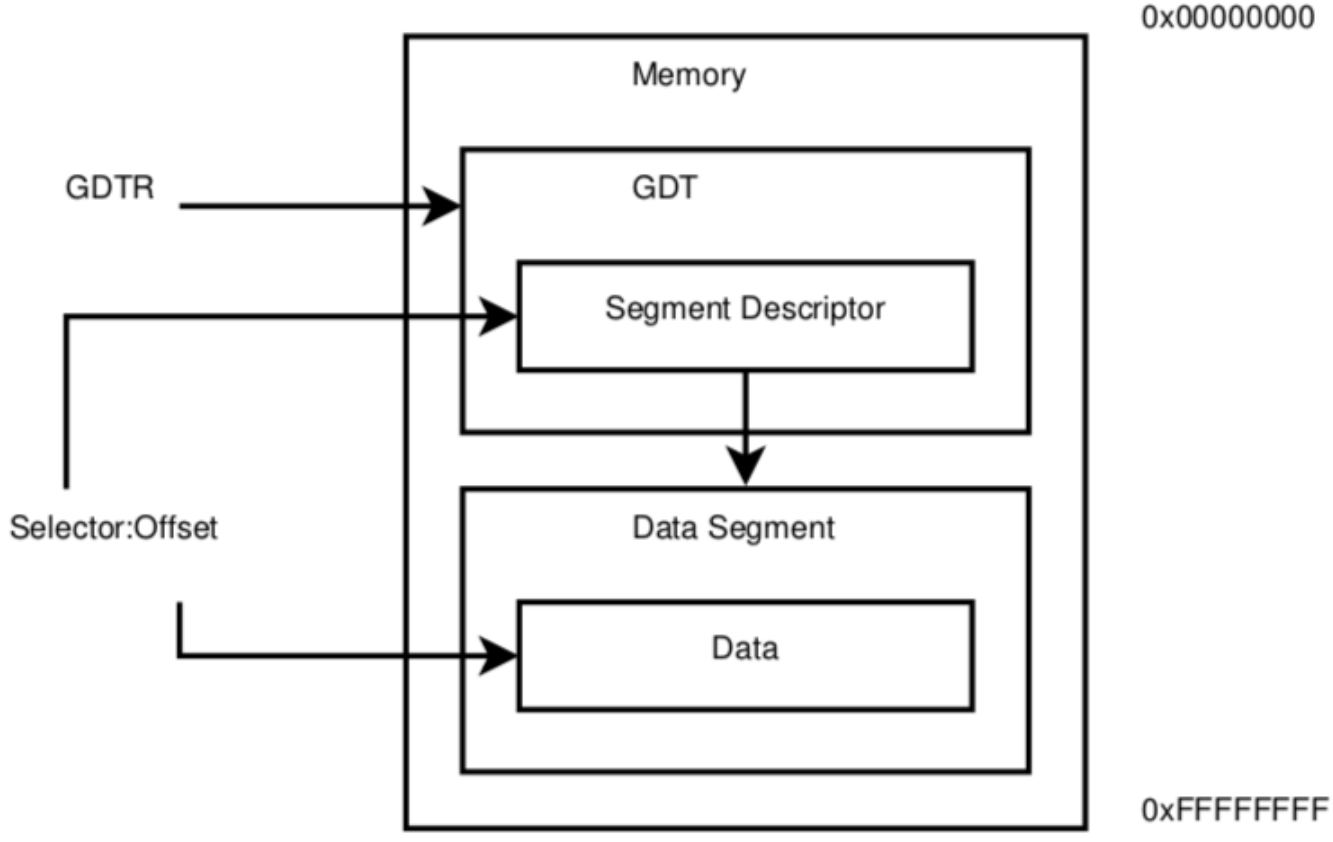
	8086 寄存器	80386 寄存器
通用寄存器	AX，BX，CX，DX SP，BP，DI，SI	EAX，EBX，ECX，EDX ESI，EDI，EBP，ESP
段寄存器	CS，DS，SS，ES	CS，DS，SS，ES，FS，GS
段描述符寄存器	无	对程序员不可见
状态和控制寄存器	FLAGS，IP	EFLAGS，EIP CR0，CR1，CR2，CR3
系统地址寄存器	无	GDTR，IDTR，TR，LDTR
调试寄存器	无	DR0，···，DR7
测试寄存器	无	TR0，···，TR7

