OS_LAB1 PINTOS_THREAD

11510351 黄哲 2018.05.02

---

1. TASK1 Scheduler based on time slice

1.1 分析

本质上是要对位于`device`文件夹中的 `timer.c` 中的函数`timer_sleep` 进行修改，使其从之前的**忙等待**更改成为**闲等待。**
之前函数核心代码如下图所示：

```
while (timer_elapsed (start) < ticks)
  thread_yield ();
```

在获取了当前时间后，通过 while 循环执行 `thread_yield` 操作，直到苏醒。而在函数`thread_yield`中，是通过不停的进行放置当前线程到`ready_list`中，再进行 `schedule ()`实现的，整个过程需要占用 CPU 进行操作，因而是 busy waiting 的。

1.2 思路

1. 考虑到对于休眠有时间限制，则在对应时间到来时唤醒即可。
2. 当调用`timer_sleep` 时，设置对应的睡眠时间，把线程放入`wait_list`中，并阻塞线程。
3. 考虑到系统有时间中断，则在中断时对`wait_list`中对每个线程进行判断，减少睡眠时间。
   若时间已到，则移除队列，设置状态为等待。

1.3 具体操作

1. 考虑进行修改，在`timer_sleep`中：
   - 对输入进行判断，若小于零则直接返回，也和 task 中的`negative`相对应。
   - 直接调用了写在 `thread` 中的函数`thread_waiting`进行后续操作。

```
void timer_sleep(int64_t ticks){
  if (ticks <= 0) return;
  ASSERT(intr_get_level() == INTR_ON);
  thread_waiting(ticks);
}
```

2. 更改`thread.h`中线程的结构体，加入两个元素
   - 分别是之后加入`wait_list`中所需用到的`elem`和用来计算休眠的 tick 数。

```
struct thread{
  struct list_elem waitelem;
  int waiting_ticks;
  }
```

3. 在`thread.c`中，加入`wait_list`的定义并对其进行初始化。
4. 在`thread_waiting`中：
   先屏蔽中断，后设置当前线程需休眠的时间，加入`wait_list` ，并阻塞线程即可。

```
void thread_waiting(int64_t ticks){
  struct thread *cur = thread_current();

  ASSERT(!intr_context());
  enum intr_level old_level = intr_disable();

  cur->waiting_ticks = ticks;
  list_insert_ordered(&wait_list, &cur->waitelem, (list_less_func *)&thread_waiting_ticks, NULL);
  thread_block();

  intr_set_level(old_level);
}
```

---

5. 在时间中断函数`timer_interrupt`中加入函数`thread_detated`用于实现检测+唤醒。
    - 本质上是一个遍历。
    - 遍历了`wait_list`，对于其中所有线程都减少了休眠时间，并唤醒已经到时的线程。

```
void thread_detated(void){
  struct list_elem *e;
  ASSERT(intr_get_level() == INTR_OFF);
  if (!list_empty(&wait_list)){
    for (e = list_begin(&wait_list); e != list_end(&wait_list); e = list_next(e)){
      struct thread *t = list_entry(e, struct thread, waitelem);
      t->waiting_ticks--;
      if (t->waiting_ticks == 0){
        list_remove(&t->waitelem);
        thread_unblock(t);
      }
    }
  }
}
```

