**Report for Lab08**

**Stu：金泽文**

**No.PB15111604**

**实验目的：**

在实验7的基础上，实现Round Robin算法的进程调度机制。

**实验内容：**

1. 实现中断初始化和中断管理，提供缺省中断处理入口
2. 实现开中断和关中断，接口enbleIRQ()、disableIRQ()
3. 实现时钟初始化和时钟中断
4. 实现时间片轮转调度算法
5. 修改osStart原语，以增加相关功能的初始化

**我的完成情况：**

我完成了所有内容：

1. 时钟中断的实现，右上角时钟的实现。
2. FCFS调度
3. Round Robin调度。

当然，完成的还有整整四天的调试经历之后的心性的提升。

时间片：1s

**关于中断机制的学习与调研，以及对应的实现：**

1. 可编程中断控制器8259A：

可屏蔽中断与CPU的关系通过8259A建立起来。8259A由主从两片组成。初始化的过程是：

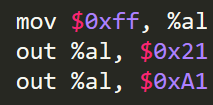
1.往端口20h（ 主片） 或A0h（ 从片） 写入ICW1。

2.往端口21h（ 主片） 或A1h（ 从片） 写入ICW2。

3.往端口21h（ 主片） 或A1h（ 从片） 写入ICW3。

4.往端口21h（ 主片） 或A1h（ 从片） 写入ICW4。

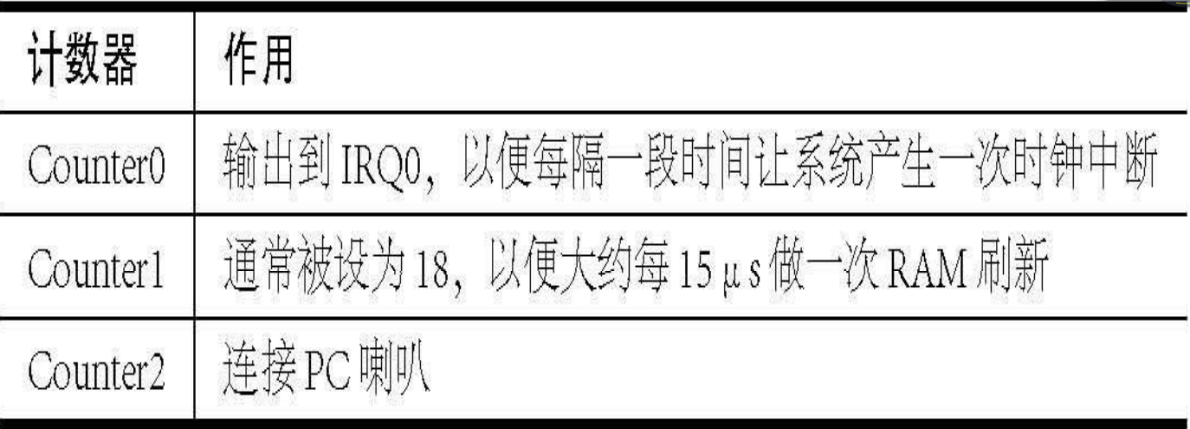
ICW的格式如下：

所以，先置ff：

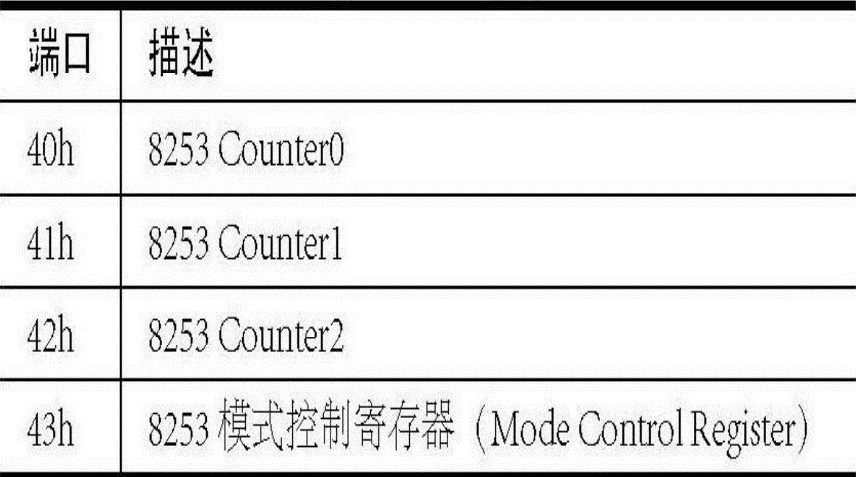
再按照格式依次设置。其中中断向量0x20对应IRQ0，0x28对应IRQ8。所以IRQ0~IRQ7对应用户定义中断的0x20~0x27，IRQ8~IRQ15对应0x28~0x2f。

