

实现时间片轮转的二态进程模型

实现时间片轮转的二态进程模型

个人信息

实验题目

实验目的

实验要求

实验内容

实验方案

所用工具

虚拟机配置

代码关键部分

内核

kfun.asm

ckfun.c

实验过程

生成com文件

写盘并且运行

运行虚拟机

实验总结

参考资料

个人信息

- 院系：数据科学与计算机学院
- 年纪：2018
- 姓名：王天龙
- 学号：18340168

实验题目

实现时间片轮转的二态进程模型

实验目的

1. 学习多道程序与CPU分时技术
2. 掌握操作系统内核的二态进程模型设计与实现方法
3. 掌握进程表示方法
4. 掌握时间片轮转调度的实现

实验要求

1. 了解操作系统内核的二态进程模型
2. 扩展实验五的的内核程序，增加一条命令可同时创建多个进程分时运行，增加进程控制块和进程表数据结构。
3. 修改时钟中断处理程序，调用时间片轮转调度算法。
4. 设计实现时间片轮转调度算法，每次时钟中断，就切换进程，实现进程轮流运行。
5. 修改save()和restart()两个汇编过程，利用进程控制块保存当前被中断进程的现场，并从进程控制块恢复下一个进程的现场。
6. 编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

实验内容

1. 修改实验5的内核代码，定义进程控制块PCB类型，包括进程号、程序名、进程内存地址信息、CPU寄存器保存区、进程状态等必要数据项，再定义一个PCB数组，最大进程数为10个。
2. 扩展实验五的的内核程序，增加一条命令可同时执行多个用户程序，内核加载这些程序，创建多个进程，再实现分时运行。
3. 修改时钟中断处理程序，保留无敌风火轮显示，而且增加调用进程调度过程。
4. 内核增加进程调度过程：每次调度，将当前进程转入就绪状态，选择下一个进程运行，如此反复轮流运行。
5. 修改save()和restart()两个汇编过程，利用进程控制块保存当前被中断进程的现场，并从进程控制块恢复下一个进程的运行。
6. 实验5的内核其他功能，如果不必要，可暂时取消服务。







实验方案

所用工具

实验平台是windows10系统，使用的软件有：虚拟机软件vmware workstation pro、汇编语言编译器tasm、c语言编译器tcc、链接器tlink、可视化编译十六进制文件内容工具winhex、代码编辑器visual studio 2019

虚拟机配置

虚拟机为1cpu，内存为4MB，使用一个1.44MB大小的软盘，选择从软盘启动

设备	摘要
 内存	4 MB
 处理器	1
 CD/DVD (IDE)	自动检测
 软盘	正在使用文件 D:\computer_op...
 网络适配器	NAT
 声卡	自动检测
 显示器	自动检测

代码关键部分

内核

- kernel.asm
 - _start 入口
- kfun.asm
 - _printc
 - _readkeypress
 - _cleanscreen
 - _loadblock
 - _save
 - _restart
 - _setIF
 - _initalStack
 - _LoadINT8H
 - _LoadINT20H
 - _LoadINT21H
 - _LoadINT22H

- SYCALL_1H
- SYCALL_2H
- SYSCALL_4CH
- ckfun.c
 - struct cpuState
 - struct PCB
 - void strcpy(char* dest,char* source)
 - void initPCBList()
 - void switchProcess()
 - int createProcess(int cs,int ip,int blockid,int numOfblock)
 - void endProcess()
 - void printf(char* s)
 - void scanf(char* s)
 - void kernelmain()

kfun.asm

_save的实现

这个函数用于切换进程时，把当前进程的cpu状态都保存起来，保存到进程控制块里面。内核中有一个进程控制块数组(11个元素，10个用户进程，1个系统进程)，用一个指针指向当前正在运行的进程控制块。

```
#define MaxProcessNum 11/*加多了一个系统PCB*/
typedef struct
{
    int ax;/*偏移0*/
    int bx;
    int cx;
    int dx;
    int di;
    int bp;
    int es;
    int ds;
    int si;
    int ss;
    int sp;
    int ip;
    int cs;
    int flags;/*偏移26*/
}cpuState;

typedef struct
{
    cpuState cpu;
    int pid;
    char name[NameLen + 1];
    char state;/*三种状态,0:PCB空闲,1:进程运行,2:进程阻塞*/
}PCB;

PCB pcbList[MaxProcessNum];/*最后一个是一个系统的进程*/

PCB *currentPCB = &pcbList[MaxProcessNum - 1];/*指向当前运行的PCB块*/
PCB *kernelPCB = &pcbList[MaxProcessNum - 1];/*指向内核的PCB*/

int numOfProcess = 0;/*用户进程个数*/
```

