**linux0.11进程调度详解**

置顶 2018年05月31日 16:47:41 [sadyjx](https://me.csdn.net/qq_33973210) 阅读数：156

* + [内核调度初始化](https://blog.csdn.net/qq_33973210/article/details/80519327#%E5%86%85%E6%A0%B8%E8%B0%83%E5%BA%A6%E5%88%9D%E5%A7%8B%E5%8C%96)
    - [sched\_init()](https://blog.csdn.net/qq_33973210/article/details/80519327#schedinit)
      * [set\_tss\_desc() & set\_ldt\_desc()](https://blog.csdn.net/qq_33973210/article/details/80519327#settssdesc-setldtdesc)
      * [ltr() & lldt()](https://blog.csdn.net/qq_33973210/article/details/80519327#ltr-lldt)
      * [LATCH](https://blog.csdn.net/qq_33973210/article/details/80519327#latch)
  + [进程调度流程](https://blog.csdn.net/qq_33973210/article/details/80519327#%E8%BF%9B%E7%A8%8B%E8%B0%83%E5%BA%A6%E6%B5%81%E7%A8%8B)
    - [中断处理函数\_timer\_interrupt](https://blog.csdn.net/qq_33973210/article/details/80519327#%E4%B8%AD%E6%96%AD%E5%A4%84%E7%90%86%E5%87%BD%E6%95%B0timerinterrupt)
    - [do\_timer()](https://blog.csdn.net/qq_33973210/article/details/80519327#dotimer)
    - [调度函数schedule()](https://blog.csdn.net/qq_33973210/article/details/80519327#%E8%B0%83%E5%BA%A6%E5%87%BD%E6%95%B0schedule)
    - [难点——任务切换switch\_to()](https://blog.csdn.net/qq_33973210/article/details/80519327#%E9%9A%BE%E7%82%B9%E4%BB%BB%E5%8A%A1%E5%88%87%E6%8D%A2switchto)
      * [跳转](https://blog.csdn.net/qq_33973210/article/details/80519327#%E8%B7%B3%E8%BD%AC)
      * [切换回来](https://blog.csdn.net/qq_33973210/article/details/80519327#%E5%88%87%E6%8D%A2%E5%9B%9E%E6%9D%A5)

**文章写的有些长，把相关的、用到的函数都列出来了，看完应该能对进程调度相关的代码有一定了解**

**内核调度初始化**

**sched\_init()**

kernel/sched.c : 385

该函数只有36~40行与进程调度有关，只想了解进程调度的同学可以忽略该函数其他部分

void sched\_init(void)

{

int i;

struct desc\_struct \* p;

if (sizeof(struct sigaction) != 16)

panic("Struct sigaction MUST be 16 bytes");

*//以下两行初始化init任务(任务0)的任务状态段描述符和局部数据表描述符*

*//详细介绍见下文*

set\_tss\_desc(gdt+FIRST\_TSS\_ENTRY,&(init\_task.task.tss));

set\_[ldt](https://www.baidu.com/s?wd=ldt&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)\_desc(gdt+FIRST\_LDT\_ENTRY,&(init\_task.task.ldt));

p = gdt+2+FIRST\_TSS\_ENTRY;*//指向gtd表中init任务的后一个任务的任务状态段描述符*

for(i=1;i<NR\_TASKS;i++) { *//清gdt表中init任务后所有描述符*

task[i] = NULL;

p->a=p->b=0;

p++;

p->a=p->b=0;

p++;

}

*/\* Clear NT, so that we won't have troubles with that later on \*/*

*//这行内联汇编利用堆栈做中间变量将标志寄存器的NT位清除，以屏蔽任务切换*

*//1.首先用pushfl指令将标志寄存器压栈*

*//2.然后通过栈顶指针sp来修改刚才压栈的标志寄存器*

*//3.然后将栈顶的内容弹出到标志寄存器中，完成修改*

\_\_asm\_\_("pushfl ; andl $0xffffbfff,(%esp) ; popfl");

*//加载任务寄存器和局部描述符表寄存器*

*//详细介绍见下文*

ltr(0);

lldt(0);

*//下面的代码用于对8253定时器初始化操作，使其每10ms发出一个定时中断*

outb\_p(0x36,0x43); *//模式设置*

outb\_p(LATCH & 0xff , 0x40);*//设置定时值的低字节(LATCH见下文)*

outb(LATCH >> 8 , 0x40); *//定时值的高字节*

set\_intr\_gate(0x20,&timer\_interrupt);*//设置定时中断处理函数，中断处理函数\_timer\_interrupt下文中有详解*

outb(inb\_p(0x21)&~0x01,0x21); *//修改中断控制器屏蔽码以开启时钟中断*

set\_system\_gate(0x80,&system\_call); *//设置系统调用中断门，与本文无关*

}

**set\_tss\_desc() & set\_ldt\_desc()**

在include/asm/system.h中最后两行定义   
set\_tss\_desc()和set\_ldt\_desc()是两个宏函数，该宏函数把传入的gdt的[偏移地址](https://www.baidu.com/s?wd=%E5%81%8F%E7%A7%BB%E5%9C%B0%E5%9D%80&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)转换成(char\*)型后，逐字节设置描述符   
源码如下

