Lab5练习三

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)

1.fork

分配并初始化进程控制块(alloc\_proc 函数)

分配并初始化内核栈(setup\_stack 函数)

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

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)

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

自此,进程已经准备好执行了,把进程状态设置为“就绪”态

设置返回码为子进程的 id

2.exec

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。

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。

3.wait

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;

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;

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。

4.exit

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）

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）

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。

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。

执行schedule()调度函数，选择新的进程执行。

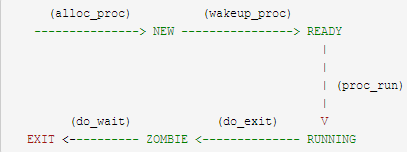
\*\*\*网上搜索参考四个函数功能\*\*\*

fork：执行完毕后，如果创建新进程成功，则出现两个进程，一个是子进程，一个是父进程。在子进程中，fork函数返回0，在父进程中，fork返回新创建子进程的进程ID。我们可以通过fork返回的值来判断当前进程是子进程还是父进程。

execve：完成用户进程的创建工作。首先为加载新的执行码做好用户态内存空间清空准备。接下来的一步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。

wait：等待任意子进程的结束通知。wait\_pid函数等待进程id号为pid的子进程结束通知。这两个函数最终访问sys\_wait系统调用接口让ucore来完成对子进程的最后回收工作

exit：会把一个退出码error code传递给ucore ，ucore通过执行内核函数do\_exit来完成对当前进程的退出处理，主要工作简单地说就是回收当前进程所占的大部分内存资源，并通知父进程完成最后的回收工作。



Lab6练习一

请理解并分析sched\_calss中各个函数指针的用法,并接合Round Robin 调度算法描述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

在内核初始化时调用sched\_init函数选择调度器并对调度器初始化。每当进程被唤醒则调用enqueue函数将其加入调度器等待调度的进程队列。每当发生进程切换时，首先调用enqueue将当前正在运行的进程加入等待调度的进程队列，然后调用pick\_next函数获取接下来将执行的进程，并调用dequeue将被选中即将执行的进程从调度队列中移除。每当产生时钟中断时，需要调用proc\_tick函数来更新调度器的时钟信息。

多级反馈队列调度算法的主要思路是，时间片大小随着优先级的增加而增加。同时，进程在当前时间片没有完成，则降到下一优先级。

算法的概要设计：

1. 初始化三个空的队列
2. 第一个队列作为8ms的队列，第二个队列作为40ms队列，第三个作为先到先服务队列。
3. 每次时间片完时检查所有队列中的进程是否完成，未完成的降级到下一个队列。
4. 执行完的进程退出队列，未执行的加入队列。