实验2-part2说明文档

陈远腾 2020k8009929041

开发过程：

1.过程简述：

(1) 10.1-5：结合课上所讲内容，仔细阅读并理解了guide_book2 task3-5的内容，同时仔细阅读了riscv手册中关于特权级指令的说明，对相应内容做了笔记以方便后续查找。
(2) 10.6-7周四周五：结合代码梳理了触发中断 --> 陷入中断 --> 结束中断 过程中函数的跳转，及过程中各个函数大致要完成什么工作。
(3) 10.8-9：利用周末时间开始代码设计，完成了task3的设计及debug。
(4) 10.10周一：完成了task4时间中断的设计。
(5) 10.12周三：完成了task5关于创建线程的设计。
(6) 10.13周四：复查代码，找出了一两处之前没有发现的bug（之前并没有导致运行错误）。
(7) 10.16周日：完成了实验2-part2的说明文档。

2.遇到的主要问题bug：

(1) 首先是初始时根本搞不清楚每个函数要干嘛及各个函数间是怎么跳转的，经仔细阅读梳理代码后画出了流程图，后续依照流程图才能顺利完成设计。

(2) PCB初始化时SSTATUS设置错误，因为我们模拟的是第一次进入内核的情形，所以应当将SSTATUS设为SR_SPIE。

(3) 没有在interrupt_helper的末尾将ra置为ret_from_exception的地址，导致中断处理完成后没有跳至ret_from_exception完成context的load，导致没有恢复上下文。

(4) 在SAVE_CONTEXT中sp指针设置错误，没有在SAVE_CONTEXT的末尾将SP恢复为原值或指向内核栈的空白部分，导致后续内核程序改变了内核栈中存储上下文的部分。

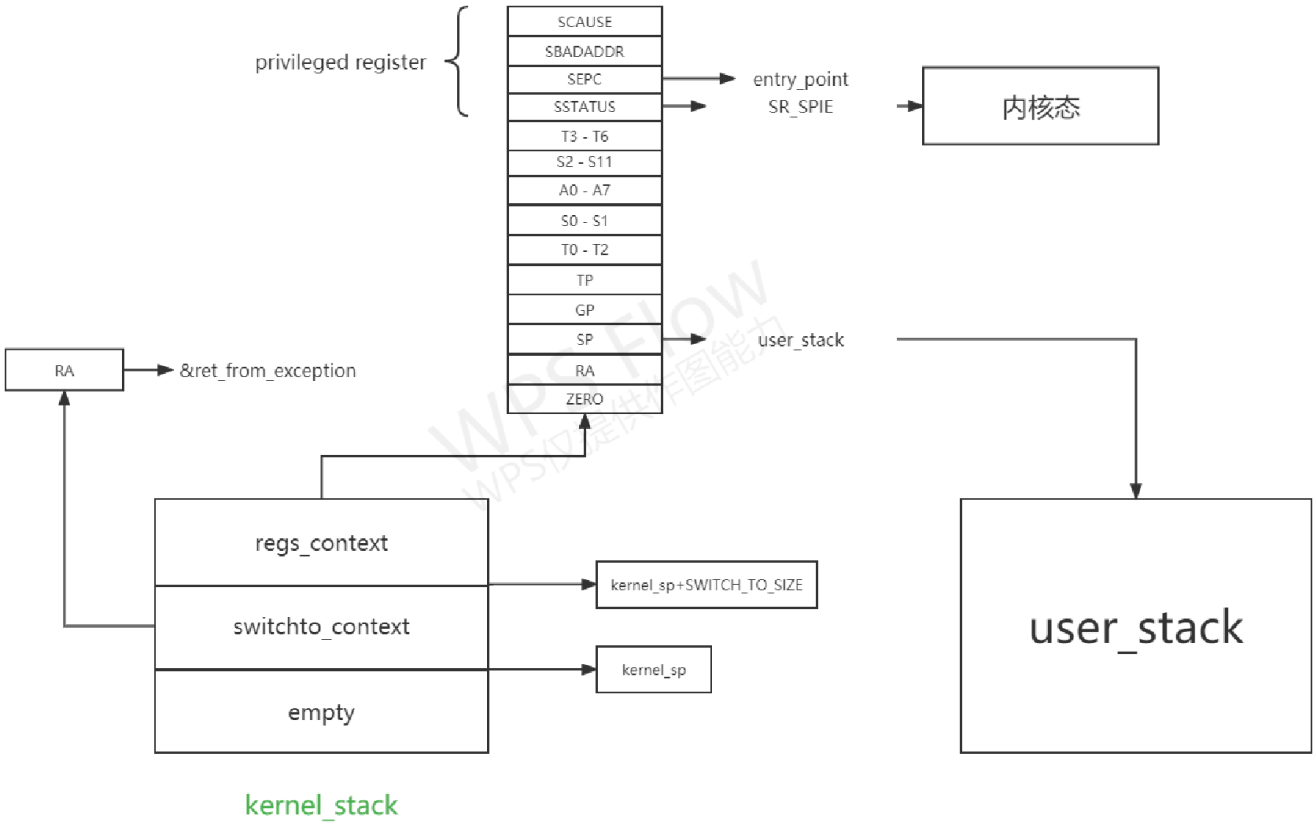
(5) 在task4中，没有在main函数中初始化tp指针，导致在第一次中断过程中SAVE_CONTEXT 和RESTORE_CONTEXT均错误。