1.4 结果

ticks	alarm-single	alarm-multiple	alarm-single-change	alarm-multiple-change
idle ticks	0	0	254	581
kernel ticks	361	894	110	316
all ticks	361	894	361	894

```
Executing 'alarm-single':
(alarm-single) begin
(alarm-single) Creating 5 threads to sleep 1 times each.
(alarm-single) Thread 0 sleeps 10 ticks each time,
(alarm-single) thread 1 sleeps 20 ticks each time, and so on.
(alarm-single) If successful, product of iteration count and
(alarm-single) sleep duration will appear in nondescending order.
(alarm-single) thread 0: duration=10, iteration=1, product=10
(alarm-single) thread 1: duration=20, iteration=1, product=20
(alarm-single) thread 2: duration=30, iteration=1, product=30
(alarm-single) thread 3: duration=40, iteration=1, product=40
(alarm-single) thread 4: duration=50, iteration=1, product=50
(alarm-single) end
Execution of 'alarm-single' complete.
Timer: 361 ticks
Thread: 254 idle ticks, 110 kernel ticks, 0 user ticks
Console: 984 characters output
Keyboard: 0 keys pressed
```

```
Executing 'alarm-multiple':
(alarm-multiple) begin
(alarm-multiple) Creating 5 threads to sleep 7 times each.
(alarm-multiple) Thread 0 sleeps 10 ticks each time,
(alarm-multiple) thread 1 sleeps 20 ticks each time, and so on.
(alarm-multiple) If successful, product of iteration count and
(alarm-multiple) sleep duration will appear in nondescending order.
(alarm-multiple) thread 0: duration=10, iteration=1, product=10
(alarm-multiple) thread 1: duration=20, iteration=1, product=20
(alarm-multiple) thread 0: duration=10, iteration=2, product=20
(alarm-multiple) thread 2: duration=30, iteration=1, product=30
(alarm-multiple) thread 0: duration=10, iteration=3, product=30
(alarm-multiple) thread 3: duration=40, iteration=1, product=40
(alarm-multiple) thread 1: duration=20, iteration=2, product=40
(alarm-multiple) thread 0: duration=10, iteration=4, product=40
(alarm-multiple) thread 4: duration=50, iteration=1, product=50
(alarm-multiple) thread 0: duration=10, iteration=5, product=50
(alarm-multiple) thread 2: duration=30, iteration=2, product=60
(alarm-multiple) thread 1: duration=20, iteration=3, product=60
(alarm-multiple) thread 0: duration=10, iteration=6, product=60
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 1: duration=20, iteration=4, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 894 ticks
Thread: 581 idle ticks, 316 kernel ticks, 0 user ticks
```

2. Round-Robin scheduler

2.1 分析

本质上是实现

1. 根据优先级确定时间片

2. 根据执行情况动态调整线程优先级
3. `Round-Robin` 的线程优先级调度

2.2 思路

1. 时间片的确定在初始化线程的时候执行即可。
2. 在时间中断时，只有在时间片都被耗尽的情况下才切换线程，同时 **recharge** 时间片，调整优先级。
3. 实现 RR 的优先队列则对所有进队操作进行修改，插入时都根据优先级进行插入即可。

2.3 具体操作

1. 更改 `thread.h` 中线程的结构体，加入三个变量
    - `time_slice` ：用来决定当前线程的时间片，一经确定不再更改
    - `rest` ：初始即为 `time_slice` ，随时间轮转消耗每次减一
    - `origin_priority` ：用于存放初始优先级

```
struct thread{
  int time_slice;
  int rest;
  int origin_priority;
  }
```

2. 更改 `thread.c` 中函数 `thread_tick` ，加入以下内容，同时注释掉本来用来调度的代码
    - 判断当前 `rest` 为1，则时间片耗尽，进行**recharge**，并将当前线程优先级减三（最低为零），之后激活调度函数。
    - 若 `rest` 不为1，则直接消耗时间片，不执行其他操作。

```
if (strcmp(t->name, "main") != 0){
    if (t->rest == 1){
        t->rest = t->time_slice;
        if (t->priority >= 3){
            t->priority = t->priority - 3;
            t->actual_priority = t->actual_priority - 3;
        }else
            t->priority = 0;
        intr_yield_on_return();
    }else
```

```
        t->rest--;
}
```

3. 更改所有执行入队操作的代码，将其转变为按优先级入队。此处以 `thread_unblock` 为例：
    - 将入队操作变更为：

```
list_insert_ordered(&ready_list, &t->elem, (list_less_func *)&compare_thread_pri, NULL);

//其中比较函数为：
bool compare_thread_pri(const struct list_elem *a, const struct list_elem *b, void *aux UNUSED){
  struct thread *sa = list_entry(a, struct thread, elem);
  struct thread *sb = list_entry(b, struct thread, elem);
  return sa->priority > sb->priority;
}
```

