操作系统

第三章 进程与处理机管理

文艳军

计算机学院

本章讲授计划

① 进程的描述与状态

② 进程的创建

③ 进程的调度时机与算法

④ 进程切换

⑤ 线程的引入

目录

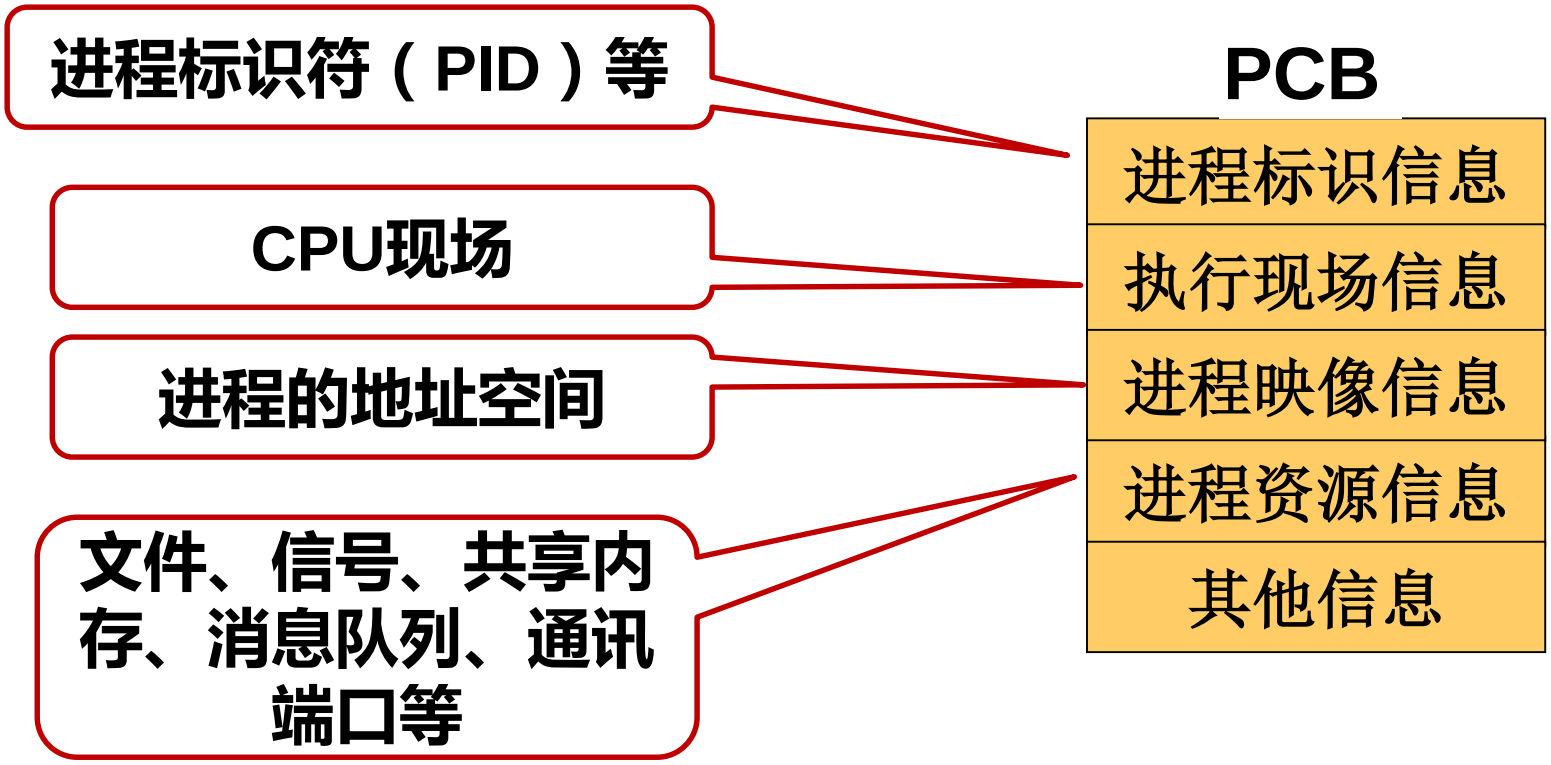
一. 进程的描述

进程： 是有独立功能的程序关于某个数据集合的一次运行活动

进程映像： 进程的程序、数据、栈的集合

进程控制块PCB

- 是OS控制进程运行用的数据结构



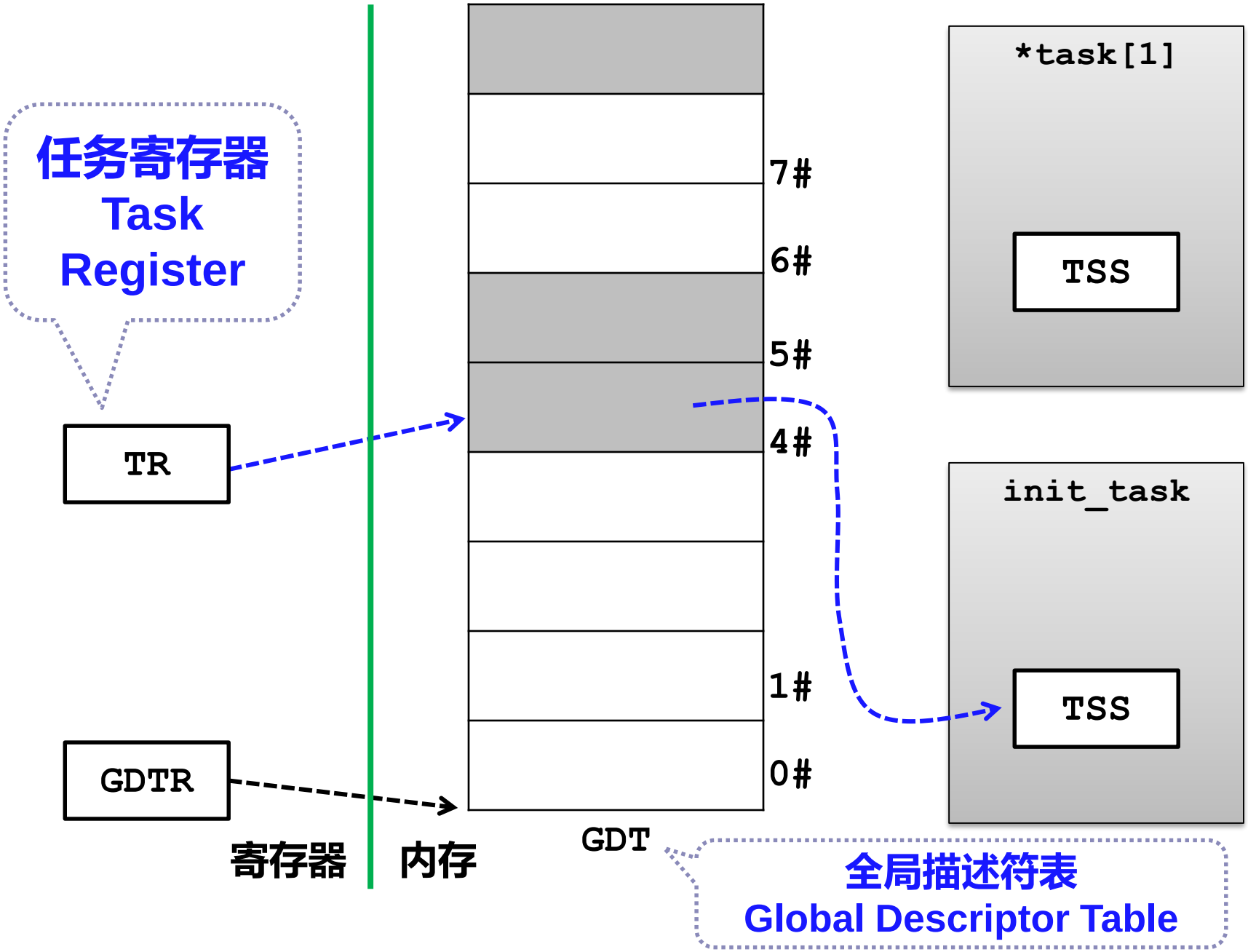
```
struct task_struct {
/* these are hardcoded - don't touch */
        long state;     /* -1 unrunnable, 0 runnable, >0 stopped */
        long counter;
        long priority;
        long signal;
        struct sigaction sigaction[32];
        long blocked; /* bitmap of masked signals */
/* various fields */
        int exit_code;
        unsigned long start_code,end_code,end_data,brk,start_stack;
        long pid,father,pgrp,session,leader;
        unsigned short uid,euid,suid;
        unsigned short gid,egid,sgid;
        long alarm;
        long utime,stime,cutime,cstime,start_time;
        unsigned short used_math;
/* file system info */
        int tty;            /* -1 if no tty, so it n
        unsigned short umask;
        struct m_inode * pwd;
        struct m_inode * root;
        struct m_inode * executable;
        unsigned long close_on_exec;
        struct file * filp[NR_OPEN];
/* ldt for this task 0 - zero 1 - cs 2 - ds&ss *
        struct desc_struct ldt[3];
/* tss for this task */
        struct tss_struct tss;
} ?  end task_struct ?  ;
```

