Home Assignment (Final)

zhengyudian09@software.nju.edu.cn

1 Introduction

OK，经过了上一次的Assignment 4之后，如果你是自己认真一步步调出来的代码，我相信你已经无论在对底层编程的能力和解决问题的能力上已经有了很高的提升，这次的作业是更Hard的一个版本，我们将仅仅使用nasm在实模式下实现简单的进程调度程序。

先说下我们上一届使用的是bcc(在linux下安装as86和ld86进行编写程序)，上一次是C+汇编共同的编译，但是比较坑爹的是你C和汇编一起写需要顾及编译器到底如何把你的C转成汇编的，因为编译器可能会自己加入一些push bp等的代码，带来很大的不便性，而且需要用gdb+qemu共同调试，qemu作为像我们bochs一样的虚拟机，gdb是连接qemu进行调试的工具。但不得不承认，C和汇编写在一起代码并不太elegant。

在于渊的书中写的非常elegant，是用gcc+nasm都编译成-elf文件，然后用ld进行链接，链接出来是可执行的-elf文件，其汇编和C是分开进行写的，就是没有汇编和C掺杂在一起的情况，但是我们为什么不用gcc呢，因为gcc文档中已经说明了不太支持实模式下的编译，如果我们进行保护模式当然用gcc+nasm绝对是个不二的选择，调试的工具于渊使用的bochs+gdb进行调试，而我们不是在保护模式下进行编程，我们操作的都是实模式下，如果使用到gcc很多情况下会出现问题，而且更会给大家带来调试的不便。

因此，为了给大家省力，我们采取最好的直接使用nasm 进行编写的方法，调试的时候大家在Windows上使用bochs直接进行调试即可。

*本次作业应该是最Hard的一次，如果你认真做完了本次作业的每个步骤，那么一定会对你的OS从初出茅庐到小试牛刀的提升，当然如果要提升到炉火纯青的地步需要大家进一步地看OS源代码和相关的经典书籍。

Special:对于本次作业希望大家认真完成，由于考虑到6月初的时候给大家来一个这个作业会让大家在各项大作业面前吃不消，

所以这次作业提前布置，在6月9日(周六)晚上6:30的时候我会去机房，待到9:30，如果大家以前作业有什么得分低的可以重新写一份交给我，最后一次收打印稿，看大家的态度，大家需要把自己想要grace 那一周的文档和代码都用A4纸打印出来(第一页写清楚你的学号姓名和要grace哪一次的作业)。第四次作业会在6月3日上午全都贴出来具体的分数，最后一次作业就不给grace了。这次的作业在后面也会进行说明，大家做到什么程度就是什么程度，需要提交一个少于等于3页的文档，写出大家对这个实验的体会，出现了哪些问题如何去解决的，有哪些问题没有解决。我们不会仅仅根据实验结果是否出现进行评分，在程序没有完成的情况下文档会给你加不少分，也就是说不想大家相互抄袭，还是希望大家自己去体验一下，把遇到的小BUG，小问题以及感想都写入文档。我们会score=f(代码,文档)来给大家最终的评分，在代码跑不过的情况下文档会占较大的分数，也就是说如果用心了，把遇到的问题写清楚，即使最终结果没出来也有可能得A的。

2 Target

本次作业实现的功能就是实现四个进程分别输出ABCD的轮换。本次作业有两个目的：

2.1 真正领略nasm的编译

这次作业的一个步骤是让大家把四个进程写入process.asm当中，如何正确的写入，哪些地方需要注意，哪些地方容易出问题需要大家对nasm的编译比较了解才可以。在上次的课中我们也提到了比较关键的org，mov label以及jmp/call这种转移指令具体在编译过后的machine code的区别。而且我们还会给大家提供比较方便的macro宏命令，当然如果利用宏命令去成功实现几个进程完全写在一个.asm文件中还是需要很多注意的地方。

2.2 了解掌握进程调度模型

进程调度说起来很简单，我上课已经跟大家讲明白，但是实现起来有非常非常多需要注意的事项。如何处理好进出栈的顺序，如何利用好四个栈之间的联系，实际我们实验中只使用了三个栈：假设进程A和进程B需要相互调度，那么进程A中的栈是一个栈，进程B中的栈也是一个栈，kernel中的PCB块需要充当我们不断push寄存器的“伪栈”，当然还有kernel中的栈，我们实验中

并未涉及kernel中的栈，因为处理的代码并不多。而在保护模式中，内核栈需要经常的使用，因为保护模式中是根据call指令以及TSS中存入的123级的ss以及esp来进行优先级的提升导致栈的转换，在我们实模式中我们并未过多涉及第四种栈。

Good Luck!!!

