练习0：填写已有实验

以下操作都是将**内容**复制到 lab6 里（都可以从 lab4 复制）**不要复制整个文件！！**

将 lab5 的 `kern/debug/kdebug.c`、`kern/init/init.c` 以及 `kern/trap/trap.c` 复制到 lab6 里

将 lab5 的 `kern/mm/pmm.c` 和 `kern/mm/default_pmm.c` 复制到 lab6 里

将 lab5 的 `kern/mm/vmm.c` 和 `kern/mm/swap_fifo.c` 复制到 lab6 里

最后将 lab5 的 `kern/process/proc.c` 复制到 lab6 里，之后还要改进代码

trap_dispatch 函数源码

写于：**kern/trap/trap.c**

```
static void
trap_dispatch(struct trapframe* tf){
    ......
    case IRQ_OFFSET + IRQ_TIMER:
#if 0
        LAB3 : If some page replacement algorithm(such as CLOCK PRA) need tick to change the priority of pages,
            then you can add code here.
#endif
            /* LAB1 YOUR CODE : STEP 3 */
            /* handle the timer interrupt */
            /* (1) After a timer interrupt, you should record this event using a global variable (increase it), such as ticks in kern/driver/clock.c
             * (2) Every TICK_NUM cycle, you can print some info using a funciton, such as print_ticks().
             * (3) Too Simple? Yes, I think so!
             */
             /* LAB5 YOUR CODE */
             /* you should upate you lab1 code (just add ONE or TWO lines of code):
              *    Every TICK_NUM cycle, you should set current process's current->need_resched = 1
              */
              /* LAB6 YOUR CODE */
              /* you should upate you lab5 code
               * IMPORTANT FUNCTIONS:
               * sched_class_proc_tick
               */
            break;
    ......
}
```

trap_dispatch 函数答案

写于：**kern/trap/trap.c**

```
static void
trap_dispatch(struct trapframe *tf) {
    ......
    case IRQ_OFFSET + IRQ_TIMER:
        ticks++;
        assert(current != NULL);
        sched_class_proc_tick(current);
        break;
    ......
}
```

alloc_proc 函数源码

写于：**kern/process/proc.c**

```
// alloc_proc - alloc a proc_struct and init all fields of proc_struct
static struct proc_struct *
alloc_proc(void) {
    struct proc_struct *proc = kmalloc(sizeof(struct proc_struct));
    if (proc != NULL) {
    //LAB4:EXERCISE1 YOUR CODE
    /*
     * below fields in proc_struct need to be initialized
     *       enum proc_state state;                      // Process state
     *       int pid;                                    // Process ID
     *       int runs;                                   // the running
times of Proces
     *       uintptr_t kstack;                           // Process kernel
stack
     *       volatile bool need_resched;                 // bool value:
need to be rescheduled to release CPU?
     *       struct proc_struct *parent;                 // the parent
process
     *       struct mm_struct *mm;                       // Process's
memory management field
     *       struct context context;                     // Switch here to
run process
     *       struct trapframe *tf;                       // Trap frame for
current interrupt
     *       uintptr_t cr3;                              // CR3 register:
the base addr of Page Directroy Table(PDT)
     *       uint32_t flags;                             // Process flag
     *       char name[PROC_NAME_LEN + 1];               // Process name
     */
     //LAB5 YOUR CODE : (update LAB4 steps)
    /*
     * below fields(add in LAB5) in proc_struct need to be initialized
     *       uint32_t wait_state;                        // waiting state
     *       struct proc_struct *cptr, *yptr, *optr;     // relations
between processes
     */
     //LAB6 YOUR CODE : (update LAB5 steps)
    /*
     * below fields(add in LAB6) in proc_struct need to be initialized
     *     struct run_queue *rq;                       // running queue
contains Process
```

```
     *     list_entry_t run_link;                    // the entry linked
in run queue
     *     int time_slice;                           // time slice for
occupying the CPU
     *     skew_heap_entry_t lab6_run_pool;          // FOR LAB6 ONLY:
the entry in the run pool
     *     uint32_t lab6_stride;                     // FOR LAB6 ONLY:
the current stride of the process
     *     uint32_t lab6_priority;                   // FOR LAB6 ONLY:
the priority of process, set by lab6_set_priority(uint32_t)
     */
    }
    return proc;
}
```

alloc_proc 函数答案

写于：**kern/process/proc.c**

```
static struct proc_struct *
alloc_proc(void) {
    struct proc_struct *proc = kmalloc(sizeof(struct proc_struct));
    if (proc != NULL) {
        proc->pid = -1;                                     // 进程ID
        memset(&(proc->name), 0, PROC_NAME_LEN);            // 进程名
        proc->state = PROC_UNINIT;                          // 进程状态
        proc->runs = 0;                                     // 进程时间片
        proc->need_resched = 0;                             // 进程是否能被
调度
        proc->flags = 0;                                    // 标志位
        proc->kstack = 0;                                   // 进程所使用的
内存栈地址
        proc->cr3 = boot_cr3;                               // 将页目录表地
址设为内核页目录表基址
        proc->mm = NULL;                                    // 进程所用的虚
拟内存
        memset(&(proc->context), 0, sizeof(struct context)); // 进程的上下文
        proc->tf = NULL;                                    // 中断帧指针
        proc->parent = NULL;                                // 该进程的父进
程
        proc->wait_state = 0;                               // 等待状态的标
志位
        proc->cptr = NULL;                                  // 该进程的子进
程
        proc->yptr = NULL;                                  // 该进程的弟进
程
        proc->optr = NULL;                                  // 该进程的兄进
程
        proc->rq = NULL;                                    // Lab6：当前进
程在运行队列中的指针
        list_init(&(proc->run_link));                       // Lab6：运行队
列的指针
        proc->time_slice = 0;                               // Lab6：占用
CPU 的时间片
        proc->lab6_run_pool.left = NULL;                    // Lab6：运行池
中的条目
```

```
25          proc->lab6_run_pool.right = NULL;                // Lab6：运行池中的条目
26          proc->lab6_run_pool.parent = NULL;               // Lab6：运行池中的条目
27          proc->lab6_stride = 0;                           // Lab6：步进值
28          proc->lab6_priority = 0;                         // Lab6：优先级(和步进值成反比)
29      }
30      return proc;
31  }
```

练习1: 使用 Round Robin 调度算法