

电子科技大学计算机科学与工程学
院（网络空间安全学院）

实 验 报 告

（实验）课程名称 计算机操作系统

电子科技大学

实验报告

学生姓名：韩博宇 学 号：2019040708023 指导教师：刘杰彦

实验地点：主楼 A2-412

实验时间：2021.11.21

一、实验室名称：主楼 A2-412

二、实验项目名称：内存地址转换实验

三、实验学时：2 学时

四、实验目的：

- (1) 掌握计算机的寻址过程
- (2) 掌握页式地址地址转换过程
- (3) 掌握计算机各种寄存器的用法

五、实验环境

Linux 内核 (0.11) +Bochs 虚拟机

六、实验内容和原理

本实验运行一个设置了全局变量的循环程序，通过查看段寄存器，LDT 表，GDT 表等信息，经过一系列段、页地址转换，找到程序中该全局变量的物理地址。

(一) 基本概念

物理地址：

把内存比作一个大的数组（为了分析方便），每个数组都有其下标，这个下标标识了内存中的地址，这个实实在在的在内存中的地址，我们称之为物理地址。但是在用于内存芯片级的单元寻址，与处理器和 CPU 连接的地址总线相对应。

逻辑地址：

与物理地址比较相对的是逻辑地址，这个地址就是在程序中我们把它放到的位置；而这个位置通常是由编译器给出的。另外一种理解是：逻辑地址指的是机器语言指令中，用来指定一个操作数或者是一条指令的地址。Intel 段式管理中：“一个逻辑地址，是由一个段标识符加上一个指定段内相对地址的偏移量表示为 [段标识符：段内偏移量]。”

虚拟地址：

Virtual Address, 简称 VA, 由于 Windows 程序时运行在 386 保护模式下，这样程序访问存储器所使用的逻辑地址称为虚拟地址。实际上因为我们现代程序中地址都是虚拟的，所以这里的虚拟地址和线性地址是等价了的。

线性地址：

线性地址（Linear Address）也叫虚拟地址(virtual address)是逻辑地址到物理地址变换之间的中间层。在分段部件中逻辑地址是段中的偏移地址，然后加上基地址就是线性地址。

（二）CPU 段式内存管理：逻辑地址转换为线性地址

1.段标识符：

也称为段选择符，属于逻辑地址的构成部分，段标识符是由一个 16 位长的字段组成，其中前 13 位是一个索引号。后面 3 位包含一些硬件细节。



图 1 段选择符

索引号:

可以看作是段的编号，也可以看做是相关段描述符在段表中的索引位置。系统中的段表有两类：GDT 和 LDT。

GDT:

全局段描述符表，整个系统一个，GDT 表中存放了共享段的描述符，以及 LDT 的描述符（每个 LDT 本身被看作一个段）。

LDT:

局部段描述符表，每个进程一个，进程内部的各个段的描述符，就放在 LDT 中。

TI 字段:

Intel 设计思想是：一些全局的段描述符，就放在“全局段描述符表(GDT)”中，一些局部的，例如每个进程自己的，就放在所谓的“局部段描述符表(LDT)”中。TI=0, 表示相应的段描述符在 GDT 中，TI=1 表示表示相应的段描述符在 LDT 中。

2.段描述符:

