Lab1简介

2008-2-19

Lab1 Outline

- 时间安排
- 实习任务
- 实验环境搭建
- **AT&T**汇编
- **GNU** 工具
- 第一周任务
- **C**语言程序结构
- 第二周任务

Lab 1 时间安排

- **Lab1时间：2月20日至3月4日**
- 第一周：
  - 安装**Linux、Bochs，熟悉环境，完成Part1**
- 第二周：
  - 完成**Part2、3，各组提交代码、文档**

Lab 1 任务

- 安装并熟悉Bochs实验环境，阅读相关背景资料，学习常用的Bochs调试指令
- 熟悉AT&T汇编
- 了解PC从加电到加载内核的整个过程
- 理解内核在内存中的布局
- 实现终端字符打印
- 实现函数调用堆栈跟踪函数

Lab 1　资料清单

- 背景知识：x86 and PC architecture.pdf
- IA32的官方资料：
  - System Programming Guide.pdf
  - Instruction Set Reference.pdf
  - Basic Architecture.pdf
- 汇编语言：
  - PC Assembly Language.pdf
  - 80x86汇编语言程序设计教程　杨秀文等编著　清华大学出版社 10.1
- 电子书：
  - Linux内核0.11完全注释
  - Linux内核源代码情景分析(上)

Lab 1　资料阅读要求

- x86 and PC architecture.pdf　了解背景知识
- Linux内核0.11完全注释
  - 阅读2.10节 Linux/Makefile文件
  - 参考阅读14.2-14.4节　bochs介绍、磁盘映像文件制作
- Linux内核源代码情景分析
  - 阅读1.5节　了解AT&T汇编
- 资料下载：http://os.pku.edu.cn
- 更多资料的可以到MIT 的开放课程网站下载 http://ocw.mit.edu/OcwWeb/index.htm

Lab 1 实习题目

- **Exercise 1~12：**
  - 必做
- **Challenge 2：控制台彩色打印**
  - 必做
- **Challenge 1：光盘启动**
  - 选做

Lab1 Outline

- 时间安排
- 实习任务
- 实验环境搭建
- **AT&T**汇编
- **GNU** 工具
- 第一周任务
- **C**语言程序结构
- 第二周任务

实验环境及工具

- 一台X86 PC机
- 较新的Linux操作系统，例如Ubuntu，Fedora等，双系统和虚拟机均可
  - 实验需在Linux环境下完成
- GNU工具链
- Xwindows开发包
  - Xorg-dev
- Bochs模拟器

Bochs安装和使用 - Outline

- 关于Bochs
- Bochs安装环境
- Bochs源码下载、解压
- Bochs配置
- Bochs源码编译
- 安装到系统目录
- Bochs的使用

关于Bochs

- C++ 开发, 开源
- IA-32 (x86) PC 模拟器:
  - Intel x86 CPU 、通用 I/O 设备、可定制的 BIOS
- 可移植:
  - compiled to emulate: 386, 486, Pentium Pro, AMD64 CPU
- 支持操作系统:
  - Linux, Windows 95, DOS, Windows NT 4, FreeBSD, MINIX
- written by Kevin Lawton

Bochs安装环境

- 安装Linux操作系统
  - Linux 2.4以上内核
  - 图形界面
- 用root管理员登录Linux
  - 新建一个目录，在该目录下完成Bochs安装过程

Bochs源码下载

- http://os.pku.edu.cn
- 将源码下载到安装目录下
  - **bochs-2.2.6.tar.gz**
- http://bochs.sourceforge.net/
  - **bochs 2.2.6 released on Jan. 29, 2006**

解压缩Bochs源码

- 键入命令

  tar –xzf bochs-2.2.6.tar.gz

- 产生bochs-2.2.6目录
- 进入该目录

Bochs模拟器配置

- 配置：

  ./configure --enable-disasm --enable-debugger --enable-new-pit --enable-all-optimizations --enable-4meg-pages --enable-global-pages --enable-pae --enable-sep --enable-cpu-level=6 --enable-sse=2 --disable-reset-on-triple-fault --with-all-libs