*#define \_set\_tssldt\_desc(n,addr,type) \*

\_\_asm\_\_ ("movw $104,%1\n\t" \

"movw %%ax,%2\n\t" \

"rorl $16,%%eax\n\t" \

"movb %%al,%3\n\t" \

"movb $" type ",%4\n\t" \

"movb $0x00,%5\n\t" \

"movb %%ah,%6\n\t" \

"rorl $16,%%eax" \

::"a" (addr), "m" (\*(n)), "m" (\*(n+2)), "m" (\*(n+4)), \

"m" (\*(n+5)), "m" (\*(n+6)), "m" (\*(n+7)) \

)

*#define set\_tss\_desc(n,addr) \_set\_tssldt\_desc(((char \*) (n)),addr,"0x89")*

*#define set\_ldt\_desc(n,addr) \_set\_tssldt\_desc(((char \*) (n)),addr,"0x82")*

**ltr() & lldt()**

在include/linux/sched.h : 153定义   
1. 将传入的任务号计算出该任务的**任务状态段/局部描述符表**在gdt中的偏移后   
2. 用 **ltr/lldt** 汇编指令将任务号对应的 **任务状态段/局部描述符表** 加载到对应的 **任务状态寄存器TR/局部描述符表寄存器LDT**

*#define FIRST\_TSS\_ENTRY 4*

*#define FIRST\_LDT\_ENTRY (FIRST\_TSS\_ENTRY+1)*

*#define \_TSS(n) ((((unsigned long) n)<<4)+(FIRST\_TSS\_ENTRY<<3))*

*#define \_LDT(n) ((((unsigned long) n)<<4)+(FIRST\_LDT\_ENTRY<<3))*

*#define ltr(n) \_\_asm\_\_("ltr %%ax"::"a" (\_TSS(n)))*

*#define lldt(n) \_\_asm\_\_("lldt %%ax"::"a" (\_LDT(n)))*

**LATCH**

在kernel/sched.c : 46行定义   
设置[时钟芯片](https://www.baidu.com/s?wd=%E6%97%B6%E9%92%9F%E8%8A%AF%E7%89%87&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)8253的定时初值   
linux0.11是通过时钟中断来进行任务调度的   
Linux希望的中断频率是100，即10ms发出一次时钟中断

*//HZ在include/linux/sched.h定义为100，1.193180MHZ为定时芯片8253的输入时钟频率*

#define LATCH (1193180/HZ)

**进程调度流程**

10ms触发中断 –> 进入中断处理函数\_timer\_interrupt –> \_do\_timer –> schedule()调度函数 –> 任务切换switch\_on()

**中断处理函数\_timer\_interrupt**

定义在kernel/system\_call.s : 176

\_timer\_interrupt:

push %ds *# save ds,es and put kernel data space*

push %es *# into them. %fs is used by \_system\_call*

push %fs

pushl %edx *# we save %eax,%ecx,%edx as gcc doesn't*

pushl %ecx *# save those across function calls. %ebx*

pushl %ebx *# is saved as we use that in ret\_sys\_call*

pushl %eax

*#以上将寄存器压栈，保护现场*

*#以下5行，将ds、es指向内核数据段，将fs指向局部数据段*

movl $0x10,%eax

mov %ax,%ds

mov %ax,%es

movl $0x17,%eax

mov %ax,%fs

incl \_jiffies *#系统启动后的时钟滴答值+1*

movb $0x20,%al *# EOI to interrupt controller #1*

outb %al,$0x20 *#发送EOI以结束硬件中断*

*#以下三行取当前特权级别，并压栈作为调用\_do\_timer的参数*

movl CS(%esp),%eax

andl $3,%eax *# %eax is CPL (0 or 3, 0=supervisor)*

pushl %eax

call \_do\_timer *# 'do\_timer(long CPL)' does everything from*

addl $4,%esp *# task switching to accounting ...*

jmp ret\_from\_sys\_call

**do\_timer()**

定义在kernel/sched.c : 305

void do\_timer(long cpl)

{

extern int beepcount;

extern void sysbeepstop(void);

*//蜂鸣器*

if (beepcount)

if (!--beepcount)

sysbeepstop();

*//根据当前特权级，将相应的运行时间递增。*

if (cpl)

current->utime++;

else

current->stime++;

*//如果有定时器正在使用，则将定时器链表的第一个定时器的定时值减一，如果定时值为0，则执行其处理程序并删除该定时器*

if (next\_timer) {

next\_timer->jiffies--;

while (next\_timer && next\_timer->jiffies <= 0) {

void (\*fn)(void);*//利用函数指针临时保存当前定时器的处理函数*

fn = next\_timer->fn;

next\_timer->fn = NULL;

next\_timer = next\_timer->next;

