***2017***



**系统能力综合训练 课程设计报告**

|  |  |
| --- | --- |
| 题 目： | X86模拟器设计 |
| 专 业： | 计算机科学与技术 |
| 班 级： | ACM1501 |
| 学 号： | U201514566 |
| 姓 名： | 董超 |
| 电 话： | 15927382865 |
| 邮 件： | [948914729@qq.com](mailto:948914729@qq.com) |
| 完成日期： | 2019-01-09 周三晚 |

目 录

[1 课程设计概述 2](#_Toc497213218)

[1.1 课设背景 2](#_Toc497213219)

[1.2 设计任务 2](#_Toc497213220)

[2 PA1 简易调试器 3](#_Toc497213221)

[2.1 功能实现的要点 3](#_Toc497213222)

[2.2 必答题 4](#_Toc497213223)

[2.3 主要故障与调试 5](#_Toc497213224)

[3 PA2 冯诺依曼计算机系统 7](#_Toc497213225)

[3.1 功能实现的要点 7](#_Toc497213226)

[3.2 必答题 9](#_Toc497213227)

[3.3 主要故障与调试 11](#_Toc497213228)

[4 PA3 异常控制流 13](#_Toc497213229)

[4.1 功能实现的要点 13](#_Toc497213230)

[4.2 主要故障与调试 15](#_Toc497213231)

[5 设计总结与心得 17](#_Toc497213232)

[5.1 课设总结 17](#_Toc497213233)

[5.2 课设心得 17](#_Toc497213234)

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字:** |

|  |
| --- |
| 二、对课程设计的学术评语（教师填写） |
|  |
| 三、对课程设计的评分（教师填写） |
| |  |  |  |  | | --- | --- | --- | --- | | 评分项目  （分值） | 报告撰写  （30分） | 课设过程  （70分） | 最终评定  （100分） | | 得分 |  |  |  | |
| **指导教师签字:** |

# **课程设计概述**

## 课设背景

NEMU受到了QEMU的启发, 结合了GDB调试器的特性, 去除了大量与课程内容差异较大的部分，该课程的学习将帮助大家了解模拟器系统开发的基本思路，熟悉掌握阅读大型软件项目代码框架的技巧和方法，通过实现X86模拟器，将对X86 CPU软硬件系统有着更为清晰的理解。

NEMU是一个X86模拟器，是一个普通的用户态应用程序, 在GNU/Linux操作系统中运行，但同时又是一个特殊的程序, 可虚拟出一个计算机系统, 其它程序可以在其中模拟执行，通过模拟器的设计让学生理解掌握下图中X86硬件基本构成以及软硬协同的机制。



图1 NEMU模拟器计算机框架示意图

学生将首先设计实现一款类似GDB的调试工具，然后利用程序模拟实现X86 CPU，寄存器组，并利用程序实现X86机器指令取指、指令译码，指令执行的动作，程序解释执行过程中，还必须实现虚拟存储系统的管理机制，完成虚拟地址到线性地址到物理地址的转换，为加速程序执行还必须完成CPU Cache机制的模拟，另外为支持程序的输入输出，还必须利用程序实现X86中断异常处理以及I/O处理机制，最终见证奇迹的剑奇侠传移植工作将向你展示了ISA如何把软件硬件联系起来, 从而支持一个游戏的运行，整个模拟器的设计实现贯穿计算机组成原理的方方面面，也涉及操作系统、编译原理相关内容，有利于进一步加深对计算机分层系统栈的理解，梳理大学3年所学的全部理论知识，大大提升学生计算机系统能力。

## 设计任务

PA项目包括一个准备实验(配置实验环境)以及4部分连贯的实验内容，具体时间和任务分布如下，该项目提供了总体框架代码，学生只需要完成框架代码中的部分代码，故总体代码量只有2000多行，但阅读代码框架是一项较大的挑战。

表1.1 NEMU模拟器任务量统计表



# **PA1 简易调试器**

## 功能实现的要点

(1) 正确地实现用于模拟寄存器的结构体CPU\_state (结构体的定义在nemu/include/cpu/reg.h中) 。

在本节任务中，我们需要实现正常的寄存器结构体。原框架给出的代码如下：

|  |
| --- |
| 1 typedef struct {  2 struct {  3 uint32\_t \_32;  4 uint16\_t \_16;  5 uint8\_t \_8[2];  6 } gpr[8];  7  8 /\* Do NOT change the order of the GPRs' definitions. \*/  9  10 /\* In NEMU, rtlreg\_t is exactly uint32\_t. This makes RTL instructions  11 \* in PA2 able to directly access these registers.  12 \*/  13 rtlreg\_t eax, ecx, edx, ebx, esp, ebp, esi, edi;  14  15 vaddr\_t eip;  16  17 } CPU\_state |