- **--enable-disasm***
  - 使得**Bochs**可以反汇编机器指令，**disasm**是**disassemble**的缩写
- **--enable-debugger***
  - 使得用户可以使用**Bochs**自带的调试器进行调试
- **--enable-new-pit**
  - 使用一个新的更加完善的**PIT**模块

Bochs模拟器配置

- **--enable-all-optimizatinons**
  - 打开所有速度优化选项
- **--enable-4meg-pages*C**
  - 支持**4M**页面扩展
- **--enable-global-pages**
  - 支持全局页面特性。避免经常使用的页面从**TLB**中移出
- **--enable-pae**
  - 支持物理地址扩展
- **--enable-sep *C**
  - 支持**SYSENTER/SYSEXIT**指令

Bochs模拟器配置

- **--enable-cpu-level=6 ***
  - 支持**X86 i686**
- **--enable-sse=2**
  - 支持**SSE2**
- **--disable-reset-on-triple-fault**
  - 不支持三次错误自动重启
- **--with-all-libs***
  - 使用所有的库
- **--enable-cdrom**
  - 支持光驱

编译安装Bochs源码

- 键入命令 **make**
  - 系统将在**Bochs**安装目录下编译**Bochs**源码，生成文件不会复制到系统目录
- 如果发生错误，需要重新编译，编译之前键入命令**make clean**
- 键入命令 **make install** 将**Bochs**安装到系统目录

Bochs的使用

- 参考Bochs自带的帮助文档
  - doc目录下
- man bochs
- 关于配置的详细说明
  - /usr/local/share/doc/bochs/bochsrc-sample.txt
- http://os.pku.edu.cn
  - 提供一份简要命令指南

Lab1 Outline

- 时间安排
- 实习任务
- 实验环境搭建
- **AT&T**汇编
- **GNU** 工具
- 第一周任务
- **C**语言程序结构
- 第二周任务

AT&T汇编语法主要规则(1)

- 操作数顺序
  - **ops source, target**
  - 操作数顺序是**AT&T**语法的主要特征
- 指令后缀
  - **movb, movw, movl, b代表byte(8bit), w代表word(16bit), l代表long(32bit)**
- 内存寻址
  - **displacement(base, index, scale)引用[base+index*scale+displacement]处的内存**
  - **movl task_struct(%ebx), %eax**
  - **movl 4(%ebp), %eax**
  - **mov 4, %eax**

AT&T汇编语法主要规则(2)

- 寄存器表示
  - **%eax, %bx**
- 立即数表示
  - **$0x123, $19, $0375**
- 变量引用
  - **movl variable, %eax; movl $variable, %eax**
  - **variable**在内存中有一个地址值，汇编器简单的将地址值替换变量名

AT&T汇编实例

- inb $0x64,%al
- testb $0x2,%al
- jnz seta20.2
- movb $0xdf,%al
- outb %al,$0x60
- movl %cr0, %eax
- orl $CR0_PE_ON, %eax
- movl %eax, %cr0

目标寄存器或端口

立即数或标号

AT&T汇编语法主要规则(3)

- 绝对转移与相对转移
  - 绝对转移指令用于C中的函数指针跳转
    - **movl $do_pgfault, %eax**
    - **jmp *%eax;**
  - 相对转移指令中将操作数作为目标地址与当前EIP的差值，将操作数与EIP相加得到目标地址
    - **jmp .-100**
    - **jmp do_pgfault**
  - 汇编器会根据跳转范围自动生成相对跳转指令的偏移量

AT&T汇编语法主要规则(4)

- 16位指令与32位指令
  - 助记符相同，但机器码不同
  - CPU工作在32位模式下时不能执行16位指令，反之亦然
  - .code16和.code32指示符表明以下代码按照16位还是32位汇编成机器码

