

# 哈工大操作系统-L15一个实际的schedule函数

## 哈工大操作系统-L15一个实际的schedule函数

counter时间片+SFJ调度(几何级数更新counter保证最长为2P)

时间片的作用

优先级的作用

几何级数保证了时间片最长为2P

讲述Linux 0.11的调度函数schedule()

## counter时间片+SFJ调度(几何级数更新counter保证最长为2P)

### 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;
```

在所有任务中，找到就绪的那些，并在就绪的那些中找出counter最大的

```
p=&task[NR_TASKS];
```

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

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

如果某个进程IO的过程

中，没有就绪的进程，那么左边的代码段会对所有的进程进行counter的调整，而IO的进程因为不消耗counter，因此会进行除以2+优先级的counter调整。

多次这样调整后，IO的进程返回时，counter会比较高

```
if(c) break; //找到了最大的counter
```

```
for(p=&LAST_TASK;p>&FIRST_TASK;--p)
```

```
(*p)->counter=( (*p)->counter>>1)
```

如果所有任务的时间片都用完了，依照这个格式更新counter

```
+(*p)->priority; }
```

```
switch_to(next); }
```

如果找到了下一个任务就break，去执行下一个任务

## 时间片的作用

## counter的作用: 时间片

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

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

每到一个时间点, 调用一次do\_timer, 对counter减少一个时间单位, 起到时间片的作用。如果counter用完了, 那么调用别的应用。

```
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

## 优先级的作用

优先级使得IO的程序优先调度

## 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

而阻塞队列中的时间片尚未用完, 此时也参与时间片的分配, 除以2加初始值的分配方式会使得阻塞队列中的进程时间片慢慢增多

## 几何级数保证了时间片最长为2P

- 一直按照时间片轮转, 且时间片长度有界这样能保证后台程序不会饥饿
- 不断的短时间片轮转, 也使得短作业优先完事, 这样也近似了SJF
- IO的优先级高也照顾了前台进程
- 只使用了一个counter就能维护

## counter作用的整理

可能有个疑问，如果IO永远阻塞下去，那么阻塞的进程一旦不阻塞了，时间片不就会很长？

但是我们根据下面的 $c(t)$ 和 $c(0)$ 可知时间片是个几何级数，收敛到 $2P$ 。因此时间片最长为 $2P$

$$\begin{aligned}c(t) &= c(t-1)/2 + p \\ c(0) &= p\end{aligned}$$

$$c(\infty) = ?$$

- **counter**保证了响应时间的界
- 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

