Day 20

我们想在我们应用程序中实现一个类似于printf之类的函数，可以输出东西到控制台。但是目前为止我们做不到，因为控制台的输出是系统控制的，应用程序无法直接在控制台上输出。所以我们要设计API，应用程序通过调用API，来把他想打印的东西告诉系统，然后系统在控制台上为他进行输出。

先来实现单字符输出。

主要思路是在系统的代码中有一个cons_putchar()函数，然后我们可以使用nask的call关键字进行跳转。我们打算把参数存在寄存器当中，但是这样又面临着另一个问题。我们操作系统中的cons_putchar()函数是用C语言来写的，C语言是无法直接获得寄存器的值的。所以我们还要给这个cons_putchar()包个壳，定义个新的nask函数asm_cons_putchar()。该函数从EAX寄存器中获取需要打印的字符，然后将他们压入栈中（作为参数），再进行call，然后将之前压入栈中的内容弹出。这样就完成了整个API调用流程。

```
_asm_cons_putchar:
    PUSH 1              ; 参数是从EPS+4开始的，所以往里面push一个int占位
    AND EAX,0xff        ; 把参数mask以下，避免应用程序错误的调用API造成错误
    PUSH EAX            ; 将EAX压入栈中作为C语言函数参数
    PUSH DWORD [0x0fec] ; 要打印的控制台地址
    CALL _cons_putchar  ; C语言函数地址
    ADD ESP,12          ; 将栈中的数据丢弃
    RET
```

控制台地址的0x0fec是怎么回事呢？应用程序没法直接获取控制台的地址，所以我们约定把这个地址存储在0x0fec位置上，到时候只要去找就ok了。

```
*((int*) 0x0fec) = (int) &cons;
```

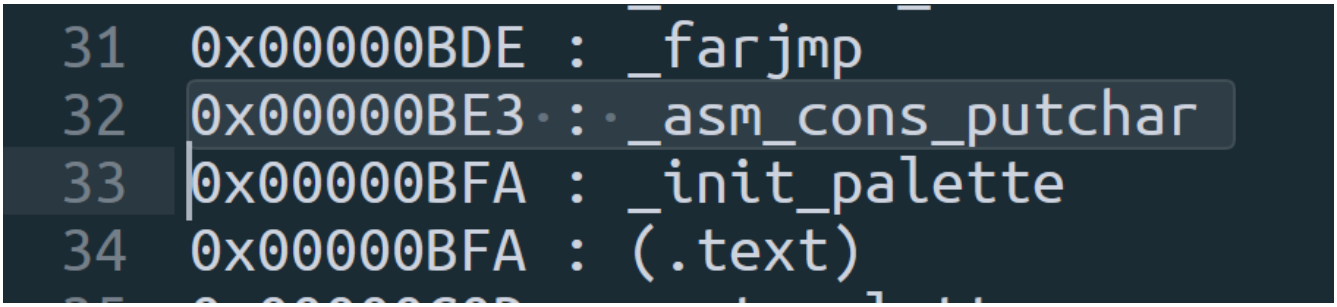
然后我们如何调用这段程序呢？注意操作系统与应用程序的编译并不相关，在编译应用程序的时候并不知道asm_cons_putchar()的存在。

这里还需要我们处理一下。我们可以先编译操作系统，然后从中得知asm_cons_putchar()函数的地址，然后把他硬写在应用程序的代码当中。

修改makefile，使他能够输出每个函数的地址。

```
bootpack.bim : $(OBJS_BOOTPACK) Makefile
    $(OBJ2BIM) @$(RULEFILE) out:bootpack.bim stack:3136k map:bootpack.map \
        $(OBJS_BOOTPACK)
```

查看bootpack.map



从中我们知道了asm_cons_putchar()的地址是0x0be3，所以我们可以按与如下方式类似的方法调用我们的API

```
MOV AL,'A'
CALL 0xbe3
```