2.8253-时钟中断的机制：

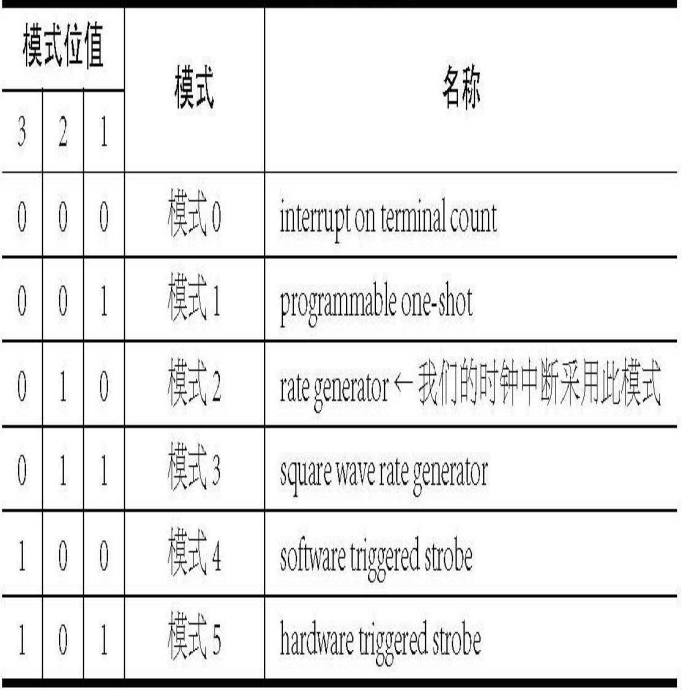
时钟中断由PIT（Programmable Interval Timer）芯片触发。在IBM XT中，这个芯片用的是Intel 8253，所以用8253称呼。

 8253有3个计数器，如图：

时钟中断由Counter0产生。

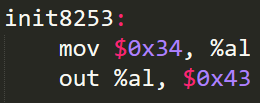
 计数器的工作原理：PC上的输入频率是1193180Hz，每个周期计数器值减1，减到0时触发。为了让系统每10ms触发一次中断，也就是让我们的时间片为10ms，所以需要设置计数器为11931。为了设置，需要知道8253的端口，如下：

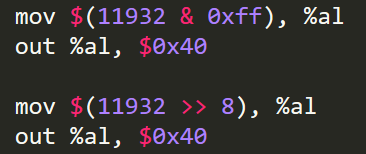
为了写我们的Counter0端口，需要先设置43h的MCR寄存器。MCR格式如下：