currentPCB保存着cpu当前状态要保存到的地址，_save函数只需要小改动即可。

```
_save proc near
    push ds
    push cs
    pop ds;ds=cs
    push si

    ;lea si,_currentPCB;si是currentPCB的地址
    mov si,word ptr DGROUP:_currentPCB;si是_currentPCB的内容，即当前PCB的地址
    pop word ptr [si+16];保存si
    pop word ptr [si+14];保存ds

    lea si,ret_temp
    pop word ptr [si];保存_save的返回点

    mov si,word ptr DGROUP:_currentPCB
    pop word ptr [si+22];保存ip
    pop word ptr [si+24];保存cs
    pop word ptr [si+26];保存flags

    mov [si+18],ss
    mov [si+20],sp

    mov si,ds
    mov ss,si
    mov sp,word ptr DGROUP:_currentPCB
    add sp,14

    push es
    push bp
    push di
    push dx
    push cx
    push bx
    push ax

    mov si,word ptr DGROUP:_kernelPCB
    mov sp,[si+20]
    mov ax,cs
    mov es,ax
    ;至此ss,sp,ds,es,cs都是内核的了

    lea si,ret_temp
    mov ax,[si]
    jmp ax
_save endp
```

_restart的实现

这个函数用于恢复进程，和_save一样，只需要小改动即可。_restart把currentPCB指向的进程恢复，然后跳转到哪里运行。所以，_save和_restart配合，修改currentPCB就可以实现安全的进程切换了。

```
_restart proc near
    mov si,word ptr DGROUP:_kernelPCB
    mov [si+20],sp
    ;lea sp,_currentPCB
```

```

mov sp,word ptr DGROUP:_currentPCB
pop ax
pop bx
pop cx
pop dx
pop di
pop bp
pop es;此时sp指向ds

lea si,ds_temp
pop word ptr [si]
lea si,si_temp
pop word ptr [si]

lea si,bx_temp
mov [si],bx;保护bx, 用它给ss赋值
;恢复栈
pop bx;此时sp指向sp
mov ss,bx
mov bx,sp
mov sp,[bx]

add bx,2;bx指向ip
;flags,cs,ip压栈
push word ptr [bx+4]
push word ptr [bx+2]
push word ptr [bx]

;恢复ds,si
push ax
push word ptr [si];把bx压栈保存
lea si,ds_temp
mov ax,[si]
lea si,si_temp
mov bx,[si]
mov ds,ax
mov si,bx

pop bx
pop ax

iret

```

`_restart` endp

`_LoadINT8H`

加载处理时间中断的响应程序。处理时间中断时，首先要调用_save把当前的进程保存起来，然后调用修改currentPCB的函数，最后用_restart恢复执行，从而实现进程切换。代码如下

```

INT8H:
    call _save

;打印风火轮部分忽略

    call _switchProcess; 修改currentPCB的函数，用c语言实现。

    mov al,20h
    out 20h,al
    out 0a0h,al

    jmp _restart;跳转到_restart恢复进程。

```

_LoadINT20H

在实验五的时候，20h号中断用于用户程序返回操作系统。这里对中断响应进行了修改。用户进程结束时，会触发20h中断，20H中断响应后，调用了c语言函数endProcess，这个函数用于结束用户进程，把用户进程的state改成0(PCB块空闲)，然后进入一个死循环阻塞，直到一个时间片结束触发了8h时钟中断，时钟中断再将进程切换。注意，再阻塞之前要把IF标志位置为1，否则不会响应时钟中断，为此增加了一个_setIF函数。

20h号中断响应

```

INT20H:
    mov ax,cs
    mov ds,ax
    mov ss,ax
    mov es,ax
    mov si,word ptr DGROUP:_kernelPCB
    mov sp,[si+20]

    jmp _endProcess;用jmp跳转而不用call
_LoadINT20H endp

```

_setIF

```

_setIF proc near
    sti
_setIF endp

```

_initalStack实现

这个汇编过程用于创建用户进程时，初始化用户进程的栈，即压一个0x0000进去。用于用户进程结束后返回代码前缀段以返回操作系统。调用时，有一个参数，即用户进程所在的段地址(ss)，并且返回栈顶位置(sp)。

```

_initalStack proc near
    push bp
    mov bp,sp
    push bx
    mov ax,[bp+4];获取用户进程所在的段
    mov ss,ax;切换到用户栈
    mov sp,0
    xor ax,ax

