**Lab5练习3**

**练习3: 阅读分析源代码，理解进程执行 fork/exec/wait/exit 的实现，以及系统调用的实 现（不需要编码）**

**请在实验报告中简要说明你对 fork/exec/wait/exit函数的分析。并回答如下问题：**

**请分析fork/exec/wait/exit在实现中是如何影响进程的执行状态的？ 请给出ucore中一个用户态进程的执行状态生命周期图（包执行状态，执行状态之间的变换关系，以及产生变换的事件或 函数调用）。（字符方式画即可）**

**执行：make grade。如果所显示的应用程序检测都输出ok，则基本正确。（使用的是qemu-1.0.1）**

Fork：

1、分配并初始化进程控制块(alloc\_proc 函数);

2、分配并初始化内核栈(setup\_stack 函数);

3、根据 clone\_flag标志复制或共享进程内存管理结构(copy\_mm 函数);

4、设置进程在内核(将来也包括用户态)正常运行和调度所需的中断帧和执行上下文(copy\_thread 函数);

5、把设置好的进程控制块放入hash\_list 和 proc\_list 两个全局进程链表中;

6、自此,进程已经准备好执行了,把进程状态设置为“就绪”态;

7、设置返回码为子进程的 id 号。

Exec：

1、首先为加载新的执行码做好用户态内存空间清空准备。如果mm不为NULL，则设置页表为内核空间页表，且进一步判断mm的引用计数减1后是否为0，如果为0，则表明没有进程再需要此进程所占用的内存空间，为此将根据mm中的记录，释放进程所占用户空间内存和进程页表本身所占空间。最后把当前进程的mm内存管理指针为空。

2、接下来是加载应用程序执行码到当前进程的新创建的用户态虚拟空间中。之后就是调用load\_icode从而使之准备好执行。

Wait：

1、 如果 pid!=0，表示只找一个进程 id 号为 pid 的退出状态的子进程，否则找任意一个处于退出状态的子进程;

2、 如果此子进程的执行状态不为PROC\_ZOMBIE，表明此子进程还没有退出，则当前进程设置执行状态为PROC\_SLEEPING（睡眠），睡眠原因为WT\_CHILD(即等待子进程退出)，调用schedule()函数选择新的进程执行，自己睡眠等待，如果被唤醒，则重复跳回步骤 1 处执行;

3、 如果此子进程的执行状态为 PROC\_ZOMBIE，表明此子进程处于退出状态，需要当前进程(即子进程的父进程)完成对子进程的最终回收工作，即首先把子进程控制块从两个进程队列proc\_list和hash\_list中删除，并释放子进程的内核堆栈和进程控制块。自此，子进程才彻底地结束了它的执行过程，它所占用的所有资源均已释放。

Exit：

1、先判断是否是用户进程，如果是，则开始回收此用户进程所占用的用户态虚拟内存空间;（具体的回收过程不作详细说明）

2、设置当前进程的中hi性状态为PROC\_ZOMBIE，然后设置当前进程的退出码为error\_code。表明此时这个进程已经无法再被调度了，只能等待父进程来完成最后的回收工作（主要是回收该子进程的内核栈、进程控制块）

3、如果当前父进程已经处于等待子进程的状态，即父进程的wait\_state被置为WT\_CHILD，则此时就可以唤醒父进程，让父进程来帮子进程完成最后的资源回收工作。

4、如果当前进程还有子进程,则需要把这些子进程的父进程指针设置为内核线程init,且各个子进程指针需要插入到init的子进程链表中。如果某个子进程的执行状态是 PROC\_ZOMBIE,则需要唤醒 init来完成对此子进程的最后回收工作。

5、执行schedule()调度函数，选择新的进程执行。

Lab6练习1

**练习1: 使用Round Robin 调度算法（不需要编码）**

**完成练习0后，建议大家比较一下（可用kdiff3等文件比较软件）个人完成的lab5和练习0完成后的刚修改的lab6之间的区别， 分析了解lab6采用RR调度算法后的执行过程。执行make grade，大部分测试用例应该通过。但执行priority.c应该过不去。**

**请在实验报告中完成：**

**请理解并分析sched\_calss中各个函数指针的用法，并接合Round Robin调度算法描ucore的调度执行过程 请在实验报告中简要说明如何设计实现”多级反馈队列调度算法“，给出概要设计，鼓励给出详细设计**

**1：执行过程：**

让所有runnable态的进程分时轮流使用CPU时间。RR调度器维护当前runnable进程的有序运行队列。当前进程的时间片用完之后，调度器将当前进程放置到运行队列的尾部，再从其头部取出进程进行调度。RR调度算法的就绪队列在组织结构上也是一个双向链表，只是增加了一个成员变量，表明在此就绪进程队列中的最大执行时间片。而且在进程控制块proc\_struct中增加了一个成员变量time\_slice，用来记录进程当前的可运行时间片段。这是由于RR调度算法需要考虑执行进程的运行时间不能太长。在每个timer到时的时候，操作系统会递减当前执行进程的time\_slice，当time\_slice为0时，就意味着这个进程运行了一段时间（这个时间片段称为进程的时间片），需要把CPU让给其他进程执行，于是操作系统就需要让此进程重新回到rq的队列尾，且重置此进程的时间片为就绪队列的成员变量最大时间片max\_time\_slice值，然后再从rq的队列头取出一个新的进程执行。

**2：函数用法：**

RR\_init完成了对进程队列的初始化

RR\_enqueue的函数实现如下表所示。即把某进程的进程控制块指针放入到rq队列末尾，且如果进程控制块的时间片为0，则需要把它重置为rq成员变量max\_time\_slice。这表示如果进程在当前的执行时间片已经用完，需要等到下一次有机会运行时，才能再执行一段时间。

RR\_dequeue的函数实现如下表所示。即把就绪进程队列rq的进程控制块指针的队列元素删除，并把表示就绪进程个数的proc\_num减一。

RR\_pick\_next的函数实现如下表所示。即选取就绪进程队列rq中的队头队列元素，并把队列元素转换成进程控制块指针。

RR\_proc\_tick的函数实现如下表所示。即每次timer到时后，trap函数将会间接调用此函数来把当前执行进程的时间片time\_slice减一。如果time\_slice降到零，则设置此进程成员变量need\_resched标识为1，这样在下一次中断来后执行trap函数时，会由于当前进程程成员变量need\_resched标识为1而执行schedule函数，从而把当前执行进程放回就绪队列末尾，而从就绪队列头取出在就绪队列上等待时间最久的那个就绪进程执行。