

中山大学数据科学与计算机学院  
操作系统实验课程

实 验 报 告

教 师	凌应标
学 号	17341038
姓 名	傅畅
实验名称	实验三（开发独立内核的操作系统）

## 一、实验目的: 用 C 和汇编实现操作系统内核

(一) 用 C 和汇编实现操作系统内核

(二) 增加批处理能力

### 1.2.1 提供用户返回内核的一种解决方案

### 1.2.2 一条在内核的 C 模块中实现

- 在磁盘上建立一个表，记录用户程序的存储安排
- 可以在控制台查到用户程序的信息，如程序名、字节数、在磁盘映像文件中的位置等
- 设计一种命令，命令中可加载多个用户程序，依次执行，并能在控制台发出命令
- 在引导系统前，将一组命令存放在磁盘映像中，系统可以解释执行

## 二、实验原理:

(一) c 函数与 **nasm** 函数的相互调用

c 语言在实现比较复杂的程序逻辑上比较简单，而 **asm** 在实现底层 IO 功能上比较直接，本次实验的许多个功能我需要两者尽可能多地需要两者相互调用，汲取各自长处 c 语言函数的调用，在汇编中的实现方法，只是将参数简单压入栈，然后将返回出口压入栈后，直接跳向函数入口。知道这一点之后，汇编代码想要直接调用 c 函数，只需要将参数压入栈即可；**asm** 代码段想要使用 c 函数传过来的参数，也只需要从  $sp + 4$  之后的位置取出即可。在函数返回值在汇编层面也只是用 **eax** 来传递

(二) c 语言与 **nasm** 交叉编译

由于现阶段暂时还只能编译 16bin 文件，**gcc** 的参数需要指明 **-m32**，同时需要关闭对关联库和 **builtin** 函数。

### (三) **obj** 文件的链接

ld 对.o 文件先后顺序敏感，被依赖的文件需要放在后面；而且编译出来在 bin 文件中相对顺序也和命令中文件顺序有关

### (四) 执行用户程序前后的寄存器保护措施

所有在用户程序中被使用的寄存器，都需要使用栈加以保存。待到模块结束 ret 后需要弹出恢复。

## 三、 代码分析:

代码 1: asm code

```
1  #include <stdio.h>
2  int main (int argc, char *argv[]){
3      printf("Hello world!");
4  }
```

代码 2: 一段 C 代码

### (一) **shell** 代码逻辑

```
1 void shell(){
2     for (;;) {
3         char p[15]= "fuchang@1038 $\0";
4         int len =strlen(p, MAXLEN),i;
5         puts(p, len, &row,&col);
6         len=0;
7         for (char ch=getchar(); ch!= '\r' && ch!='\n'; ch=getchar()){
8             if ( ch==0x8 ){
9                 if ( len){
10                     --col;
11                     putchar(' ', &row,&col), s[--len] = 0;
```

```

12         --col;setCursor(row, col);
13     }
14 }else{
15     putchar(ch, &row,&col);s[len++] = ch;
16 }
17 }
18 Enterline(&row, &col);
19 for (i=0; s[i]!=' ' && i<len;++i);
20 if (i<len)exec_single(s+i, len-i);
21 }
22 }

```

代码 3: 一段 C 代码

整个 shell 的逻辑比较简单，不断输入字符并在屏幕上追踪显示；如果输入了删除符 0x8 则退格显示。

等到输入回车换行符之后，光标换行 Enterline()，并扫描命令行，跳进命令解释函数 exec\_single()

## (二) ls 现实当前文件信息

```

1 void dir(int showall){
2     char *p;
3     for (volatile int i=0; i<32; ++i){
4         p=readItem(i);
5         if (p[0]==0) break;
6         volatile int len = p[7]=='\0'?strlen(p,8):8,num=0;
7         if ( !showall){
8             if ( i) {putchar(' ', &row,&col);}
9             puts(p, len, &row, &col);
10        }else{
11            puts(p,len, &row, &col);

```

```

12         puts(" section:",9, &row, &col);putnum(sectionLoc(p), &
        row,&col);
13         puts(" size:",6,&row, &col); putnum(fileSiz(p)<<9, &row
        , &col);
14         num=createTim(p);
15         puts(" crtime:",8, &row, &col);
16         putnum(num/60,&row,&col);putchar(':',&row,&col);putnum(
        num%60, &row,&col);
17         puts(" filetype:",10,&row, &col);
18         num=fileTyp(p);
19         if ( num==1) puts("bin",3, &row, &col); else
20         if ( num==2) puts("bat",3, &row, &col); else
21         if ( num==0) puts("ker",3, &row, &col);
22
23         Enterline(&row, &col);
24     }
25 }
26 if ( !showall)Enterline(&row,&col);
27 }