```

```

push ax;用户栈里压一个0x0000
mov ax,sp;返回值
mov bx,cs;恢复到内核栈
mov ss,bx
sub bp,2;ss:bp==bx
mov sp,bp
pop bx
pop bp
ret
_initalStack endp

```

ckfun.c

这个文件包含了内核的主要代码。增加了一些新函数，用于创建进程、切换进程、结束进程。同时，对内核进行了修改。

创建进程

创建进程使用的是createProcess函数，要创建进程，首先要找到一个空闲的PCB块，，然后对cs, ip, ss, ds, es, sp, flags寄存器的值根据进程加载的位置进行初始化，同时还要对PCB块的id, 别名赋值，并且从软盘中加载程序到内存，最后把PCB块的state置为2(阻塞)，等待时间片结束切换运行。

```

/*创建成功，则返回pid，否则返回-1*/
/*参数的含义：cs:ip为程序加载位置，blockid是程序在软盘的起始块号，numOfblock说明程序占用了多少块*/
int createProcess(int cs, int ip, int blockid, int numOfblock)
{
    int i;
    for (i = 0; i < MaxProcessNum; i++)
    {
        if (pcbList[i].state == 0)/*找到空闲的PCB块*/
        {
            break;
        }
    }
    if (i == MaxProcessNum)/*没有空闲的PCB块了*/
    {
        printf("Error: create process failed\r\n");
        return 0;
    }
    /*初始化寄存器*/
    pcbList[i].cpu.cs = cs;
    pcbList[i].cpu.ds = cs;
    pcbList[i].cpu.es = cs;
    pcbList[i].cpu.ip = ip;
    /*初始化进程的栈(压一个0x00)*/
    pcbList[i].cpu.ss = cs;
    pcbList[i].cpu.sp = initalStack(cs);
    /*初始化标志寄存器*/
    pcbList[i].cpu.flags = 512;

    pcbList[i].pid = i;
    pcbList[i].name[0] = '0' + i;
    pcbList[i].name[1] = 0;
    LoadPSP(cs);/*加载PSP段*/
    loadblock(cs, ip, blockid, numOfblock);/*把程序装入内存*/
    pcbList[i].state = 2;
    numOfProcess += 1;/*用户进程个数加一*/
}

```

```

    return i;
}

```

切换进程

切换进程由switchProcess()函数完成，由时钟中断响应调用。switchProcess()首先判断用户进程的个数，如果为0，就直接切换到内核进程。如果不为0，则从PCB数组里找到下一个阻塞的进程，然后切换。在这里，用户进程和内核进程视为一样，即用户进程和内核进程交替执行。具体代码如下

```

void switchProcess()
{
    int i;
    if (numOfProcess == 0)/*如果用户进程的个数为0，则直接切换为内核进程*/
    {
        currentPCB = &pcbList[MaxProcessNum - 1];
        currentPCB->state = 1;
        return;
    }
    for (i = (currentPCB->pid + 1) % MaxProcessNum; i != currentPCB->pid; i++, i
    %= MaxProcessNum)/*寻找一个阻塞的进程*/
    {
        if (pcbList[i].state == 2)
        {
            currentPCB->state = 2;
            pcbList[i].state = 1;
            currentPCB = &pcbList[i];
            return;
        }
    }
}

```

结束进程

结束进程由endProcess()函数完成。要结束一个用户进程，首先把PCB块的state置为0(空闲)，并且把用户进程的个数减一，然后开中断，让cpu可以响应时钟中断，这是因为endProcess是由20h号中断调用，响应中断时，IF已经被置为0，最后进入一个死循环阻塞。直至一个时间片结束，让内核把进程切换走，至此，该用户进程结束。

```

void endProcess()/*结束进程*/
{
    currentPCB->state = 0;
    numOfProcess--;
    /*要在这里开中断*/
    setIF();
    while (1);/*阻塞在这里，直至一个时间片结束*/
}

```

对内核主函数的修改

对内核主函数的修改主要是为了让内核适应多进程。主要修改在于让内核调用创建进程的函数，并且让内核等待用户进程全部结束时，再读取新的命令。修改如下

```

void kernelmain()
{
    ....
}