具体描述了一个段。在段表中，存放了很多段描述符。我们可以

通过段标识符的前 13 位，直接在段描述符表中找到一个具体的段描述符，也就是说，段标识符的前 13 位是相关段描述符在段表中的索引位置。



图 2 GDT 或 LDT 示例

每一个段描述符由 8 个字节组成：

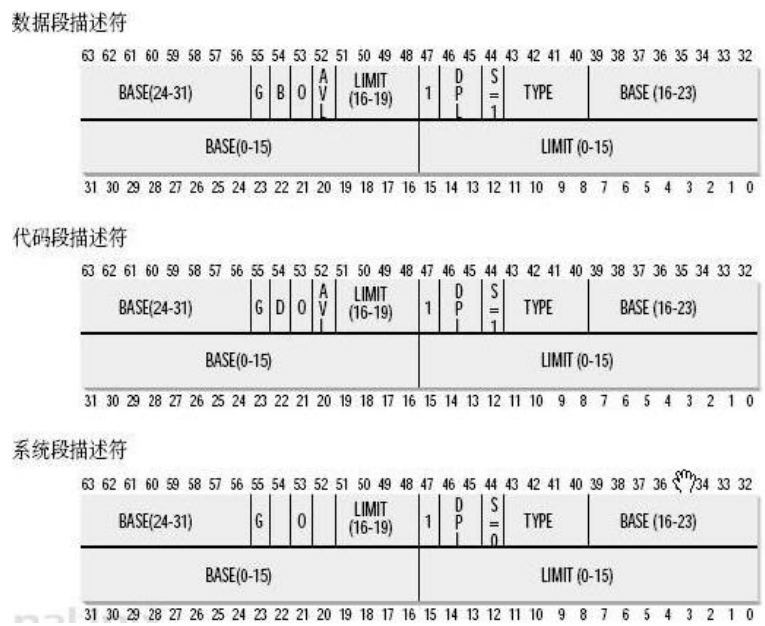


图 3 段描述符

Base 字段：

描述了一个段的开始位置：段基址。Base(24-31)：基地址的高 8 位，Base(16-23)：基地址的中间 8 位，Base(0-15)：基地址的低 16 位。（这里的段基址，不是相应的段在内存中的起始地址，而是程序编译链接以后，这个段在程序逻辑(虚拟)地址空间里的起始位置。）

3.从逻辑地址到线性地址的转换过程：

(1) 从 GDTR 中获得 GDT 的地址，从 LDTR 中获得 LDT 在 GDT 中的偏移量，查找 GDT，从中获取 LDT 的起始地址；

(2) 从 DS 中的高 13 位获取 DS 段在 LDT 中索引位置，查找 LDT，获取 DS 段的段描述符，从而获取 DS 段的基地址；

(3) 根据 DS 段的基地址+段内偏移量，获取所需单元的线性地址。

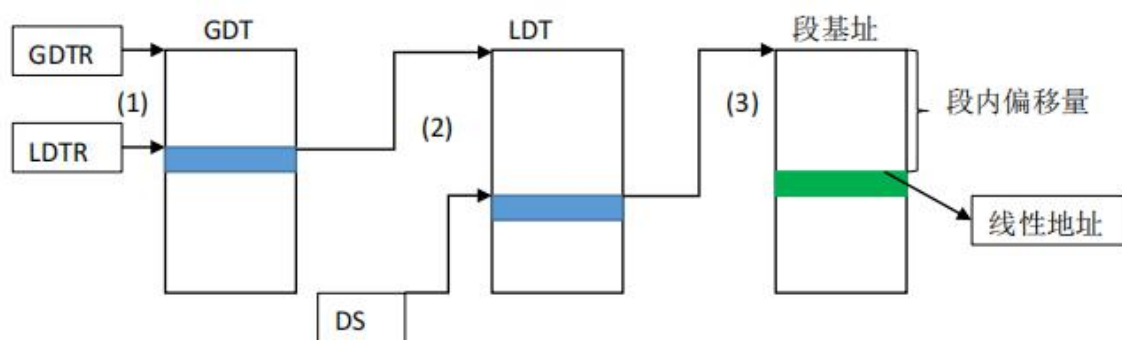


图 4 逻辑地址到线性地址的转换

(三) CPU 的页式内存管理：线性地址到物理地址的转换

(1) 因为页目录表的地址放在 CPU 的 cr3 寄存器中，因此首先从 cr3 中取出进程的页目录表（第一级页表）的起始地址（操作系统负责在调度进程的时候，已经把这个地址装入对应寄存器）；

(2) 根据线性地址前十位，在页目录表（第一级页表）中，找到对应的索引项，因为引入了二级管理模式，线性地址的前十位，是第一级页表中的索引值，根据该索引，查找页目录表中对应的项，该项即保存了一个第二级页表的起始地址。

