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的实现

fork的功能是创建一个新进程，具体地说是创建一个新进程所需的控制信息。我们以用户程序forktest为例，来分析fork的调用过程。

##### 从用户态的fork到内核态的do\_fork

user/forktest.c的main调用fork来创建新进程，从fork到do\_fork的调用过程如下：

fork -> sys\_fork(位于user/lib/syscall.c) -> syscall(SYS\_fork) -> sys\_fork(kern/syscall/syscall.c) -> do\_fork

#### xec的实现

exec的功能是在已经存在的进程的上下文中运行新的可执行文件，替换先前的可执行文件。在ucore中exec对应的函数是do\_execve。

1. do\_execve首先检查用户态虚拟内存空间是否合法，如果合法且目前只有当前进程占用，则释放虚拟内存空间，包括取消虚拟内存到物理内存的映射，释放vma，mm及页目录表占用的物理页等。
2. 调用load\_icode函数来加载应用程序

3.重新设置当前进程的名字，然后返回

#### wait的实现

wait的功能是等待子进程结束，从而释放子进程占用的资源。在ucore中wait对应的函数是do\_wait。

1.遍历进程链表proc\_list，根据输入参数寻找指定pid或任意pid的子进程，如果没找到，直接返回错误信息。

2.如果找到子进程，而且其状态为ZOMBIE，则释放子进程占用的资源，然后返回。

3.如果找到子进程，但状态不为ZOMBIE，则将当前进程的state设置为SLEEPING、wait\_state设置为WT\_CHILD，然后调用schedule函数，从而进入等待状态。等再次被唤醒后，重复寻找状态为ZOMBIE的子进程。

#### exit的实现

exit的功能是释放进程占用的资源并结束运行进程。在ucore中exit对应的函数是do\_exit。

1.释放页表项记录的物理内存，以及mm结构、vma结构、页目录表占用的内存。

2.将自己的state设置为ZOMBIE，然后唤醒父进程，并调用schedule函数，等待父进程回收剩下的资源，最终彻底结束子进程。

请分析fork/exec/wait/exit在实现中是如何影响进程的执行状态的？

当程序执行fork(),exec(),wait(),exit()函数时，会调用sys\_xxxx（）函数，这些函数又会调用syscall()函数，syscall()函数嵌入了内联汇编指令，int产生系统调用与中断。  
而kernel则会对这些系统调用进行统一的处理。这些函数会产生中断，如果此时current -> need\_resched ==1,则会进行调度。  
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）。

进程创建(fork()函数) -> 进程就绪（proc -> state == RUNNABLE）-> 进程执行（schedule()函数) -> 进程退出（do\_exit()） -> 进程结束(do\_wait()回收kstack和proc\_struct)

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\_class中各个函数指针的用法，并结合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

sched\_calss中各个函数指针的用法：

1. void (\*init)(struct run\_queue \*rq)：初始化run queue队列 2.void(\*enqueue)(struct run\_queue \*rq, struct proc\_struct \*proc)：将当前的线程入队

3.void (\*dequeue)(struct run\_queue \*rq, struct proc\_struct \*proc)：将proc线程出队

4.struct proc\_struct \*(\*pick\_next)(struct run\_queue \*rq)：在rq队列中选择一个符合条件的队列。

5.void (\*proc\_tick)(struct run\_queue \*rq, struct proc\_struct \*proc)：在产生时钟中断时将proc的时间片减少，并对其进行判断是否需要调度。

ucore的调度过程：ucore通过schedule()函数进行调度，在这个函数中，当需要进行线程的切换时，他会先将当前的线程通过sched\_class\_enqueue()函数加入队列，然后通过sched\_class\_pick\_next()函数选择下一个要执行的线程并将其从队列中删除。

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.进程在进入待调度的队列等待时，首先进入优先级最高的Q1队列等待；

2.首先调度优先级高的队列中的进程。若高优先级中队列中已没有调度的进程，则调度次优先级队列中的进程；

3.对于同一个队列中的各个进程，按照时间片轮转法调度。比如Q1队列的时间片为N，那么Q1中的作业在经历了N个时间片后若还没有完成，则进入Q2队列等待，若Q2的时间片用完后作业还不能完成，一直进入下一级队列，直至完成；

4.在低优先级的队列中的进程在运行时，又有新到达的作业，那么在运行完这个时间片后，CPU马上分配给新到达的作业。