**实验三 用户进程管理**

**一、实验目的**

1.了解第一个用户进程创建过程

2.了解系统调用框架的实现机制

3.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

4.理解操作系统的调度管理机制

5.熟悉 ucore 的系统调度器框架，以及缺省的Round-Robin 调度算法

**二、实验内容**

将创建用户进程，让用户进程在用户态执行，且在需要ucore支持时，可通过系统调用来让ucore提供服务。为此需要构造出第一个用户进程，并通过系统调用sys\_fork/sys\_exec/sys\_exit/sys\_wait来支持运行不同的应用程序，完成对用户进程的执行过程的基本管理。

**三、实验题目**

**练习1: 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**

**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。

do\_fork的实现

1.分配一个进程控制块，设置其state为UNINIT

2.为内核栈分配2页的内存空间，并将其地址记录在进程控制块的kstack字段中

3.复制父进程的内存空间到新进程

4.为新进程分配pid

5.设置新进程的父进程、子进程等关系信息

6.将新进程添加到进程链表proc\_list和哈希表hash\_list中

7.设置新进程的state为RUNNABLE，从而将其唤醒。

**exec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。

**系统调用的实现**

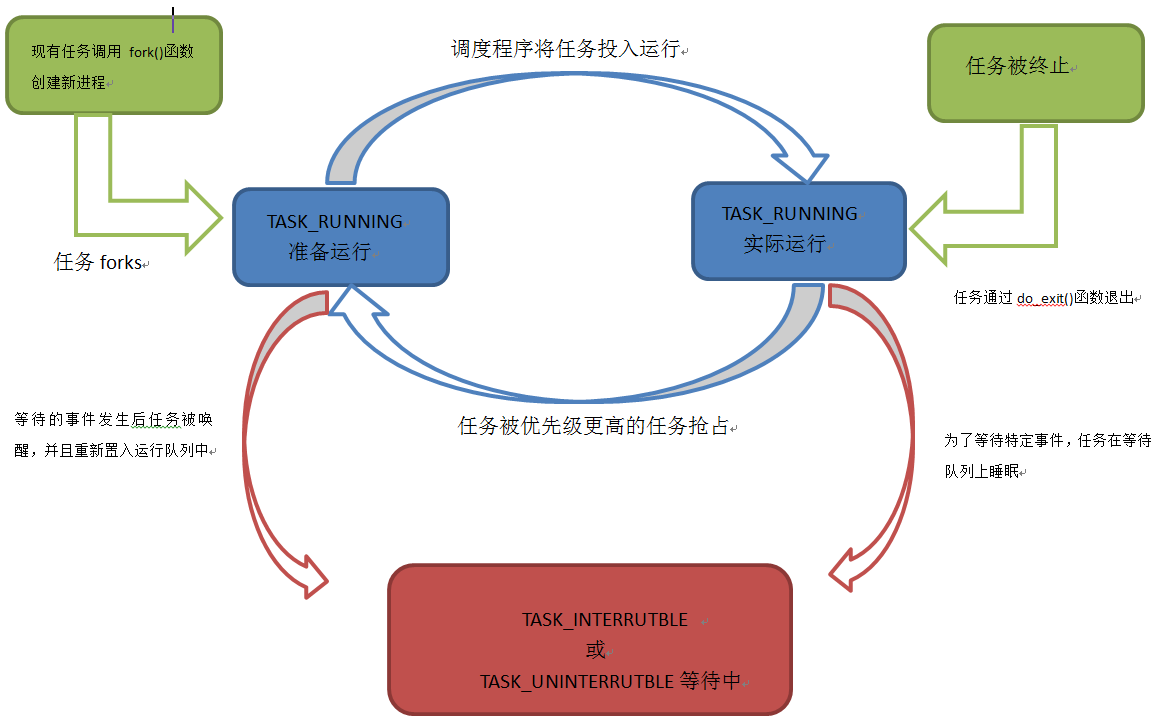
ucore实现系统调用分为以下几步：

1.在idt\_init函数中初始化系统调用对应的中断描述符。

2.在user/libs/syscall.c中封装了syscall接口，简化应用程序访问系统调用的复杂性。

3.在kernel/syscall/syscall.c中用函数数组来存储各系统调用对应的处理函数的地址，并封装了syscall接口，用于根据系统调用号索引函数数组，找到对应的处理函数来运行。

**请给出ucore中一个用户态进程的执行状态生命周期图**



**练习2: 使用 Round Robin 调度算法**

**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**

**各个指针的用法**

1.init函数：这个函数被封装为sched\_init函数，用于调度算法的初始化，使用grep命令可以知道，该函数仅在ucore入口的init.c里面被调用进行初始化；

2.enqueue函数：该函数的功能为将指定的进程的状态置成RUNNABLE，并且放入调用算法中的可执行队列中，被封装成sched\_class\_enqueue函数，可以发现这个函数仅在wakeup\_proc和schedule函数中被调用，前者为将某个不是RUNNABLE的进程加入可执行队列，而后者是将正在执行的进程换出到可执行队列中去；

3.dequeue函数：该函数的功能为将某个在队列中的进程取出，其封装函数4.sched\_class\_dequeue仅在schedule中被调用，表示将调度算法选择的进程从等待的可5.执行的进程队列中取出进行执行；

6.pick\_next函数：该函数的封装函数同样仅在schedule中被调用，功能为选择要执行的下个进程；

7.proc\_tick函数：该函数表示每次时钟中断的时候应当调用的调度算法的功能，仅在进行时间中断的ISR中调用；

**ucore的调度执行过程**

1.在ucore中调用调度器的主体函数的代码仅存在在wakeup\_proc和schedule，前者的作用在于将某一个指定进程放入可执行进程队列中，后者在于将当前执行的进程放入可执行队列中，然后将队列中选择的下一个执行的进程取出执行；

2.当需要将某一个进程加入就绪进程队列中，则需要将这个进程的能够使用的时间片进行初始化，然后将其插入到使用链表组织的队列的对尾；这就是具体的Round-Robin enqueue函数的实现；

3.当需要将某一个进程从就绪队列中取出的时候，只需要将其直接删除即可；

4.当需要取出执行的下一个进程的时候，只需要将就绪队列的队头取出即可；

5.每当出现一个时钟中断，则会将当前执行的进程的剩余可执行时间减1，一旦减到了0，则将其标记为可以被调度的，这样在ISR中的后续部分就会调用schedule函数将这个进程切换出去；

**请在简要说明如何设计实现”多级反馈队列调度算法“**

1.在proc\_struct中添加总共N个多级反馈队列的入口，每个队列都有着各自的优先级，编号越大的队列优先级约低，并且优先级越低的队列上时间片的长度越大，为其上一个优先级队列的两倍；并且在PCB中记录当前进程所处的队列的优先级；

2.处理调度算法初始化的时候需要同时对N个队列进行初始化；

3.在处理将进程加入到就绪进程集合的时候，观察这个进程的时间片有没有使用完，如果使用完了，就将所在队列的优先级调低，加入到优先级低1级的队列中去，如果没有使用完时间片，则加入到当前优先级的队列中去；

4.在同一个优先级的队列内使用时间片轮转算法；

5.在选择下一个执行的进程的时候，有限考虑高优先级的队列中是否存在任务，如果不存在才转而寻找较低优先级的队列；（有可能导致饥饿）

6.从就绪进程集合中删除某一个进程就只需要在对应队列中删除即可；

7.处理时间中断的函数不需要改变；

至此完成了多级反馈队列调度算法的具体设计；