(3) 查找第二级页表，根据线性地址的中间十位，在该页表中找到数据页的起始地址；

(4) 将页的起始地址与页内偏移量（即线性地址中最后 12 位）相

加，得到最终我们想要的物理地址；

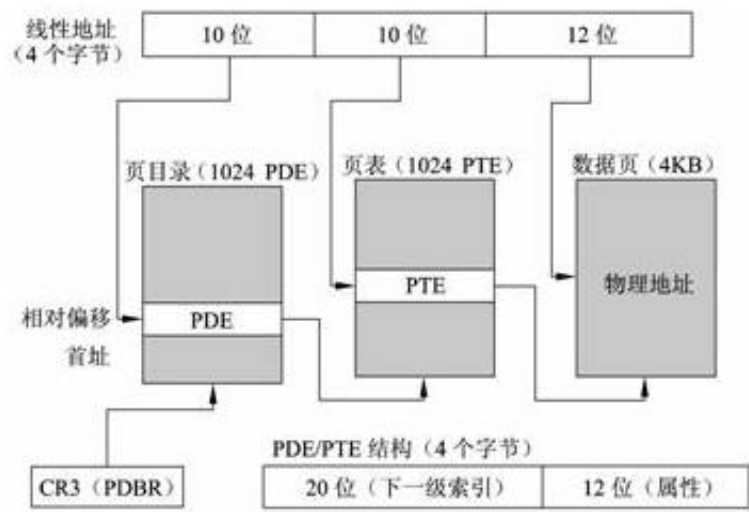


图 5 线性地址结构

七、实验步骤及结果分析：

- 1. 运行“bochs.exe”文件安装bochs。
- 2. 拷贝 bootimage-0.11-hd、diska.img、hdc-0.11-new.img、mybochsrc-hd.bxrc 至安装目录。
- 3. 在安装目录中找到 bochsdbg.exe 程序，并运行。
- 4. 在弹出的界面中，点击“Load”加载配置文件“mybochsrc-hd.bxrc”。随后，点击 “Start” 启动 Bochs 虚拟机。虚拟机启动后，出现两个窗口，一个为 Bochs 控制窗口，另一个为 Linux 操作系统运行窗口（主显示窗口）。
- 5. 在控制窗口输入 “c” 后回车，加载 Linux 操作系统。