(fn)(); *//执行该定时器的处理函数*

}

}

*//软盘相关，与本文无关*

if (current\_DOR & 0xf0)

do\_floppy\_timer();

*//如果当前进程时间片不为0，则退出继续执行当前进程*

if ((--current->counter)>0) return;

current->counter=0;

*//如果当前特权级表示发生中断时正在内核态运行，则返回(内核任务不可被抢占)*

if (!cpl) return;

*//执行调度函数*

schedule();

}

**调度函数schedule()**

定义在kernel/sched.c : 104

void schedule(void)

{

int i,next,c;

struct task\_struct \*\* p;

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

*//遍历任务数组，如果任务设置过定时值alarm并且已经超时，则把信号位图中的SIGALRM置位*

*//LAST\_TASK和FIRST\_TASK在include/linux/sched.h第7、8行定义，分别指向任务数组的最后一个和第一个元素*

for(p = &LAST\_TASK ; p > &FIRST\_TASK ; --p)

if (\*p) {

if ((\*p)->alarm && (\*p)->alarm < jiffies) {

(\*p)->signal |= (1<<(SIGALRM-1));

(\*p)->alarm = 0;

}

*//如果信号位图中有已经置位的信号，并且任务处于可中断状态，则把任务置位就绪态*

if (((\*p)->signal & ~(\_BLOCKABLE & (\*p)->blocked)) &&

(\*p)->state==TASK\_INTERRUPTIBLE)

(\*p)->state=TASK\_RUNNING;

}

*/\* this is the scheduler proper: \*/*

while (1) {

c = -1;

next = 0; *//保存选出的任务的任务号*

i = NR\_TASKS;*//include/linux/sched.h : 4 将该宏定义为64(任务数组长度最大64)*

p = &task[NR\_TASKS];

*//遍历任务数组，选出就绪态的、时间片最大的任务*

while (--i) {

if (!\*--p)

continue;

if ((\*p)->state == TASK\_RUNNING && (\*p)->counter > c)

c = (\*p)->counter, next = i;

}

*//如果所有任务时间片都是0，则执行下面代码为所有任务重新分配时间片，否则跳出当前while(1)循环*

if (c) break;

*//遍历任务数组重新分配时间片，新分配的时间片为counter/2+优先级，所以优先级越高分配到的时间片越大*

for(p = &LAST\_TASK ; p > &FIRST\_TASK ; --p)

if (\*p)

(\*p)->counter = ((\*p)->counter >> 1) +

(\*p)->priority;

}

switch\_to(next);

}

***难点*——任务切换switch\_to()**

在include/linux/sched.h : 171定义   
该函数将当前函数切换到所传参数任务号对应的任务数组中的任务。

#define switch\_to(n) {\

struct {long a,b;} \_\_tmp; \

\_\_asm\_\_("cmpl %%ecx,\_current\n\t" \ *//ecx传入的任务号对应的任务是否为当前任务*

"je 1f\n\t" \ *//若是，则跳出*

"movw %%dx,%1\n\t" \ *//新任务TSS选择符赋值给第一个参数\_\_tmp.b*

"xchgl %%ecx,\_current\n\t" \ *//交换指令，交换后ecx为被切换的任务，current为要切换到的任务*

"ljmp %0\n\t" \ *//跳转至\_\_tmp(新任务的段选择符)，切换到该TSS对应的进程(\_\_tmp.a实际上没用)，实际上的指令是ljmp \_\_tmp.b低字节:\_\_tmp.a；a做偏移地址为0*

"cmpl %%ecx,\_last\_task\_used\_math\n\t" \ *//以下为再次切换回来后检查是否使用过协处理器*

"jne 1f\n\t" \

"clts\n" \

"1:" \

::"m" (\*&\_\_tmp.a),"m" (\*&\_\_tmp.b), \ *//*

"d" (\_TSS(n)),"c" ((long) task[n])); \ *//\_TSS(n)传入给dx，任务号n对应的任务传入给ecx*

}

以下为对switch\_on()函数的详细介绍   
该函数一共两个难点   
1. 怎么跳转到新任务   
2. 任务切换后，怎么执行到协处理器检测代码

**跳转**

* 首先进入函数后定义了一个八字节结构体\_\_tmp，我们只用到了其中的六个字节
* 成员a值为0，成员b的低二字节在第五行被赋值为要切换到任务的TSS **段选择符**
* 跳转到TSS段选择符即会造成任务切换到该TSS对应的任务
* 第七行的跳转代码相当于，跳转到\_\_tmp.b对应的TSS选择符的0偏移处

**切换回来**

* 在任务切换时CPU会保存寄存器现场
* 当前被切换任务的寄存器现场将保存在该任务的TSS结构中
* 需注意在切换前，IP寄存器将指向切换指令的下一条指令，同样IP寄存器也将被CPU保存到结构中
* 当该备切换的任务再一次被内核调度运行时，它将从IP寄存器指向的那条指令开始运行