**实现 Stride Scheduling 调度算法**

static int

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

{

//通过进程控制块指针取得进程a

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

//通过进程控制块指针取得进程b

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

//步数相减，通过正负比较大小关系

int32\_t c = p->lab6\_stride - q->lab6\_stride;

if (c > 0) return 1;

else if (c == 0) return 0;

else return -1;

}

static void

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

/\* LAB6: YOUR CODE \*/

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

rq->lab6\_run\_pool = NULL; //初始化当前进程运行队列为空

rq->proc\_num = 0; //设置运行队列为空

}

static void

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

/\* LAB6: YOUR CODE \*/

#if USE\_SKEW\_HEAP

//将进程加入就绪队列

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 ++; //进程数加一

}

static inline skew\_heap\_entry\_t \*

skew\_heap\_insert(skew\_heap\_entry\_t \*a, skew\_heap\_entry\_t \*b,

compare\_f comp)

{

skew\_heap\_init(b); //初始化进程b

return skew\_heap\_merge(a, b, comp);//返回a与b进程结合的结果

}

static inline void

skew\_heap\_init(skew\_heap\_entry\_t \*a)

{

a->left = a->right = a->parent = NULL; *//初始化相关指针*

}

static inline skew\_heap\_entry\_t \*

skew\_heap\_merge(skew\_heap\_entry\_t \*a, skew\_heap\_entry\_t \*b,

compare\_f comp)

{

if (a == NULL) return b;

else if (b == NULL) return a;

skew\_heap\_entry\_t \*l, \*r;

if (comp(a, b) == -1) //a进程的步长小于b进程

{

r = a->left; //a的左指针为r

l = skew\_heap\_merge(a->right, b, comp);

a->left = l;

a->right = r;

if (l) l->parent = a;

return a;

}

else

{

r = b->left;

l = skew\_heap\_merge(a, b->right, comp);

b->left = l;

b->right = r;

if (l) l->parent = b;

return b;

}

}

static void

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

/\* LAB6: YOUR CODE \*/

rq->lab6\_run\_pool =

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

rq->proc\_num --;

}

static inline skew\_heap\_entry\_t \*

skew\_heap\_remove(skew\_heap\_entry\_t \*a, skew\_heap\_entry\_t \*b,

compare\_f comp)

{

skew\_heap\_entry\_t \*p = b->parent;

skew\_heap\_entry\_t \*rep = skew\_heap\_merge(b->left, b->right, comp);

if (rep) rep->parent = p;

if (p)

{

if (p->left == b)

p->left = rep;

else p->right = rep;

return a;

}

else return rep;

}

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);

#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) //优先级为0

p->lab6\_stride += BIG\_STRIDE; //步长设置为最大值

//步长设置为优先级的倒数

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

return p;

}

static void

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

/\* LAB6: YOUR CODE \*/

if (proc->time\_slice > 0) { //到达时间片

proc->time\_slice --; //执行进程的时间片time\_slice减一

}

if (proc->time\_slice == 0) { //时间片为0

//设置此进程成员变量need\_resched标识为1,进程需要调度

proc->need\_resched = 1;

}

}

首先是初始化函数stride\_init。 开始初始化运行队列，并初始化当前的运行队。然后是入队函数stride\_enqueue，这里函数主要是初始化刚进入运行队列的进程 proc 的stride属性，然后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。最后初始化时间片，然后将运行队列进程数目加一。然后是出队函数stride\_dequeue，即完成将一个进程从队列中移除的功能，这里使用了优先队列。最后运行队列数目减一。接下来就是进程的调度函数stride\_pick\_next，观察代码，它的核心是先扫描整个运行队列，返回其中stride值最小的对应进程，然后更新对应进程的stride值，将步长设置为优先级的倒数，如果为0则设置为最大的步长。最后是时间片函数stride\_proc\_tick，主要工作是检测当前进程是否已用完分配的时间片。如果时间片用完,应该正确设置进程结构的相关标记来引起进程切换。