实验六 ： 二状态多进程

目录

一/二.实验目的与要求

1，在内核实现多进程的二状态模型，理解简单进程的构造方法和时间片轮转调度过程。
2，实现解释多进程的控制台命令，建立相应进程并能启动执行。
3，至少一个进程可用于测试前一版本的系统调用，搭建完整的操作系统框架，为后续实验项目打下扎实基础。

三.实验方案

1.实验工具与环境（gcc+nasm+ld）

nasm 编译汇编代码生成.o 文件，gcc 编译 C 模块生成.o 文件；ld 链接.o 文件生成.bin 文件；dd 将.bin 写入映像文件.img；vmware 上添加.img 文件到软盘后运行。

2.方案的思想

1.相关原理：

①关于进程状态：

进程状态可分为运行态，就绪态，阻塞态等状态。本实验是二状态多进程操作系统，只分为运行态和就绪态。一个进程在任意时刻只能处于一种状态，每次只有一个程序处于运行状态。进程状态的切换会在时钟中断中执行。

②关于进程控制块

进程控制块是操作系统控制进程所必须的，用于存储进程的所有状态。它也是进程恢复执行的必要条件。**它包括了一个独有的进程标识，所有的寄存器的值，一个独立的进程栈，另外还有一个进程状态。**进程标识用于独特的标识本进程，寄存器的值用于恢复进程的执行。而进程栈则是一个空间。进程状态用于代表本进程的状态，即运行态和就绪态。这些都是实现多进程所必须的。

③进程表

进程表存放了当前正在执行的进程的标识，以及所有就绪态的进程的标识，

每次在进程切换时，可以从进程表中获知下一个要执行的进程是哪个。然后恢复该进程的执行。

2.实验内容

（由于这次的实验中子程序不可占用键盘输入，我将上次实验的 4 个中断换为了子程序，然后将 5 个系统调用中的两个改成了中断程序）

1.建立进程表保存可执行进程的标识，并在进程切换时返回新的进程标识。

2.创建进程控制块以保存必要的进程内容

要注意pcb中的初始值，cs和ip的初始值应该是该进程在内存中的地址，而ss和sp的初始值应该是该进程的栈的地址。

3.编写save函数保存进程状态：

每次进入时钟中断，进马上保存所有寄存器的值，包括cs，ip以及栈地址。

4.编写restart函数进行恢复进程：

每次即将离开时钟中断，恢复所有寄存器的值，然后将标志寄存器，cs，ip压入栈中，那么执行iret指令后就会回到相应的地址。

3.实现方法：

①save函数实现方法：

先将调用save函数的地址pop到一个暂时的变量处。先判断当前进程是哪个进程，将寄存器的值传送到pcb中，将cs，ip，flags寄存器pop到pcb中，将ss，sp寄存器mov到pcb中。

②切换进程的方法：

递增进程标识，若该进程处于就绪则将运行标识改为该进程的标识，并返回。

③restart函数实现方法：

同save函数类似，只是顺序相反。

3.程序流程：

主程序过程：
1.引导程序加载内核。
2.内核初始化进程状态表，除内核进程外都为阻塞态。
3.用户通过键盘输入将子程序放入就绪队列，或者将子程序从就绪队列中移出。
4.在就绪对列中的进程通过时间片，进行切换进程运行。