```

代码 4: 一段 C 代码

该 ls 命令会将表中的所有文件的文件名都输出

另外我设计的 ls 命令带有一个可选参数 *-al*，该参数下 ls 会将文件的所有详情信息全部输出。

为了方便读取文件信息，我对每一条文件信息都写了对应的函数

```

1
2 typedef char* itemPtr;
3 int sectionLoc(itemPtr p){return (((int)p[9])<<8) + p[8];}
4 int fileSiz(itemPtr p){ return (((int)p[11])<<8)+p[10];}
5 int createTim(itemPtr p){return (( (int)p[13])<<8)+p[12];}

```

```
6 int fileType(itemPtr p){return (((int)p[15])<<8)+p[14];}
```

代码 5: 一段 C 代码

### (三) 对批处理文件与 **bin** 文件的执行

```
1 void exec_single(char *cmd, int len){
2     if ( strcmp(cmd, ".\"",2)==0){
3         int i=2,j,k;
4         for (; i<len && cmd[i]!=' ';++i);
5         for (j=i; cmd[j]!=' ' && j<len; ++j);
6         int exist=0;
7         itemPtr t=readItem(k);
8         for (k=0; k<32; ++k,t+=16) if ( strcmp(cmd+i,t, j-i)==0){ // bug ,t+=16?
9             if ( fileType(t) ==1)run_user_prog(k);else
10             if ( fileType(t) ==2)exec_batch(get_user_bat(k),fileSiz(t)<<9);
11             exist = 1;
12         }
13         if ( !exist) puts("file not found!",15, &row,&col);
14
15     } else
16     if ( strcmp(cmd, "ls",2)==0){
17         int i=2, j;
18         for (; i<len && cmd[i]!=' '; ++i);
19         for (j=i; cmd[j]!=' ' && j<len; ++j);
20         if ( j-i>0){
21             if (j-i==3 && strcmp(cmd+i, "-a",3)==0)
22                 dir(1);
23             else puts("parameter not found!",21, &row, &col);
24         }else dir(0);
25     } else
```

```

26     puts("command not found!",18, &row, &col);
27 }

```

代码 6: 一段 C 代码

该函数的解释过程其实就是对两个指令字符串的匹配过程。整个过程比较冗余的  
还是对非法输入的判断。

对用户 **bin** 文件的执行和用户 **batch** 文件的执行还是有不同的，用户 **batch** 文件本质  
上还是一堆指令字符串

```

1     bat.sh_start:
2     db  "./prog2",0xa,"./prog1",0xa
3     times 512-($-bat.sh_start) db 0

```

代码 7: asm code

调用 **nasm** 中的用户程序,需要加载其到对应的内存并跳转执行,这里的 **run\_user\_prog**  
和 **get\_user\_bat** 用 **nasm** 写的都是得到目的地地址或者命令地址然后直接执行

```

1 run_user_prog:           ;debug
2     ; run user's bin prog
3     push ax
4     push bx
5
6     mov bx, sp
7     push word [bx+0x8]
8     call dword Load_prog
9     xor ebx, ebx
10    mov bx, ax
11    call dword ebx
12
13    add sp, 0x2           ;bug , fail to match the push—pop bracket
14    pop bx
15    pop ax
16
17 o32 ret
18
19 get_user_bat:
20     ;run user's bat order
21
22     push bx              ;each take 2 units of stack memo
23
24     mov bx, sp

```

```

25         push word [bx+0x6]
26         call dword Load_prog
27         add sp,0x2
28
29         pop bx
30
31 o32 ret

```

代码 8: asm code

#### (四) IO 函数设计

我主要在 nasm 中封装了 4 个函数:

##### 3.4.1 putchar

调用 10 号中断，将字符串输出到第 0 页的指定位置

```

1  asm_putchar:                ; debugged
2  push ax
3  push bx
4  push cx
5  push dx
6  push ds
7  push es
8  push bp    ; bug, every register used in the prog must be protected !!!
9
10     mov ax, cs                ; 置其他段寄存器值与CS相同
11     mov ds, ax                ; 数据段
12     mov bp, sp                ; BP=当前串的偏移地址
13     add bp, 0x12
14     mov ax, ds                ; ES:BP = 串地址
15     mov es, ax                ; 置ES=DS
16     mov cx, 1 ; CX=串长
17     mov ax, 1301h             ; AH = 13h (功能号)、AL = 01h (光标置于串尾)
18     mov bx, 0007h             ; 页号为0(BH = 0) 黑底白字(BL = 07h)
19     mov dh, [bp+0x4]           ; 行号=0
20     mov dl, [bp+0x8]           ; bug , each is 64bit !!!
21     int 10h                   ; BIOS的10h功能: 显示一行字符
22
23 pop bp
24 pop es
25 pop ds
26 pop dx
27 pop cx

```



```

28 pop bx
29 pop ax
30 o32 ret

```

代码 9: asm code

### 3.4.2 getchar

getchar 实现比较简单，调用 16h 中断后，所得 ascii 码直接存在了 al 中，作为返回值可以直接 ret

```

1  getchar:                ; debugged
2      mov ax,0x0
3      int 16h
4  o32 ret

```

代码 10: asm code

### 3.4.3 屏幕下滚

```

1  ScrollDown:             ; debugged
2
3      push ax
4      push bx
5      push cx
6      push dx
7
8      mov  ah,6            ;6=屏幕初始化或上卷 , 7=屏幕初始化或下卷
9      mov  al,1;          AL = 上卷行数AL =0全屏幕为空白
10     mov  bh,0;          BH = 卷入行属性
11     mov  cx,0;          CH = 左上角行号 CL = 左上角列号
12     mov  dx,0x184f       ;DH = 右下角行号 DL = 右下角列号(24,79)
13     int  10h
14
15     pop  dx
16     pop  cx
17     pop  bx
18     pop  ax
19  o32 ret

```

代码 11: asm code

### 3.4.4 设置光标

```
1  setCursor: ; debugged
2      push dx
3      push bx
4      push bp
5      push ax
6
7      xor ax, ax
8      xor bx, bx
9      mov ah, 2h
10     mov bp, sp
11     mov dh, [bp+0xc] ; mad to calc the real site in stack
12     mov dl, [bp+0x10]
13     int 10h
14
15     pop ax
16     pop bp
17     pop bx
18     pop dx
19     o32 ret
```

代码 12: asm code

然后在 c 语言中就可以比较方便地实现 puts, putnum, Enterline 等函数:

```
1 void putchar(char ch, int *r, int *c){ // used for
2     asm_putchar(ch, *r, *c);
3     ++(*c);
4 }
5
6 void puts(char *s, int len, int *r, int *c){
7     for (volatile int i=0; i<len && s[i]; ++i, ++(*c))
8         asm_putchar(s[i], *r, *c);
9 } // print a string at the (r,c), used for terminal
10
11 extern void Enterline(int *r, int *c){
12     for (;(*r)>=24; --(*r))ScrollDown();
13     ++(*r); *c=0;
```

```

14 }
15 void putnum(int num,int *r, int *c){
16     tmpLen=0;
17     if ( num==0) putchar('0', r, c);    else{
18         for (; num ;num/=10) tmp[tmpLen++] = num%10;
19         for (volatile int i=tmpLen-1; ~i; --i)
20             putchar(tmp[i]+'0',r, c);
21     }
22 }

