实验2-part1说明文档

陈远腾 2020k8009929041

开发过程：

1.过程简述:

(1) 9.15-19：预习了实验2-part1的内容，同时
在阅读xv6代码时对pcb、内核栈及scheduler设计有了一定了解
(2) 9.20-23：结合课上所讲内容，仔细阅读guide_book2，初步完成对
pcb结构、内核栈等结构的设计草稿。
(3) 9.24-26：利用周末时间开始代码设计，利用前两天完成了task 1，后一天完成了task2.

2.遇到的问题bug：

(1) 初始时对kernelallocpage函数及内核栈指针的理解不深，没搞清楚kernelallocpage函数返回的是分配的内存的低地址还是高地址，以及kernel_sp应该指向内核栈的哪里，导致设计出的代码没有让kernel_sp(0)指向上下文结构体的底部——sp指针，最终导致switch出问题

(2) 对双向循环队列的设计有问题，导致一个循环下来空头向队尾和队头的指针断裂了，后来仔细检查了出队函数的逻辑，发现少考虑了一种情况，导致队列断裂。

(3) 在设计switch_to时，出了很奇怪的bug，即3个程序只能循环调度一轮，之后就卡死。gdb很久也找不出问题，只能注释掉一部分内容进行尝试：反复尝试发现将sd s2, SWITCH_TO_S2(t0)后的sd指令注释掉就能正常运行，这是非常难以理解的。与同学交流后才知道问题出在switch_to开头的调整sp指针设置栈部分，原有代码的设计逻辑与我思路里的设计逻辑不同，我的设计并不需要再switch_to里开栈导致后面的bug，因此将首尾的开设栈，消除栈注释掉即可。

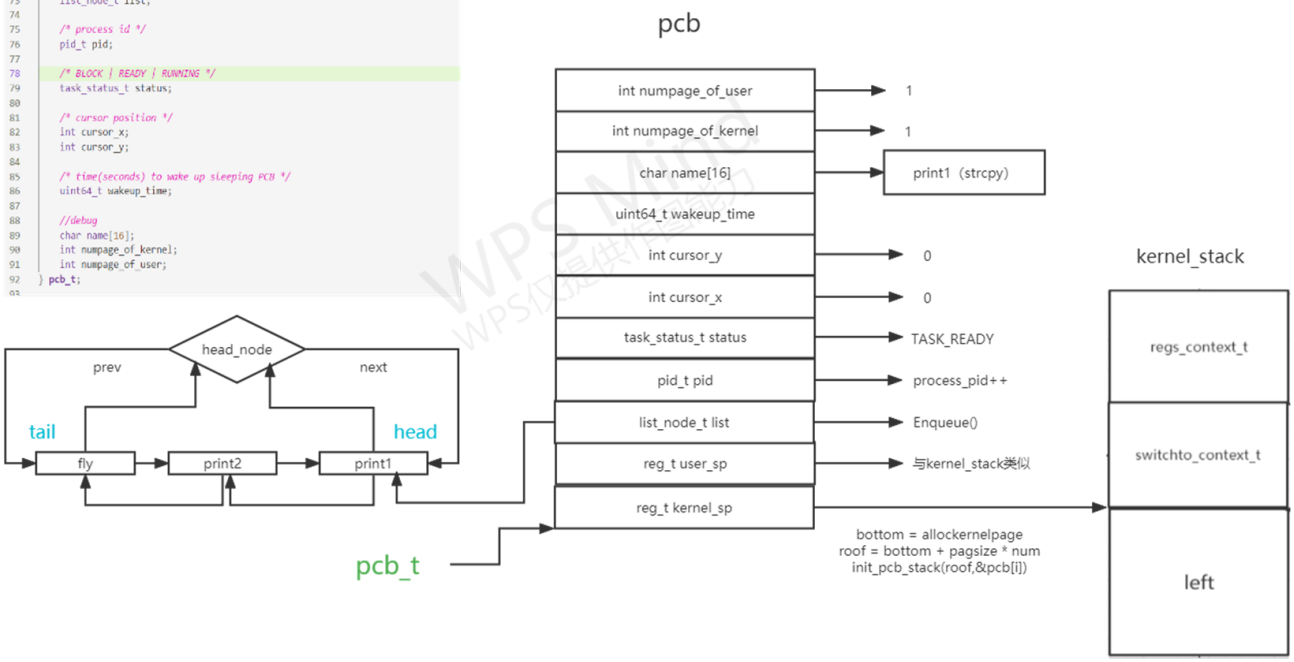
(4) 在设计互斥锁时遇到的最主要的一个bug是只在第一轮调度循环过程中能锁住冲突的进程，而在后续的调度循环中，两个互斥过程能同时运行，说明实际上后面没有锁住。经仔细检查发现原因是当一个进程尝试获得锁失败后，直到原本占有该锁的进程释放该锁后，也没有获得该锁。因此我在mutex_acquire函数"能否获得锁"条件判断失败所执行的代码末位递归调用mutex_acquire()，因此进程在锁被释放后能够正常获得该锁。