切换进程过程：
1.进入时钟中断程序。
2.运行 save 函数保存寄存器内容
3.获取要运行的新进程
4.Restart 函数恢复进程状态。

4.程序关键模块：

1.pcb 表模块：

```
PCB:
    si0_save    dw      0
    si1_save    dw      0
    si2_save    dw      0
    si3_save    dw      0
    si4_save    dw      0
    di_save     dw      0, 0, 0, 0, 0
    bp_save     dw      0, 0, 0, 0, 0
    sp_save     dw      0, 0x1a00, 0x2a00, 0x3a00, 0x4a00
    bx_save     dw      0, 0, 0, 0, 0
    ax_save     dw      0, 0, 0, 0, 0
    cx_save     dw      0, 0, 0, 0, 0
    dx_save     dw      0, 0, 0, 0, 0
    ss_save     dw      0, 0, 0, 0, 0
    gs_save     dw      0, 0, 0, 0, 0
    fs_save     dw      0, 0, 0, 0, 0
    es_save     dw      0, 0, 0, 0, 0
    ds_save     dw      0, 0, 0, 0, 0
    ip_save     dw      0x8100, 0x1000, 0x2000, 0x3000, 0x4000
    cs_save     dw      0, 0, 0, 0, 0
    flags_save  dw      0, 0, 0, 0, 0
    status      dw      1, 0, 0, 0, 0
```

```
int running_proc = 0;
int status[5] = {1, 0, 0, 0, 0};// 进程表   0代表内核进程
```

running_proc 表示当前运行的程序标识，status 中的 1 代表是就绪态的进程，0 代表是阻塞的。

2.切换进程模块：

```
int getnextproc()
{
    int i;
    if (running_proc < 5)
    {
        i = running_proc+1;
        while(i%5 != running_proc)
        {
            if(status[i%5] == 1)
            {
                running_proc = i%5;
                break;
            }
            i++;
        }
    }
    return running_proc;
}
```

切换进程时调用这个函数，将返回切换的进程的标识。函数搜索下一个处于就绪态的进程，将运行进程标志改为该进程的标识。

3.save 函数模块：

```
;保存当前运行态进程的上下文
save:
    pop word[cs:ret_save]          ;先保存函数返回地址
    cmp byte[cs:a_running_proc], 0 ;判断现在运行中的进程是哪个
    jz proc_kernel
    cmp byte[cs:a_running_proc], 1
    jz proc1
    cmp byte[cs:a_running_proc], 2
    jz proc2
    cmp byte[cs:a_running_proc], 3
    jz proc3
    cmp byte[cs:a_running_proc], 4
    jz proc4
    push word[cs:ret_save]         ;若不是并行进程则不进行save,test
    ret
;保存si的值，以si作为偏移量选择进程
proc_kernel:
    mov word[cs:si0_save], si
    mov si, 0
    jmp saveother
proc1:
    mov word[cs:si1_save], si
    mov si, 2
    jmp saveother
proc2:
    mov word[cs:si2_save], si
    mov si, 4
    jmp saveother
```

```
    mov ax, ds
    mov word[cs:ds_save+si],ax
    pop word[cs:ip_save+si]
    pop word[cs:cs_save+si]
    pop word[cs:flags_save+si]
    mov word[cs:sp_save+si], sp
    mov ax, ss
    mov word[cs:ss_save+si],ax
;切换成内核栈
    mov ax, cs
    mov ss, ax
    mov sp, temp_stack
    push word[cs:ret_save]
    ret
```

要注意保存寄存器的顺序，避免某些寄存器的值被篡改了

4.restart 程序模块：

```
;切换进程
restart:
    pop word[cs:ret_save]
    mov bx, 2
    mov ax, word[cs:a_running_proc]
    mul bl
    mov si, ax
    ;切换栈用户
    mov ax, word[cs:ss_save+si]
    mov ss, ax
    mov sp, word[cs:sp_save+si]
    mov ax, word[cs:gs_save+si]
```

