**操作系统实验报告3**

171491221 姜世廷

**练习1**: **使用 Round Robin 调度算法（不需要编码）**

**完成练习0后，建议大家比较一下（可用kdiff3等文件比较软件）个人完成的lab5和练习0完成后的刚修改的lab6之间的区别，分析了解lab6采用RR调度算法后的执行过程。执行make grade，大部分测试用例应该通过。但执行priority.c应该过不去。**

**请在实验报告中完成：**

**请理解并分析sched\_calss中各个函数指针的用法，并接合Round Robin 调度算法描**

**ucore的调度执行过程。**

在rr调度中，系统将所有的就绪进程按先来先服务的原则排成一个队列，每次调度时，把CPU分配给队首进程，并令其执行一个时间片。时间片的大小从几ms到几百ms。当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序便据此信号来停止该进程的执行，并将它送往就绪队列的末尾；然后，再把处理机分配给就绪队列中新的队首进程，同时也让它执行一个时间片。这样就可以保证就绪队列中的所有进程在一给定的时间内均能获得一时间片的处理机执行时间。换言之，系统能在给定的时间内响应所有用户的请求。

Make grade后：

可见，执行priority.c有错误。

RR\_init函数利用指针为零完成了对进程队列的初始化

把进程的进程控制块指针放入到rq末尾，如果进程控制块的时间片为0，则需要把它重置为max\_time\_slice。再依次调整rq和rq的进程数目加一。

把rq的队列元素删除，并将proc\_num减一。

选取rq中的队头队列元素，并把队列元素转换成进程控制块指针。

每一次时间片到时，当前执行进程的时间片time\_slice便减一。如果time\_slice降到零，则设置此进程成员变量need\_resched标识为1

**练习2: 实现 Stride Scheduling 调度算法（需要编码） 首先需要换掉RR调度器的实现，即用default\_sched\_stride\_c覆盖default\_sched.c。然后根据 此文件和后续文档对Stride度器的相关描述，完成Stride调度算法的实现。**

1、比较器定义

/\* You should define the BigStride constant here\*/

/\* LAB6: YOUR CODE \*/

#define BIG\_STRIDE 0x7FFFFFFF /\* 定义一个大整数处以优先级 \*/

/\* The compare function for two skew\_heap\_node\_t's and the

\* corresponding procs\*/

static int

proc\_stride\_comp\_f(void \*a, void \*b)

{

struct proc\_struct \*p = le2proc(a, lab6\_run\_pool);

struct proc\_struct \*q = le2proc(b, lab6\_run\_pool);

int32\_t c = p->lab6\_stride - q->lab6\_stride;//步数相减，通过正负比较大小关系

if (c > 0) return 1;

else if (c == 0) return 0;

else return -1;

}

2、运行队列初始化

/\*

\* stride\_init initializes the run-queue rq with correct assignment for

\* member variables, including:

\*

\* - run\_list: should be a empty list after initialization.

\* - lab6\_run\_pool: NULL

\* - proc\_num: 0

\* - max\_time\_slice: no need here, the variable would be assigned by the caller.

\*

\* hint: see proj13.1/libs/list.h for routines of the list structures.

\*/

static void

stride\_init(struct run\_queue \*rq) {

/\* LAB6: YOUR CODE \*/

list\_init(&(rq->run\_list));//初始化调度器类的信息

rq->lab6\_run\_pool = NULL;//初始化当前的运行队列为一个空的容器结构。

rq->proc\_num = 0;//设置rq->proc\_num为 0

}

3、入队操作

/\*

\* stride\_enqueue inserts the process ``proc'' into the run-queue

\* ``rq''. The procedure should verify/initialize the relevant members

\* of ``proc'', and then put the ``lab6\_run\_pool'' node into the

\* queue(since we use priority queue here). The procedure should also

\* update the meta date in ``rq'' structure.

\*

\* proc->time\_slice denotes the time slices allocation for the

\* process, which should set to rq->max\_time\_slice.

\*

\* hint: see proj13.1/libs/skew\_heap.h for routines of the priority

\* queue structures.