```

代码 13: 一段 C 代码

#### (五) makefile

### 四、实验结果与分析（含重要数据结果分析或核心代码流程分析）：

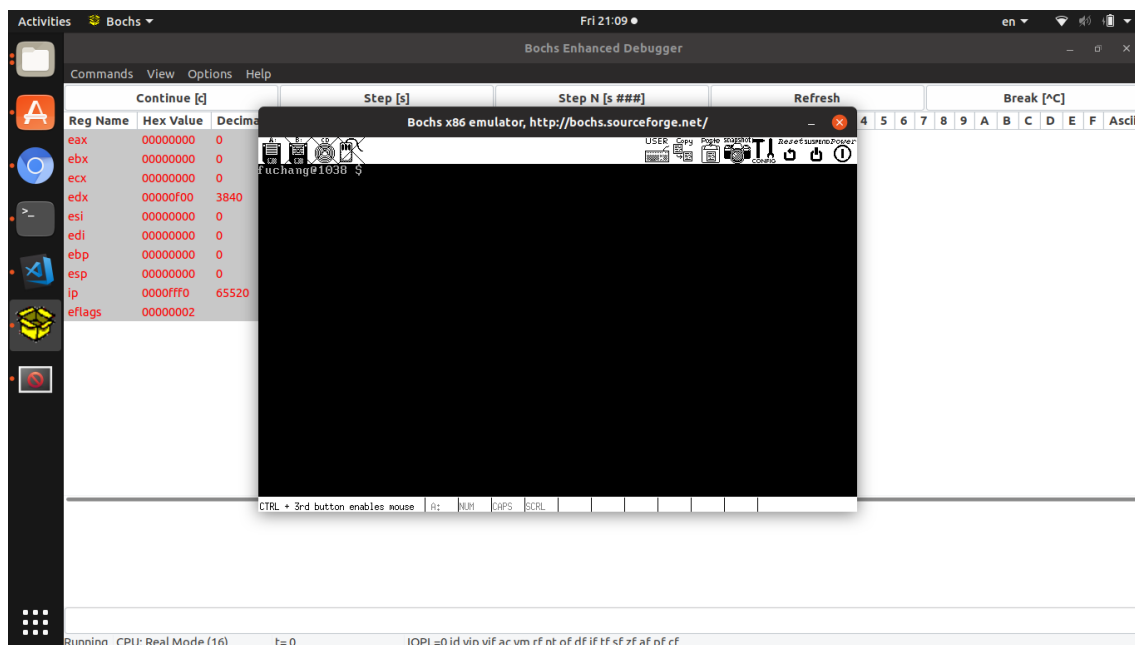


图 1: 打开程序显示命令行

(自行填写。每个实验项目的格式范例：

- 关键流程分析
- 实验结果

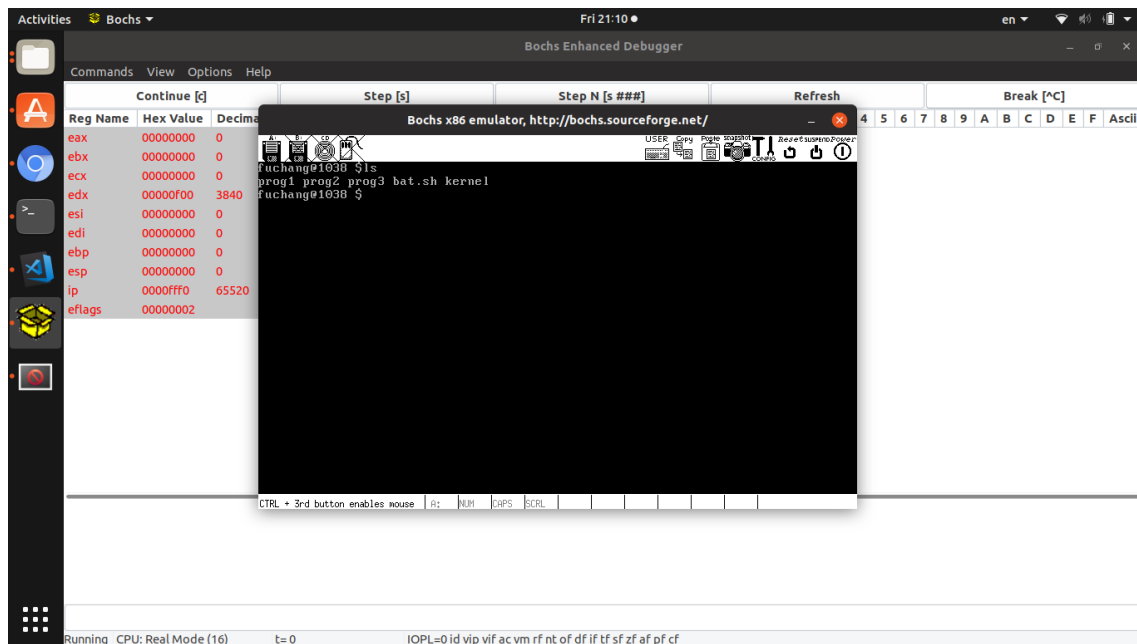


图 2: 输入 ls 指令

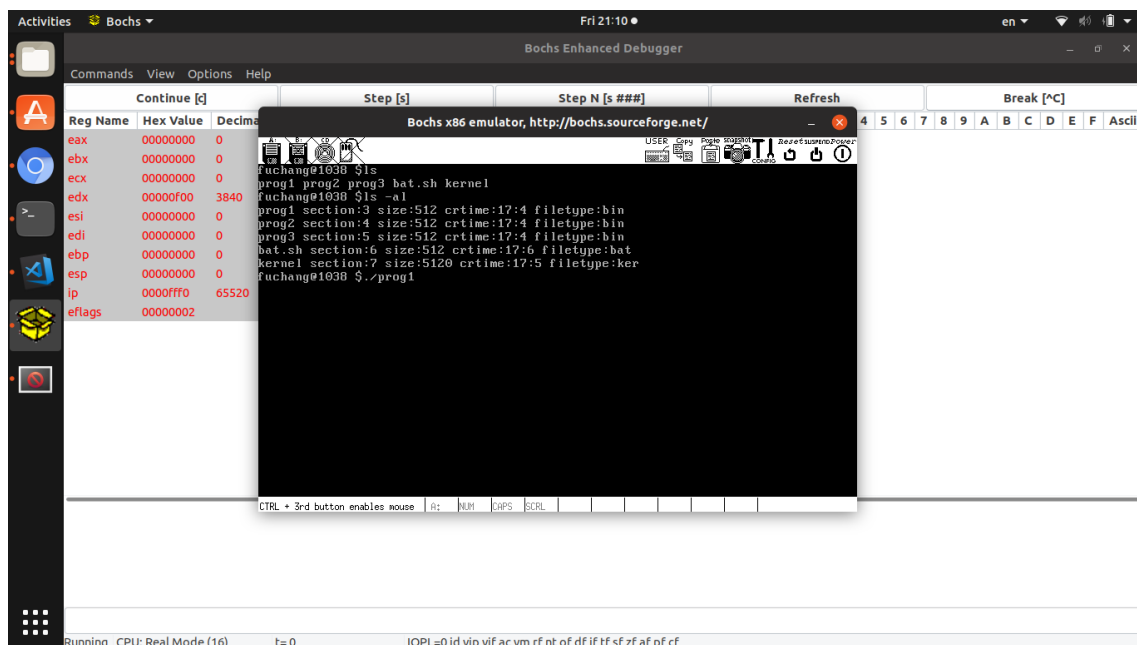