3 Detail

这里详细讲一下我们给大家的每个.asm文件，前两个文件不需要大家去填，后两个文件需要大家去填

3.1 boot.asm

首先boot.asm是加载到空的a.img(1.44M)的前512个字节的，可以通过dd命令:

*dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc

或者上节课大家实现的四种linux,windows系统调用和库函数来实现都可以。这里我们使用于渊的源代码，具体功能是512个字节最后两个字节是象征引导的aa55，其中内容加载到内存0x7c00处是在软盘中寻找loader.bin的文件并把这个文件加载到内存的

90100h(9000h:0100h)处，然后跳转到loader.bin:

jmp 0900h:0100h

用nasm去编译的时候大家注意需要使用到fat12hdr.inc，所以大家需要用:

*nasm boot.asm -I fat12hdr.inc所在的文件夹+'\' -o boot.bin

3.2 loader.asm

这里是我在于渊源码基础上已经帮大家实现好的程序，注意此个文件位于上面的四个label:

```
1 BaseOfKernel equ 08000h ;KERNEL.BIN段地址
2 OffsetOfKernel equ 0100h ;KERNEL.BIN偏移地址
3 BaseOfProcess equ 07000h ;PROCESS.BIN段地址
4 OffsetOfProcess equ 0100h ;PROCESS.BIN偏移地址
```

这几个label给出了含义，这个loader.bin在引导代码加载到内存之后所做的操作就是把process.bin加载到70100h(7000h:0100h)处，把kernel.bin加载到80100h(8000h:0100h)处。

3.3 *process.asm

这里是我们四个进程需要实现的地方，我们知道loader.bin要加载这个文件到0700h:0100h处，所以我们的org 需要设置成0100h我们四个进程是这么分布的：四个进程分别的功能是各自输出ABCD(进程一不断输出A，进程二不断输出B，进程三不断输出C，进程四不断输出D)，因为我们需要看到连续的效果，所以我们需要把显存要写入的位置存在某个特定的内存中，我们规定这个内存为090000h,在kernel.asm开始我们会设置这个位置的(word)为0，即

mov word [9000h:0000h],0 ;这是伪码

那么我们在这四个进程中需要使用这个位置的值进行写入显存，注意写完特定位置的显存后需要把这个位置+2，方便下一个要写的时候进行读取。这样如果我们实现了进程的轮转，我们就可以看到连续的输出ABCD了。

下面我们需要知道这四个进程的cs:ip的位置(即起始位置)以及栈的位置是如何分配的。我们规定每个进程初始的cs:ip和ss:sp是一样的，当然这样子不冲突，因为栈是往地址小的地方增长，而进行的代码是随着地址高的地方走。

进程一初始的cs:ip和ss:sp = 7000h:0100h
进程二初始的cs:ip和ss:sp = 7020h:0100h
进程三初始的cs:ip和ss:sp = 7040h:0100h
进程四初始的cs:ip和ss:sp = 7060h:0100h

(*Tips:使用times命令进行中间间隔值的赋值，不要出现mov label这样子的汇编代码)

编译的时候直接通过nasm process.asm -o process.bin 进行编译即可。

*大家需要注意的是如何通过macro会使我们的代码更简单,具体的宏定义:

```
%macro HONG 3
%1:
    mov dx,0
%2:
    mov al,%3
    loop %2
%endmacro
```

这样子我们通过HONG a1,b1,'A' 的调用实际产生的是:

```
a1:
```

```
    mov dx,0
b1:
    mov al,'A'
    loop b1
```