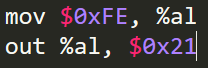
模式位如下：

 读写位如下：

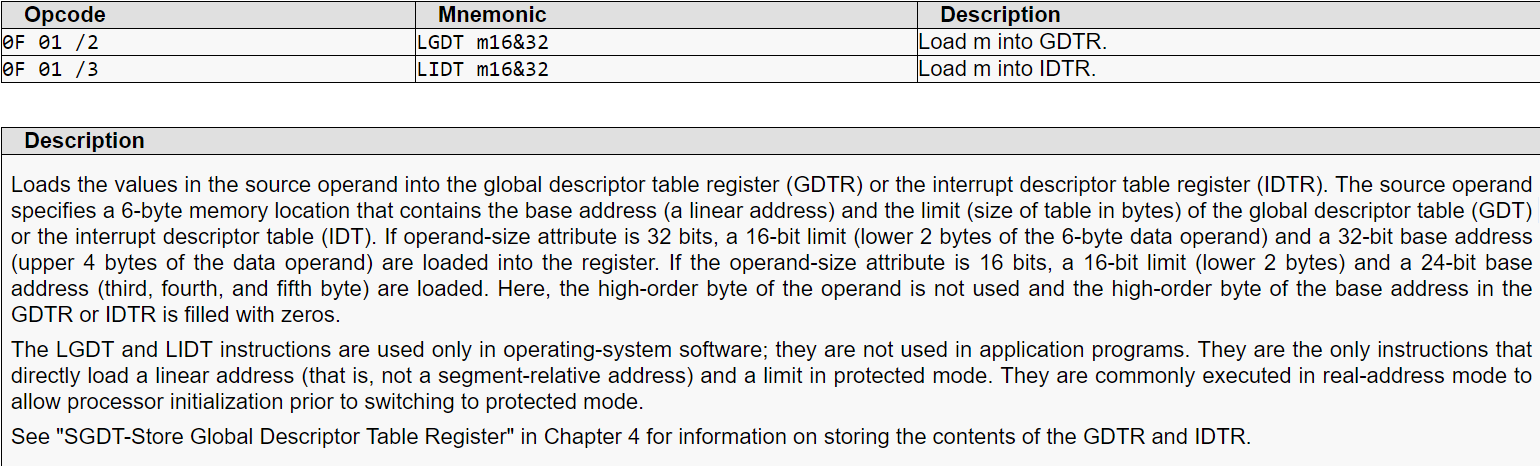
 选择位如下：

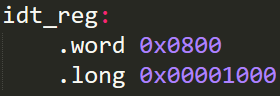
 根据以上格式，0x43处应该写入0x34。

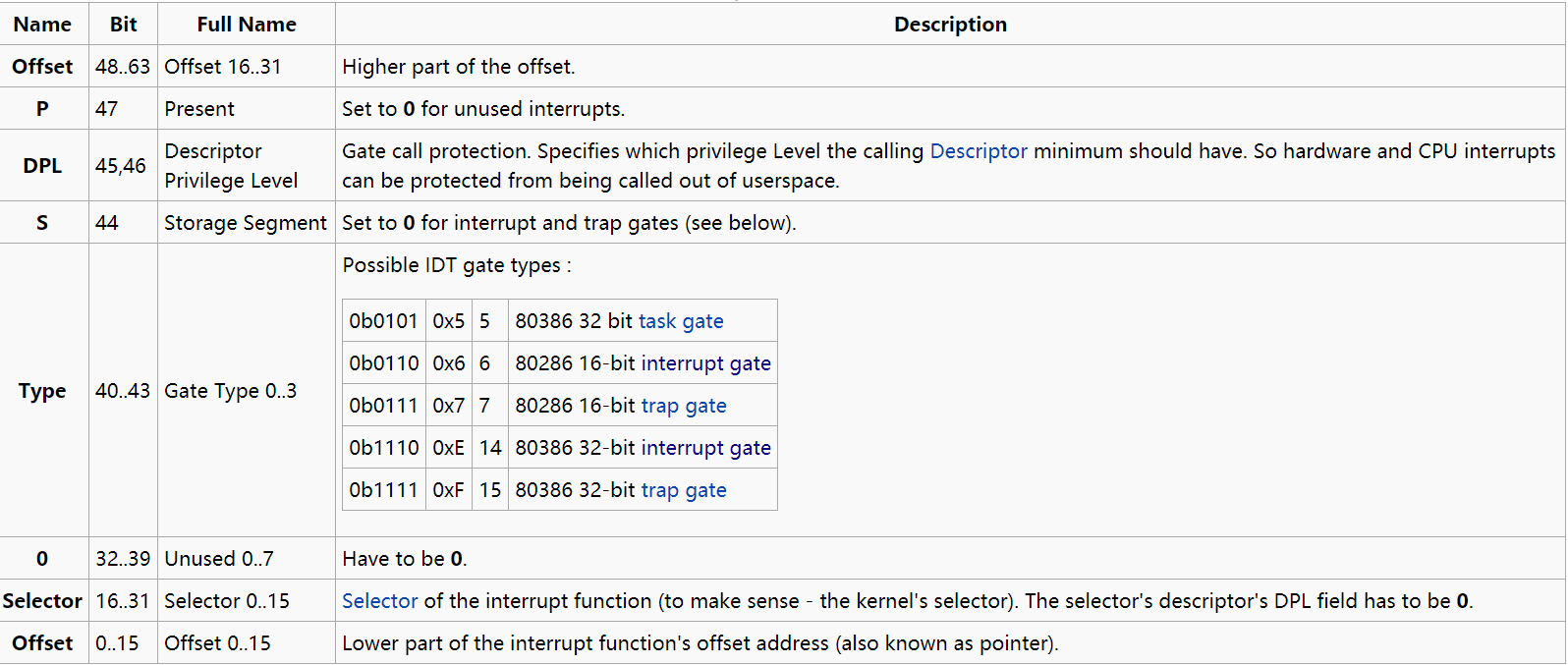
 按字节输出到0x40。

 最后，为了打开时钟中断，需要

2.建立并初始化IDT

 根据google到的下图，得到lidt指令的用法。

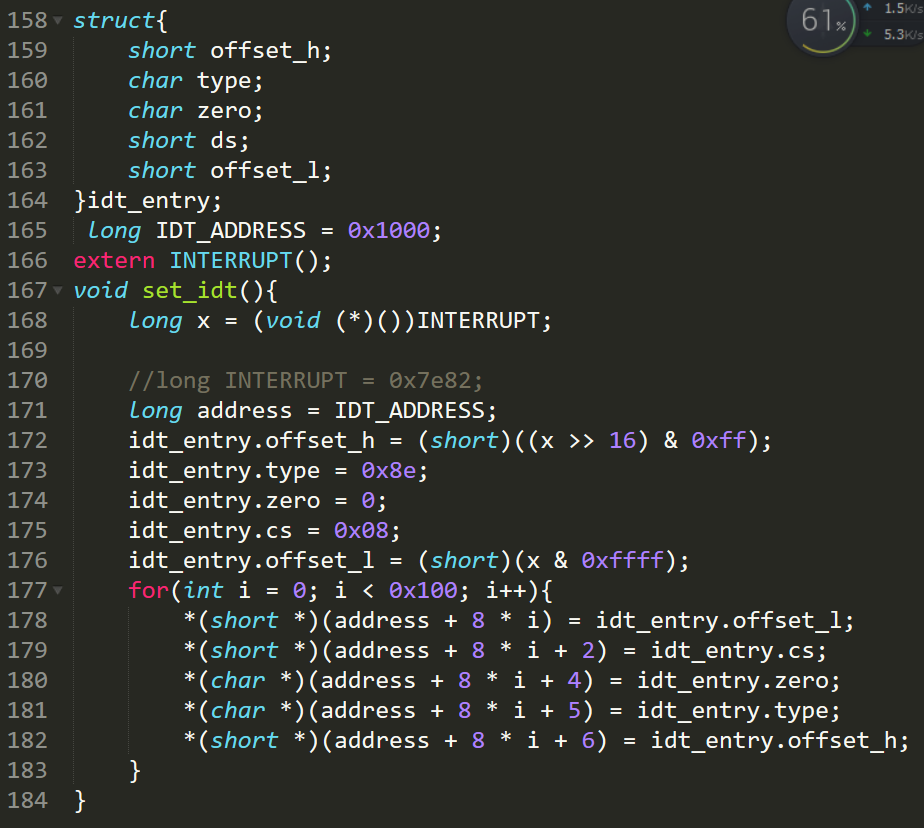
 所以得到：

查到idt中entry的格式：

这一部分我选择的方案是编写c函数然后在32.s中调用