对了！还有一个问题，仅仅这样是不行的，仅仅这样call，是call不到我们的api的。操作系统所在的段时2，所以我们还需要补充一个段号。改成

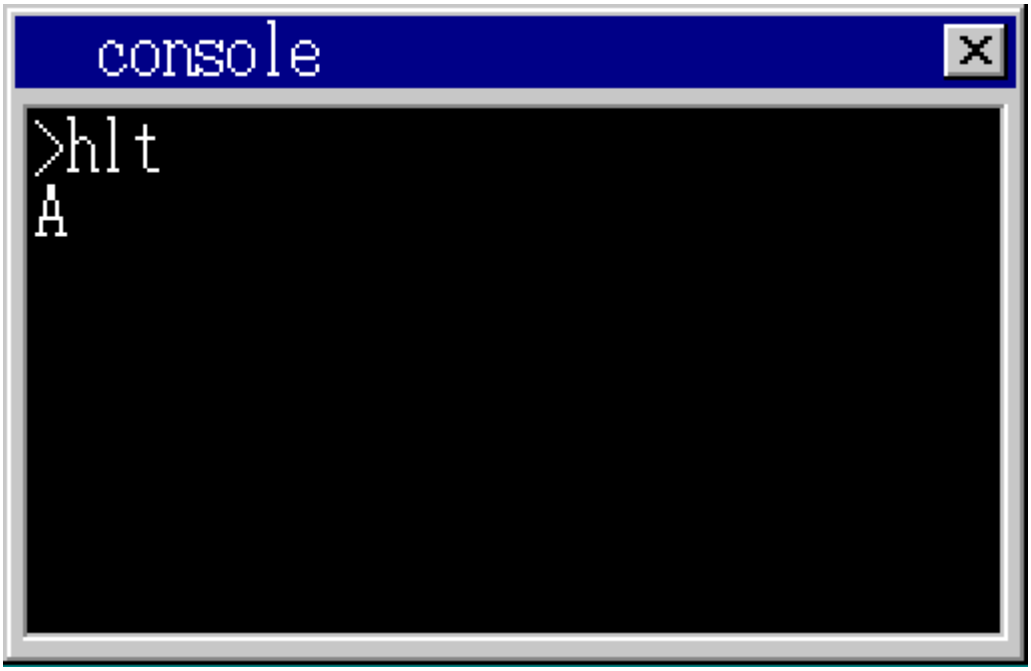
```
CALL 2*8: 0xbe3
```

使用了farcall，对应的也应该使用far ret。所以我们还要修改asm_cons_putchar()

将最后的RET修改成RETF就ok了。

```
_asm_cons_putchar:
    PUSH 1                ; 参数是从EPS+4开始的，所以往里面push一个int占位
    AND EAX,0xff          ; 把参数mask以下，避免应用程序错误的调用API造成错误
    PUSH EAX              ; 将EAX压入栈中作为C语言函数参数
    PUSH DWORD [0x0fec] ; 要打印的控制台地址
    CALL _cons_putchar  ; C语言函数地址
    ADD ESP,12            ; 将栈中的数据丢弃
    RETF
```

跑一下吧



成功了!

不过现在应用程序运行起来仍然有问题，我们无法从应用程序返回操作系统了。这可不妙。为了系统能够正常的返回，我们应当使用call和ret。由于应用程序和操作系统不在相同的段中，所以我们要用far call和far ret。

为了在c语言中实现far call，需要编制naskfunc。

```
_farcall: ; void farcall(int eip, int cs);
    CALL FAR [ESP+4] ; eip, cs
    RET
```

然后调用程序的代码修改成为

```
farcall(0, 1003 * 8);
```

应用程序代码也需要修改，把hlt换成retf就ok了。同时需要注意，由于我们修改了操作系统的代码，API的地址会发生变化，所以我们需要重新查找asm_cons_putchar()的地址并写入

```
/* --- hlt.nas --- */
    MOV AL,'A'
    CALL 2*8:0xbe8
    RETF
```

不过这样好烦啊，每次迭代源码都需要重新查表修改应用程序源码，当API变多、API调用变多的时候，这将成为一场灾难。

记得我们之前通过设置IDT来用C语言来处理中断吗？IDT上还有好多空闲的中断号没有使用，我们可以用其中的一个中断号来代替我们的API地址。这样，调用API只需要触发随营的软中断就好了。

要做的工作有：

- 在IDT中注册asm_cons_putchar()

- 修改asm_cons_putchar()推出语句为IRETD
- 在asm_cons_putchar()开始处进行sti（这是因为CPU会自动关闭中断，导致无法响应键盘、鼠标等操作）