程序设计

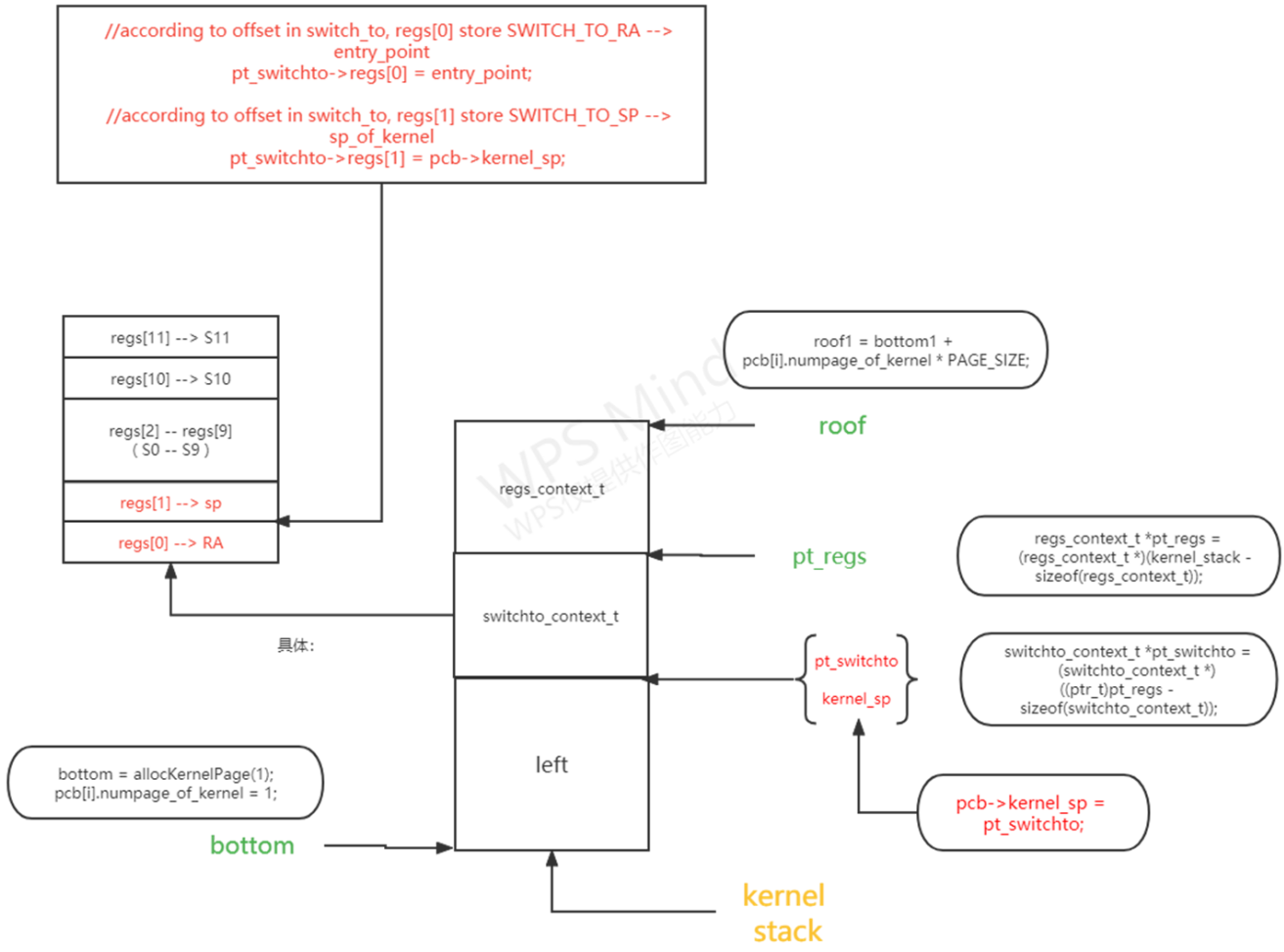
task1：

1.PCB结构体设计

```c
typedef struct pcb
{
    /* register context */
    // NOTE: this order must be preserved, which is defined in regs.h!!
    reg_t kernel_sp;
    reg_t user_sp;

    /* previous, next pointer */
    list_node_t list;

    /* process id */
    pid_t pid;

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

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

    /* time(seconds) to wake up sleeping PCB */
    uint64_t wakeup_time;

    //debug
    char name[16];
    int numpage_of_kernel;
    int numpage_of_user;
} pcb_t;
```



2.kernel stack设计



3.switch_to设计

Switch_to