2.4 结果

```
(alarm-priority) begin

begin
(30,30) (30,30) (30,30) (30,30) (29,29) (29,29) (29,29) (28,28) (28,28) (27,27) (27,27) (27,27) (27,27) (27,27) (27,27) (27,27) (27,27) (30,27) (3
0,27) (30,27) (30,27) (26,26) (26,26) (26,26) (26,26) (26,26) (26,26) (26,26) (29,26) (29,26) (29,26) (25,25) (25,25) (25,25) (25,25) (25,25) (25,
25) (28,25) (28,25) (24,24) (24,24) (24,24) (24,24) (24,24) (27,24) (27,24) (27,24) (27,24) (27,24) (27,24) (27,24) (27,24) (30,24) (30,24) (30,24
) (30,24) (23,23) (23,23) (23,23) (23,23) (26,23) (26,23) (26,23) (26,23) (26,23) (26,23) (26,23) (29,23) (29,23) (29,23) (22,22) (22,22) (22,22)
(25,22) (25,22) (25,22) (25,22) (25,22) (25,22) (28,22) (28,22) (21,21) (21,21) (24,21) (24,21) (24,21) (24,21) (24,21) (27,21) (27,21) (27,21) (2
7,21) (27,21) (27,21) (27,21) (27,21) (30,21) (30,21) (30,21) (30,21) (23,20) (23,20) (23,20) (23,20) (26,20) (26,20) (26,20) (26,20) (26,20) (26,
20) (26,20) (29,20) (29,20) (29,20) (22,19) (22,19) (22,19) (25,19) (25,19) (25,19) (25,19) (25,19) (25,19) (28,19) (28,19) (21,18) (21,18) (24,18
) (24,18) (24,18) (24,18) (24,18) (27,18) (27,18) (27,18) (27,18) (27,18) (27,18) (27,18) (27,18) (30,18) (30,18) (30,18) (30,18) (23,17) (23,17)
(23,17) (23,17) (26,17) (26,17) (26,17) (26,17) (26,17) (26,17) (26,17) (29,17) (29,17) (29,17) (22,16) (22,16) (22,16) (25,16) (25,16) (25,16) (2
5,16) (25,16) (25,16) (28,16) (28,16) (21,15) (21,15) (24,15) (24,15) (24,15) (24,15) (24,15) (27,15) (27,15) (27,15) (27,15) (27,15) (27,15) (30,
15) (30,15) (30,15) (30,15) (23,14) (23,14) (23,14) (23,14) (26,14) (26,14) (26,14) (26,14) (26,14) (26,14) (26,14) (29,14) (29,14) (29,14) (22,13
) (22,13) (22,13) (25,13) (25,13) (25,13) (25,13) (25,13) (25,13) (28,13) (28,13) (21,12) (21,12) (24,12) (24,12) (24,12) (24,12) (24,12) (30,12)
(30,12) (30,12) (30,12) (23,11) (23,11) (23,11) (23,11) (26,11) (26,11) (26,11) (26,11) (26,11) (26,11) (29,11) (29,11) (29,11) (22,10) (22,10) (2
2,10) (25,10) (25,10) (25,10) (25,10) (25,10) (25,10) (28,10) (28,10) (21,9) (21,9) (24,9) (24,9) (24,9) (24,9) (24,9) (30,9) (30,9) (30,9) (30,9)
 (23,8) (23,8) (23,8) (23,8) (29,8) (29,8) (29,8) (22,7) (22,7) (22,7) (28,7) (28,7) (21,6) (21,6) (24,6) (24,6) (24,6) (24,6) (24,6) (30,6) (30,6
) (30,6) (30,6) (23,5) (23,5) (23,5) (23,5) (29,5) (29,5) (29,5) (22,4) (22,4) (22,4) (28,4) (28,4) (21,3) (21,3) (24,3) (24,3) (30,3) (30,3) (30,
3) (30,3) (23,2) (23,2) (23,2) (23,2) (29,2) (29,2) (29,2) (22,1) (22,1) (22,1) (28,1) (28,1) (21,0) (21,0) (30,0) (30,0) (30,0) (30,0) (23,0) (23
,0) (23,0) (29,0) (29,0) (29,0) (22,0) (22,0) (22,0) (28,0) (28,0) (21,0) (21,0) (30,0) (30,0) (29,0) (29,0) (29,0) (22,0) (22,0) (22,0) (28,0) (2
8,0) (21,0) (21,0) (29,0) (29,0) (29,0) (22,0) (22,0) (22,0) (28,0) (28,0) (21,0) (21,0) (29,0) (29,0) (22,0) (22,0) (22,0) (28,0) (28,0) (21,0) (
21,0) (22,0) (22,0) (22,0) (28,0) (28,0) (21,0) (21,0) (22,0) (22,0) (28,0) (28,0) (21,0) (21,0) (28,0) (28,0) (21,0) (21,0) (28,0) (21,0) (21,0)
(21,0) (21,0) (21,0) (21,0)
end
(alarm-priority) end
```