80386保护模式

- 寻址方式的变化
  - 在保护模式下，分段机制是利用一个称作段选择子的偏移量到全局描述符表中找到需要的段描述符，而这个段描述符中就存放着真正的段的物理首地址，该物理首地址加上偏离量即可得到最后的物理地址
  - 一般保护模式的寻址可用 0xMMMM:0xNNNNNNNN 表示，其中 0xMMMM 表示段选择子的取值，16 位(其中高 13 位 表示其对应的段描述符在全局描述符表中的索引，低 3 位 表示权限等信息)，0xNNNNNNNN 表示偏移量的取值，32 位
  - 段选择子为 CS，DS，SS，ES，FS，GS 这些段寄存器

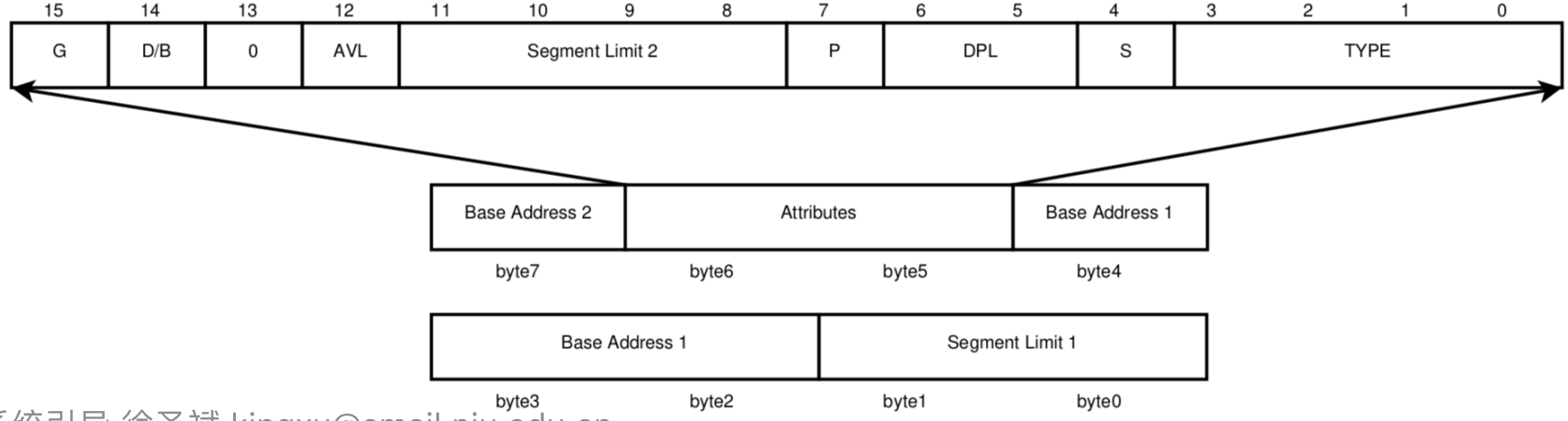
80386保护模式

- 80386 及以后的处理器专门设计了一个寄存器 GDTR (Global Descriptor Table Register)用于存储全局描述符表在内存中的基地址以及其表长界限



80386保护模式

- 段描述符
  - 每个段描述符为 8 个字节，共 64 位
  - 段基址为第 2，3，4，7 字节，共 32 位
  - 段限长为第 0，1 字节及第 6 字节的低 4 位，共 20 位，表示 该段的最大长度
  - 当属性 G 为 0 时，20 位段限长为实际段的最大长度(最大为 1MB);当属性 G 为 1 时，该 20 位段限长左移 12 位后加上 0xFFF 即为实际段的最大长度(最大为 4GB)



80386保护模式

- 段描述符的属性
  - D/B:对于不同类型段含义不同
    - 在可执行代码段，该位叫做 D 位，D 为 1 使用 32 位地址和 32/8 位操作数，D 为 0 使用 16 位地址和 16/8 位操作数
    - 在向下扩展的数据段中，该位叫做 B 位，B 为 1 段的上界为 4GB，B 为 0 段的上界为 64KB
    - 在描述堆栈段的描述符中，该位叫做 B 位，B 为 1 使用 32 位 操作数，堆栈指针用 ESP，B 为 0 使用 16 位操作数，堆 栈指针用 SP