Lab1 Outline

- 时间安排
- 实习任务
- 实验环境搭建
- **AT&T**汇编
- **GNU** 工具
- 第一周任务
- **C**语言程序结构
- 第二周任务

Linux环境开发工具

- GCC编译器前端
  - 常用选项
    - -Wall 开启所有警告信息
    - -g加入调试信息
    - -O2打开2级优化
    - -S将C程序编译成汇编文件
- AS汇编器
- LD链接器
- OBJDUMP查看目标文件信息

Lab1 Outline

- 时间安排
- 实习任务
- 实验环境搭建
- **AT&T**汇编
- **GNU** 工具
- 第一周任务
- **C**语言程序结构
- 第二周任务

Bochs模拟器—使用

- 常用命令：
  - 调试： **vb addr, lb addr, b(pb) addr**
  - 运行： **s, c**
  - 查看内存： **x(xv)/nuf addr, xp/nuf addr**
  - 查看寄存器： **info r, info eflags**
  - 查看**GDT**内容： **info gdt [a[b]]**
  - 查看**CPU**状态： **info cpu, dump_cpu**

Bochs调试

- 实习源码编译后，会在obj目录的相应位置出现*.asm汇编文件，在其中找到希望跟踪的C代码对应的汇编代码地址，设置断点跟踪
- 实现Lab1中简易Shell，完成自制调试功能

BIOS启动过程

- 知识点
  - 实模式下的寻址方式
  - PC内存布局情况及其形成历史
- 主要步骤
  - 用Bochs查看PC执行的第一条指令
  - 跟踪BIOS中若干条指令（熟悉bochs的单步跟踪，断点设置的使用）

Part 1 PC Bootstraps

- **Exercise1：X86汇编基础**

  阅读**PC Assembly Language**书中（以下部分可以忽略：第一章**1.3.5**之后部分，第**5**、**6**章，**7.2**）

  阅读**Brennan's Guide**中的**Syntax**部分

Bochs模拟器—使用

- **Exercise 2：** 熟悉bochs基本命令的使用
  - 从0xf:0xfff0开始单步跟踪BIOS的执行
  - 在初始化位置0x7c00设置实地址断点，测试断点正常
  - 从0x7c00开始跟踪代码运行，将单步跟踪反汇编得到的代码与boot/boot.S和obj/boot/boot.asm进行比较
  - 在obj目录下自己找一个内核中的代码位置，设置断点并进行测试

BIOS启动过程

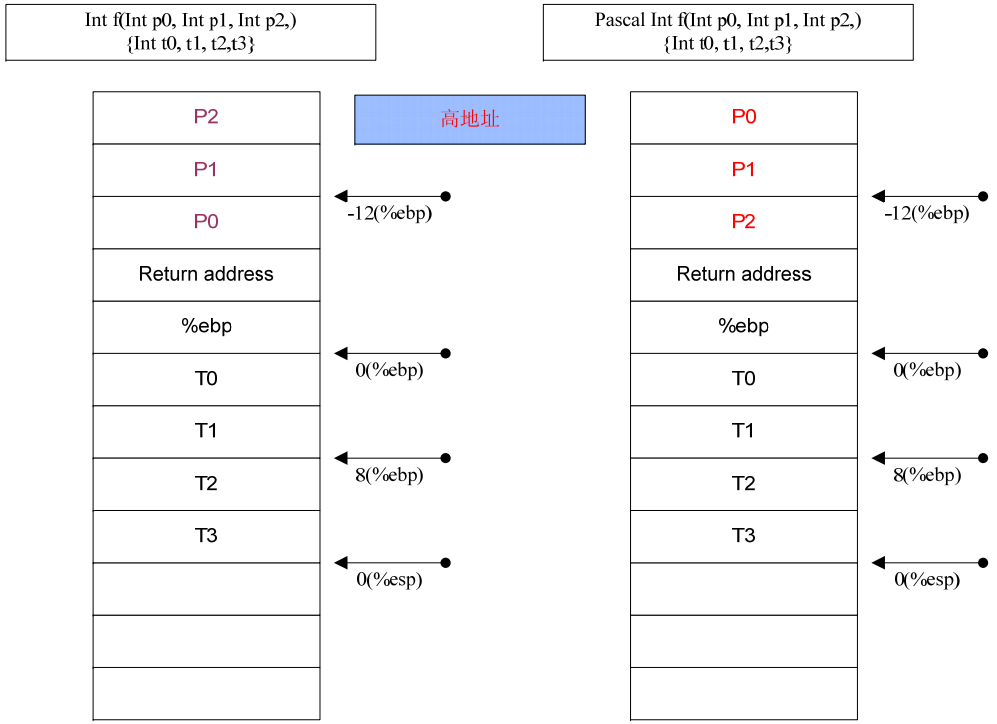
- **Exercise 3：** 查看BIOS中前5~6条命令的内容，参考Phil Storrs I/O Ports Description大致了解这些命令的作用