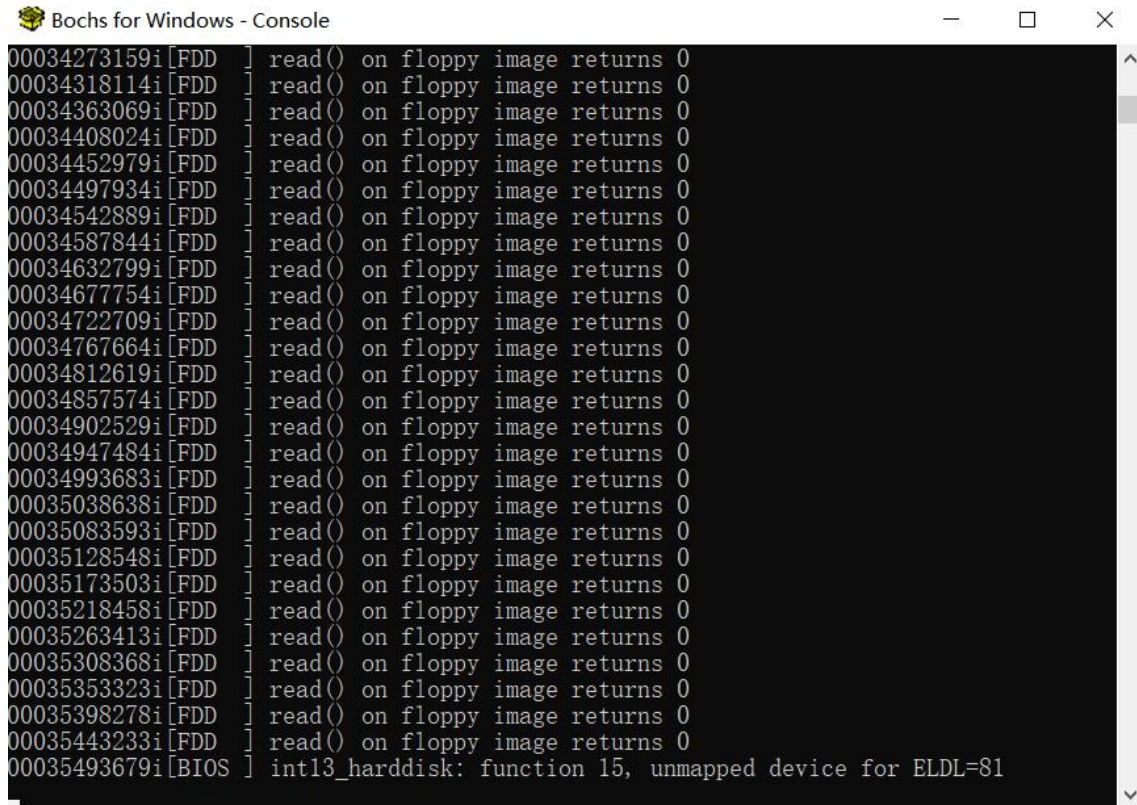


图 6 控制窗口

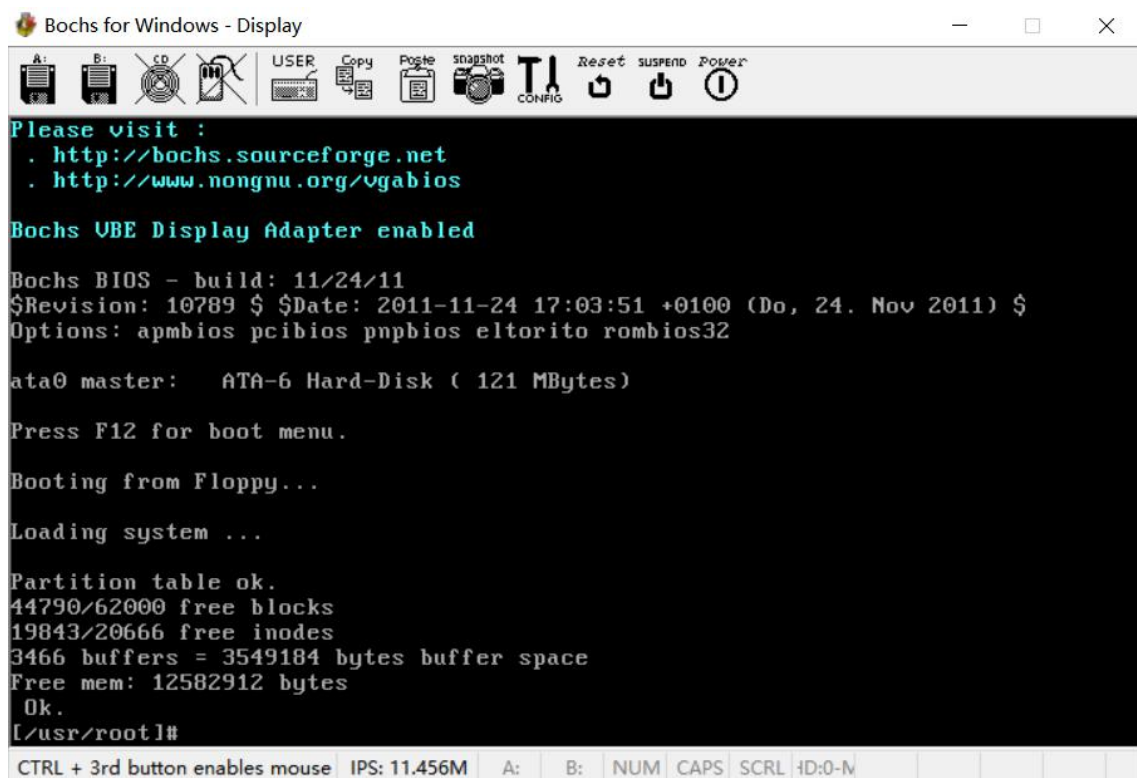


图 7 Linux 操作系统

6. 在 Linux 操作系统中，使用 vi 工具编写 main.c 源文件。

先输入指令“vi main.c”，然后按 Enter 键，进入代码编辑，输出代码，我的学号为 2019040708023，因此 j 取学号后 8 位，即：40708023。截图如下：



```
#include <stdio.h>
int j=0x40708023;

int main(){
    printf("the address of j is 0x%x\n",&j);
    while(j);
    printf("program terminated normally!\n");
    return 0;
}
```

图 8 main.c文件内容

然后按 Esc 键，输入指令“:wq”，保存文件并退出。

7. 随后输入“gcc -o main main.c”，编译并生成“main”可执行文件。

8. 在 Linux 操作系统中，输入“./main”指令运行该可执行文件，运行结果如下图所示。



```
[/usr/root]# gcc -o main main.c
[/usr/root]# ./main
the address of j is 0x3004
```