CPU包含8个基本寄存器和1个eip寄存器，每个寄存器总大小为4字节，并且可以按照2字节，1字节访问。显然，这里应该使用匿名联合。

|  |
| --- |
| 1 typedef struct {  2 union{  3 union {  4 uint32\_t \_32;  5 uint16\_t \_16;  6 uint8\_t \_8[2];  7 } gpr[8];  8  9 /\* Do NOT change the order of the GPRs' definitions. \*/  10  11 /\* In NEMU, rtlreg\_t is exactly uint32\_t. This makes RTL instructions  12 \* in PA2 able to directly access these registers.  13 \*/  14 struct {  15 rtlreg\_t eax, ecx, edx, ebx, esp, ebp, esi, edi;  16 };  17 };  18 vaddr\_t eip;  19 } CPU\_state; |

在上述代码中，gpr的类型为匿名联合，作用是使同一寄存器变量指向的内存地址相同。而较外层的联合，作用是给gpr数组中元素重命名为eax，ecx…等寄存器名称。

(2)完成简易调试器

简易调试器是NEMU中一项非常重要的基础设施. NEMU是一个用来执行其它客户程序的程序, 这意味着, NEMU可以随时了解客户程序执行的所有信息. 然而这些信息对外面的调试器(例如GDB)来说, 是不容易获取的。

例如在通过GDB调试NEMU的时候, 很难在NEMU中运行的客户程序中设置断点, 但对于NEMU来说, 这是一件不太困难的事情。为了提高调试的效率, 同时也作为熟悉框架代码的练习, 我们在monitor中实现一个具有如下功能的简易调试器。

表1.2 NEMU调试器命令格式和功能对照表

| **命令** | **格式** | **使用举例** | **说明** |
| --- | --- | --- | --- |
| 帮助(1) | help | help | 打印命令的帮助信息 |
| 继续运行(1) | c | c | 继续运行被暂停的程序 |
| 退出(1) | q | q | 退出NEMU |
| 单步执行 | si [N] | si 10 | 让程序单步执行N条指令后暂停执行,  当N没有给出时, 缺省为1 |
| 打印程序状态 | info SUBCMD | info r info w | 打印寄存器状态 打印监视点信息 |
| 表达式求值 | p EXPR | p $eax + 1 | 求出表达式EXPR的值, EXPR支持的 运算请见[调试中的表达式求值](https://nju-ics.gitbooks.io/ics2018-programming-assignment/content/1.6.html)小节 |
| 扫描内存(2) | x N EXPR | x 10 $esp | 求出表达式EXPR的值, 将结果作为起始内存 地址, 以十六进制形式输出连续的N个4字节 |
| 设置监视点 | w EXPR | w \*0x2000 | 当表达式EXPR的值发生变化时, 暂停程序执行 |
| 删除监视点 | d N | d 2 | 删除序号为N的监视点 |

在上述功能中，监视点的设置和删除即操作链表，其他功能都比较容易实现，只有表达式求值比较困难。

为了解决表达式求值的问题，我们需要完成这样几个任务：词法分析，递归求值，寻找主运算符。

词法分析的任务比较简单，我们使用正则表达式完成词法识别，并且保存为不同类型的结点(使用token结构记录)。

递归求值的主要思路：

a.如果表达式中只有一个token，直接返回这个token的，否则转b；

b.如果表达式被括号包围，去掉括号重新求值，否则转c；

c.寻找主操作符，分别计算主操作符左右两边的值，在计算。

上述过程我们设计到主操作符的使用，主操作符的定义非常简单，即最后进行的运算符号。我们可以通过下述方式寻找主操作符：

a.非运算符的token不是主运算符；

b.出现在一对括号中的token不是主运算符。注意到这里不会出现有括号包围整个表达式的情况。

c.主运算符的优先级在表达式中是最低的。这是因为主运算符是最后一步才进行的运算符。当有多个运算符的优先级都是最低时，根据结合性，最后被结合的运算符才是主运算符. 一个例子是1 + 2 + 3, 它的主运算符应该是右边的+.

上述过程大致描述了计算表达式的思路，但接下来有一个重要的问题，如何判断是否被括号包围以及如何寻找主操作符。

我们给每个表达式构造一个辅助数组给定一个算式（1+2）\*（3\*4），括号匹配的方式如下图所示。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ( | 1 | + | 2 | ) | \* | ( | 3 | \* | 4 | ) |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

图1.2 括号匹配辅助数组示意图

在上图中，第一行是给定的算式，第二行是构造的辅助数组。辅助数组与算式等长，位置一一对应。当算式中对应的符号为“(”时，数组相应内容为前一位加一；为“)”时，内容为前一位减一。假设第零位是0。

我们可以从辅助数组中看出括号匹配情况。当且仅当辅助数组所有内容非负，并且最后一位为0时，括号匹配正常。并且可以寻找主操作符，在对应辅助数组值为0的运算符中优先级最低的符号（相同优先级时选择在最右侧的符号），即为主操作符。

通过上述过程，我们可以完成不含单目运算的表达式的计算。如果表达式需要支持单目运算（如负号，取址等），我们可以通过判断符号前方的token类型来判断运算符号的类型。

## 必答题

a.理解基础设施 我们通过一些简单的计算来体会简易调试器的作用. 首先作以下假设:

假设你需要编译500次NEMU才能完成PA.