因为如果我们在循环中直接放入打印ABCD的话闪的会非常快，因为时钟中断在BIOS初始会设置大概每隔55ms发送一次，也就是一秒调用大概18.2次，而bochs对时钟模拟的还不是很精确，可能1s调用远远超过18.2次，所以我们的的程序如果不加限制则会在55ms中疯狂打印某个字母（如果是进程一则疯狂打印A），所以我们对程序进行一下设置，在打印A的字符的进程代码前面加上空循环，让循环每次空跑2*ffffh次，防止字符的过快输出。

在kernel.asm(下面)中实现的时钟中断处理函数代码中我们也需要设置跟这里类似的一个参数TimeCount为300，当然目的不一样，在kernel.asm中我们在中断处理函数中的目的是在时钟中断中既让进程进行切换，又保证进程不频繁地切换，即每进入时钟中断300次才进行进程的切换。

当然实现的方法也不一样:

在时钟中断处理函数中，我们专门开辟一个内存来存放现在的数字记录是多少（初始值300，每次进入减一，如果不到0则直接返回；如果到了0重新变成300，而且进入进程切换）。

在打印字符进程当中我们直接用某个寄存器来记录次数，注意因为是2*ffffh次，所以大家需要用两个寄存器来记录，在这里你也可以利用跟时钟中断处理函数一样开辟内存的方法，不过这样子比较麻烦，如果大家能处理的好宏命令则也可以使用此个方法。其实打印字符进程中不采用与上面相同的做法是因为这里是把四个进程通过宏命令捏合在一个.bin文件当中，如果跟上面相同的方式处理会出现问题。

3.4 *kernel.asm

进入到最关键的一个.asm了，核心的.asm。

首先上课的时候我已经跟大家阐述了进程调度模型。而且因为loader.bin会把我们的kernel.bin加载到80100h(8000h:0100h)处，所以这里我们的org需要设置成0100h，为了带来不必要的干扰，首先我们明确一些东西：在底下我们注释标明的数据区里，这些是你们需要用到的数据，这段的代码不要更改，不要自己声明equ或者dw（包括在需要你填的代码中），充分利用好提供给你们的这个数据区，很多常量认为你们是已知的，包括PCB块的

长度（34byte，在底下会贴出详细的PCB块，但是你们不能把这个PCB块复制四个完全贴在数据区中，数据区的所有代码是不能动的），计时器的数字300，以及BaseOfKernel 08000h，我们提供的数据区中的变量有：

```
AllProcNum equ 4
TimeCount dw 300
Pro_Ready dw PCB
BaseOfPro dw 07000h,07020h,07040h,07060h
PCB dw 0
```

其中TimeCount是规定我们每300次进入时钟中断处理函数的时候进行进程切换，在process.asm中已经给大家了注释，Pro_Ready标明的是下一个进程的进程块的地址，默认我们是使用第一个进程块，进程块在PCB处，实际我们需要填满34*4个byte，但是我们只给出了两个byte（PCB dw 0）这个不影响，我们只是为了标识，在kernel.asm开始的时候我们需要用代码初始这四个进程块，其实实际是AllProcNum 个进程块，如果你程序编好的话随意更改进程块的数量，另外在process.asm中继续加入进程块就能看到ABCDE或者ABCDEF等等的交替打出。如果进行进程切换的时候，假设要切换到下一个进程，那么我们的Pro_Ready这里就需要做：add word [Pro_Ready],34 这样的指令。BaseOfPro当中主要是方便大家在PCB块中初始化的时候需要使用到的每个进程的基址，每个进程的偏移地址以及sp的值都为0100h。下面贴出来PCB块中具体有哪些内容（这34个byte）：

```
    GS
    FS
    ES
    DS
    DI
    SI
    BP
    SP
    BX
    DX
    CX
    AX
    IP
    CS
    Flags
```

```
    SS
    ID
```