(6) 在task5中，在main的syscall表中需要用到新定义的宏THREAD_CREATE，这个宏已经在syscall.h中定义，但在main中始终找不到，后来发现需要在arch/riscv/include/asm/unistd.h中定义这个宏。

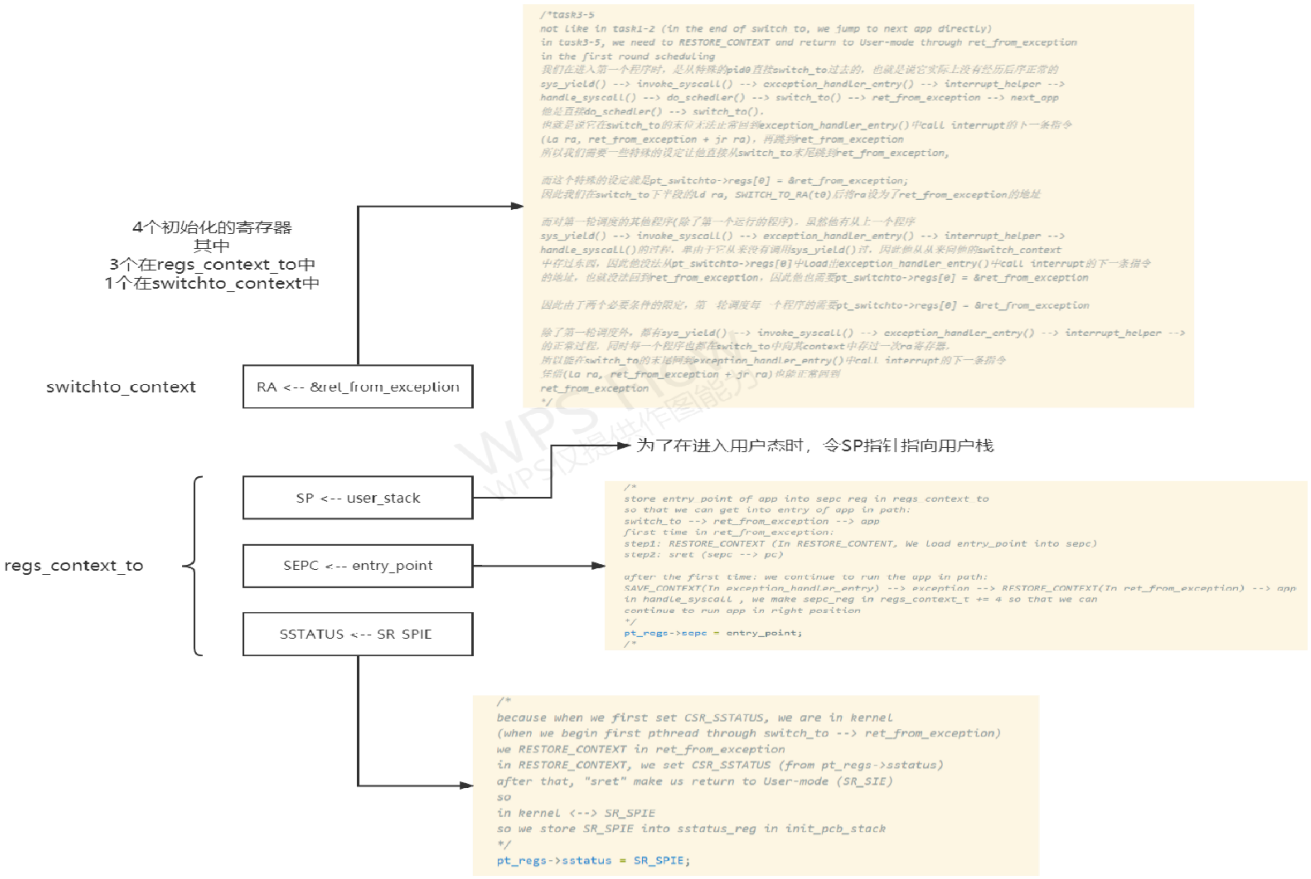
程序设计

task3：

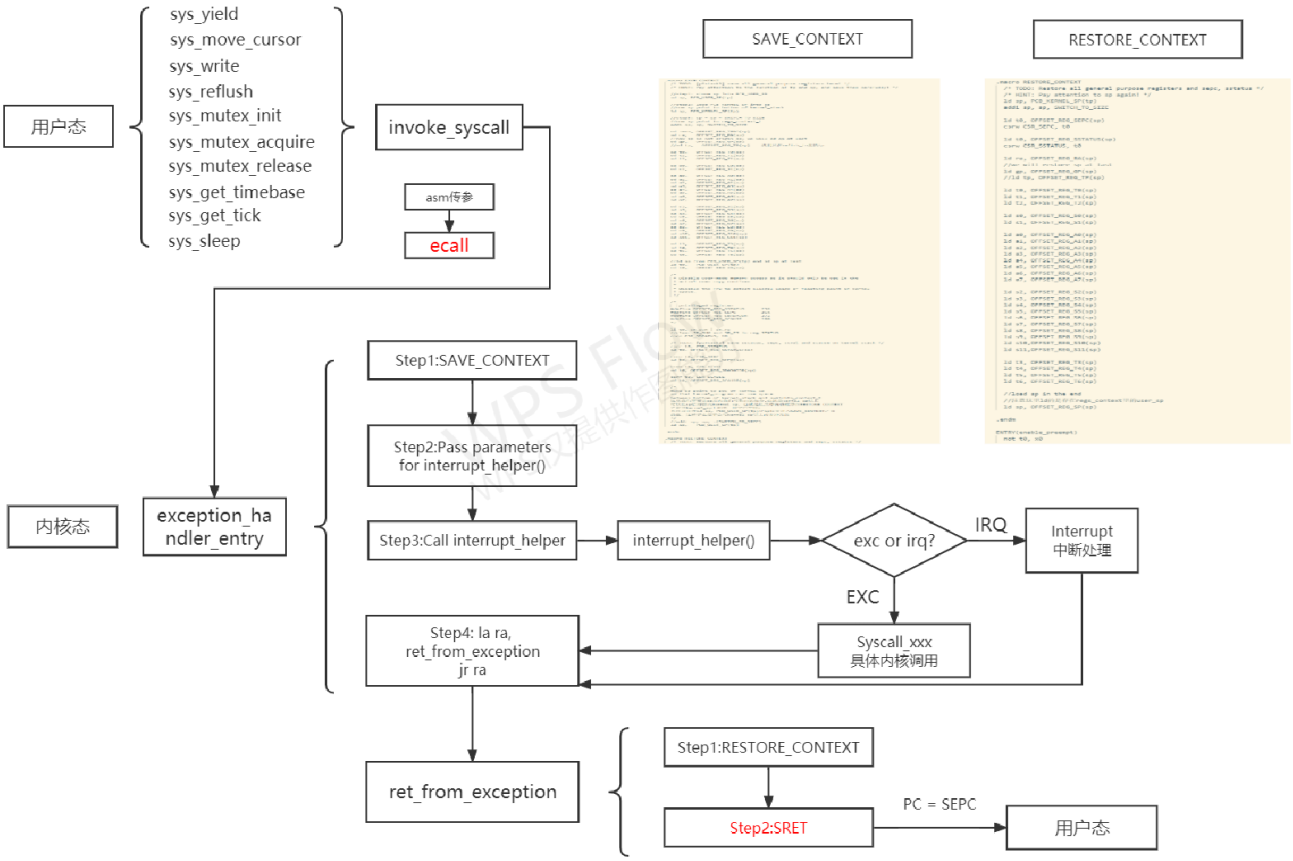
1.PCB中regs_context部分寄存器的初始化：



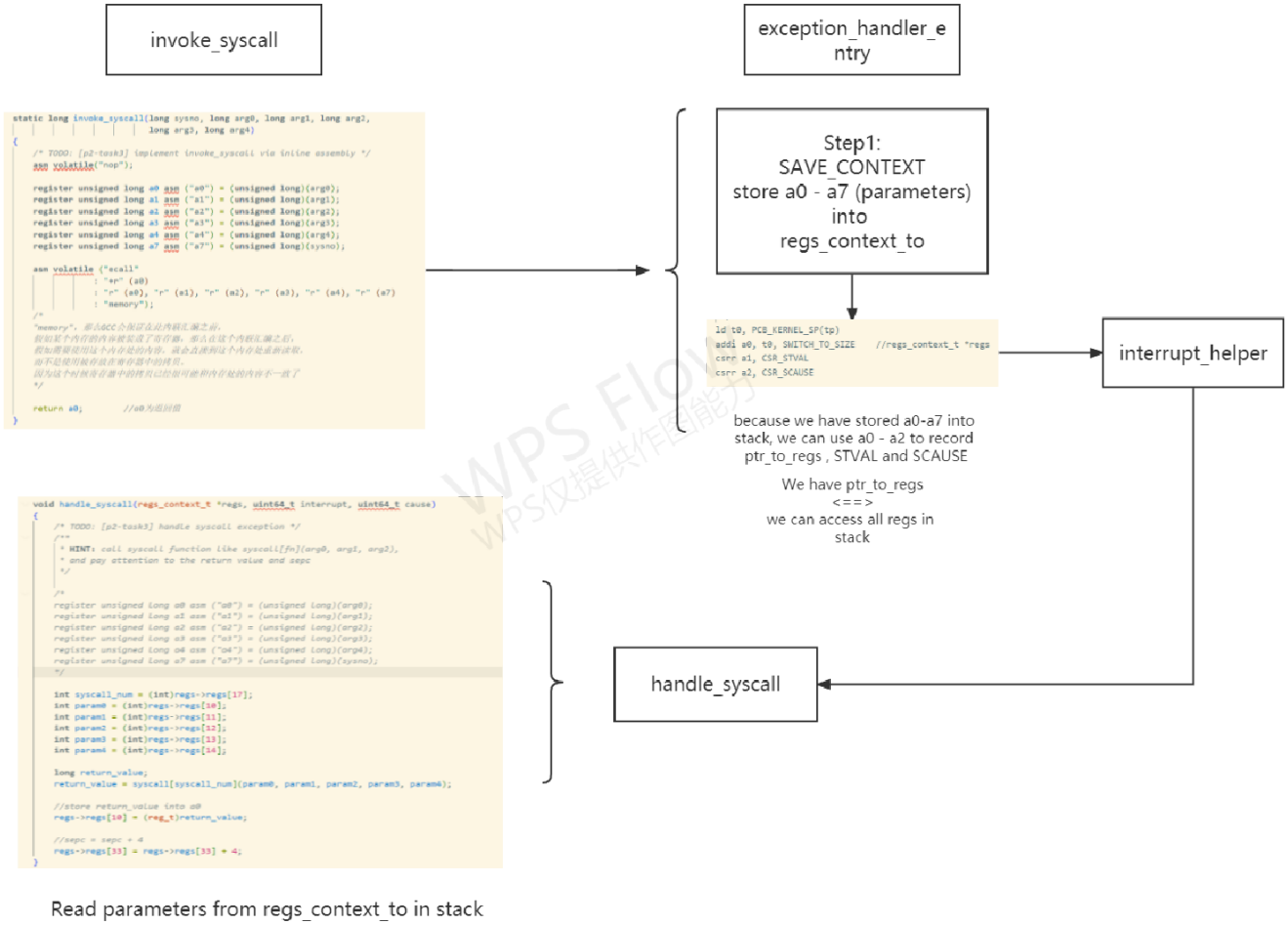
初始化说明：



2.完成一次 触发中断 --> 陷入中断 --> 恢复 的完整过程：

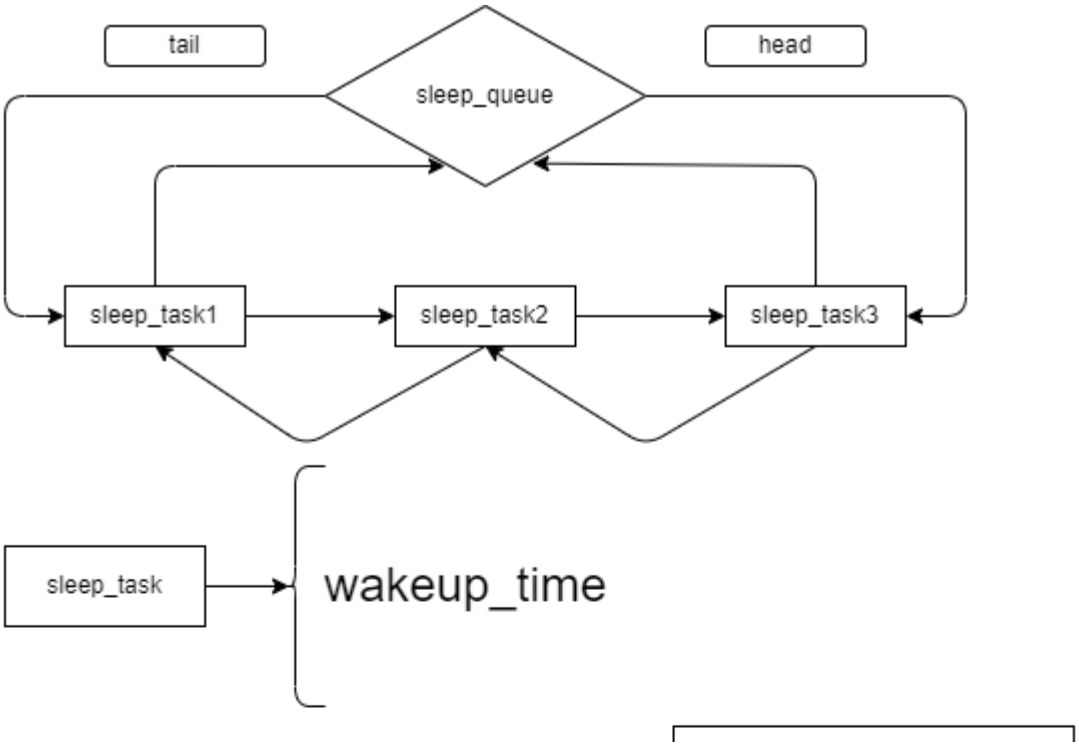


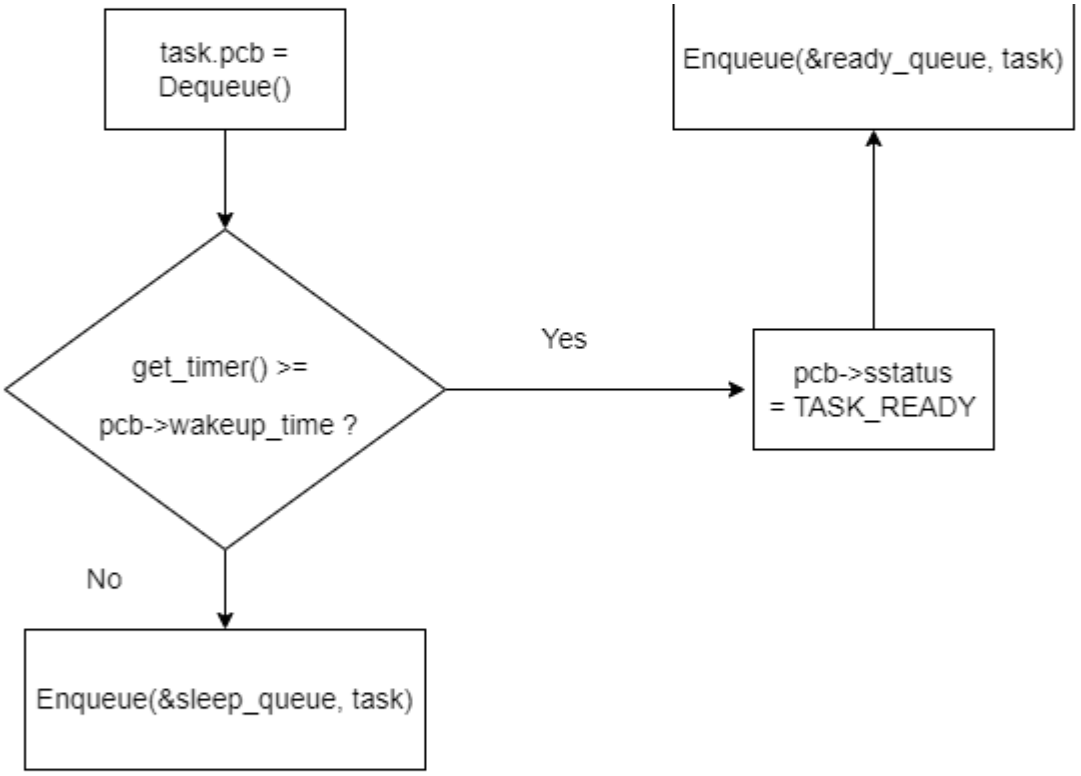