局部描述符表
进程的地址空间

任务状态段
task state segment
进程现场区

```
struct tss_struct {
    long    back_link;   /* 16 high bits zero */
    long    esp0;
    long    ss0;         /* 16 high bits zero */
    long    esp1;
    long    ss1;         /* 16 high bits zero */
    long    esp2;
    long    ss2;         /* 16 high bits zero */
    long    cr3;
    long    eip;
    long    eflags;
    long    eax,ecx,edx,ebx;
    long    esp;
    long    ebp;
    long    esi;
    long    edi;
    long    es;          /* 16 high bits zero */
    long    cs;          /* 16 high bits zero */
    long    ss;          /* 16 high bits zero */
    long    ds;          /* 16 high bits zero */
    long    fs;          /* 16 high bits zero */
    long    gs;          /* 16 high bits zero */
    long    ldt;         /* 16 high bits zero */
    long    trace_bitmap;   /* bits: trace 0, bitmap 16-31 */
    struct i387_struct i387;
} ? end tss_struct ? ;
```

0号进程的核心栈
栈底


任务寄存器
Task
Register
TR
GDTR
寄存器
内存
7#
6#
5#
4#
1#
0#
GDT
*task[1]
TSS
init_task
TSS
全局描述符表
Global Descriptor Table

一. 进程的描述

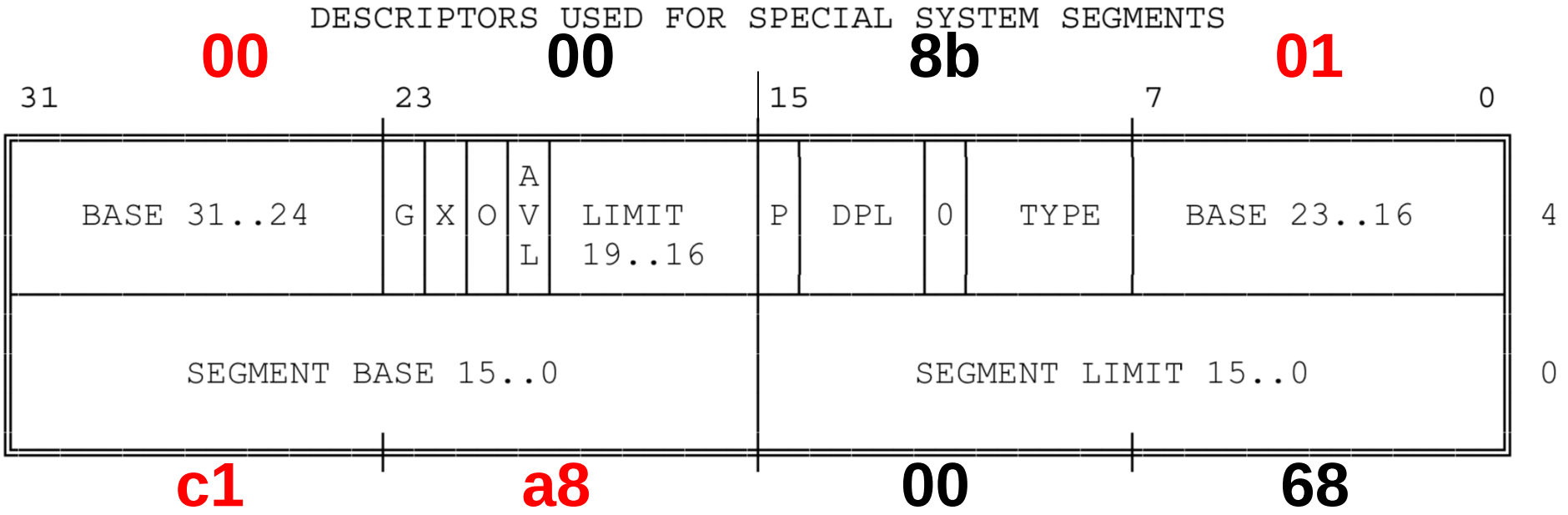
- 演示1：

  #0进程的PCB、进程现场和核心栈

  - 位置：task0入口

段描述符格式（系统段）

- 系统段：包括任务状态段、LDT段等



目录

独学&讨论

学习（《Linux内核完全剖析》）：

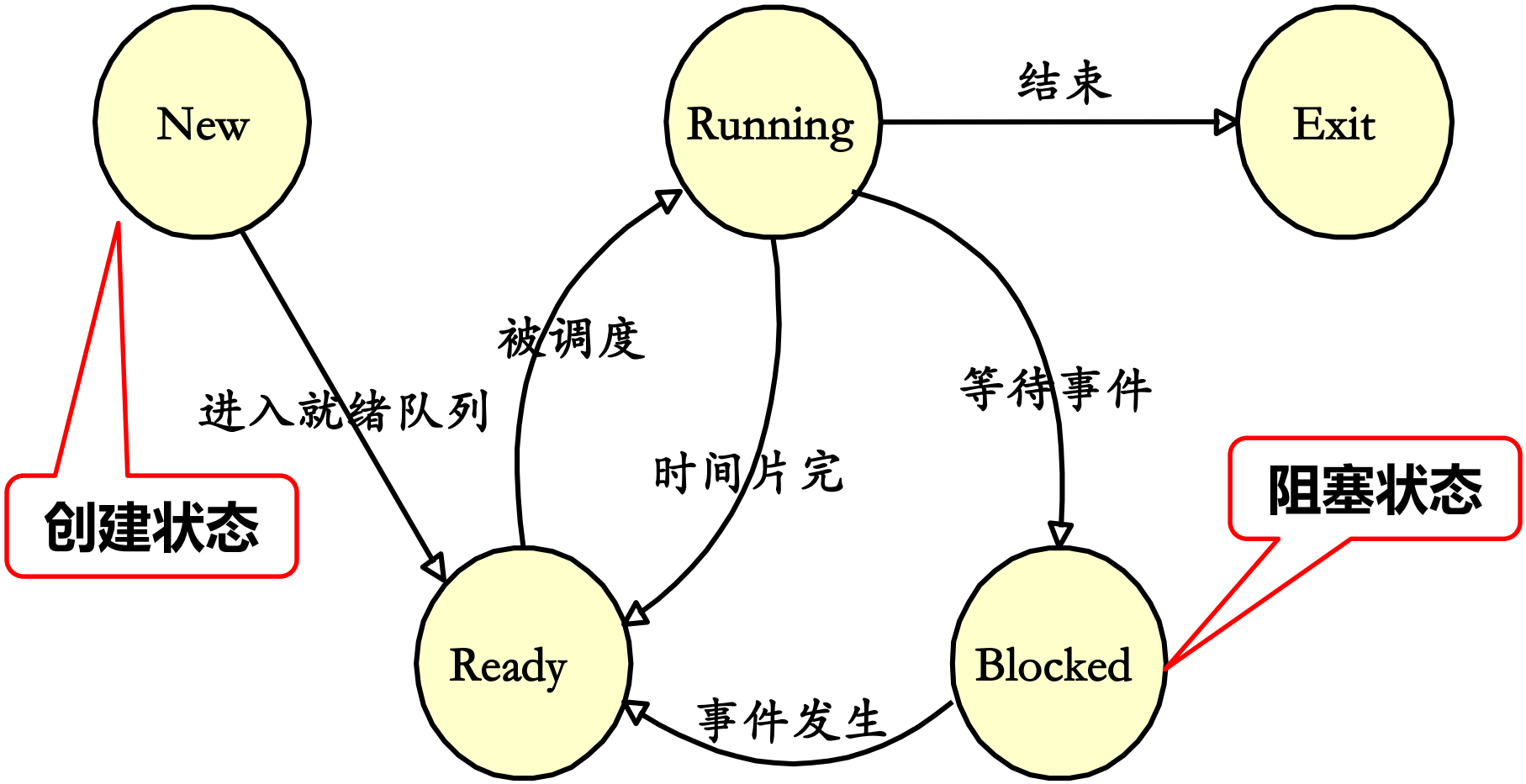
- §4.7.1：P126始，任务的结构和状态
- §4.7.3：P330始，任务管理数据结构