Lab1 Outline

- 时间安排
- 实习任务
- 实验环境搭建
- **AT&T**汇编
- **GNU** 工具
- 第一周任务
- **C**语言程序结构
- 第二周任务

子程序结构(1)

- 基于堆栈的子程序结构



子程序结构(2)

- 堆栈对齐
  - IA-32中堆栈是4字节对齐，IA-64中可以是8字节对齐
  - 压入堆栈的数据必须对齐，byte数据在压入堆栈之前必须扩展为四字节
- 堆栈生长方向
  - IA-32中堆栈向下生长，压入数据堆栈指针减少，弹出数据堆栈指针增加

子程序结构(3)

- 堆栈指针
  - IA-32中堆栈指针指向栈顶第一个可用地址
- C语言函数结构
  - 活动记录
    - 函数参数、返回地址、局部变量
  - 变量上下文环境
    - C语言两层函数结构，不是全局变量，便是局部变量
    - 全局变量在数据段，局部变量位于堆栈

子程序结构(4)

- 参数传递方式
  - 参数传递顺序
    - Pascal从左向右压入参数
    - C从右向左压入参数
  - 清除堆栈上传递的参数
    - Pascal由callee函数清除参数
    - C由caller清除参数
  - C方式的优点
    - 可变参数列表容易实现
    - 汇编语言编写C函数
      - 简化了汇编语言子程序的复杂性，考虑需要使用可变参数的汇编子程序

子程序结构(5)

- 返回值传递方式
  - 简单类型，指针类型，**%eax**寄存器或者**%eax**和**%edx**
  - 结构体类型，函数原型中返回类型作为形式参数列表的一部分；**struct my f(int k)**将转化为**void f(struct my *p, int k);** 实际调用时将返回对象的赋值目标地址作为参数

数据存储(1)

- 数据对齐
  - 系统存储结构有关，页对齐
    - 段起始地址要求页对齐，便于缺页处理
  - 减少Cache miss ， Cache line 对齐
    - 结构体常要求cache line对齐，提高访问效率
  - 减少内存读写次数，字对齐
    - 普通变量要求字对齐，减少内存读写次数

数据存储(2)

- 可执行程序在内存中的结构
  - **.text** 代码段
    - 存放代码
  - **.data** 数据段
    - 存放数据
  - **.rodata** 只读数据段
    - 存放只读数据，如C中的字符串和其它常量
  - **.bss** 未初始化数据段
    - 存放未初始化的全局或静态数据，这些内存必须初始化为0

C语言与汇编语言接口(1)