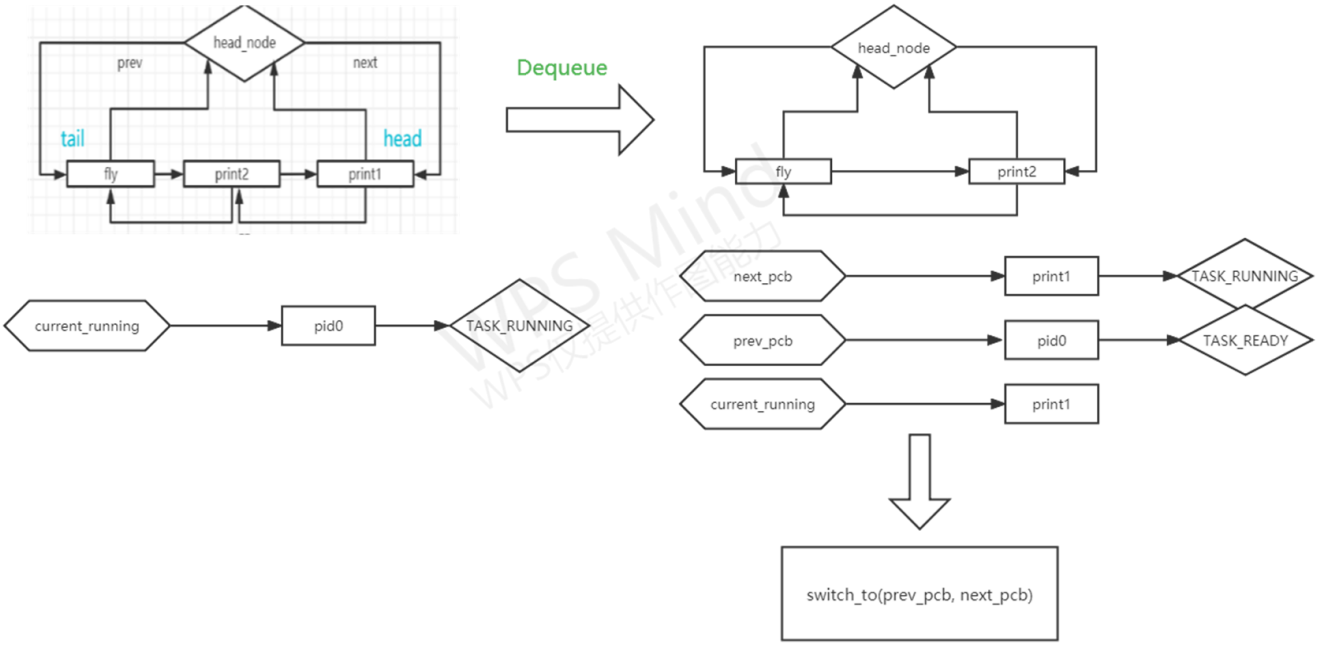
```
switch_to(prev_pcb,next_pcb);
```

```
ld t0, PCB_KERNEL_SP(a0)

/*
*now we show the struct of kernel
*sure now it is the stack of prev pcb

*  / \  reg S11
*   |     .
*   |     .
*   |     .
*   |   reg S0
*   |   reg SP
*   |   reg RA    <-- (kernel_sp of previous pcb <==> t0)
*/

sd ra, SWITCH_TO_RA(t0)
sd sp, SWITCH_TO_SP(t0)
sd s0, SWITCH_TO_S0(t0)
sd s1, SWITCH_TO_S1(t0)
sd s2, SWITCH_TO_S2(t0)
sd s3, SWITCH_TO_S3(t0)
sd s4, SWITCH_TO_S4(t0)
sd s5, SWITCH_TO_S5(t0)
sd s6, SWITCH_TO_S6(t0)
sd s7, SWITCH_TO_S7(t0)
sd s8, SWITCH_TO_S8(t0)
sd s9, SWITCH_TO_S9(t0)
sd s10, SWITCH_TO_S10(t0)
sd s11, SWITCH_TO_S11(t0)
```

