## 操作系统

## 实验报告

# 班号：\_\_\_\_1603102\_\_\_\_\_\_

# 学号：\_\_\_1160301023\_\_\_\_\_\_\_

# 姓名：\_\_安宏展\_\_\_\_\_\_\_\_

1. 请简述head.s 的工作原理。
2. 请记录head.s 的内存分布状况，写明每个数据段，代码段，栈段的起始与终止的内存地址。
3. 简述Head.s 57~62行在做什么？简述iret执行后，pc如何找到下一条指令？并记录iret执行前后，栈是如何变化的。
4. 当任务进行系统调用时，即int 0x80时，记录栈的变化情况。当执行完system\_interrupt 函数，执行153行iret时，记录栈的变化情况。此外，当进入和退出system\_interrupt时，都发生了模式切换，请总结模式切换时，特权级是如何改变的？栈切换吗？如何进行切换的？
5. 当时钟中断发生，进入到timer\_interrupt程序，请**详细记录**从任务0切换到任务1，以及又过了10ms，从任务1切换回到任务0，整个流程是怎样的？TSS是如何变化的？各个寄存器的值是如何变化的？请详细总结任务切换的过程。
6. 请简述head.s 的工作原理。

利用中断实现字符打印以及任务切换

内存只有一个，而且空间有限，我们要在一个有限的空间里完成任务的执行，就需要用合理的方式管理内存

1. 请记录head.s 的内存分布状况，写明每个数据段，代码段，栈段的起始与终止的内存地址。

代码段

|  |  |  |
| --- | --- | --- |
| 存储内容 | 开始地址 | 结束地址 |
| Startup\_32 | 0x0 | 0xac |
| Setup\_gdt | 0xad | 0xb4 |
| Setup\_idt | 0xb5 | 0xe3 |
| Write\_char | 0xe4 | 0x113 |
| Ignore\_int | 0x114 | 0x129 |
| Nop | 0x12a | 0x12b |
| Timer\_interrupt | 0x12c | 0x167 |
| System\_interrupt | 0x168 | 0x17e |
| Task0 | 0x10e0 | 0x10f3 |
| Task1 | 0x10f4 | 0x1100 |

数据段：

|  |  |  |
| --- | --- | --- |
| 存储内容 | 开始地址 | 结束地址 |
| Idt | 0x198 | 0x997 |
| Gdt | 0x998 | 0x9D7 |
| Ldt0 | 0xbe0 | 0xbf7 |
| Ldt1 | 0xe60 | 0xe77 |
| TSS0 | 0xbf8 | 0xc60 |
| TSS1 | 0xe78 | 0xee0 |

栈段:(高地址开始，低地址结束)

|  |  |  |
| --- | --- | --- |
| 存储内容 | 开始地址 | 结束地址 |
| Init\_Stack | 0xbd8 | 0x9d8 |
| Task0\_Kernel\_Stack | 0xe60 | 0xc60 |
| Task1\_Kernel\_Stack | 0x10e0 | 0xee0 |
| User\_Stack | 0x1301 | 0x1101 |

1. 简述Head.s 57~62行在做什么？简述iret执行后，pc如何找到下一条指令？并记录iret执行前后，栈是如何变化的。

(1)

push $0x17 ;将任务0的栈段的选择子入栈

push $init\_stack(0xbd8) ;将任务0的栈段的偏移地址入站

pushf ;将标志寄存器eflags入栈

push $0x0f ;将当前局部空间代码段选择子cs入栈

push $task0(0x10e0) ;将当前局部空间代码段偏移入栈

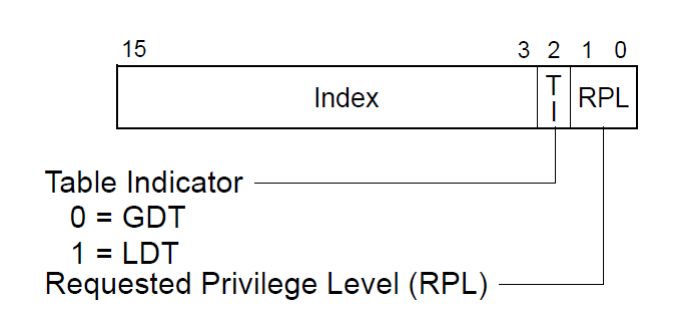
iret ;执行中断返回指令，使用弹栈的方式讲 ;当前的特权等级从0变为3

1. iret执行后，cs与esp中会存储特权等级在内核空间时候的基地址以及偏移量，根据cs中选择子找出当前的基地址，然后与esp中存储的偏移量相加就可以得到当前的pc值

