OS lab7 Report

name：刘乐奇

sid：12011327

Ubuntu用户名：lynchrocket

阅读Lab7代码，详细描述user/rr.c中产生的进程是如何进行调度的。描述中需要包含各进程的执行顺序，何时进入被调度的队列，何时被切换，执行结束时发生了什么（重复的内容只需描述一遍）。

父进程在此处 `fork()` 产生了五个子进程

```
if ((pids[i] = fork()) == 0)
```

并且在 `fork()` 时产生中断，由trap.c的 `schedule()` 的 `sched_class_enqueue()` 将进程加入调度的队列中。
并且由于if条件为假，会直接跳至此处

```
cprintf("main: fork ok,now need to wait pids.\n");
```

在此处父进程会调用 `waitpid()` 等待各个子进程，并进入休眠状态，产生中断，在trap.c中调用 `schedule()` 函数，调度到子进程。

```
for (i = 0; i < TOTAL; i ++) {
    status[i]=0;
    waitpid(pids[i],&status[i]);
    //cprintf("main: pid %d, acc %d, time %d\n",pids[i],status[i],gettime_msec());
}
```

这时候3号子进程开始执行，3号子进程会在fork语句的位置开始执行，此时fork返回值为0，这时候就会进入到循环。由于子进程运行的时间大于一个时间片，每次时钟中断剩余的时间片就会-1；如果为零，则会设置为需要调度的状态，在trap.c中的 `trap()` 函数中会进行调度。

```
// kern/trap/trap.c
void
trap(struct trapframe *tf) {

    if (current == NULL) {
        trap_dispatch(tf);
    } else {
        struct trapframe *otf = current->tf;
        current->tf = tf;

        bool in_kernel = trap_in_kernel(tf);

        trap_dispatch(tf);

        current->tf = otf;
        if (!in_kernel) {
            if (current->flags & PF_EXITING) {
                do_exit(-E_KILLED);
            }
            if (current->need_resched) {
                schedule();
            }
        }
    }
}


// kern/trap/trap.c/interrupt_handler()
        case IRQ_S_TIMER:
            clock_set_next_event();
            if (++ticks % TICK_NUM == 0 ) {
                //print_ticks()
            }
            if (current){
                sched_class_proc_tick(current);
            }
            break;


// kern/schedule/sched.c
static void
RR_proc_tick(struct run_queue *rq, struct proc_struct *proc) {
    if (proc->time_slice > 0) {
        proc->time_slice --;
    }
    if (proc->time_slice == 0) {
        proc->need_resched = 1;
    }
}
```

这时候就调度到4号进程。重复这个过程。3，4，5，6，7，3，4，5，6，7，.....（RR调度算法），直到某一个进程执行结束。

```
// kern/schedule/sched.c
void
schedule(void) {
    bool intr_flag;
    struct proc_struct *next;
    local_intr_save(intr_flag);
    {
        current->need_resched = 0;
        if (current->state == PROC_RUNNABLE) {
            sched_class_enqueue(current);
        }
        if ((next = sched_class_pick_next()) != NULL) {
            sched_class_dequeue(next);
        }
        if (next == NULL) {
            next = idleproc;
        }
        next->runs ++;
        if (next != current) {
            cprintf("The next proc is pid:%d\n",next->pid);
            proc_run(next);
        }
    }
    local_intr_restore(intr_flag);
}
```

当某一进程结束时，在 `do_exit()` 函数中会设置自己的状态为僵尸状态，并唤醒父进程，将父进程插入到运行队列中，进行调度。此时要么调度到其他进程执行一个时间片，要么进程结束再次调度，最终调度到父进程。在 `do_wait()` 函数中父进程会判断子进程是否为僵尸进程，对僵尸进程进行清除打扫，回收资源。如此进行5次回收完所有子进程资源，然后父进程返回，执行结束。