Lab7

> 12011702 张镇涛

Q1

> 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）。

总结调度流程：

父进程执行到waitpid语句，主进程会找到pids[0]=3对应的进程。因为3号进程此时不处于僵尸态，父进程进入休眠状态，调用schedule函数，调度到该子进程。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。由于我们的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。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，由于我们创建的子进程是相同的，这里是3号进程最先执行结束。

进程最后会进入do_exit函数，设置自己的状态为僵尸状态，唤醒父进程，将父进程插入到运行队列，然后进行调度。经过几次调度，会调度到父进程， 父进程这时候还在wait系统调用中，由于子进程全部为僵尸进程，因此回收所有资源并结束wait()。

执行顺序：

我们执行代码：

```
setup timer interrupts
The next proc is pid:1
The next proc is pid:2
kernel_execve: pid = 2, name = "rr".
Breakpoint
main: fork ok,now need to wait pids.
...
The next proc is pid:3
The next proc is pid:4
The next proc is pid:5
The next proc is pid:6
The next proc is pid:7
The next proc is pid:3
child pid 3, acc 5000000, time 10010
The next proc is pid:4
child pid 4, acc 4948000, time 10010
The next proc is pid:5
child pid 5, acc 5008000, time 10010
The next proc is pid:6
child pid 6, acc 5012000, time 10010
The next proc is pid:7
child pid 7, acc 4932000, time 10010
The next proc is pid:2
```

```
main: wait pids over
The next proc is pid:1
all user-mode processes have quit.
The end of init_main
kernel panic at kern/process/proc.c:414:
    initproc exit.
```

首先通过父进程创建出3，4，5，6，7子进程，然后按照该顺序在每一个时间剩余片为0时进行调度，当所有进程结束时，会再次调度到父进程。

何时进入被调度的队列：

```
//schedule/sched.c
if (current->state == PROC_RUNNABLE) {
      sched_class_enqueue(current);
}
```

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, 该进程会加入到队列队尾。

何时被切换：

剩余时间片为0 或者 进程结束。

执行结束时发生了什么：

由于每个子进程完全相同，3号进程率先执行，所以3号进程会首先结束。进程最后会进入do_exit函数，设置自己的状态为僵尸状态，唤醒父进程，将父进程插入到运行队列，然后进行调度。经过几次调度，会调度到父进程， 父进程这时候还在wait系统调用中，由于子进程全部为僵尸进程，因此回收所有资源并结束wait()。