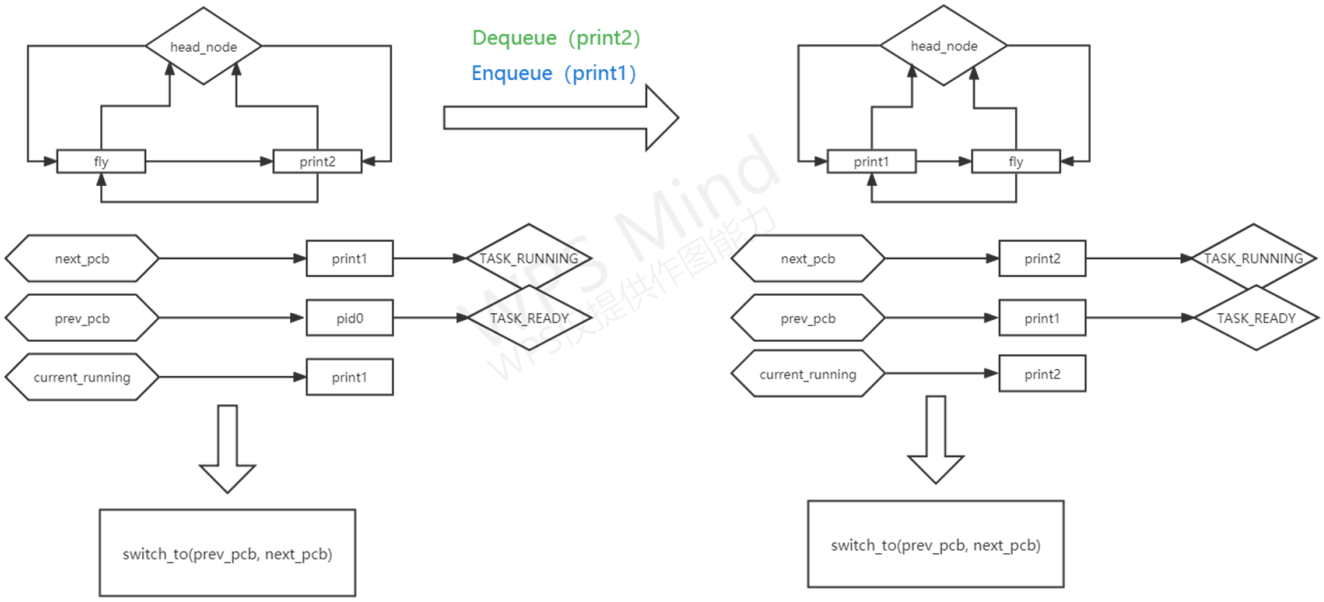
```
ld t0, PCB_KERNEL_SP(a1)

/*
*now we show the struct of kernel
*sure now it is the stack of next pcb

*  / \  reg S11
*   |     .
*   |     .
*   |     .
*   |   reg S0
*   |   reg SP
*   |   reg RA    <-- (kernel_sp of next pcb <==> t0)
*/

ld ra, SWITCH_TO_RA(t0)
ld sp, SWITCH_TO_SP(t0)
ld s0, SWITCH_TO_S0(t0)
ld s1, SWITCH_TO_S1(t0)
ld s2, SWITCH_TO_S2(t0)
ld s3, SWITCH_TO_S3(t0)
ld s4, SWITCH_TO_S4(t0)
ld s5, SWITCH_TO_S5(t0)
ld s6, SWITCH_TO_S6(t0)
ld s7, SWITCH_TO_S7(t0)
ld s8, SWITCH_TO_S8(t0)
ld s9, SWITCH_TO_S9(t0)
ld s10, SWITCH_TO_S10(t0)
ld s11, SWITCH_TO_S11(t0)
```