根据上图，p设置为1，dpl，s设置为0，type设置为e；selector设置为ds，

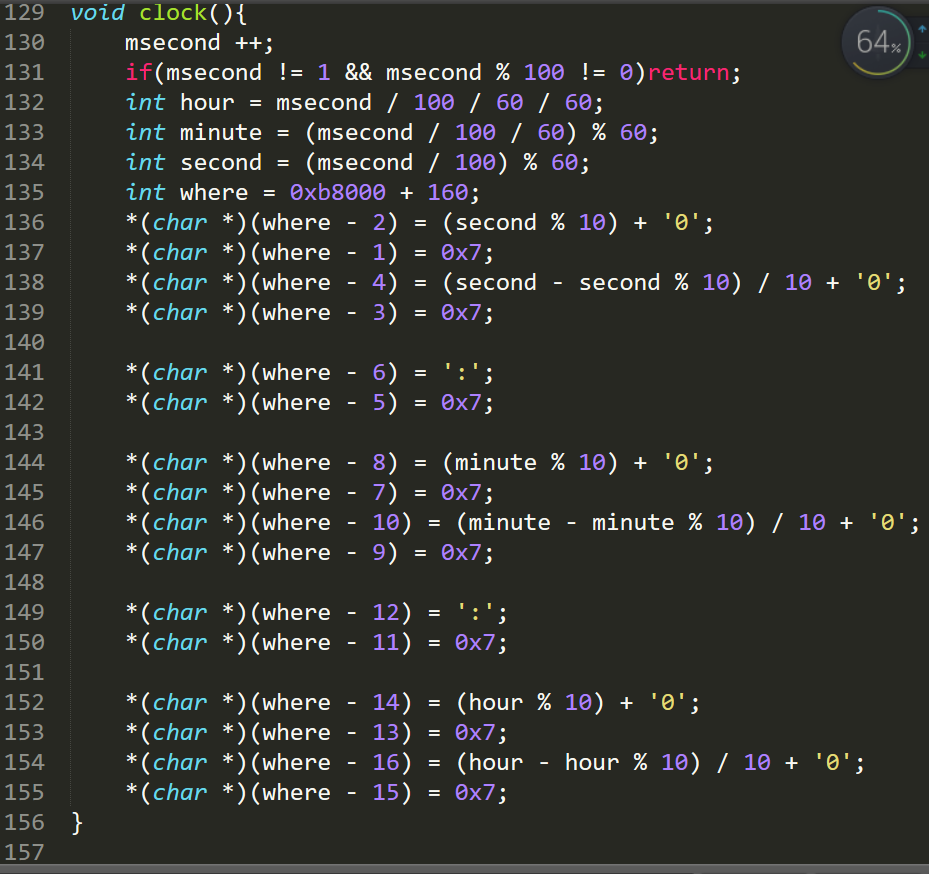
offset为中断调用的函数，所以得到：

期间因为外部变量声明，外部函数声明，变量类型转换，花了大量的时间，内核级的调试还是不够熟练，也暴露了对c的掌握的不足。

**中断机制的检查与时钟的实现：**

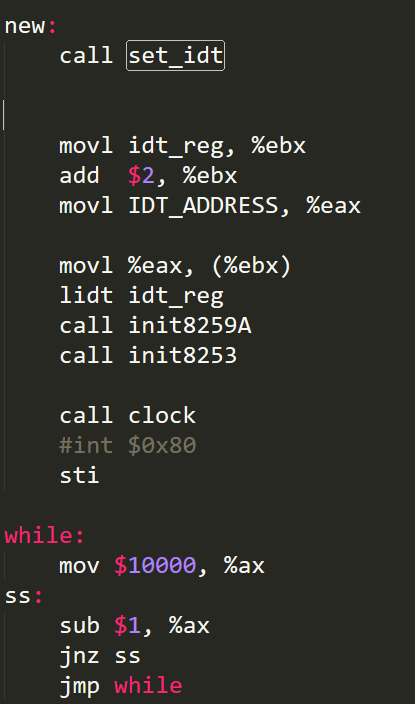
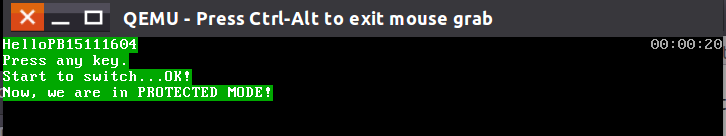
在上述调研与实现之后，先检查一下是否有bug。检查的方法是设计一个clock函数，并且在32.s里的中断处理程序中调用，通过不调用myMain而是死循环观察时钟。