假设这500次编译当中, 有90%的次数是用于调试.

假设你没有实现简易调试器, 只能通过GDB对运行在NEMU上的客户程序进行调试. 在每一次调试中, 由于GDB不能直接观测客户程序, 你需要花费30秒的时间来从GDB中获取并分析一个信息.

假设你需要获取并分析20个信息才能排除一个bug.

那么这个学期下来, 你将会在调试上花费多少时间?

由于简易调试器可以直接观测客户程序, 假设通过简易调试器只需要花费10秒的时间从中获取并分析相同的信息. 那么这个学期下来, 简易调试器可以帮助你节省多少调试的时间?

答：500 \* 90% \* 30 \* 20 = 270,000 = 75h

75 \* (30 - 10) / 30 = 50h

b.查阅i386手册 理解了科学查阅手册的方法之后, 请你尝试在i386手册中查阅以下问题所在的位置, 把需要阅读的范围写到你的实验报告里面:

EFLAGS寄存器中的CF位是什么意思?

ModR/M字节是什么?

mov指令的具体格式是怎么样的?

答：CF用于判断进位标记（page419）

ModR/M字节用于取址。0、1、2字节为R/M部分，3、4、5为REG/OPCODE部分，6、7为MOD部分。MOD和R/M部分用于确定目的操作数的地址，REG/OPCODE为源操作数或操作指令。（page242-245）

mov指令的包含多种具体格式，具体可参考Inter 80386手册P345页。

c.shell命令 完成PA1的内容之后, nemu/目录下的所有.c和.h和文件总共有多少行代码? 你是使用什么命令得到这个结果的? 和框架代码相比, 你在PA1中编写了多少行代码? (Hint: 目前pa1分支中记录的正好是做PA1之前的状态, 思考一下应该如何回到统计工程的代码行数, 例如敲入make count就会自动运行统计代码行数的命令. 再来个难一点的, 除去空行之外, nemu/目录下的所有.c和.h文件总共有多少行代码?

答：统计.c文件：find . -name "\*.c" |xargs cat|wc –l



统计.h文件：find . -name "\*.h" |xargs cat|wc –l



忽略空行统计：

find . -name "\*.c" |xargs cat|grep -v ^$|wc –l

find . -name "\*.h" |xargs cat|grep -v ^$|wc -l

d.使用man 打开工程目录下的Makefile文件, 你会在CFLAGS变量中看到gcc的一些编译选项. 请解释gcc中的-Wall和-Werror有什么作用? 为什么要使用-Wall和-Werror?

答：

-Wall 打开gcc的所有警告;

-Werror，它要求gcc将所有的警告当成错误进行处理。

使用Wall和Werror可以提高我们程序的正确率，避免后期由于前面的错误而拖慢进度。

## 主要故障与调试

a.union匿名联合的使用

在寄存器设计的过程中，我需要是用匿名联合。刚开始设计时我使用了下列代码。但发现结果不正确，因而仔细查看，发现问题。所有寄存器所在的匿名联合包含的成员为：gpr，eax，ecx，edx，ebx，esp，ebp，esi，edi，这九个元素共享一块内存，显然这是不符合要求的。

|  |
| --- |
| 1 typedef struct {  2 union{  3 union {  4 uint32\_t \_32;  5 uint16\_t \_16;  6 uint8\_t \_8[2];  7 } gpr[8];  8  9 /\* Do NOT change the order of the GPRs' definitions. \*/  10  11 /\* In NEMU, rtlreg\_t is exactly uint32\_t. This makes RTL instructions  12 \* in PA2 able to directly access these registers.  13 \*/  14  15 rtlreg\_t eax, ecx, edx, ebx, esp, ebp, esi, edi;  16  17 };  18 vaddr\_t eip;  19 } CPU\_state; |

解决方法：将eax，ecx，edx，ebx，esp，ebp，esi，edi，八个寄存器再放在同一结构体中，该结构体与gpr位于同一匿名联合，共享一片内存。

# **PA2 冯诺依曼计算机系统**

## 功能实现的要点

## 必答题

## 主要故障与调试

# **异常控制流**

## 功能实现的要点

## 必答题

## 主要故障与调试