

第6章 进程调度

-  6.1进程调度概念
-  6.2典型调度算法
-  6.3 Linux进程调度

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

《操作系统原理》

6.3 Linux进程调度



教师：苏曙光

华中科技大学软件学院

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

Linux进程类型

普通进程

- 采用动态优先级来调度
- 调度程序周期性地修改优先级（避免饥饿）

实时进程

- 采用静态优先级来调度
- 由用户预先指定，以后不会改变

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

Linux进程的优先级

静态优先级

- 进程创建时指定或由用户修改。

动态优先级

- 在进程运行期间可以按调度策略改变。
- 非实时进程采用动态优先级，由调度程序计算。
- 只要进程占用CPU，优先级就随时间流失而不断减小。
- task_struct的counter表示动态优先级。

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

调度策略（结合task_struct结构）

task_struct->policy指明进程调度策略

```
/*  
 * Scheduling policies  
 */  
#define SCHED_OTHER  
#define SCHED_FIFO  
#define SCHED_RR
```

```
/*  
 * This is an additional bit set when we want to  
 * yield the CPU for one re-schedule..  
 */  
#define SCHED_YIELD 0x10
```



网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

调度策略

■ 实时进程

■ SCHED_FIFO（先进先出）✓

- ◆ 当前实时进程一直占用CPU直至退出或阻塞或被抢占
- ◆ 阻塞后再就绪时被添加到同优先级队列的末尾。

■ SCHED_RR（时间片轮转）✓

- ◆ 与其它实时进程以Round-Robin方式共同使用CPU。
- ◆ 确保同优先级的多个进程能共享CPU。

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

调度策略

■ 非实时进程（普通进程）

- SCHED_OTHER（动态优先级）
- counter成员表示动态优先级

■ 调度策略的改变

- 系统调用sched_setscheduler()改变调度策略。
- 实时进程的子孙进程也是实时进程。

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

进程调度的依据

task_struct

■ policy —

- ◆ 进程的调度策略,用来区分实时进程和普通进程
- ◆ `SCHED_OTHER(0) || SCHED_FIFO(1) || SCHED_RR(2)`

■ priority ✓

- ◆ 进程(包括实时和普通)的静态优先级

■ rt_priority ✓

- ◆ 实时进程特有的优先级: `rt_priority+1000`

■ counter —

- ◆ 进程能连续运行的时间

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

动态优先级与counter

counter值的含义

- 进程能连续运行的时间，单位是时钟滴答tick

◆ 时钟中断周期tick为10ms。若counter=60，则能连续运行600ms。

- 较高优先级的进程一般counter较大。
- 一般把counter看作动态优先级。

counter的初值与priority有关

- 普通进程创建时counter的初值为priority的值。

counter的改变

- 时钟中断tick时，当前进程的counter减1，直到为0被阻塞。

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

子进程新建时的counter

■ 新建子进程counter从父进程时间片counter中继承一半。

```
p->counter = (current->counter + 1) >> 1;  
current->counter >>= 1;
```

■ 防止用户无限制地创建后代进程而长期占有CPU资源

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

调度时机

■ 中断处理过程中直接调用schedule()

- 时钟中断、I/O中断、系统调用和异常
- 内核被动调度的情形

■ 中断处理过程返回用户态时直接调用schedule()

- 必须根据need_resched标记

■ 内核线程可直接调用schedule()进行进程切换

- 内核主动调度的情形

■ 用户态进程只能通过陷入内核后在中断处理过程中被动调度

- 必须根据need_resched标记

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

进程切换

■ 概念

- 内核挂起当前CPU上的进程并恢复之前挂起的某个进程
- 任务切换、上下文切换

■ 与中断上下文的切换有差别

- 中断前后在同一进程上下文中，只是用户态转向内核态执行

■ 进程上下文包含了进程执行需要的所有信息

- 用户地址空间：包括程序代码，数据，用户堆栈等
- 控制信息：进程描述符，内核堆栈等
- 硬件上下文（注意中断也要保存硬件上下文只是保存的方法不同）

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

进程调度和切换的流程

- schedule()函数
 - 选择新进程 `next = pick_next_task(rq, prev);` //进程调度算法
 - 调用宏 context_switch (rq, prev, next) 切换进程上下文
 - prev：当前进程，next：被调度的新进程
 - 调用 switch_to (prev, next) 切换上下文

网址：www.icourses.cn，主页搜索“苏曙光”即可进入MOOC课堂

两个进程A,B切换的基本过程

- 1. 正在运行用户态进程A
- 2. 发生中断（譬如时钟中断）
 - 保存current当前进程的cs:eip/esp/eflags到内核堆栈
 - 从内核堆栈装入ISR中断服务例程的cs:eip和ss:esp
- 3. SAVE_ALL //保存现场，已进入内核中断处理过程
- 4. 中断处理过程中或中断返回前调用了schedule()
 - 其中的switch to做了进程上下文切换
- 5. 运行用户态进程B (B曾经通过以上步骤被切换出去过)
- 6. RESTORE_ALL //恢复现场
- 7. iret //中断返回 pop cs:eip/ss:esp/eflags
- 8. 继续运行用户态进程A

华中科技大学.苏曙光老师.《操作系统原理》MOOC课程组版权所有