幸亏发现及时， 发现了很多很多小bug并且一一排除，这一过程用了一天半的时间。

 实现的clock函数如下：

外部有一个msecond全局变量。

32．S中代码如图：

得到的结果：

结果合理。

可以开始设计schedule机制了。

**Round Robin的设计与实现：**

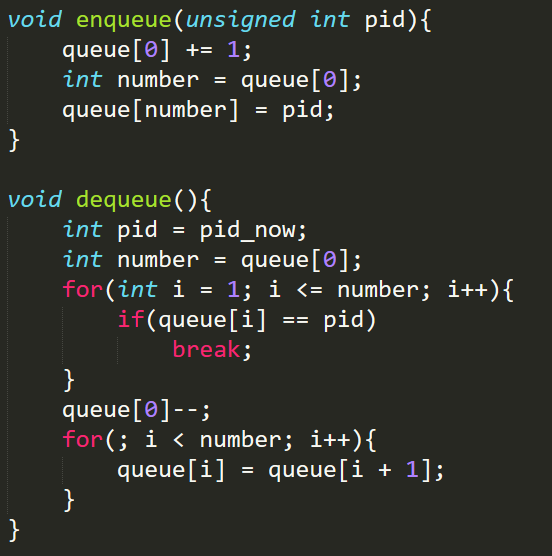
Round Robin的实现有以下几个难点：（出去上面的时钟中断的使用）

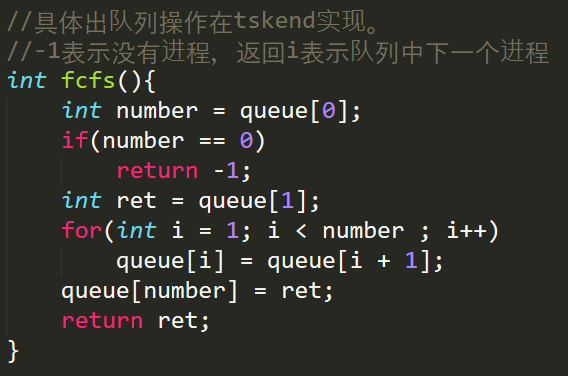
1. **Round Robin中进程的顺序跟pid不完全一致**。由于队列顺序会改变，不能只维护一个顺序数组。
2. **保存现场以返回**。不同于lab6，lab7，这次的进程调度室用round robin，之前都是fcfs，不用考虑返回之前运行过的函数的问题，而这次必须保存好。
3. **iret**。调用中断routine的时候，如果中途call schedule，那么有可能会导致栈溢出。因为中断次数很多。所以应避免直接call。

