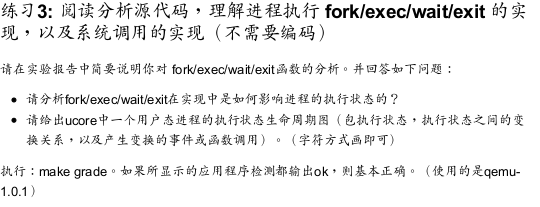
**实验五 用户进程管理**

实验目的：

了解ucore如何实现系统调用sys\_fork/sys\_exec/sys\_exit/sys\_wait来进行进程管理。

实验内容：



1. 对fork/exec/wait/exit影响进程执行状态的分析：

Fork 不会影响当前进程的执行状态，但是会将子进程的状态标记为RUNNALB，使得可以在后续的调度中运行起来；

Exec 不会影响当前进程的执行状态，但是会修改当前进程中执行的程序；

Wait 系统调用取决于是否存在可以释放资源（ZOMBIE）的子进程，如果有的话不会发生状态的改变，如果没有的话会将当前进程置为SLEEPING态，等待执行了exit的子进程将其唤醒；

Exit 会将当前进程的状态修改为ZOMBIE态，并且会将父进程唤醒（修改为RUNNABLE），然后主动让出CPU使用权；

**fork**

当程序执行fork时，使用系统调用SYS\_fork,系统调用SYS\_fork主要是由do\_fork和wakeup\_proc来完成。

**exec**

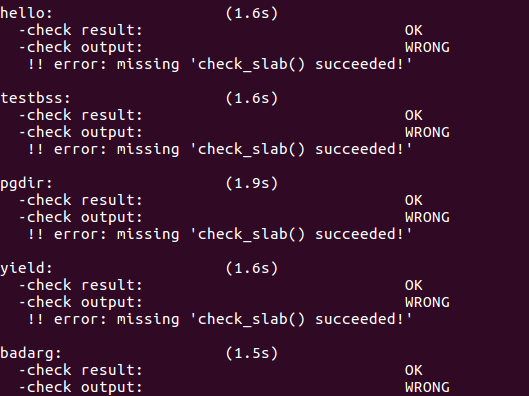
当应用程序执行的时候，会系统调用SYS\_exec,当ucore收到时，就会使用do\_execve()函数来实现。

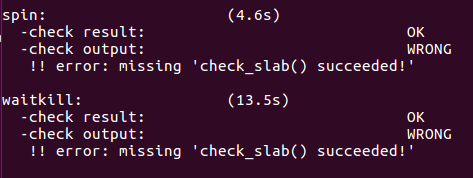
**wait**

当执行wait功能的时候，会系统调用SYS\_wait，功能主要由do\_wait函数实现，完成回收子进程的内核栈和进程控制块所占内存空间工作。

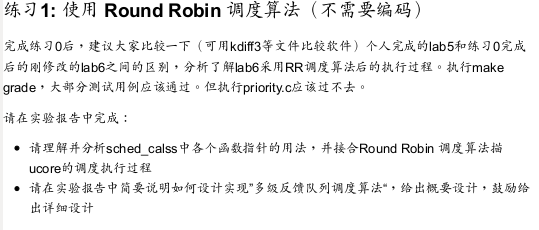
**exit**

当执行exit功能的时候，会系统调用SYS\_exit，系统调用的功能主要由do\_exit函数实现。实现过程：首先，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；然后，设置当前进程的中hi性状态为PROC\_ZOMBIE，设置当前进程的退出码为error\_code。这表示此时这个进程已经无法再被调度，只能等待父进程来完成最后的回收工作；如果当前父进程已经处在等待子进程的状态（父进程的wait\_state被置为WT\_CHILD），则此时可以唤醒父进程，让父进程来帮助子进程完成最后的资源回收工作；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，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。最后，执行schedule()调度函数，选择新的进程执行。





**实验六 调度器**



1.RR\_init函数

完成了对对于进程队列的初始化步骤；

2. RR\_enqueue函数

把进程的进程控制块指针放入到rq队列末尾，如果进程控制块的时间片为0，则需要把它重置为max\_time\_slice（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）。然后在依次调整rq和rq的进程数目加一。

3. RR\_pick\_next函数

选取函数，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，即置为当前占用CPU的程序。

4. RR\_proc\_tick函数

每一次时间片到时的时候，当前执行进程的时间片time\_slice便减一。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。