Project2 A Simple Kernel 设计文档（Part I）

中国科学院大学
郑旭舟
2020 年 10 月 18 日

1. 任务启动与 Context Switch 设计流程

1.1.1. PCB 包含的信息

项目	简介
regs_context_t kernel_context, user_context	内核态/用户态上下文信息
uint64_t kernel_stack_top, user_stack_top	内核态/用户态栈顶
void *prev, *next	进程控制块前项/后项指针
queue_t lock_queue	进程持有的锁
uint32_t prior	进程优先级
pid_t pid	进程标识符
task_type_t type	task 的类型（用户态/内核态，进程/线程）
task_status_t status	task 的状态
int cursor_x, cursor_y	光标的坐标

1.1.2. 如何启动一个 task

想要启动一个 task，需要先获得它的入口地址。观察 main.c，可以发现实验中程序的主要行为是：

① PCB 初始化，装载需要运行的 task；

② 使用 do_scheduler()进行进程调度。

在初始化 PCB 的过程中，函数 set_pcb()将 task 的入口地址装入 PCB 的 user_context 中 ra 寄存器对应的位置。main 中运行 do_scheduler()函数时，待运行进程的 PCB 中所存的上下文信息将载入寄存器，ra 寄存器被存入的是 task 的入口地址，而 do_scheduler()函数的下一条汇编是 jr ra，因此 PC 直接跳到 task 的入口地址开始执行。

1.1.3. Context switch 时的寄存器保存

上下文切换的过程中保存了除 at 寄存器和$25,$26 号寄存器之外的 29 个通用寄存器。它们被保存在内存中的进程控制块内，当再次切换到该进程时，它们从相应位置读出上下文内容并加载到寄存器中，从而实现有切换情形下进程的正常运行。

2. Mutex lock 设计流程

2.1.1. 无法获得锁时的处理流程

当使用 do_mutex_lock_aquire(mutex_lock_t *lock)函数来请求锁但无法获得锁时，调用 do_block(queue_t *queue)函数将请求锁的进程挂起到该锁的阻塞队列 lock->blocked 中，直到该锁被释放。

2.1.2. 被阻塞的 task 何时再次执行

被阻塞的 task 在该锁被释放后从阻塞队列中移出并重新进入就绪队列后再次执行。

3. 关键函数功能

3.1. 进程切换相关函数

3.1.1. do_scheduler()函数

do_scheduler()保存当前 current_running 用户态 context 到进程控制块中，调用 scheduler 函数切换到新的 current_running，并将该进程的 PCB 中保存的上下文恢复到寄存器中，然后跳到 ra 开始执行对应进程。

代码如下：

```
NESTED(do_scheduler, 0, ra)
    # save context
    SAVE_CONTEXT(USER)
    # current_running
    jal scheduler
    # restore context
    RESTORE_CONTEXT(USER)
    # jump to entry
    jr ra
END(do_scheduler)
```

3.2. 互斥锁相关函数

3.2.1. 互斥锁的初始化

初始化一个锁时，设置其状态为 UNLOCKED，prev, next 指针设为 NULL，并初始化其对应的阻塞队列。代码如下：

```
// mutex initialization ([*lock])
void do_mutex_lock_init(mutex_lock_t *lock)
{
    lock->status = UNLOCKED;
    lock->prev = NULL;
    lock->next = NULL;
    queue_init(&(lock->blocked));
}
```

3.2.2. 互斥锁的请求

一个进程请求一个互斥锁时：

1. 若该锁已被占用，则将该进程挂起到该锁的阻塞队列，仅当该锁变为 UNLOCKED 状态时，该进程才有机会再次进入就绪队列。
2. 若该锁未被占用，则请求成功，将该锁的状态置为 LOCKED，并将锁加入该进程 PCB 的锁队列中。

代码如下：

```
// mutex acquire ([*lock])
void do_mutex_lock_acquire(mutex_lock_t *lock)
{
    while (lock->status == LOCKED)
    // chk after lock_acquire failed
    {
        do_block(&(lock->blocked));
    }
    // UNLOCKED: success
    lock->status = LOCKED;
    queue_push(&(current_running->lock_queue), (void *)lock);
}
```

3.2.3. 互斥锁的释放：

当需要释放一个互斥锁时，需要依次做这些事：

1. 置锁的状态为 UNLOCKED；
2. 将该锁阻塞队列中的所有进程释放，即加入就绪队列中；
3. 将该锁移出当前进程的锁队列。

代码如下：

```
// mutex release ([*lock])
void do_mutex_lock_release(mutex_lock_t *lock)
{
    lock->status = UNLOCKED;
    do_unblock_all(&(lock->blocked));
    queue_remove(&(current_running->lock_queue), (void *)lock);
}
```

参考文献

无