针对这些难点，解决方法与实现如下：

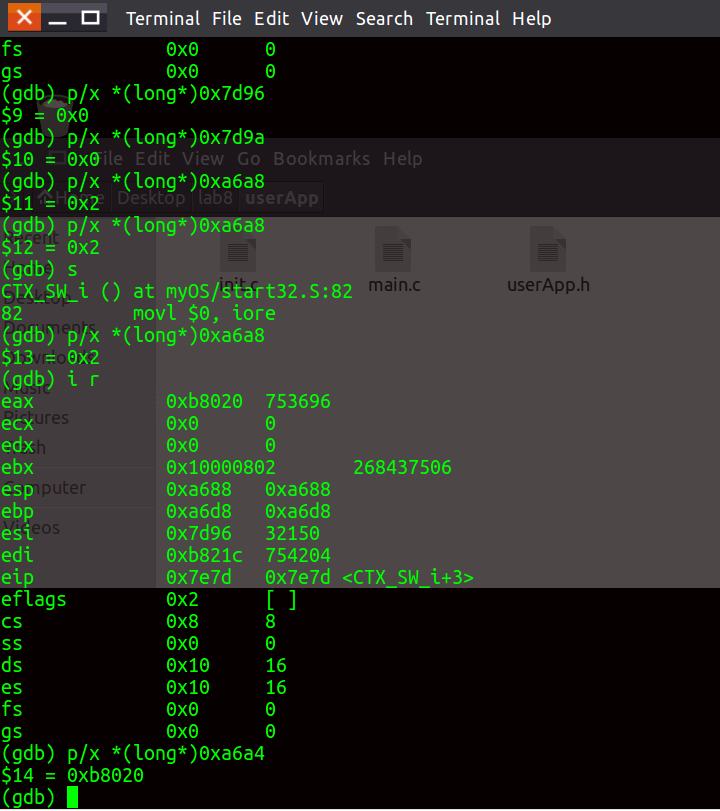
**维护队列**

 在之前进程数组jobs之外，设置fcfs队列：

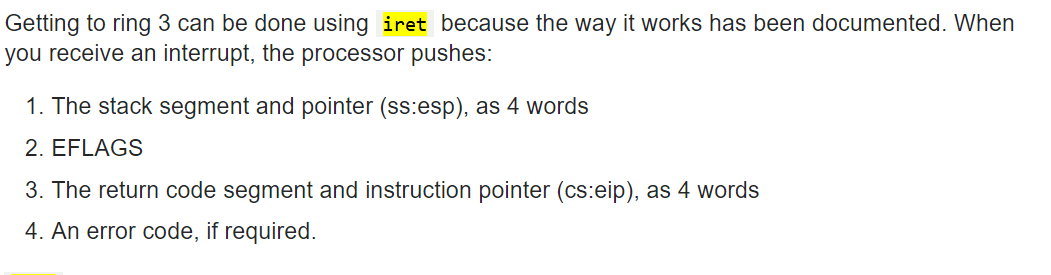
 出入队列：

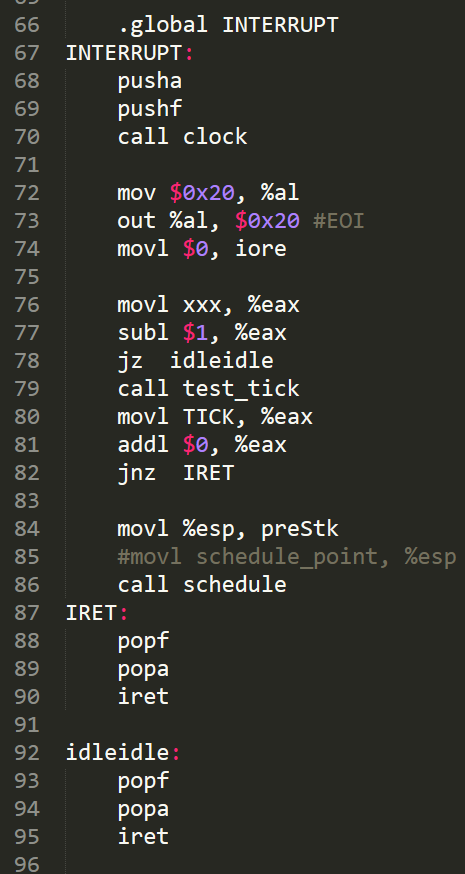
 以及得到下一个进程的fcfs函数：

**现场保存**

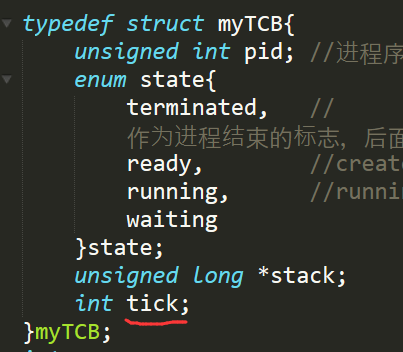
这一步需要pushf，在gdb调试过程中发现，只要push的顺序是pushf,pusha，那么就是先存eflags，再存其他八个regs，pop的时候顺序相反即可。

现在需要知道的就是iret都干了什么。

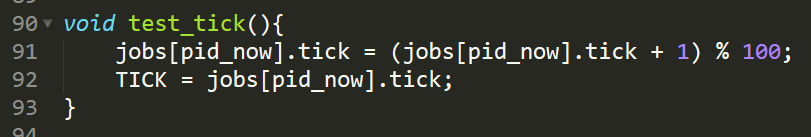
根据google的结果：

只是多了一个eflags。所以只需要：

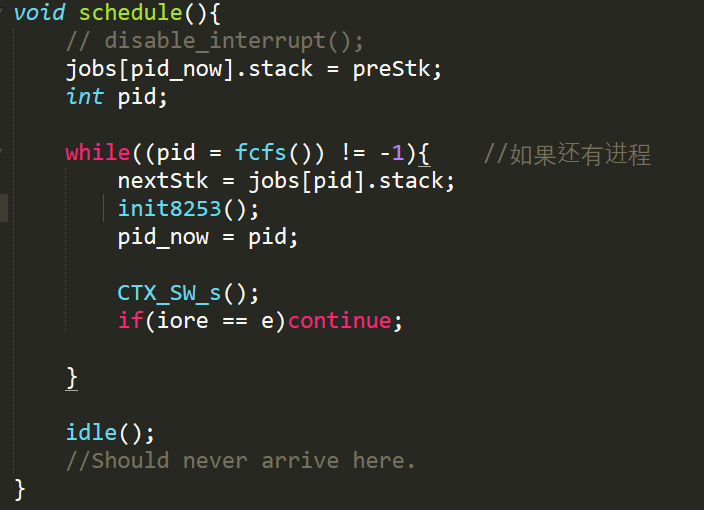
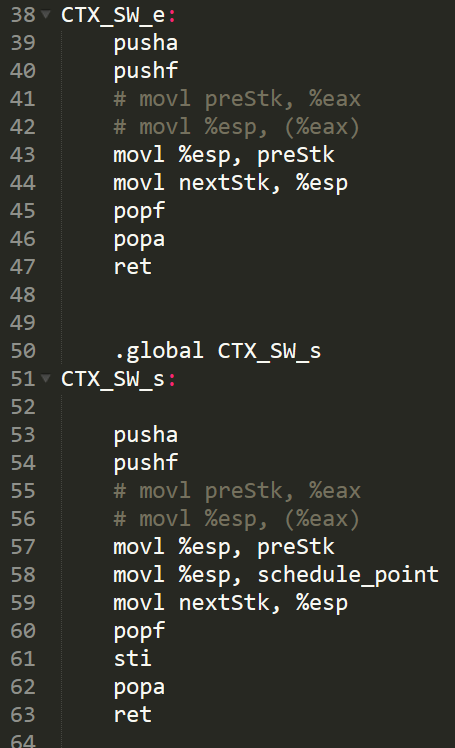
这里，xxx是为了判断是否调度完所有程序，在idle中的中断。如果是idle中的中断，那么当然没有必要tick，没有必要schedule，所以刷新clock之后就可以pop并iret了。