问题：

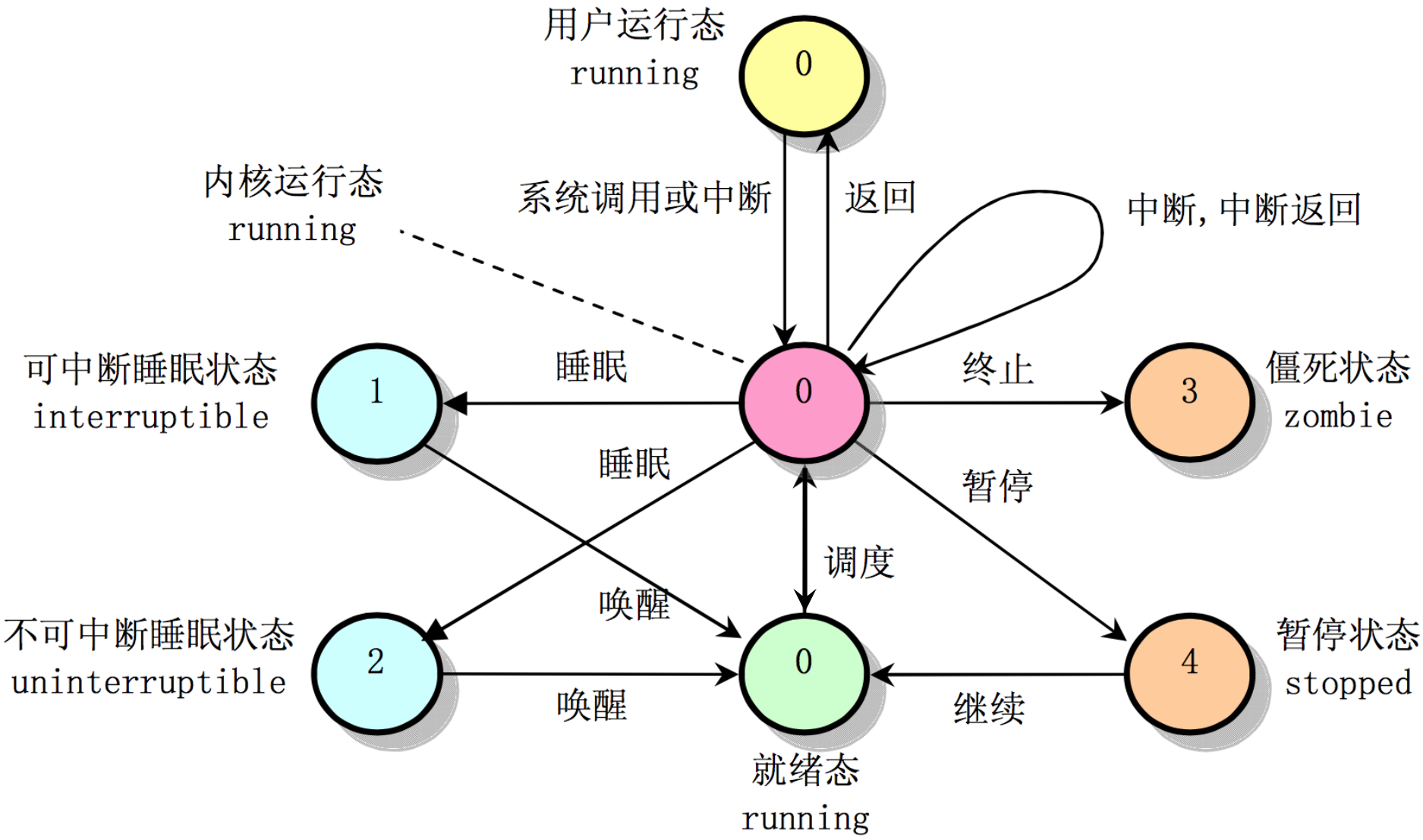
- TSS中记录了所有CPU寄存器的值吗？
- TSS.ESP指向哪里？
- 如何根据TSS找到用户栈的位置？

