Project2 A Simple Kernel 设计文档（Part I）

中国科学院大学
[姓名] 李国峰
[日期] 2021 年 1 月 24 日

1. 任务启动与 Context Switch 设计流程

（1） PCB 包含的信息

```
typedef struct pcb
{
    /* register context */
    regs_context_t user_context;
    regs_context_t kernel_context;

    uint32_t user_stack_top;
    uint32_t kernel_stack_top; //$29

    /* previous, next pointer */
    void *prev;
    void *next;
    /* task in which queue */


    /* What tasks are blocked by me, the tasks in this
     * queue need to be unblocked when I do_exit(). */


    /* holding lock */


    /* block related */


    /* priority */

    // name
    char name[32];

    /* process id */
    pid_t pid;

    /* kernel/user thread/process */
    task_type_t type;

    /* BLOCK | READY | RUNNING */
    task_status_t status;

    /* cursor position */
    int cursor_x;
    int cursor_y;

} pcb_t;
```

（2） 如何启动一个 task，包括如何获得 task 的入口地址，启动时需要设置哪些寄存

器等

实现非抢占式调度时，使用 set_pcb()函数，调用参数包括 process_id（全局变量，初始化为 1，每次启动一个 pcb 都自增一次），给定的 pcb 数组（用于给启动的任务的 PCB 分配空间）和任务信息（在 sched1_tasks[]中给出），task 的入口地址就包含在任务信息中，即 task_info 结构体中的 entry_point。启动时需要设置的寄存器是 29 号寄存器和 31 号寄存器，分别存储栈指针和函数调用返回地址。对于初次启动的任务来说，这样做无需执行其他操作就可以通过返回直接跳转到该任务的入口地址开始执行；同时，调用任务结束后也可以根据 31 号寄存器的内容返回到调用者运行。

（3） context switch 时保存了哪些寄存器，保存在内存什么位置，使得进程再切换回来后能正常运行？

```
typedef struct regs_context
{
    /*32 mips general registers*/
    /* 32 * 8B = 256B */
    uint64_t regs[32];

    /*special registers*/
    /* 7 * 8B = 56B */
    uint64_t cp0_status;
    uint64_t cp0_badvaddr;
    uint64_t cp0_cause;
    uint64_t cp0_epc;
    uint64_t hi;
    uint64_t lo;
    uint64_t pc;

} regs_context_t; /* 256 + 56 = 312B */
```

需要保存除$k0 和$k1 外的通用寄存器和一些协处理器寄存器。

2. Mutex lock 设计流程

（1） 无法获得锁时的处理流程

当一个进程申请获得锁时，如果锁是被持有的，那就阻塞这个进程。

```
void do_mutex_lock_acquire(mutex_lock_t *lock)
{
    if(lock->status == LOCKED)
    {
        do_block(&block_queue);
    }
    lock->status = LOCKED;
}
```

（2） 被阻塞的 task 何时再次执行？

当锁被释放时，操作系统从阻塞队列中弹出一个进程来获得这个锁，顺序遵循 FCFS。

```
void do_mutex_lock_release(mutex_lock_t *lock)
{
    if(!queue_is_empty(&block_queue))
    {
        do_unblock_one(&block_queue);
        lock->status = LOCKED;
    }
    else
        lock->status = UNLOCKED;

}
```

3. 关键函数功能

（1） scheduler()调度流程

```
void scheduler(void)
{
    // TODO schedule
    // Modify the current_running pointer.
    pcb_t *next_running;
    //判空，如果为空则current_running指向pcb[0]
    if(queue_is_empty(&ready_queue))
        next_running = &pcb[0];
    else
        next_running = (pcb_t*)queue_dequeue(&ready_queue);

    if(current_running->status != TASK_BLOCKED)//current_running->status = TASK_RUNNING
    {
        current_running->status = TASK_READY;
        if(current_running->pid != 0)
        {
            queue_push(&ready_queue, current_running);
        }
    }

    current_running = next_running;
    current_running->status = TASK_RUNNING;
    return ;
}
```

参考文献

[1] 中国科学院大学操作系统实验手册 Project2-MIPS（2020 秋季学期）