```



```

while (1)
{
    printf("m for message, l to list, c to clean screen, 1-8 to select
program: ");
    scanf(&c);
    switch (c)
    {

        .....

        case '8':
            /*创建多个用户进程*/
            createProcess(0x2000, 0x100, 7, 1);
            createProcess(0x3000, 0x100, 8, 1);
            createProcess(0x4000, 0x100, 9, 1);
            createProcess(0x5000, 0x100, 10, 1);
            break;
            .....
    }
    while (numOfProcess != 0);/*在这里阻塞，直至用户进程都完成再读取新的命令*/
}
}

```

用户程序和之前没有变化，可以直接使用。

实验过程

生成com文件

在DoxBox里使用命令

```

tasm kernel.asm
tasm kfun.asm
tcc -c ckfun.c
tlink /3 /t kernel.obj kfun.obj ckfun.obj, kernel.com

```

即可生成内核程序的com执行体。用户程序也类似。

写盘并且运行

写盘式，引导程序在首扇区，内核在2、3、4、5、6扇区，用户程序在7-15号扇区，可以用指令查看用户程序所在的扇区

运行虚拟机

风火轮显示效果

```
m for message, l to list, c to clean screen, 1-5 to select program: _
```

运行测试程序1，输入6

```
M for message, l to list, c to clean screen, 1-7 to select program: 6
ch=c
str=hello,world!
a=12345
c
hello,world!
ch=c,a=12345,str=hello,world!
M for message, l to list, c to clean screen, 1-7 to select program:
```

运行测试程序2，输入7

```
m for message, l to list, c to clean screen, 1-7 to select program: 7
m for message, l to list, c to clean screen, 1-7 to select program: _
```

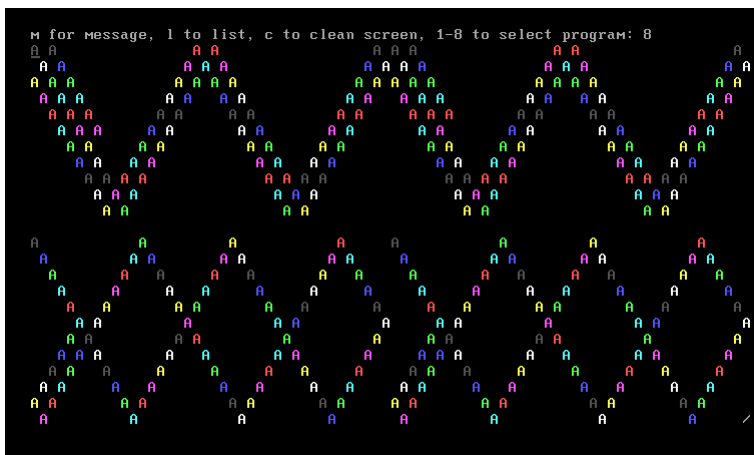
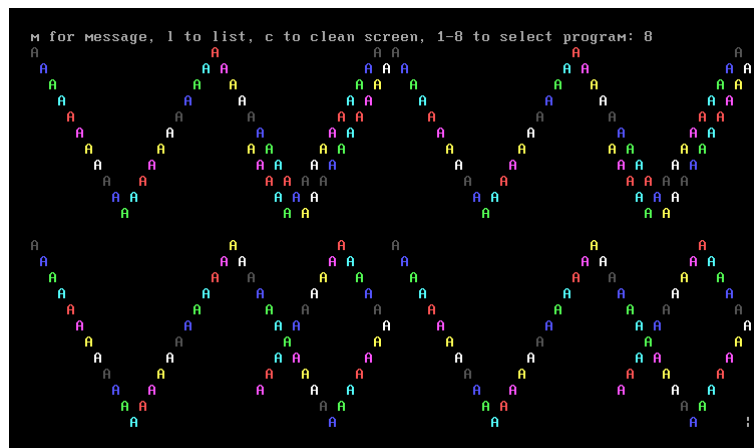
INT22H

运行正常

字符碰壁，输入1-4

```
m for message, l to list, c to clean screen, 1-7 to select program: 3
m for message, l to list, c to clean screen, 1-7 to select program: _
```

多进程运行，输入8



实验总结

这次实验中出现的bug有很多，都是由时间中断的_save和_restart，以及切换进程的switchProcess函数不协调引起的。由于这些都发生在时间中断了，也没有什么好的方法使用断点调试，只能一步一步地排查，把不合理的地方改过来。

参考资料