3. Priority scheduler based on time slice

3.1 分析

本质上是实现优先级捐赠的问题，其中会遇到很多特殊情况，如递归提高优先级，线程释放后确定新优先级等。

3.2 思路

1. 一开始考虑的是通过 `lock` 确定。即 `lock` 的结构体中如下：

- 添加了元素：`threads`，用于存放所有需要获取这个锁的线程
- 而元素`holder`为当前得到这个锁的线程
- 但后面考虑这样会导致在 **递归捐赠** 和 **多锁捐赠** 的时候难以实现。

```
struct lock{
    struct thread *holder;
    struct semaphore semaphore;
    struct list_elem elem;
    struct list threads;
};
```

2. 查了资料，改变思路用比较简单的方式实现。
   - 每次需求锁和释放锁时，都会对所有的 **线程-锁** 关系更新优先级。即不再保存历史的捐赠记录，缺点是计算量比较大。
   - 在设置优先级时，不对提高的优先级进行调整。

3.3 具体操作

1. 在`lock_acquire`中进行调整，加入函数`finding_nesting`
   - 递归进行捐赠

```
void finding_nesting(struct lock *lock){
  struct thread *cur = thread_current();
  cur->lock = lock;
  struct lock *l = lock;
  int pri = cur->priority;
  if (lock->holder == NULL)
    return;

  while (l != NULL ){
    if (pri < l->mp)
        break;
    l->mp = pri;
    progigation_pri(l->holder);
    l = l->holder->lock;
  }
}
```

2. 更改函数`thread_set_priority`，考虑捐赠期间更改优先级的问题。

- 先进行中断，之后若无锁则直接更新优先级。若有锁且新优先级比旧优先级高也更新优先级。不然则先讲新设置的优先级赋值给`origin_priority`。

```
void thread_set_priority(int new_priority){
  enum intr_level old_level = intr_disable();
  struct thread *cur = thread_current();
  cur->origin_priority = new_priority;
  if (list_empty(&cur->locks) || new_priority > cur->prio
rity)
    cur->priority = cur->origin_priority;
  thread_yield();
  intr_set_level(old_level);
}
```

3. 在释放锁`lock_release`的时候也进行相应操作。
   - 先屏蔽中断，之后把锁从线程的持有列表中移除，再更新一次所有线程的优先级。

```
void lock_release(struct lock *lock){
  ASSERT(lock != NULL);
  ASSERT(lock_held_by_current_thread(lock));
  enum intr_level old_level = intr_disable();
  list_remove(&lock->elem);
  lock->holder = NULL;
  update_priority(thread_current());
  sema_up(&lock->semaphore);
  intr_set_level(old_level);
}
```

3.4 结果

在注释了`thread_tick`中部分代码之后，跑`make check`结果。

```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
FAIL tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-lower
FAIL tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
FAIL tests/threads/priority-condvar
pass tests/threads/priority-donate-chain
FAIL tests/threads/mlfqs-load-1
FAIL tests/threads/mlfqs-load-60
FAIL tests/threads/mlfqs-load-avg
FAIL tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
FAIL tests/threads/mlfqs-nice-2
FAIL tests/threads/mlfqs-nice-10
FAIL tests/threads/mlfqs-block
10 of 27 tests failed.
```