- 标识符
  - 较早的C编译器将C语言先编译成汇编语言，对应的汇编标识符有前导下划线
  - C中引用汇编标识符
    - 汇编程序定义供C使用的标识符时必须加下划线
  - 汇编引用C中的标识符
    - C中定义的标识符在汇编中引用时须加下划线
  - C中定义变量 int abc，汇编程序引用该变量方法为_abc
  - 汇编程序中定义变量_def，C中引用为def，如果汇编定义变量def，则C中无法引用
  - 较新的编译器有控制选项可以取消这种约定

C语言与汇编语言接口(2)

- 数据类型
  - 汇编中的word类型为16位，尽管CPU字长为32位
- 函数原型
  - C中声明函数原型，规定参数类型和个数，汇编中只需声明标号即可，但是编写汇编代码是必须参考C中的函数原型

C语言与汇编语言接口(3)

- 寄存器功能分组
  - 活动记录寄存器
    - %ebp
  - caller-saved寄存器
    - %eax, %ebx, %edx, %ecx
  - callee-saved寄存器
    - %ebp, 用到的其它非caller-saved寄存器
  - 返回值寄存器
    - %eax/%edx
- 局部可变数组
  - C99标准新增内容，允许在堆栈上开辟动态数组

Lab1 Outline

- 时间安排
- 实习任务
- 实验环境搭建
- **AT&T**汇编
- **GNU** 工具
- 第一周任务
- **C**语言程序结构
- 第二周任务

Part 2 Boot Loader

- 主要步骤
  - 用bochs跟踪boot loader第一条指令的地址和内容
  - 阅读boot/boot.S，了解如何为boot/main.c预留堆栈，从实模式到保护模式的切换的步骤
  - 阅读boot/main.c，了解如何加载ELF文件
- 知识点
  - 从实模式到保护模式的切换
  - Boot Loader的功能
  - 了解ELF文件格式（boot/main.c中用到的基本格式）
  - 链接地址和装载地址的区别
  - x86段式寻址机制（GDT的设置）
- 主要变化
  - 内核映像使用了ELF格式取代a.out格式

ELF文件格式

- **ELF (Executable and Linking Format)**
- 三种类型：
  - Relocatable file
  - **Executable file（boot/main.c中使用的类型）**
  - Shared object file
- 详细的格式定义参见：
  - the ELF specification：
    http://pdos.csail.mit.edu/6.828/2005/readings/elf.pdf

Boot Loader – 光盘启动

- 光盘一个扇区为**2KB**，故在**boot/sign.pl**中，放置**0x55AA**的位置需要调整
- 在硬盘占据**IDE0:0**时，**boot/main.c:void readsect()**中使用的端口需要变化
- 使用**mkisofs**生成光盘镜像
- 编译**Bochs**时加入光驱支持**(--enable-cdrom)**
- 需要根据**.bochsrc**的格式要求加入光盘镜像的对应设置

- 具体可参见实习要求中相应的介绍和链接

Part 3 The Kernel

- 主要内容
  - 理解内核在内存中的布局
  - 实现终端字符打印
  - 理解函数调用时的堆栈情况

内核布局

- 主要步骤
  - 阅读**kern/entry.S**文件，了解堆栈设置和**ebp**的初始化
- 知识点
  - 理解段式寻址机制如何把内核映像从低端物理地址映射到高端虚拟地址
  - 为什么要采取这种内存布局方式？
  - 了解**link**地址和**load**地址的区别

内存布局 – inc/memlayout.h

- **0x0~0xefffffff**：用户内存空间
  - 用户只读的系统信息
  - 进程地址空间，代码段、数据段、文件描述符等
- **0xf0000000~0xffffffff**：内核内存空间
  - 内核映像
  - 内核栈
  - 初始化内存
  - 从**0x0**开始排布物理内存

Link Address V.S. Load Address

- **Link Address**
  - 编译器指定代码所需要放置的内存地址
  - 由链接器配置，一般由操作系统决定
  - 决定所有直接跳转和内存地址访问的位置
- **Load Address**
  - 程序被实际加载到内存的位置
  - 由程序加载器配置，一般从可执行文件中获得