图 3: Shot3

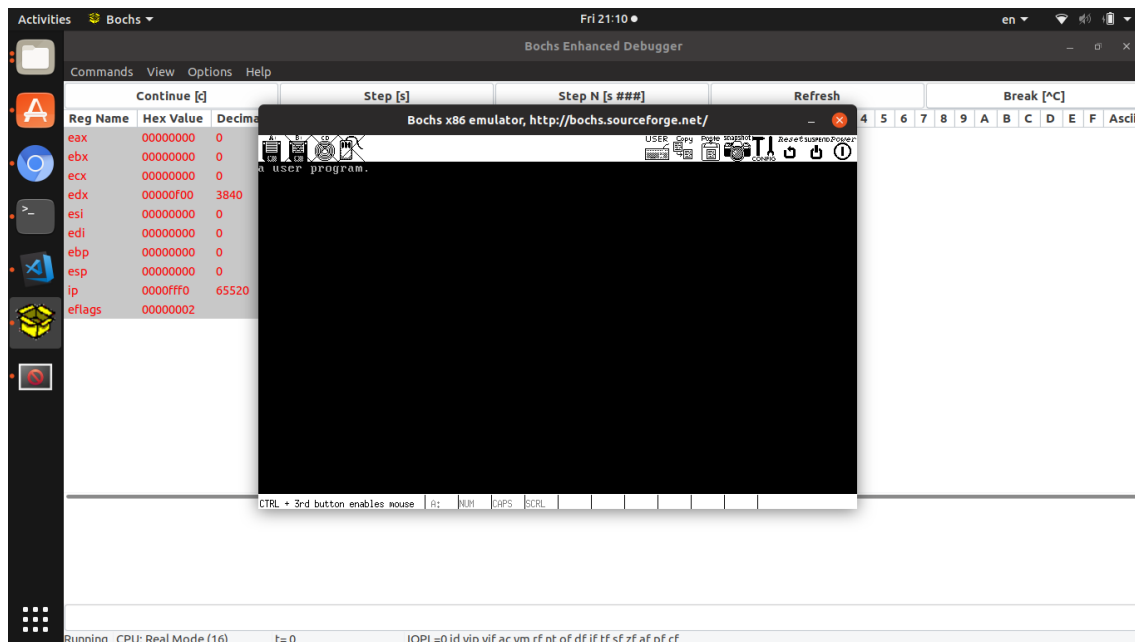


图 4: Shot3

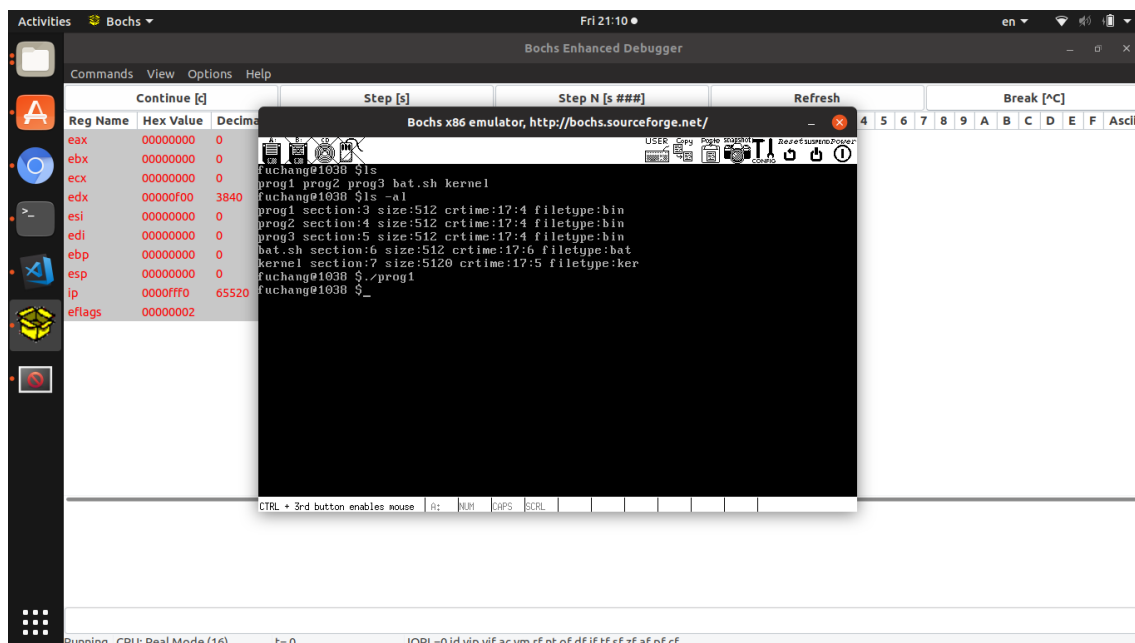


图 5: Shot3

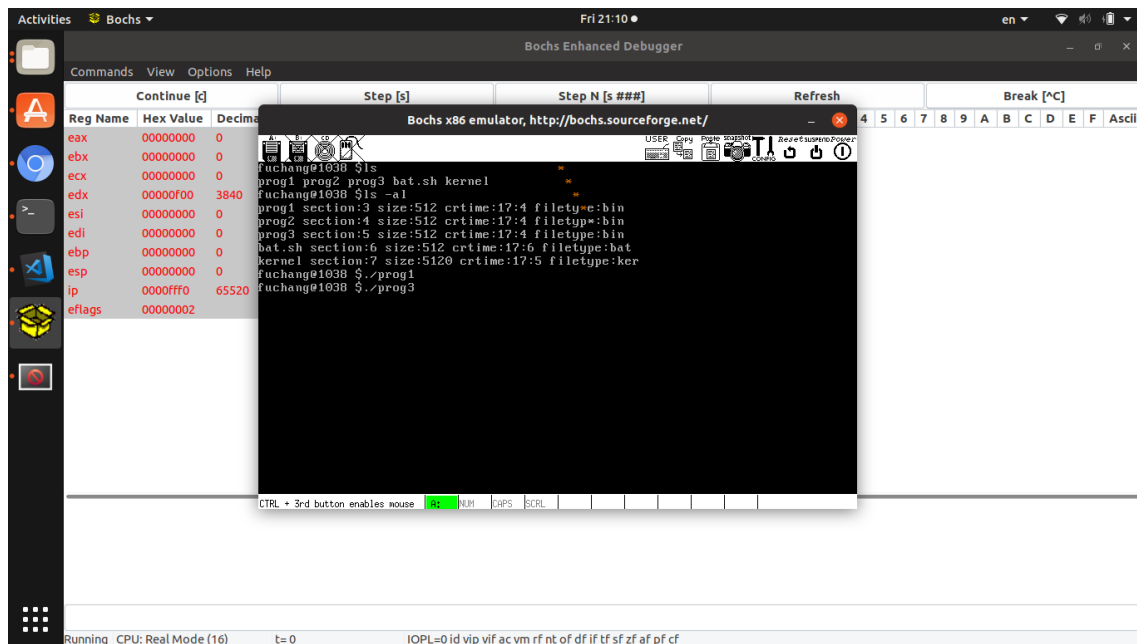


图 6: Shot3

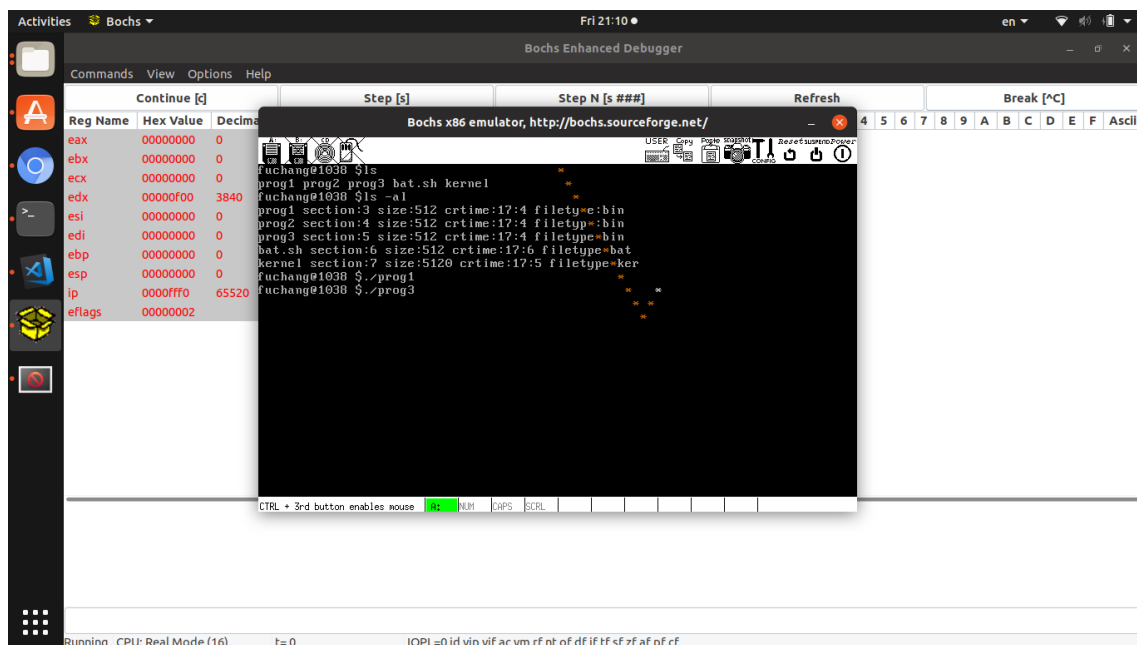


图 7: Shot3