3.系统调用中如何传递参数，参数如何在内核中保存、传递、使用、恢复：



task4：

1.在check_sleeping中如何将所有该唤醒的进程唤醒：





Check_sleeping:
遍历sleep_queue：一次将每个sleep_task从队头出队，比较当前时间(get_timer())与其wakeup_time的大小，若当前时间大于wakeup_time，则将该task加入ready_queue，反之则从队尾入队(sleep_queue的任务顺序是没有影响的)。这样就将所有符合唤醒要求的任务出队并加入了ready_queue。

task5：

1.为线程创建PCB，用PCB模拟tcb：

整个过程与在main中初始化进程pcb的过程基本相同。

```
void thread_create(ptr_t function, int pid)
{
    /*param
    ptr_t function: 新创建线程要执行的函数
    pid: 新创建线程的线程号
    */

    //step1: 为新进程创建其对应的pcb
```

```
    //step1-1: 分配内核栈和用户栈
    ptr_t bottom1, bottom2, roof1, roof2;
    bottom1 = allocKernelPage(1);
    roof1 = bottom1 + pcb[pid-1].numpage_of_kernel * PAGE_SIZE;
    bottom2 = allocUserPage(1);
    roof2 = bottom2 + pcb[pid-1].numpage_of_user * PAGE_SIZE;

    tcb[pid-1].numpage_of_kernel = 1;
    tcb[pid-1].numpage_of_user = 1;

    regs_context_t *pt_regs =
        (regs_context_t *)(roof1 - sizeof(regs_context_t));

    pt_regs->regs[1] = (reg_t)function;   //ra
    pt_regs->regs[2] = roof2;             //sp
    pt_regs->sstatus = SR_SPIE;           //status
    pt_regs->sepc    = (reg_t)function;   //sepc

    switchto_context_t *pt_switchto =
        (switchto_context_t *)((ptr_t)pt_regs - sizeof(switchto_context_t));

    tcb[pid-1].kernel_sp = pt_switchto;
    tcb[pid-1].user_sp = roof2;

    pt_switchto->regs[0] = (reg_t)ret_from_exception;   //ra
    pt_switchto->regs[1] = pt_switchto;                 //sp

    tcb[pid-1].pid = pid;
    tcb[pid-1].status = TASK_READY;
    tcb[pid-1].cursor_x = 0;
    tcb[pid-1].cursor_y = 0;
    //tcb[pid-1].name
    tcb[pid-1].wakeup_time = 0;

    Enqueue(&ready_queue, &tcb[pid-1].list);
}
```