make run下，运行顺利

接下来我们要让系统支持输入文件名，运行对应的程序。

具体思路如下 如果我们输入的命令是

- mem
- cls
- dir
- type

其中之一的话，我们执行对应的控制台功能，否则我们查找是否存在<命令>或者<命令>.hrb文件，如果存在的话，我们就执行该文件。

由此，需要编制一个cmd_app函数，他找到并执行一个程序

```
int cmd_app(struct CONSOLE *cons, int *fat, char *cmdline)
{
    struct MEMMAN *memman = (struct MEMMAN *) MEMMAN_ADDR;
    struct FILEINFO *finfo;
    struct SEGMENT_DESCRIPTOR *gdt = (struct SEGMENT_DESCRIPTOR *) ADR_GDT;
    char name[18], *p;
    int i;
    for (i = 0; i < 13; i++) {
        if (cmdline[i] <= ' ') {
            break;
        }
        name[i] = cmdline[i];
    }
    name[i] = 0;
    finfo = file_search(name, (struct FILEINFO *) (ADR_DISKIMG + 0x002600), 224);   // 尝试
查找文件
    if (finfo == 0 && name[i - 1] != '.') { // 找不到，加后缀名再试一遍
        name[i    ] = '.';
        name[i + 1] = 'H';
        name[i + 2] = 'R';
        name[i + 3] = 'B';
        name[i + 4] = 0;
        finfo = file_search(name, (struct FILEINFO *) (ADR_DISKIMG + 0x002600), 224);

    if (finfo != 0) {   // 找到则读取文件到内存并执行
        p = (char *) memman_alloc_4k(memman, finfo->size);
        file_loadfile(finfo->clustno, finfo->size, p, fat, (char *) (ADR_DISKIMG +
0x003e00));
        set_segmdesc(gdt + 1003, finfo->size - 1, (int) p, AR_CODE32_ER);
        farcall(0, 1003 * 8);
        memman_free_4k(memman, (int) p, finfo->size);
        cons_newline(cons);
        return 1;   // 找到返回1
```

```
    }
    return 0;       // 找不到返回0
}
```

之前我们改用中断来做API其实漏了一个东西，我们应该保存CPU现场的，否则会出问题

再中断服务程序开始的地方加PUSHAD，结束的地方加POPAD。这样就OK了

我们来实现更多的API吧。由于API可能会有很多，一个API对应一个中断号不太现实。

根据课堂上学到的知识我们知道，现代操作系统的API基本上都是通过中断号加功能号来实现的，我们也来使用这样的设计。

我们编写两个新的API，一种是显示一串字符，遇到字符编码0则结束；另一种是先指定好要显示的字符串的长度再显示。

```
void cons_putstr0(struct CONSOLE *cons, char *s)
{
    for (; *s != 0; s++) {
        cons_putchar(cons, *s, 1);
    }
    return;
}
void cons_putstr1(struct CONSOLE *cons, char *s, int l)
{
    int i;
    for (i = 0; i < l; i++) {
        cons_putchar(cons, s[i], 1);
    }
    return;
}
```

然后我们分配功能号

功能号	功能
1	显示单个字符（AL = 字符编码）
2	显示字符串 0（EBX = 字符串地址）
3	显示字符串 1（EBX = 字符串地址，ECX = 字符串长度）

为API引入统一的入口。

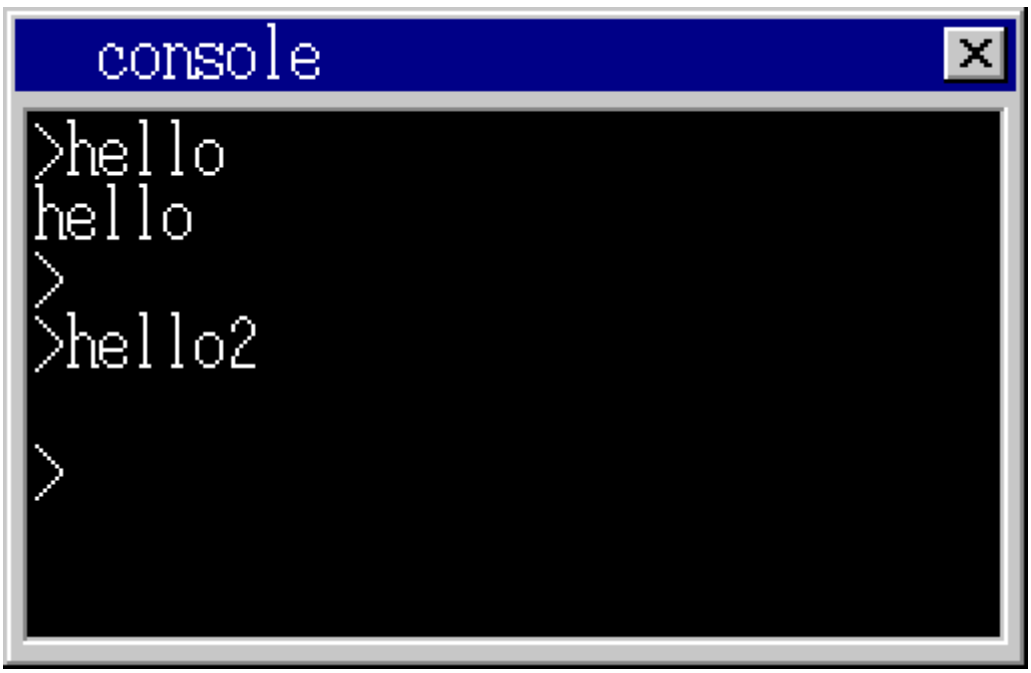
```
_asm_hrb_api:
        STI
        PUSHAD ; 用于保存寄存器值的PUSH
        PUSHAD ; 用于向hrb_api传值的PUSH
        CALL _hrb_api
        ADD ESP,32
        POPAD
        IRETD
```

```
void hrb_api(int edi, int esi, int ebp, int esp, int ebx, int edx, int ecx, int eax)
{
    struct CONSOLE *cons = (struct CONSOLE *) *((int *) 0x0fec);
    if (edx == 1) {
        cons_putchar(cons, eax & 0xff, 1);
    } else if (edx == 2) {
        cons_putstr0(cons, (char *) ebx);
    } else if (edx == 3) {
        cons_putstr1(cons, (char *) ebx, ecx);
    }
    return;
}
```