具体内容：cs 中存储0x0f，0x0f转换成二进制为：



选择子的格式如下：



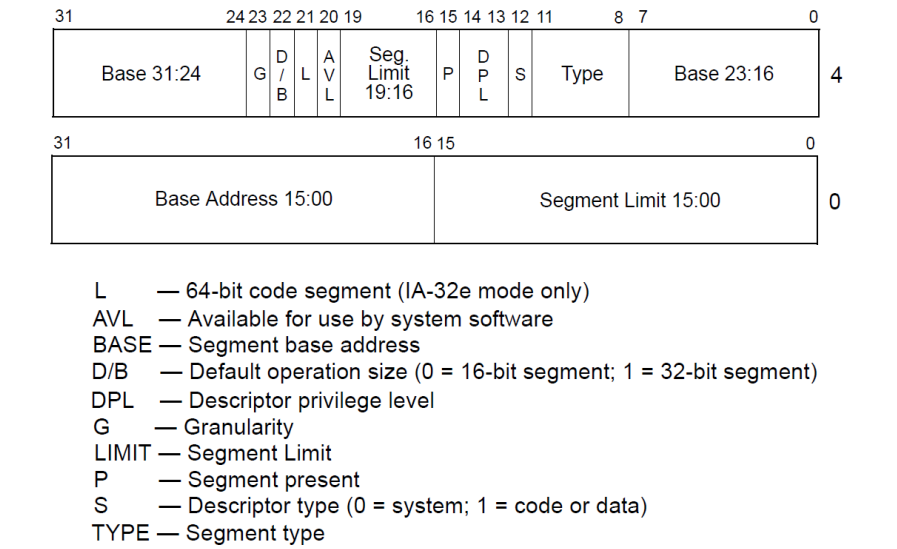
0x17对应TI位为1，说明要去查询LDT表格，索引为0x10，说明查询GDT的第二项，第二项的内容为



将其转换为为二进制如下：



GDT中存储的是描述符，描述符的结构如下：



我们关注其中基地址，将其32位基地址拼凑出来会发现基地址为0x0，eip中存储的是0x10e0，故PC的下一次会跳转到0x10e0+0x0的位置去去指令

1. 栈的变化：

iret执行之前，我们可以观察虚拟机中：

在执行第一句push的时候，ss中的选择子为0x10，按照选择子的结构以及描述符的结构转换过来是基地址为0的数据段，esp中存储的是0xbd8,此时栈中没有任何内容，将所有的push指令执行后会发现栈中的内容如下：

|  |
| --- |
| 0x00000BC4 : 0x10e0 ;$task0 |
| 0x00000BC8 : 0x0f ;cs |
| 0x00000BC0 : 0x246 ;eflags |
| 0x00000BD0 : 0xbd8 ;esp |
| 0x00000BD4 : 0x17 ;ss |
| 0x00000BD8 : 0xbd8 |

iret执行之后栈继续变为空

|  |
| --- |
| 0x00000BD8 : 0xbd8 |

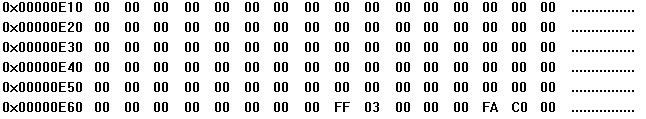
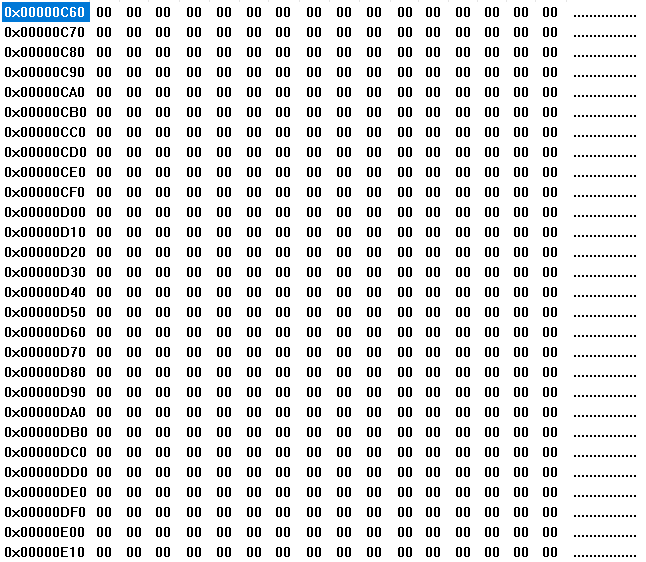
4、当任务进行系统调用时，即int 0x80时，记录栈的变化情况。当执行完system\_interrupt 函数，执行153行iret时，记录栈的变化情况。此外，当进入和退出system\_interrupt时，都发生了模式切换，请总结模式切换时，特权级是如何改变的？栈切换吗？如何进行切换的？

