操作系统 Operating Systems

L15 一个实际的schedule函数

schedule( )函数

lizhijun_os@hit.edu.cn
综合楼411室

授课教师：李治军

Linux 0.11的调度函数schedule()

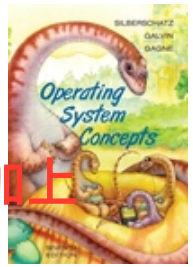
```
void Schedule(void)   //在kernel/sched.c中
{ while(1) { c=-1; next=0; i=NR_TASKS;
    p=&task[NR_TASKS];
    while(--i){ if((*p->state == TASK_RUNNING&&(*p)->counter>c)
      c=(*p)->counter, next=i; }
    if(c) break; //找到了最大的counter
    for(p=&LAST_TASK;p>&FIRST_TASK;--p)
      (*p)->counter=((*p)->counter>>1)
                +(*p)->priority; }
    switch_to(next);}
```

指针指到数组的末尾
各种状态的进程都在
这个数组中

counter是优先级，找到了最大的优先级的任务
用一个counter即作为时间片，又作为优先级

找到了一个，
就直接切换

如果运行态任务的时间片都用完了
那么就将所有的进程的时间片变为时间片初值加上
当前剩余时间片的一半。
就绪态变为处置了，执行IO的变得比初值大，
所以最终IO完成的进程优先级会高，会优先调度！



counter的作用: 时间片

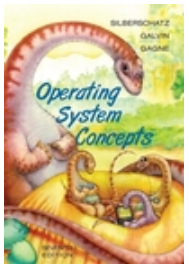
```
void do_timer(...)    //在kernel/sched.c中
{   if((--current->counter>0) return;
    current->counter=0;
    schedule(); }
```

每次时钟中断就
将当前进程的counter—

```
_timer_interrupt: //在kernel/system_call.s中
   ...
   call _do_timer
```

```
void sched_init(void) {
   set_intr_gate(0x20, &timer_interrupt);
```

- counter是典型的时间片，所以是轮转调度，保证了响应



counter的另一个作用: 优先级

```
while(--i){ if((*p->state == TASK_RUNNING&&(*p)->counter>c)
        c=(*p)->counter, next=i; }
```

- 找counter最大的任务调度，counter表示了优先级

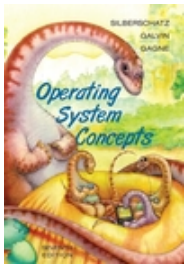
```
for(p=&LAST_TASK;p>&FIRST_TASK;--p)
   (*p)->counter=((*p)->counter>>1)+(*p)->priority; }
```

- counter代表的优先级可以动态调整

  阻塞的进程再就绪以后优先级高于非阻塞进程，为什么？

  进程为什么会阻塞？I/O，正是前台进程的特征

阻塞得越久，counter就会越大，优先级就会越高



counter作用的整理

除2是为了保证有界！
而且除2计算会很快
除3也行

- counter保证了响应时间的界

$$c(t) = c(t-1)/2 + p$$
$$c(0) = p$$

$c(\infty) = ?$ 2p

- 经过IO以后，counter就会变大；IO时间越长，counter越大(为什么?)，照顾了IO进程，变相的照顾了前台进程
- 后台进程一直按照counter轮转，近似了SJF调度
- 每个进程只用维护一个counter变量，简单、高效

- **CPU调度: 一个简单的算法折中了大多数任务的需求，这就是实际工作的schedule函数**