然后修改IDT注册部分。

make run一下

啥也没有？？？？？



作者要把这个放到day 21？

不成，今天必须给弄了。

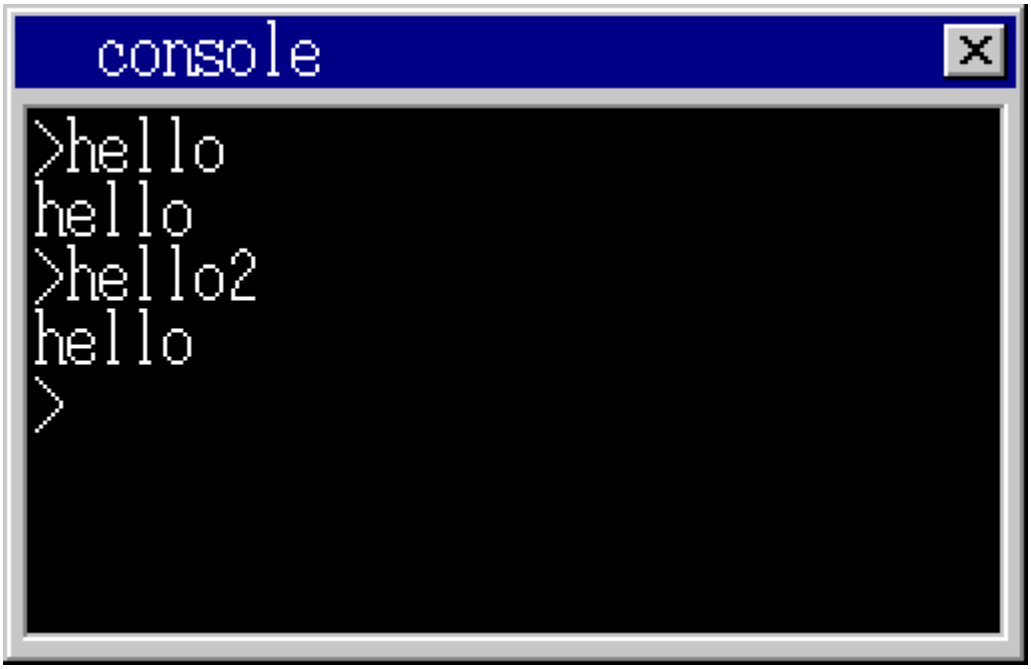
通过阅读Day21的内容，了解到原来是代码段没有正确的设置而导致了这个错误，我们只要正确的设置代码段就可以了。

我们想办法把我们之前为应用程序准备的内存地址给传过去

```
/* --- partial content of cmd_app --- */
if (finfo != 0) {
        p = (char *) memman_alloc_4k(memman, finfo->size);
        *((int *) 0xfe8) = (int) p; // 把分配的内存地址存到0xfe8位置
        file_loadfile(finfo->clustno, finfo->size, p, fat, (char *) (ADR_DISKIMG +
0x003e00));
        set_segmdesc(gdt + 1003, finfo->size - 1, (int) p, AR_CODE32_ER);
        farcall(0, 1003 * 8);
        memman_free_4k(memman, (int) p, finfo->size);
        cons_newline(cons);
        return 1;
    }
```

```
void hrb_api(int edi, int esi, int ebp, int esp, int ebx, int edx, int ecx, int eax)
{
    struct CONSOLE *cons = (struct CONSOLE *) *((int *) 0x0fec);
    int cs_base = *((int *) 0xfe8); // 取出段地址
    if (edx == 1) {
        cons_putchar(cons, eax & 0xff, 1);
    } else if (edx == 2) {
        cons_putstr0(cons, (char *) ebx + cs_base); // 加上段地址
    } else if (edx == 3) {
        cons_putstr1(cons, (char *) ebx + cs_base, ecx); // 加上段地址
    }
    return;
}
```

make run



OK，成功