**Lab5——用户进程管理**

**Lab6——调度器**

实验目的

1. 了解第一个用户进程创建过程 了解系统调用框架的实现机制
2. 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
3. 理解操作系统的调度管理机制
4. 熟悉 ucore 的系统调度器框架，以及缺省的Round-Robin 调度算法
5. 基于调度器框架实现一个(Stride Scheduling)调度算法来替换缺省的调度算法

实验内容

1. 实验4完成了内核线程，但到目前为止，所有的运行都在内核态执行。实验5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。相关原理介绍可看附 录B。
2. 实验五完成了用户进程的管理，可在用户态运行多个进程。但到目前为止，采用的调度策略 是很简单的FIFO调度策略。本次实验，主要是熟悉ucore的系统调度器框架，以及基于此框架 的Round-Robin（RR） 调度算法。然后参考RR调度算法的实现，完成Stride Scheduling调度 算法。

实验结果

Lab5练习3：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。

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：