\*/

static void

stride\_enqueue(struct run\_queue \*rq, struct proc\_struct \*proc) {

/\* LAB6: YOUR CODE \*/

#if USE\_SKEW\_HEAP

rq->lab6\_run\_pool = //在使用优先队列的实现中表示当前优先队列的头元素

skew\_heap\_insert(rq->lab6\_run\_pool, &(proc->lab6\_run\_pool), proc\_stride\_comp\_f);//比较队头元素与当前进程的步数大小，选择步数最小的运行

#else

assert(list\_empty(&(proc->run\_link)));

list\_add\_before(&(rq->run\_list), &(proc->run\_link));//将 proc插入放入运行队列中去

#endif

if (proc->time\_slice == 0 || proc->time\_slice > rq->max\_time\_slice) {//初始化时间片

proc->time\_slice = rq->max\_time\_slice;

}

proc->rq = rq;

rq->proc\_num ++;

}

4、出队操作

/\*

\* stride\_dequeue removes the process ``proc'' from the run-queue

\* ``rq'', the operation would be finished by the skew\_heap\_remove

\* operations. Remember to update the ``rq'' structure.

\*

\* hint: see proj13.1/libs/skew\_heap.h for routines of the priority

\* queue structures.

\*/

static void

stride\_dequeue(struct run\_queue \*rq, struct proc\_struct \*proc) {

/\* LAB6: YOUR CODE \*/

#if USE\_SKEW\_HEAP

rq->lab6\_run\_pool =

skew\_heap\_remove(rq->lab6\_run\_pool, &(proc->lab6\_run\_pool), proc\_stride\_comp\_f);// 在斜堆中删除相应元素

#else

assert(!list\_empty(&(proc->run\_link)) && proc->rq == rq);

list\_del\_init(&(proc->run\_link));// 从运行队列中删除相应元素

#endif

rq->proc\_num --;

}

5、选择进程调度

/\*

\* stride\_pick\_next pick the element from the ``run-queue'', with the

\* minimum value of stride, and returns the corresponding process

\* pointer. The process pointer would be calculated by macro le2proc,

\* see proj13.1/kern/process/proc.h for definition. Return NULL if

\* there is no process in the queue.

\*

\* When one proc structure is selected, remember to update the stride

\* property of the proc. (stride += BIG\_STRIDE / priority)

\*

\* hint: see proj13.1/libs/skew\_heap.h for routines of the priority

\* queue structures.

\*/

static struct proc\_struct \*

stride\_pick\_next(struct run\_queue \*rq) {

/\* LAB6: YOUR CODE \*/

#if USE\_SKEW\_HEAP

if (rq->lab6\_run\_pool == NULL) return NULL;

struct proc\_struct \*p = le2proc(rq->lab6\_run\_pool, lab6\_run\_pool); //找到相应指针指向rq->lab6\_run\_pool

#else

list\_entry\_t \*le = list\_next(&(rq->run\_list));

if (le == &rq->run\_list)

return NULL;

struct proc\_struct \*p = le2proc(le, run\_link);

le = list\_next(le);

while (le != &rq->run\_list)

{

struct proc\_struct \*q = le2proc(le, run\_link);

if ((int32\_t)(p->lab6\_stride - q->lab6\_stride) > 0)

p = q;

le = list\_next(le);

}

#endif

if (p->lab6\_priority == 0)//优先级设置

p->lab6\_stride += BIG\_STRIDE;//步长为0则设置为最大步长保持相减的有效性

else p->lab6\_stride += BIG\_STRIDE / p->lab6\_priority;//步长设置为优先级的倒数

return p;

}

6、时间片部分同RR算法思想。

/\*

\* stride\_proc\_tick works with the tick event of current process. You

\* should check whether the time slices for current process is

\* exhausted and update the proc struct ``proc''. proc->time\_slice

\* denotes the time slices left for current

\* process. proc->need\_resched is the flag variable for process

\* switching.

\*/

static void

stride\_proc\_tick(struct run\_queue \*rq, struct proc\_struct \*proc) {

/\* LAB6: YOUR CODE \*/

if (proc->time\_slice > 0) {

proc->time\_slice --;

}

if (proc->time\_slice == 0) {

proc->need\_resched = 1;

}

}

7、定义一个c语言类的实现，提供调度算法的切换接口

struct sched\_class default\_sched\_class = {

.name = "stride\_scheduler",

.init = stride\_init,

.enqueue = stride\_enqueue,

.dequeue = stride\_dequeue,

.pick\_next = stride\_pick\_next,

.proc\_tick = stride\_proc\_tick,

};

实验结果：