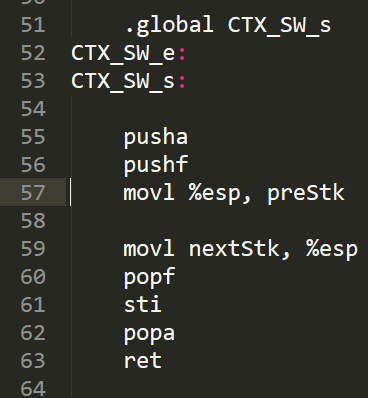
在非idle时，需要更新进程的tick，并且判断是否泡足了一个时间片，也就是1s。所以：

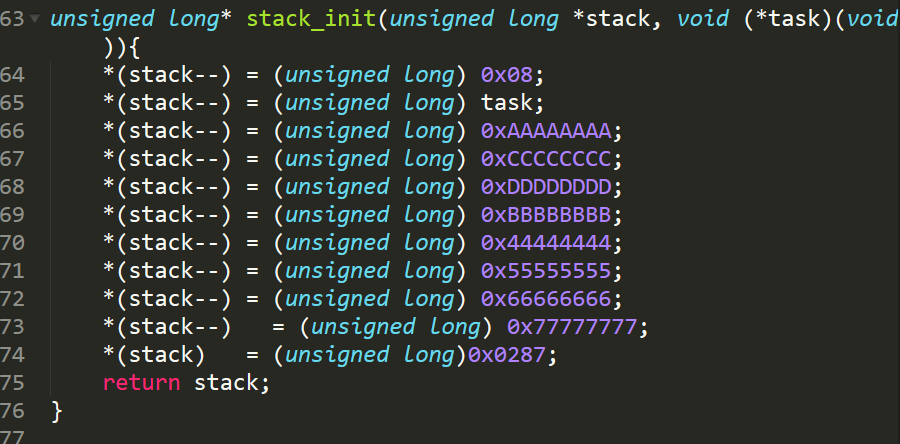
myTCB需要增加tick变量。

test\_tick函数如下：

调度函数schedule如下：注意每次调度之前需要打开中断，

另外，根据CTX\_SW的调用点的不同，设置了两个:CTX\_SW\_e，CTX\_SW\_s，前者是TskEnd函数调用的，后者是schedule调用的。有一些代码是中途调试所用。

后来优化之后发现没有必要，得到：

另外，eflags的初始值。之前因为eflags设置错误，查了一天多的bug，最后经小妞提醒改成了0287。

**原语流程:**

从32.s的call myMain之前的新内容：

Set\_idt建立idt表，将中断处理程序与entry关联 → 8259A设置芯片对应的中断 → 8253初始化计数器，10ms一次触发时钟中断。

call myMain之后：

清屏 → 输出helloworld → osstart → createTsk(initTskBody) → pMemInit → dPartitionInit → clock输出初始时钟→ schedule调度 → 找到下一个进程 → CTX\_sw → 保存现场 → 开中断 → 切入进程。

