实验六：实现时间片轮转的二态进程模型

(15分)

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。

4、修改时钟中断处理程序，调用时间片轮转调度算法。

5、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。

define MaxProcessNo=10

typedef strut {

int ax;

int bx;

int cx;

int dx;

int cs;

int ds;

int es;

int ss;

int sp;

int bp;

int di;

int si;

int ip;

int flag;

} cpuRegisters

typedef struct {

cpuRegisters cpuRegs;

int pid;

char pname[10];

char pstate;

…

} PCB;

PCB pcblist[MaxProcessNo]

1. 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
2. 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

Timer:

save()

call showWingFireWheel() ；无敌风火轮显示

call \_schedule() ; 调用进程调度过程

jmp restart

1. 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。

void schedule(){

CurrentProcessNo++；

if (CurrentProcessNo=MaxProcessNo)

CurrentProcessNo=0；

}

1. 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。

;参考程序

;Minix的save和restart

; save

;=====================================================

save: ; save the machine state in the proc table.

push ds ; stack: psw/cs/pc/ret addr/ds

push cs ; prepare to restore ds

pop ds ; ds has now been set to cs

mov ds,ker\_ds ; word 4 in kernel text space contains ds value

pop ds\_save ; stack: psw/cs/pc/ret addr

pop ret\_save ; stack: psw/cs/pc

mov bx\_save,bx ; save bx for later ; we need a free register

mov bx,dgroup:proc\_ptr ; start save set up; make bx point to save area

add bx,OFF ; bx points to place to store cs

pop PC-OFF[bx] ; store pc in proc table

pop csreg-OFF[bx] ; store cs in proc table

pop PSW-OFF[bx] ; store psw

mov ssreg-OFF[bx],ss ; store ss

mov spreg-OFF[bx],sp ; sp as it was prior to interrupt

mov sp,bx ; now use sp to point into proc table/task save

mov bx,ds ; about to set ss

mov ss,bx ; set ss

push ds\_save ; start saving all the registers, sp first

push es ; save es between sp and bp

mov es,bx ; es now references kernel memory too

push bp ; save bp

push di ; save di

push si ; save si

push dx ; save dx

push cx ; save cx

push bx\_save ; save original bx

push ax ; all registers now saved

mov sp,offset dgroup:k\_stack ; temporary stack for interrupts

add sp,K\_STACK\_BYTES ; set sp to top of temporary stack

mov splimit,offset dgroup:k\_stack ; limit for temporary stack

add splimit,8 ; splimit checks for stack overflow

mov ax,ret\_save ; ax = address to return to

jmp ax ; return to caller; Note: sp points to saved ax

;==================================================

; restart

;==================================================

restart: ; This routine sets up and runs a proc or task.

cmp dgroup:cur\_proc,IDLE; restart user; if cur\_proc = IDLE, go idle

je \_idle ; no user is runnable, jump to idle routine

cli ; disable interrupts

mov sp,dgroup:proc\_ptr ; return to user, fetch regs from proc table

pop ax ; start restoring registers

pop bx ; restore bx

pop cx ; restore cx

pop dx ; restore dx

pop si ; restore si

pop di ; restore di

mov lds\_low,bx ; lds\_low contains bx

mov bx,sp ; bx points to saved bp register

mov bp,SPLIM-ROFF[bx] ; splimit = p\_splimit

mov splimit,bp ; ditto

mov bp,dsreg-ROFF[bx] ; bp = ds

mov lds\_low+2,bp ; lds\_low+2 contains ds

pop bp ; restore bp

pop es ; restore es

mov sp,spreg-ROFF[bx] ; restore sp

mov ss,ssreg-ROFF[bx] ; restore ss using the value of ds

push PSW-ROFF[bx] ; push psw (flags)

push csreg-ROFF[bx] ; push cs

push PC-ROFF[bx] ; push pc

lds bx,DWORD PTR lds\_low ; restore ds and bx in one fell swoop

iret ; return to user or task

1. 实验5的内核其他功能，如果不必要，可暂时取消服务。

用户程序d

用户程序c

用户程序a

用户程序b

监控程序(内核)

引导程序