目录

三. 进程状态变化模型


New
Running
Exit
结束
被调度
进入就绪队列
等待事件
时间片完
创建状态
阻塞状态
Ready
Blocked
事件发生

Linux 0.11的进程状态变化模型



```
        sched_init();
        buffer_init(buffer_memory_end);
        hd_init();
        floppy_init();
        sti();    /* wyj */
        move_to_user_mode();

        if (!fork()) {          /* we count on this going ok */
                (void)mysignal(SIGALRM, SIG_IGN);
                for(;;) {
                        task1(); /* wyj */
                        alarm(1);
                        pause();
                }
/*              init();  */
        }
/*
 *   NOTE! !    For any other task 'pause()' would mean we have to get a
 * signal to awaken, but task0 is the sole exception (see 'schedule()')
 * as task 0 gets activated at every idle moment (when no other tasks
 * can run). For task0 'pause()' just means we go check if some other
 * task can run, and if not we return here.
 */
        for(;;) {
                task0();  /* wyj */
                pause();
        }
} ? end main ?
```

1号进程

#1进程的状态变化

#1进程的状态变化

```
void schedule(void)
{
    int i,next,c;
    struct task_struct ** p;

/* check alarm, wake up any interruptible tasks that have got a signal */

    for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
        if (*p) {
            if ((*p)->alarm && (*p)->alarm < jiffies) {
                    (*p)->signal |= (1<<(SIGALRM-1));
                    (*p)->alarm = 0;
                }
            if ((*p)->signal & ~(_BLOCKABLE & (*p)->blocked)) &&
            (*p)->state==TASK_INTERRUPTIBLE)
                (*p)->state=TASK_RUNNING;
        }
```

Linux 0.11的进程状态变化模型

- 演示2：

#1进程的状态变化

- 位置：task1，sys_pause
  sched.c:120

目录

独学&讨论

学习(《Linux内核完全剖析》):

- 源码: sys_pause, schedule, sys_signal, main

问题:

- sys_signal的功能是什么?

小结

一. 进程的描述

task_struct, TSS

二. 进程的状态变化模型

五种基本状态，Linux 0.11的两种睡眠态