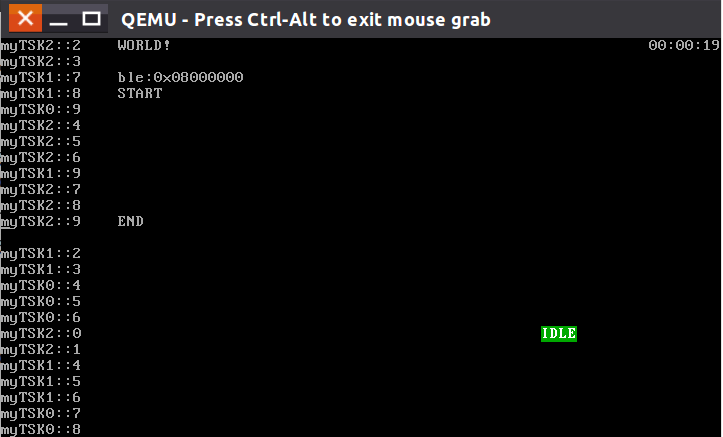
如果进程结束，→ tskend → schedule继续。

如果进程被中断→设置进程的tick，如果满足一个时间片，即1s →则转入调度。

如果不满足→则直接恢复中断。

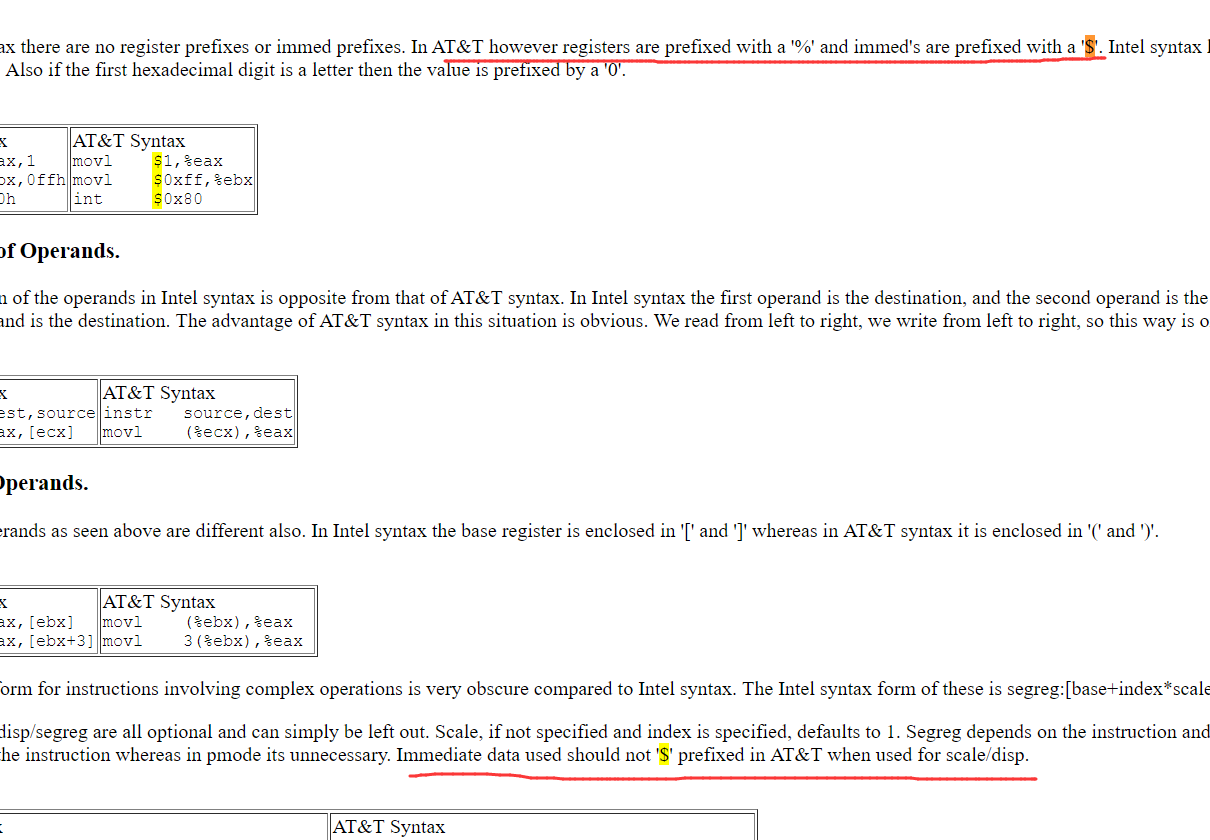
最后idle 开中断 → 死循环 → 触发中断 → 更新clock →恢复中断，继续死循环。

**测试结果：**

**实验总结（主要是bug与debug）**

实验总结的内容主要是bug与调试的回顾吧，因为90%的时间都在调试。Bug太多，说一些印象比较深刻的：

1. x86汇编立即数前忘了+‘$’符号，查得：
2. 不同源文件之间变量名的引用。

在c文件中使用s文件中的变量时，必须在s里声明globl，同时尽量使用extern声明，简单的数值变量可能只报warning，但是把s中的label当成数值变量时，必须先生命为extern （）函数之后经过类型转换再使用，否则会报找不到变量名。

1. 其他的大多是逻辑上的bug。包括对原理和机制的错误理解：

比如iret没有想象的复杂，当时做到从中断返回的时候是在晚上一整天以及前一天都在调一些智障的bug，心态很不好，所以就产生了不良心态。之后理清思路看之后好了很多。

还有整个中断机制的理解。确实是理解了机制之后bug就会消除大部分。

1. 还有debug技能的积累，学到了一些新的操作：

gdb里的比如x/10i 看内存指令，比如b \*0x00f0098设置内存断点，再比如wa \*(long \*)variable等等。Gdb的操作熟练了很多。

期间还熟练了一些grep，objdump等操作。

可以说整个调试的技能与心态的处理强大了很多。

这是最后一个实验了。

最后要感谢一下陈老师和三位助教的积极的引导、指导，还有无尽的付出。

结课辣！！！！

么么哒！！！

C:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gif !祝老师和助教假期愉快！

C:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gifC:\Users\zevin\AppData\Local\Temp\(DV468J7KSO56V{TG{DXHO1.gif