(1)当任务进行系统调用时，栈的情况（我们当前执行的是任务0）：

在执行0x80号中断之前栈中是空的，系统栈如下：

|  |
| --- |
| 0x00000BD8 : 0xbd8 |

任务0的内核栈(0xe60~0xc60)



执行int 0x80时，会将当前的状态压入栈中，包括ss、esp、eflags、cs、下一条指令的eip，然后去执行system\_interrupt()函数

|  |
| --- |
| 0x00000E4C : 0x10eb ;$task0 + b  b大概是从task0第一句执行到int 0x80指令之间的长度 |
| 0x00000E50 : 0x0f ;cs |
| 0x00000E54 : 0x246 ;eflags |
| 0x00000E58 : 0xbd8 ;esp |
| 0x00000E5C : 0x17 ;ss |
| 0x00000E60 : 0x0 |

（2）在执行system\_interrupt()函数的iret之前，栈的情况如下

|  |
| --- |
| 0x00000E4C : 0x10eb ;eip |
| 0x00000E50 : 0x0f ;cs |
| 0x00000E54 : 0x246 ;eflags |
| 0x00000E58 : 0xbd8 ;esp |
| 0x00000E5C : 0x17 ;ss |
| 0x00000E60 : 0x0 |

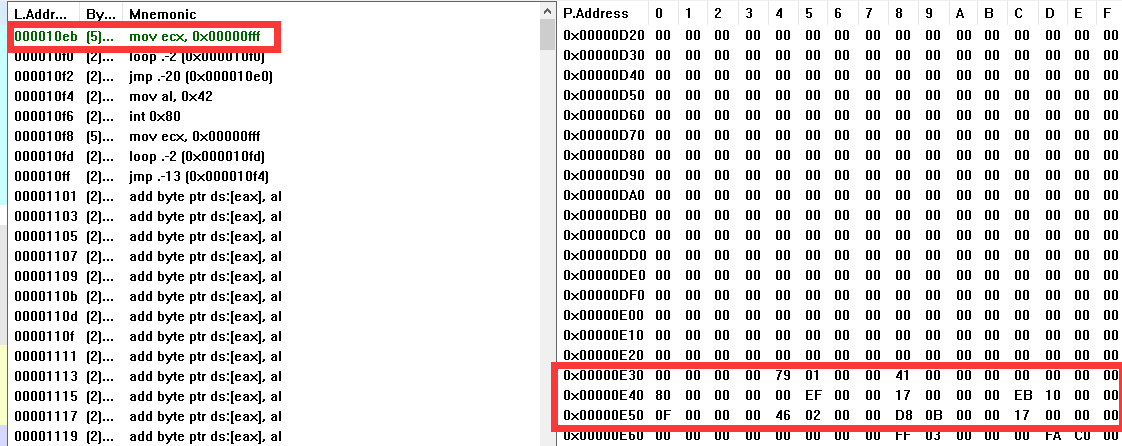
会发现与刚刚进入system\_interrupt()函数之前完全相同

执行iret之后：使用的栈会切换为系统栈，系统栈依旧为空

|  |
| --- |
| 0x00000BD8 : 0xbd8 |



iret执行过后的语句的地址为0x10eb，也印证我们前面所说的eip的内容，此时观察用户内核栈中的内容(0xe60~0xc60)，会发现虽然发生弹栈将寄存器中的内容返还给寄存器，说明弹栈操作只是将指针的位置改变，内存中的内容并未改变



