操作系统Lab6实验报告

2012011350 计24 周琳钧

练习0：填写已有实验

1. 实验过程：本实验是在实现Lab5用户进程切换的基础上进一步扩展调度器，将原来的FIFO算法扩展为现代计算机常用的Round Robin算法和Stride Scheduling算法。这一部分练习是在原有基础的代码上略作修改使其更贴近于框架。主要修改两个部分，第一部分是在proc.c中，由于proc\_struct新引入了6个变量我们要将其初始化，第二部分是在中断处理程序trap.c中，我们对时钟中断进行了进一步处理，即每次tick都调用一次run\_timer\_list()函数，该函数用于进行进程的调度。

练习1：使用Round Robin调度算法（RR）

1. 实验原理：RR算法的原理是时间片轮转，即给每个进程分配一个固定大小的时间片，维护一个进程队列，每当该进程时间片用完后，就暂停该进程执行队列中的下一进程，并将当前进程重新推入该队列
2. 思考题：
3. 用函数指针的目的是为了实现调度算法的可扩展性，考虑到调度算法的框架不会发生变化，只是调度算法会有改进空间。因此当我们修改调度算法时只需修改五个关键函数：init、enqueue、dequeue、pick\_next、proc\_tick函数即可，对应的五个函数具体实现在default\_sched.c中，并通过sched\_class\_default \_sched\_class建立一一对应关系。

下面我们来说一下ucore的调度器运行过程。每当一个时钟tick到来时，会产生一个时钟中断，然后调用run\_timer\_list()函数，这个函数会记录当前进程还有多少时间可用，然后每次调用减1。当时间用完后，如果下一进程不是RUNNABLE状态，会调用wake\_up()函数将其推入等待队列，否则无需改动，然后将当前进程的resch值赋为1，执行schedule()，schedule()的执行步骤是：将当前进程的resch重新赋为0，将其推入等待队列尾部，并将头部进程取出执行，整个进程调度完毕。

1. 多级反馈队列调度算法描述：进程在进入待调度的队列等待时，首先进入[优先级](http://baike.baidu.com/subview/1035789/1035789.htm)最高的Q1等待。首先调度优先级高的队列中的进程。若高优先级中队列中已没有调度的进程，则调度次[优先级队列](http://baike.baidu.com/view/2539614.htm)中的进程。例如：Q1,Q2,Q3三个队列，只有在Q1中没有进程等待时才去调度Q2，同理，只有Q1,Q2都为空时才会去调度Q3。对于同一个队列中的各个进程，按照[时间片轮转](http://baike.baidu.com/view/399160.htm)法调度。比如Q1队列的[时间片](http://baike.baidu.com/subview/359518/359518.htm" \t "_blank)为N，那么Q1中的作业在经历了N个时间片后若还没有完成，则进入Q2队列等待，若Q2的时间片用完后作业还不能完成，一直进入下一级队列，直至完成。在低[优先级](http://baike.baidu.com/subview/1035789/1035789.htm)的队列中的进程在运行时，又有新到达的作业，那么在运行完这个时间片后，CPU马上分配给新到达的作业（[抢占式](http://baike.baidu.com/view/1448318.htm" \t "_blank)）。因此实现这个系统首先要定义多个队列并且每个队列按优先级分配给不同的时间片，其余基本和单队列一样，这里有可能出现时间片到了之后，将一个队列中的进程dequeue出来然后enqueue到另一个进程的情况，然后pick\_next的执行过程是按优先级取下一个要运行的进程，只有当某队列中为空才会执行下一优先级的队列的进程。

练习2：实现Stride Scheduling算法

1. 实验原理：stride scheduling（以下简称SS）比RR算法好在它能根据进程优先级的不同动态分配时间片。而且与优先级呈反比（假设1为最高优先级，则其分配时间更多），方案是设定一个stride参数和pass参数，P.pass = BIG\_STRIDE / P.priority，其中BIG\_STRIDE是预先设定好的常数，只需小于等于0x7FFFFFFF即可保证程序运行正常，然后每次选取stride最小的进程执行，并将该进程stride加上pass值即可，循环这个步骤即可完成调度
2. 实现伪代码：

init(rq):

– Initialize rq->run\_list

– Set rq->lab6\_run\_pool to NULL

– Set rq->proc\_num to 0

enqueue(rq, proc)

– Initialize proc->time\_slice

– Insert proc->lab6\_run\_pool into rq->lab6\_run\_pool

– rq->proc\_num ++

dequeue(rq, proc)

– Remove proc->lab6\_run\_pool from rq->lab6\_run\_pool

– rq->proc\_num --

pick\_next(rq)

– If rq->lab6\_run\_pool == NULL, return NULL

– Find the proc corresponding to the pointer rq->lab6\_run\_pool

– proc->lab6\_stride += BIG\_STRIDE / proc->lab6\_priority

– Return proc

proc\_tick(rq, proc):

– If proc->time\_slice > 0, proc->time\_slice --

– If proc->time\_slice == 0, set the flag proc->need\_resched

1. 实现代码：见程序