四.实验过程与结果

1.实验过程：

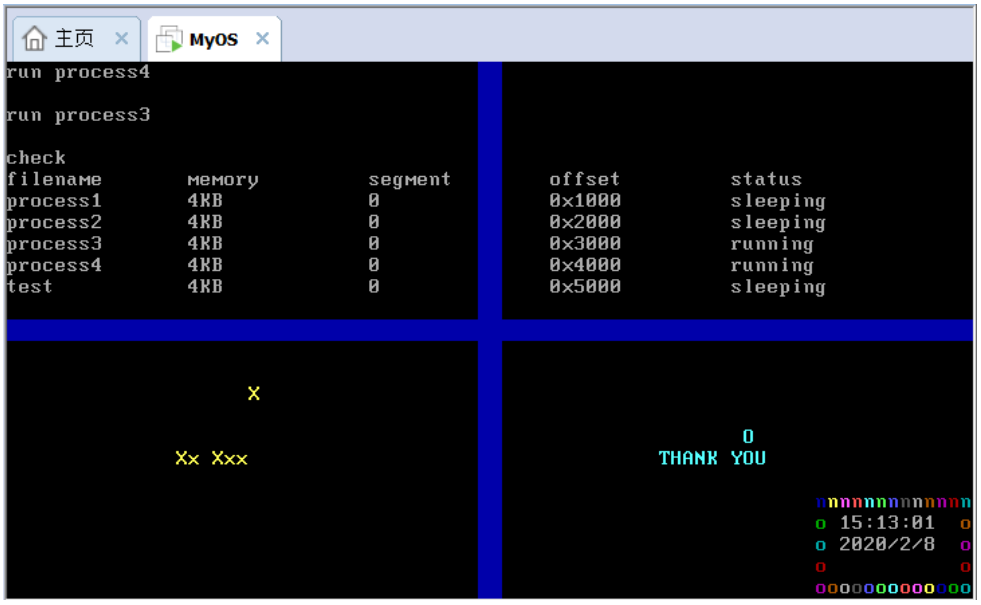
【编译链接的命令】
与上次实验一致。

【磁盘和内存安排方案】
内核程序存放于磁盘第 2 到第 36 个扇区，共 35 个扇区。由引导程序加载首地址 08100H 处。栈空间起始地址是 0。
四个用户程序连续存放在第 37 到第 52 个扇区，每个用户程序占 4 个扇区，大小 2KB。分别加载到 0x1000, 0x2000, 0x3000, 0x4000。栈空间起始地址是 0x1a00, 0x2a00, 0x3a00, 0x4a00。
测试程序存放于第 53 到第 54 个扇区，大小 1KB。加载到内存地址 0x5000。