我们在kernel.asm中开始的代码初始PCB块后第三个进程的结果应该是这样的。

在kernel.asm中我们需要的步骤一共有这么几步（如果需要填写的会在代码处进行标明）：

3.4.1 preprocessing

a. 先初始9000h:0000h处的word为0，为了方便各个进程从这里取出值作为打印字符的位置。

b. sti（开可屏蔽中断），进行PCB块的初始化操作，利用loop循环设置cx为AllProcNum，初始化相对应的PCB块，例如进程三我们初始化后应该为：

```
B800h     ;GS
7040h     ;FS
7040h     ;ES
7040h     ;DS
0000h     ;DI
0000h     ;SI
0000h     ;BP
0100h     ;SP
0000h     ;BX
0000h     ;DX
0000h     ;CX
0000h     ;AX
0100h     ;IP
7040h     ;CS
****h     ;Flags
Flags值根据各位前面的代码利用pushf可能会有差别
,不过确保sti指令即可
7040h     ;SS
0003h     ;ID
```

c. 设置cli（关可屏蔽中断），把Clock_handler设置成时钟中断向量(08h)对应的位置。注意：时钟中断向量是08h，BIOS初始的时候08h中是有代码的，clock频率55ms，即1s调用大概18.2次时钟中断，在08H处理函数里面会调用1ch的中断处理函数，初始的

时候1ch只有iret指令，我们在上一个assignment当中已经进行了说明。但是改写中断处理函数如果需要进程调度的操作大家一般是直接对08h进行改写，即(020h放ip以及022h放cs)。

*上一次我们Advanced1代码中是改的1ch中断，而实模式下编写进程调度通用的做法是直接改写8h中断，因为时钟中断其实是8h，8h处理代码中调用了1ch，1ch只是一条iret指令。我们把时钟中断中加入了进程调度是对时钟中断较大的改变，所以直接改写8h就好，但是注意需要手动发送EOI(后面会给出具体代码)。

d. 为Process1进行准备，我们把寄存器都设置成Process1对应的寄存器（除了cs和ip），为了直接跳入Process1。

e. sti开中断，跳入Process1，jmp 7000h:0100h 。

3.4.2 Clock_handler

下面进入我们比较关键的Clock_handler函数了，在这里面我们需要：

a. 保存寄存器(见下面*) 后来我们看一下TimeCount里面的值减一，如果不是0的话说明我们现在不进行Pro_schedule，那么我们跳入a2进行直接的返回操作，如果进行Pro_schedule那么我们跳入a1中，在a1中首先把TimeCount置成300，然后进行Pro_schedule的操作。

*第一步比较free，就是大家可以根据a1以及a2后面区分的不同路以及再往后的代码决定这里要保存什么寄存器，我们安排PCB表(那34个byte)的顺序是大多数编写OS安排寄存器的顺序，因为比较方便的是pusha指令，即把部分寄存器全部压入栈，顺序是ax,cx,dx,bx,sp,bp,si,di,其中sp是原来sp的位置即没压入ax之前的sp的位置，也就是我们需要保存的位置。可以在这里调用pusha或者在以后调用，我们根据自己的需要把自己可能改变的寄存器压入进栈，这些寄存器大家可以自行去设计。但是无论如何要注意这里的栈是刚才执行进程的栈，并不是kernel的栈

a2. 这里我们不进行Pro_schedule，那么我们直接进行跳出即可，注意此时要结合a步可能压入的寄存器，此处最后几条指令我们有比较严格的要求，与下面d所要求的一样，需要为：

```
push ax
mov al,20h
out 20h,al
out 0A0h,al
pop ax
iret
```

*前面几条指令是发送EOI到8259A，代表我们这次时钟中断已经结束了，因为我们是重写8h中断，8H中断中也有相应的指令告诉8259A我已经处理完一些事情了。8059A是可编程IO，初始的设置在ICW4中会设置正常（手动）的EOI而非自动EOI，所以需要我们自己发送EOI信号告诉我们时钟中断已经处理结束，最后的iret指令是中断特殊象征性的结束符(pop ip,pop cs,popf)

a1. 在这里我们首先把TimeCount置成300，然后跳入Pro_schedule中。

b. 这里是Pro_schedule，在这里我们首先把我们进入中断前进程x的寄存器都放入对应的PCB块中，由Pro_Ready给出对应的PCB块的地址，注意此时要结合a步可能压入的寄存器，a步push的寄存器都在刚才执行进程的栈空间中，而我们复制的目的地址是在kernel的PCB块中，注意做好寄存器的保护措施。

c. 在这里我们判断要进行schedule的进程PCB，我们通过：

```
    add word [Pro_Ready],34
    cmp word [Pro_Ready],PCB+34*AllProcNum
    jnz step_d
    mov word [Pro_Ready],PCB
step_d:
```