4.Do_Schduler

初始时：

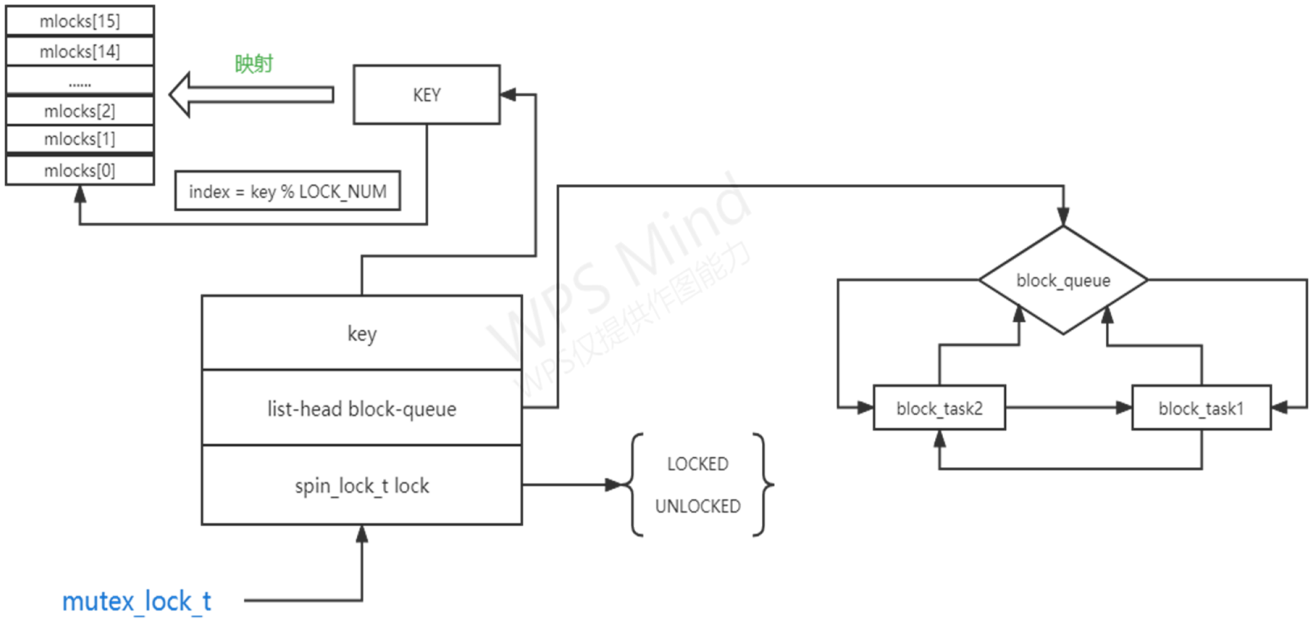


后序：



task2：

1.mutex_lock互斥锁结构：



2.互斥锁acquire_release调度过程：