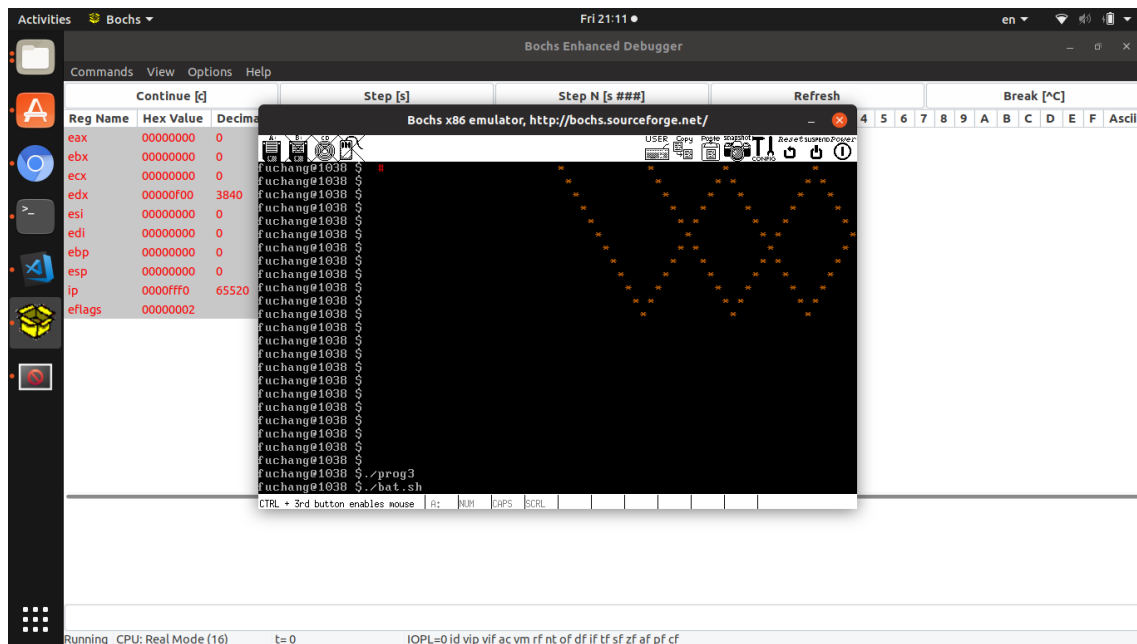


图 10: Shot3

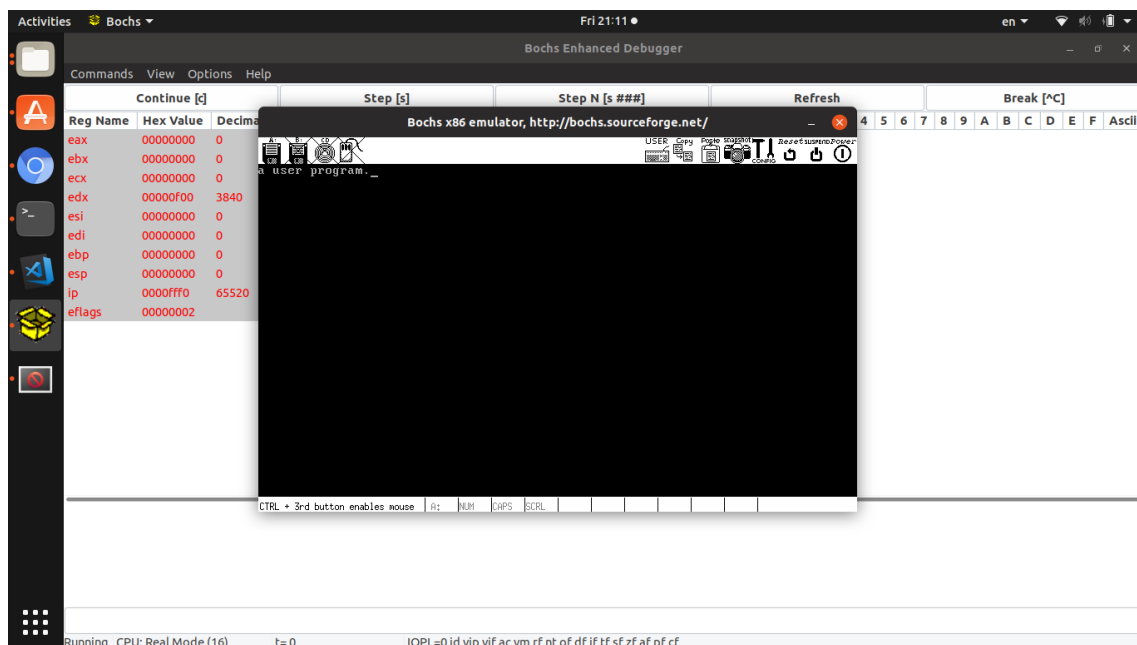


图 11: Shot3



文字描述或者截图（所作的图）。必须有截图，截图数量不少于 2 幅。

- 结果分析

对每一个结果，必须有相应分析，如解释图表反映的内涵、缘由，是否达到预期目标，是否可改进等等。)

## 五、 总结及心得体会:

(自行填写。必须写点什么，不能写“无”)

## 六、 对本实验过程及方法、手段的改进建议:

(自行填写。必须写点什么，不能写“无”)

(注意：八，九部分能反映出实验的态度、方法和效果，应重点阐述，字数勿少，独立完成，勿参考其他报告，避免雷同)

报告评分： X X X

指导教师签字： X X X

## 附录一 代码示例

```
1 #include <stdio.h>
2 int main (int argc, char *argv[]){
3     printf("Hello world!");
4 }
```

代码 14: 一段 C 代码

## 附录二 表格示例

表 1: 一个简单的表格

功能	WEB	APP
注册	√	√
登录	√	√
推送	×	√

表 2: 自定义表格

功能	WEB	APP
注册	√	√
登录	√	√
推送	×	√

附录三 图片示例



图 12: 电子科技大学

---

**算法 1** 某个算法

---

输入: 某个输入

输出: 某个输出

1: **function** 函数名 (参数列表)

2:     某个变量  $\leftarrow$  某个变量

3: **end function**

---

## 附录四 字体示例

黑体 华文行楷