即通过+34的位置与最后的位置进行对比，如果相等的话说明此时的PCB是我们最后的PCB了，那么我们需要把第一个PCB块的地址，即PCB(在data里面给出了)给到[Pro_Ready]当中。

d. 在这里我们需要启动下一进程，即把下一进程在PCB块中的寄存器全部拿出，即从kernel中的PCB 块中拿出，作为下一个进程的状态。注意此处最后的几句代码就像我们在a2中提到的那样，必须要符合如下：

```
push ax
mov al,20h
out 20h,al
out 0A0h,al
pop ax
iret
```

到了这里，我们就大功告成了！！！我们还可以后续进行调整AllProNum的值以及自己在process.asm中加入新的进程来进行测试，目前的测试结果应该如下图：

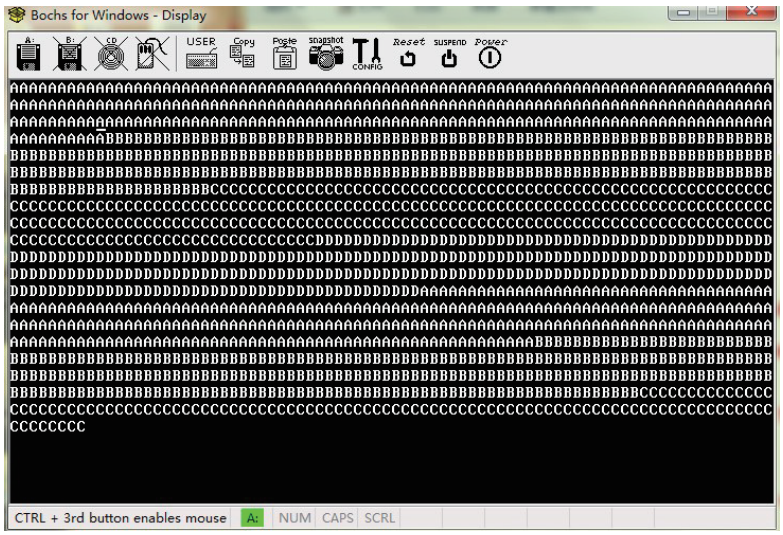

Figure 1: 测试结果

Good Luck!!!

4 Format

需要大家提交以下三项内容打包到.rar中
1.文档.pdf（不多于3页）
2.process.asm
3.kernel.asm

5 Deadline

(TSS-Time)2012年6月20日23:59:59

6 FeedBack

2012年7月17日12:00:00
*届时会在TSS Assignment&FeedBack 版块贴出大家的（序号，得分，评语，批卷人）

7 End of the OS-experimental class

到了这里，本学期的OS实验课也算基本结束了，我相信大家会根据自己分配在OS实验课的时间或多或少都接触了很多OS实验的知识，这些知识对大家以后的学习应该会有一些潜移默化的帮助。在这个实验课上一路走来我从你们身上也学到了很多东西，也非常感谢葛老师能给我一个这么好的锻炼自己的机会，同时也衷心希望你们以后如果有可能当助教的时候能够多为学院出点力，良好的环境需要大家共同去塑造。

就我自身感受而言，我觉得无论什么东西只有用心，不畏艰难才能取得让自己略微满意的结果，有时我会为一个很小的bug调上8个小时，但是也许就是这8个小时让我真正了解了debug的技巧，让我真正学会一门语言。过程越困难的事情到最后你会越佩服自己战胜困难的勇气和决心。最后，预祝大家以后在人生道路上都能走出自己的精彩！