80386保护模式

- 段描述符的属性
  - AVL:Available and Reserved Bit，通常设为 0
  - P:存在位，P 为 1 表示段在内存中
  - DPL:描述符特权级，取值 0-3 共 4 级;0 特权级最高，3 特权级最低，表示访问该段时 CPU 所处于的最低特权级，后续实验会详细讨论
  - S:描述符类型标志，S 为 1 表示代码段或数据段，S 为 0 表示系统段(TSS，LDT)和门描述符

80386保护模式

- 段描述符的属性
  - TYPE:当 S 为 1，TYPE 表示的代码段，数据段的各种属性如下表所示

bit 3	Data/Code	0 (data)
bit 2	Expand-down	0 (normal) 1 (expand-down)
bit 1	Writable	0 (read-only) 1 (read-write)
bit 0	Accessed	0 (hasn't) 1 (accessed)

bit 3	Data/Code	1 (code)
bit 2	Conforming	0 (non-conforming) 1 (conforming)
bit 1	Readable	0 (no) 1 (readable)
bit 0	Accessed	0 (hasn't) 1 (accessed)

80386保护模式

- 安全性问题
  - x86 平台 CPU 有 0、1、2、3 四个特权级，其中 level0 是最高特权级，可以执行所有指令
  - level3 是最低特权级，只能执行算数逻辑指令，很多特殊操 作(例如 CPU 模式转换，I/O 操作指令)都不能在这个级别下进行
  - 现代操作系统往往只使用到 level0 和 level3 两个特权级，操作系统内核运行时，系统处于 level0 (即 CS 寄存器的低两 位为 00b)，而用戶程序运行时系统处于 level3 (即 CS 寄存器的低两位为 11b)

80386保护模式

- 安全性问题
  - x86 平台使用 CPL、DPL、RPL 来对代码、数据的访存进行特权级检测
    - CPL(current privilege level)为 CS 寄存器的低两位，表示当前指令的特权级
    - DPL(discriptor privilege level)为描述符中的 DPL 字段，表示访存该内存段的最低特权级(有时表示访存该段的最高特权级)
    - RPL(requested privilege level)为 DS、ES、FS、GS、SS 寄存器的低两位，用于对 CPL 表示的特权级进行补充
    - 一般情况下，同时满足 CPL ≤ DPL，RPL ≤ DPL 才能实现对内存段的访存，否则产生 #GP 异常

系统的启动

- BIOS
  - 在 PC 启动时，首先会在实模式下运行 BIOS
  - BIOS 进行系统自检等工作
  - 完成上述工作后，BIOS 会读取磁盘的 MBR(Master Boot Record，磁盘的 0 柱面，0 磁头，0 扇区的 512 字节)到内存的 0x7C00(被装入的程序一般称为 Bootloader)，紧 接着执行一条跳转指令，将 CS 设置为 0x0000，IP 设置 为 0x7C00，运行被装入的 Bootloader

系统的启动

- Bootloader
  - 代码框架 lab/bootloader/中包含 start.s(AT&T 语法的 x86 汇编)以及 boot.c(C 语言)两个源文件
    - start.s 作用是将 80386 处理器从实模式切换至 32 位的保护模式，并完成各个段寄存器以及栈顶指针 ESP 的初始化
    - boot.c 作用是将存储在 MBR 之后的磁盘扇区中的程序加载至内存的特定位置并跳转执行

系统的启动

- 实模式切换保护模式
  - 关闭中断
  - 打开 A20 地址线
  - 加载 GDTR 寄存器
  - 设置 CR0 寄存器的 PE 位(第 0 位)为 1(表示进入保护模式)
  - 长跳转进入保护模式代码(由于无法直接修改 CS 寄存器，这里通过长跳转修改)
  - 初始化 DS，SS，ES，FS，GS 这些段寄存器，初始化 ESP

作业提交

- 本次作业仅需提交保护模式下加载磁盘中的 Hello World 程序并运行的相关源码与报告
- **截止时间:2017-3-18 23:59:59**
- 如果你无法完成实验, 可以选择不提交, 作为学术诚信的奖励, 你将会获得10%的分数; 但若发现抄袭现象, 抄袭双方(或团体)在本次实验中得0分，后续可能有其他惩罚
- **本实验的最终解释权由助教所有**