处理机调度

注：这个算法我写不出来，是在网上百度的，我在努力看懂它。

**一．该算法的 基本思想可以考虑如下：**

1. 为每个runnable的进程设置一个当前状态stride，表示该进程当前的调度权。另外定 义其对应的pass值，表示对应进程在调度后，stride 需要进行的累加值。

2. 每次需要调度时，从当前 runnable 态的进程中选择stride最小的进程调度。

3. 对于获得调度的进程P，将对应的stride加上其对应的步长pass（只与进程的优先权 有关系）。

4. 在一段固定的时间之后，回到 2.步骤，重新调度当前stride最小的进程。

**二．具体内容：**

**初始化函数stride\_init**

static void

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

list\_init(&(rq->run\_list));

rq->lab6\_run\_pool = NULL;

rq->proc\_num = 0;

}

**入队函数stride\_enqueue**

static void

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

#if USE\_SKEW\_HEAP

rq->lab6\_run\_pool = skew\_heap\_insert(rq->lab6\_run\_pool, &(proc->lab6\_run\_pool), proc\_stride\_comp\_f);

#else

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

list\_add\_before(&(rq->run\_list), &(proc->run\_link));

#endif

if (proc->time\_slice == 0 || proc->time\_slice > rq->max\_time\_slice) {

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

}

proc->rq = rq;

rq->proc\_num ++;

}

**出队函数stride\_dequeue**

static void

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

#if USE\_SKEW\_HEAP

rq->lab6\_run\_pool =skew\_heap\_remove(rq->lab6\_run\_pool, &(proc->lab6\_run\_pool), proc\_stride\_comp\_f);

#else

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

list\_del\_init(&(proc->run\_link));

#endif

rq->proc\_num --;

}

**调度函数stride\_pick\_next**

static struct proc\_struct \*

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

#if USE\_SKEW\_HEAP

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

struct proc\_struct \*p = le2proc(rq->lab6\_run\_pool, 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)

p->lab6\_stride += BIG\_STRIDE;

else p->lab6\_stride += BIG\_STRIDE / p->lab6\_priority;

return p;

}

**优先队列比较函数proc\_stride\_comp\_f**

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;

if (c > 0) return 1;

else if (c == 0) return 0;

else return -1;

}