Link Address V.S. Load Address

- 一般由可执行文件结构信息和加载器来保证两个地址相同
- **Link和Load地址不同会导致：**
  - 直接跳转位置错误
  - 直接内存访问(只读数据区或bss等直接地址访问)错误
  - 堆和栈等的使用不受影响，但是可能会覆盖程序、数据区域

终端字符打印

- 主要步骤
  - 参考**Lions book**的第五章，以便理解**printf**的实现过程和细节
  - 阅读**kern/printf.c, lib/printfmt.c, kern/console.c**三个文件，搞清楚各自实现的功能以及它们之间的关联
  - 实现**8**进制数字打印的函数
- 知识点
  - **printf**不定长参数的获取和解析机制

堆栈

- 主要步骤
  - 编写打印出堆栈操作历史踪迹的内核监视函数**test_backtrace**（可参考**kern/monitor.c**中的函数原型，并可以使用**inc/x86.h** 下**read_ebp()**函数）
  - 把这个函数挂到内核监视器的命令列表中，供用户使用
- 知识点
  - 内核堆栈的初始化
  - 了解**esp**和**ebp**的作用
  - **C**语言函数调用栈的规范

Part 2 Boot Loader

Exercise 4： 在**boot**扇区被加载到的**0x7c00**地址处设置断点，在**bochs**中跟踪代码的执行，并将其与反汇编后的文件对照

跟踪**boot/main.c**中的**read_sector()**，将**c**文件中的语句与汇编语句对应起来；找出汇编文件中与**cmain**函数中读取剩余扇区循环对应的第一条语句以及最后一条语句，并跟踪剩下的语句

Boot Loader

Exercise 5： 阅读Lions注释的第三章“Reading C Programs”，特别注意其中关于指针使用的例子。（为了避免以后不必要的麻烦，不要跳过这一个练习）

Exercise 6： 在BIOS进入boot loader以及boot loader进入内核的两个地方设置断点；查看当时0x00100000地址开始的8个word的内容；为什么不一样？第二个断点处的内容是什么？

Boot Loader

Exercise 7： 再次跟踪**boot loader**，预测由于链接地址设置错误而产生问题的第一条指令；在**boot/Makefrag**中修改该链接地址来验证你的想法；最后不要忘了改回正确的链接地址

Part 3 The Kernel

- **Exercise 8：** 内核布局
  - 使用 **Bochs**来跟踪**JOS kernel**，找到新的段式地址映射起作用的地方；使用**bochs**查看 **GDT**表的值，猜测在虚实地址转换发生错误的时候，第一条会出错的指令地址；修改**GDT**表的相关值来验证你的想法；最后恢复正确的值

终端字符打印

- **Exercise 9：**补充刻意漏掉的代码，用"%o"打印出8进制数字.
- 并确保能回答出讲义上的6个问题

堆栈

- **Exercise 10：** 了解内核在哪里初始化它的堆栈，并知道堆栈被定位到内存的什么地方；以及内核如何为它的堆栈预留空间；以及在这片预留的空间中，哪一端是堆栈初始化后的栈顶(由栈顶指针来指向)?

堆栈

- **Exercise 11：** 熟悉GCC的调用规范，在obj/kern/kernel.asm中找到test_backtrace函数的地址，用 Bochs在那里设置断点；然后可以了解在内核启动后每次该函数被调用所发生的情况
- **Exercise 12:** 按照文档描述的格式要求实现 backtrace 函数，输出采用文档所述的标准形式

代码提交说明

- 使用测试脚本**grade.sh**来测试自己的代码。只要在源代码目录**~/lab1**下**make grade**，脚本会自己执行**bochs**并设置断点，判断输出，并根据输出结果来打分
- 如果系统没有**gmake**的话可以使用**make**代替：
  - **cd /usr/bin**
  - **ln –s make gmake**