（3）

进入system\_interrupt()时，从用户模式切换到内核模式，特权等级从3变为0，栈从系统栈转换到当前任务的内核栈

退出system\_interrupt()时，从内核模式切换到用户模式，特权等级从0变为3，栈从当前任务的内核栈转换到系统栈

5、当时钟中断发生，进入到timer\_interrupt程序，请**详细记录**从任务0切换到任务1，以及又过了10ms，从任务1切换回到任务0，整个流程是怎样的？TSS是如何变化的？各个寄存器的值是如何变化的？请详细总结任务切换的过程。

由于前面已经记录了执行0x80号中断时栈的变化，所以在这里不予重复

任务0->0x08号中断

时钟中断发生前操作系统处于用户模式，使用系统栈，栈空

|  |
| --- |
| 0x00000BD8 : 0xbd8 |

时钟中断发生时，操作系统从用户模式切换到内核模式，栈变为任务0对应的内核栈(0xe60~0xc60)，将中断位置处系统的状态压栈保存，如图所示：

|  |
| --- |
| 0x00000E4C : 0x10f0 ;eip |
| 0x00000E50 : 0x0f ;cs |
| 0x00000E54 : 0x246 ;eflags |
| 0x00000E58 : 0xbd8 ;esp |
| 0x00000E5C : 0x17 ;ss |
| 0x00000E60 : 0x0 |

执行到jmp指令之前又存储了两项：

|  |
| --- |
| 0x00000E44 : 0x41 ;eax |
| 0x00000E48 : 0x17 ;ds |

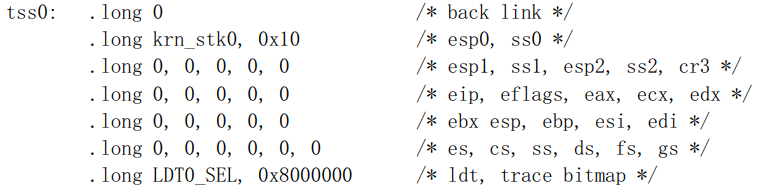
寄存器的值（我们这里只记录ss、esp、eflags、cs、eip）：

|  |  |  |
| --- | --- | --- |
| 寄存器名 | 变化前的值 | 变化后的值 |
| ss | 0x17 | 0x17 |
| esp | 0xbd8 | 0xe44 |
| eflags | 0x246 | 0x46 |
| cs | 0x0f | 0x08 |
| eip | 0x10f0 | 0x12e |

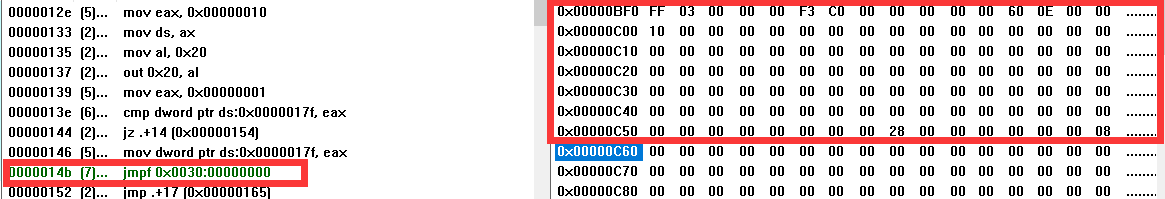
0x08号中断->任务一

这个切换过程中任务0的状态会保存在TSS0中，故只有TSS0更新

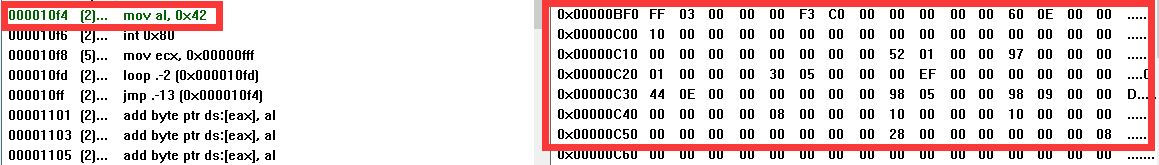
由于第一次执行任务0时寄存器的内容是从栈中弹出的，所以TSS0预先设置时大部分内容都初始化为0，如下图所示：