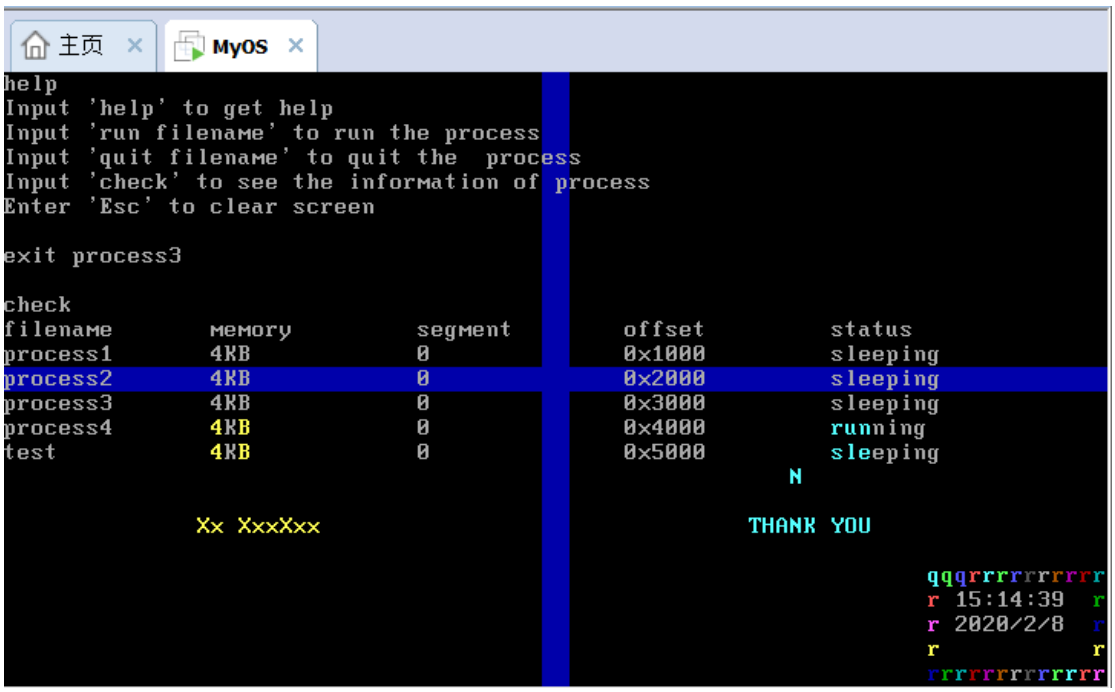
2.运行结果：

运行子程序


run process4
run process3
check
filename memory segment offset status
process1 4KB 0 0x1000 sleeping
process2 4KB 0 0x2000 sleeping
process3 4KB 0 0x3000 running
process4 4KB 0 0x4000 running
test 4KB 0 0x5000 sleeping
i
Wu Ziyang
Y
THANk YOU
21:08:26
2018/4/27

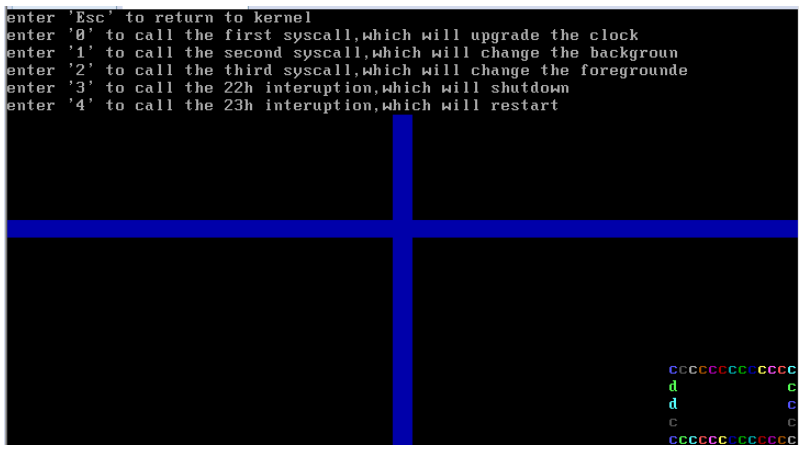

主页
MyOS
run process4
run process3
check
filename memory segment offset status
process1 4KB 0 0x1000 sleeping
process2 4KB 0 0x2000 sleeping
process3 4KB 0 0x3000 running
process4 4KB 0 0x4000 running
test 4KB 0 0x5000 sleeping
X
Xx Xxx
O
THANK YOU
15:13:01
2020/2/8

退出子程序:

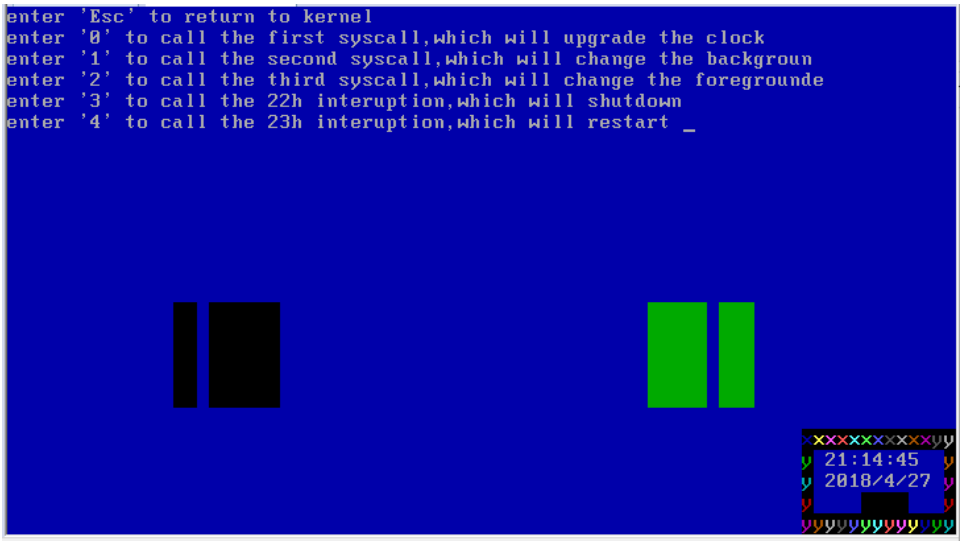


左下角的程序停止了，而右下角的还在运行

测试系统调用与中断：



切换颜色



左下和右下的颜色差是由于之前调用的子程序导致其背景颜色不同。

3.遇到的问题及解决情况：

①可以进入第二个程序，但是在第二个子程序中死循环，发现时钟中断的字符滚动没有发生，猜测是第二个子程序没有进入时钟中断，考虑到无法进入中断的原因。又因为pcb表中标志寄存器的初始值为0，在**子程序中加入了sti开中断**，果然程序可以正常运行了。一开始需要先在pcb中为子程序设定初始值，这时要注意段地址和偏移量，另外还有标志寄存器。

② 当程序在等待输入时，切换进程就会死机，在不断的注释和尝试后我发现在这两个指令里，会死机

Mov ah，0

Int 16h

我猜测是在16h中断中的跳转出了问题，但是我没有找到解决方法，我最后是用检测输入缓冲区来避免让程序停在16h内部。即，

```
mov ah, 1
int 16h
jz _Getchar
mov ah, 0
int 16h
```

来避免问题的

五.实验创新点

1. 可以指定运行与退出子程序。

2. 可以使用键盘输入

六.实验总结：

本次实验，主要设计了多进程的并发机制，即分时系统。

我感觉本次实验的难度集中在环境保存与恢复。本次实验要为每个进程分配一个相应的栈空间。save 和 restart 的问题不大，我是在时钟中断里掉坑的。因为在内核的时候是不需要切换进程的，所以对于内核进程，不需要 save 和 restart，而我为了区分进程，使用了较多的跳转，另外之前的时间中断里有较多 push 和 pop。而我不小心调换了 restart 和 pop 的位置，就使得栈出错了。正确的应该是:进入中断马上 save,然后可以 pushad 保存寄存器,返回前 popad，再 restart。而我先 restart 后再 pop 了。

另外本次实验花费了很多时间在小 bug 上，例如段超越前缀不正确，没有初始化某些变量。

本次实验中，地址的跳转也是比较有难度的事情，进程的切换就涉及到了栈的地址切换以及指令地址的跳转。而 pcb 也涉及到了偏移量的问题。甚至我怀疑地址的改变，特别是段地址，影响到了变量的值。

实验中有些问题是我没能理解的，在实验的完成中我没能解决，最后选择了其他方法避免了。在上面我就提到了在 16h 中断中进程切换会死机，后来我发现主要原因是我的进程表被破坏了。在进程切换程序中，我对进程表做了输出，然后我发现进程的状态值会变成一些奇怪的值,即使后来我使用其他方法解决了进程切换的问题，但是我发现进程的状态值还是会不断变成一些奇怪的值，然后又变回来。而且奇怪的是，它丝毫不影响我程序的运行。而我也不明白为什么它的值会发生改变。目前怀疑是段地址或者是栈改变导致了变量的寻址出现了某些问题。

参考文献：

1．《80x86 汇编语言程序设计教程》 （杨季文）
2．《nasm 中文手册》 （csdn）

3．《一个操作系统的实现》　于渊
4．《汇编语言（第三版）》　　王爽