图 9 main.c文件编译及运行结果

9. 控制窗口按 Ctrl+c，中断当前运行，进入调试状态。

10. 在控制窗口中输入 sreg 命令，查看段的具体信息。结果如下图所示。

```

(0) [0x000000000fa7065] 000f:0000000000000065 (unk. ctxt): cmp dword ptr ds:0x3004, 0x0
<bochs:2> sreg
es:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
    Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
cs:0x000f, dh=0x10c0fb00, dl=0x00000002, valid=1
    Code segment, base=0x10000000, limit=0x00002fff, Execute/Read, Accessed, 32-bit
ss:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
    Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
ds:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=3
    Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
fs:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
    Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
gs:0x0017, dh=0x10c0f300, dl=0x00003fff, valid=1
    Data segment, base=0x10000000, limit=0x03ffffff, Read/Write, Accessed
ldtr:0x0068, dh=0x000082fa, dl=0xa2d00068, valid=1
tr:0x0060, dh=0x00008bfa, dl=0xa2e80068, valid=1
gdtr:base=0x00000000000005cb8, limit=0x7ff
ldtr:base=0x000000000000054b8, limit=0x7ff

```

图 10 段的具体信息

ds 段的段标识符信息为 0x0017，转化为二进制为 0000 0000 0001 0111，其中，索引号后一位（倒数第三位）为 TI 位，可得 TI=1，表明段描述符存放在 LDT 中，右移 3 位之后为 0x02，因此局部描述符表 LDT 的偏移量为 2。

11. 查看 LDTR 寄存器信息，其中存放了 LDT 在 GDT 中的位置。由上图可得 0x0068，即 0000 0000 0110 1000，右移 3 位之后为 0000 0000 0000 1101，转化为 16 进制为 0x000D，故在 GTD 中的索引为 13。

12. GDTR 中存放了 GDT 的起始地址，输入“xp /2w 0x00005cb8+13*8”指令，查看 GDT 中对应表项，得到 LDT 段描述符，如图所示。

```

<bochs:3> xp /2w 0x00005cb8+13*8
[bochs]:
0x00000000000005d20 <bogus+      0>:  0xa2d00068      0x000082fa

```

图 11 LDT段描述符

其中，LDT 基址的 24-31 位用 LDT 段描述符高 32 位的最高 8 位表示，低 24 位用 LDT 段描述符的低 32 位的 24-31 位和高 32 位的 0-7 位表示。经过拼接可得 LDT 的基址为 0x00faa2d0。

13. 输入 “xp /2w 0x00faa2d0+16”，查看 LDT 中第 2 项段描述符，结果如图所示。

```
<bochs:4> xp /2w 0x00faa2d0+16
[bochs]:
0x0000000000faa2e0 <bogus+      0>:  0x00003fff  0x10c0f300
```

图 12 LDT中第2项段描述符

可以发现，得到的 ds 段的段描述符信息与 ds 寄存器（dl、dh）中的结果完全相同。

14. 根据上述结果，可以得到 ds 段的基地址为 0x10000000。

15. 根据程序运行结果 “the address of j is 0x3004”，可以计算线性地址为 $0x10000000 + 0x00003004 = 0x10003004$ 。转化为二进制为 0001 0000 0000 0000 0011 0000 0000 0100，其中，高 10 位为页目录索引，为 00 0100 0000，即 0x40；低 12 位为偏移，为 0000 0000 0100，即 0x40；中间 10 位为页表索引，为 00 0000 0011，即 0x03。因此，页目录索引为 0x40，页表索引为 0x03，偏移量为 0x04。

16. 输入 “creg” 指令，结果如图所示。

```
<bochs:5> creg
CR0=0x8000001b: PG cd nw ac wp ne ET TS em MP PE
CR2=page fault laddr=0x0000000010002fa8
CR3=0x0000000000000000
   PCD=page-level cache disable=0
   PWT=page-level write-through=0
CR4=0x00000000: smep osxsave pcid fsgsbase smx vmx osxmmexcpt osfxsr pce p
EFER=0x00000000: ffxsr nxe lma lme sce
```

图 13 creg查看寄存器值

寄存器 CR3 的值为 0，即页目录表的起始地址为 0。

17. 由于页目录表的起始地址为 0，故 PDE 的地址为 $0 + 64 * 4$ ，因此，输入 “xp /w 64*4” 指令，结果如图所示。

```
<bochs:6> xp /w 64*4
[bochs]:
0x00000000000000100 <bogus+      0>: 0x00fa3027
```