也可以从内存中查看，TSS0在内存中起始位置为0xbf8，共104个字节，如下所示：



可以看出还未执行jmp指令，即还未跳转到task1，此时TSS0的内容与预先设置的相同，执行jmp指令之后



任务1->0x08号中断

任务一执行中断之前处于用户模式，使用内核栈，栈空

|  |
| --- |
| 0x00000BD8 : 0xbd8 |

执行中断跳转到内核模式，使用任务一自己的内核栈(0x10e0~0xee0)，栈中存储着任务一在跳转时的ss、esp、eflags、cs、eip

|  |
| --- |
| 0x000010CC : 0x10fd ;eip |
| 0x000010D0 : 0x0f ;cs |
| 0x000010D4 : 0x202 ;eflags |
| 0x000010D8 : 0x1301 ;esp |
| 0x000010DC: 0x17 ;ss |
| 0x000010E0 : 0x17b8 |

执行到跳转指令之前又压入两项内容：

|  |
| --- |
| 0x000010C4 : 0x42 ;eax |
| 0x000010C8 : 0x17 ;ds |

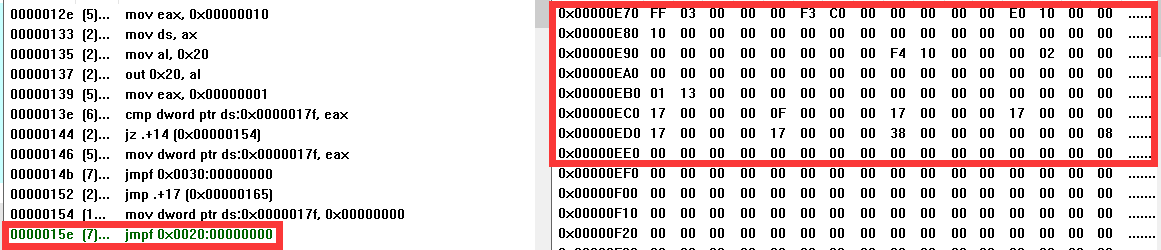
寄存器的值（我们这里只记录ss、esp、eflags、cs、eip）：

|  |  |  |
| --- | --- | --- |
| 寄存器名 | 变化前的值 | 变化后的值 |
| ss | 0x17 | 0x10 |
| esp | 0x1301 | 0x10c4 |
| eflags | 0x202 | 0x02 |
| cs | 0x0f | 0x08 |
| eip | 0x10fd | 0x12e |

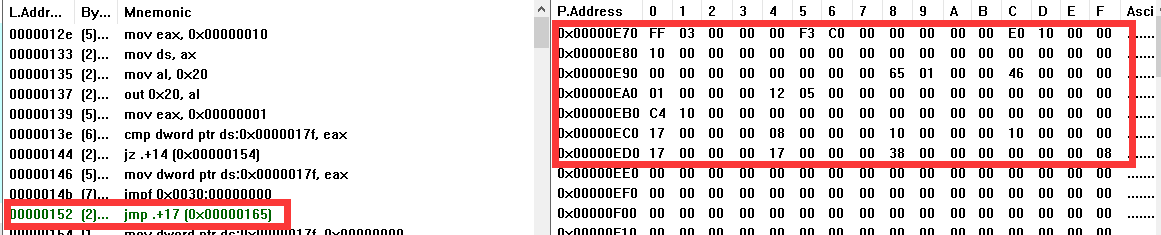
0x08号中断->任务0

跳转的时候需要将task1的状态保存在TSS1中，TSS1(0xe78~0xee0)

内容如下：



执行过jmp之后，恢复到之前task0之前中断的状态



此时发生了栈的切换，切换到task0的内核栈，此时栈中的内容：

|  |
| --- |
| 0x000010C4 : 0x42 ;eax |
| 0x000010C8 : 0x17 ;ds |
| 0x00000E4C : 0x10f0 ;eip |
| 0x00000E50 : 0x0f ;cs |
| 0x00000E54 : 0x246 ;eflags |
| 0x00000E58 : 0xbd8 ;esp |
| 0x00000E5C : 0x17 ;ss |
| 0x00000E60 : 0x0 |

继续向下执行是两句pop指令，然后是iret指令

|  |
| --- |
| 0x00000E4C : 0x10f0 ;eip |
| 0x00000E50 : 0x0f ;cs |
| 0x00000E54 : 0x246 ;eflags |
| 0x00000E58 : 0xbd8 ;esp |
| 0x00000E5C : 0x17 ;ss |
| 0x00000E60 : 0x0 |

iret指令执行之后系统跳转到用户模式，栈切换为内核栈，依旧为空

|  |
| --- |
| 0x00000BD8 : 0xbd8 |