






# OS\_review2-1

 Assign	
 Property	
 tag	homework
 姓名	
 学号	

## 3.关于调度

- 由于本次的任务不存在进程之间的互相抢占，故调度只发生在如下两种情况
  - 内核线程主动调度
  - 进程自己放弃
- 由于现阶段动用调度时，进程都是处在running态，那么在进行调度时，直接将之转变为ready态，放在ready queue里即可
  - 把下一个进程从ready queue里移出来
  - 改变进程状态
  - 换掉正在运行的线程
  - 把上一个线程放到ready queue里
  - 上下文切换
    - 保存上文
    - 恢复上文

```
void do_scheduler(void)
{
    *sch_timer = get_ticks();
    if (!list_empty(&ready_queue)){
        pcb_t *next_running = dequeue(&ready_queue);
        pcb_t *temp = current_running;
        if(current_running->status == TASK_RUNNING){
            enqueue(&ready_queue, current_running);
            current_running->status = TASK_READY;
        }
        next_running->status = TASK_RUNNING;
        current_running->add_tick = get_ticks();
    }
```

```

        current_running = next_running;
        process_id = current_running->pid;
        switch_to(temp, next_running);
    }
}

```

## 4.关于锁

- 向 OS 申请锁
  - 没被占用则直接访问，访问结束后释放锁，进程进入ready queue
  - 被占用则加入该锁的阻塞队列，等待锁被释放后访问，进程进入ready queue

```

void do_mutex_lock_init(mutex_lock_t *lock)
{
    init_head(&lock->block_queue);
    lock->lock.status = UNLOCKED;
}

void do_mutex_lock_acquire(mutex_lock_t *lock)
{
    if(lock->lock.status == LOCKED)
        do_block(current_running, &lock->block_queue);
    else
        lock->lock.status = LOCKED;
}

void do_mutex_lock_release(mutex_lock_t *lock)
{
    if(list_empty(&lock->block_queue))
        lock->lock.status = UNLOCKED;
    else{
        do_unblock(&lock->block_queue);
    }
}

```