图 14 查看PDE的值

可以得到，PDE 的值为 0x00fa3027，低 12 位置零作为下一级的索引，即 0x00fa3000。则 PTE 的地址为 0x00fa3000+3*4。

18. 输入 “xp /2w 0x00fa3000+3*4” 指令，结果如图所示。

```
<bochs:11> xp /2w 0x00fa3000+3*4
[bochs]:
0x0000000000fa300c <bogus+      0>: 0x00fa2067 0x00000000
```

图 15 查看PTE的值

得到 PTE 的值为 0x00fa2067，低 12 位置零作为下一级的索引，即 0x00fa2000。因此，物理地址为 0x00fa2000+4。

19. 输入 “xp /w 0x00fa2000+4” 指令，结果如图所示。

```
<bochs:12> xp /w 0x00fa2000+4
[bochs]:
0x0000000000fa2004 <bogus+      0>: 0x40708023
```

图 16 查看j中的值是否为学号后8位

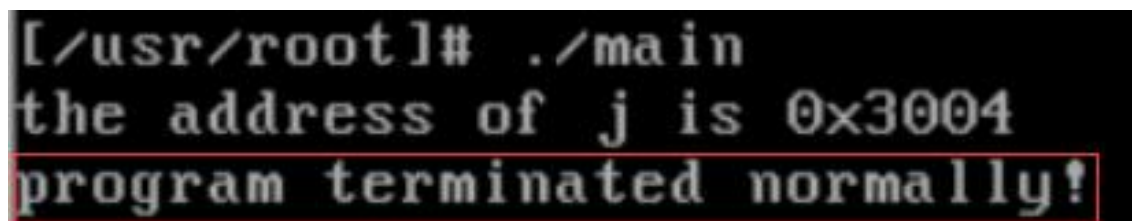
得到的结果与 main.c 中定义的 j 的值相同，即我的学号40708023，说明正确找到了 j 的物理地址。

20. 接下来对 j 的值进行修改，输入 “setpmem 0x00fa2004 4 0” 指令，将该物理地址开始的 4 个字节的值全部清零。清零之后，可以再次使用 “xp /w 0x00fa2000+4” 指令查看该物理地址存放的当前值，发现已经成功清零。截图如下：

```
<bochs:13> setpmem 0x00fa2004 4 0
<bochs:14> xp /w 0x00fa2000+4
[bochs]:
0x0000000000fa2004 <bogus+      0>: 0x00000000
```

图 17 修改j的值为0

21. 输入命令“c”，继续运行。回到 Display 窗口，显示程序已正常结束。表明本次通过物理地址对 j 值的修改有效，成功完成实验。

A terminal window with a black background and white text. The text shows a shell prompt [usr/root]#, followed by the command ./main. The output consists of two lines: 'the address of j is 0x3004' and 'program terminated normally!'. The second line is highlighted with a red rectangular border.

```
[usr/root]# ./main
the address of j is 0x3004
program terminated normally!
```

图 18 检验对j值的修改有效

八、实验结论：

本次实验通过 Linux 内核和 Bochs 虚拟机，在 main.c 文件中以学号后 8 位定义了一个变量，通过查看段寄存器、LDT 表、GDT 表等信息，经过一系列段、页地址转换，先后完成了逻辑地址到线性地址的转换和线性地址到物理地址的转换，找到程序中该全局变量的物理地址，并通过物理地址的访问对其进行修改，使得程序能按照预期顺利退出。

九、总结及心得体会：

本实验通过 Linux 内核和 Bochs 虚拟机，让我实际体会到了计算机的寻址过程和页式地址的地址转换过程，也对计算机中的寄存器有了更加深入的了解。实验不但帮助我巩固了寻址过程、地址转换等课堂上学到的操作系统基本知识，也让我通过实际操作，加深了对计算机工作过程的理解和认识，提高了动手能力。

由于之前学过 Linux 课程，该实验还帮我复习了一些 Linux 课程上学到的知识，可谓是一举两得。总之，本次实验让我收获很大，不论是对理论知识有了更扎实的掌握，也对实际过程和实践操作有了一

定的了解。

十、对本实验过程及方法、手段的改进建议：

本实验提供了非常详细的指导，整个实验完成起来比较简单。我觉得可以增设一些挑战性内容，帮助学生更深入地了解计算机寻址过程、地址转换过程等理论，通过自己的思考在解决问题的过程中强化对于相关知识的理解和掌握。

报告评分：

指导教师签字：