****

**《操作系统原理实验》**

**实验报告**

**（综合实验）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 数据科学与计算机学院 | | | | | |
| **专业（班级）** | **：** | 17软件工程2班 | | | | | |
| **学生姓名** | **：** | 赵孟阳 | | | | | |
| **学号** | **：** | 17343160 | | | | | |
| **时间** | **：** | 2019 | 年 | 8 | 月 | 10 | 日 |

目录

[一、 实验题目 2](#_Toc16310268)

[二、 实验目的 2](#_Toc16310269)

[三、 实验内容 2](#_Toc16310270)

[四、 实验要求 3](#_Toc16310271)

[五、 实验过程 3](#_Toc16310272)

[1. 实验内容1：完成ucore文件系统 3](#_Toc16310273)

[ 练习0：填写已有实验 3](#_Toc16310274)

[ 练习1：完成读文件操作的实现 4](#_Toc16310275)

[ 练习2：完成基于文件系统的执行程序机制的实现 7](#_Toc16310276)

[ 运行结果 11](#_Toc16310277)

[ 实验总结 12](#_Toc16310278)

[2. 实验内容2：修改ucore调度器为采用多级反馈队列调度算法 13](#_Toc16310279)

[六、 实验心得 18](#_Toc16310280)

[七、 附件 18](#_Toc16310281)

# 实验题目

UCORE综合实验

# 实验目的

* 考察对操作系统的文件系统的设计实现了解；
* 考察操作系统进程调度算法的实现。
* 考察操作系统内存管理的虚存技术的掌握

# 实验内容

1. 在前面ucore实验lab1-lab7的基础上，完成ucore文件系统(参见ucore\_os\_docs.pdf中的lab8及相关视频)；

2. 在上述实验的基础上，修改ucore调度器为采用多级反馈队列调度算法的，队列共设6个优先级（6个队列），最高级的时间片为q(使用原RR算法中的时间片），并且每降低1级，其时间片为上一级时间片乘2（参见理论课）；

3. （选做，加分题）在上述实验的基础上，修改虚拟存储中的页面置换算法为某种工作集页面置换算法，具体如下：

* 对每一用exec新建进程分配3帧物理页面；
* 当需要页面置换时，选择最近一段时间缺页次数最少的进程中的页面置换到外存；
* 对进程中的页面置换算法用改进的clock页替换算法。
* 在一段时间（如1000个时间片）后将所有进程缺页次数清零，然后重新计数。

# 实验要求

1. 最终提交doc、docx或pdf文件和源码
2. 填写各个基本练习中要求完成的报告内容
3. 完成实验后，请分析ucore\_lab中提供的参考答案，并请在实验报告中说明你的实现与参考答案的区别
4. 列出你认为本实验中重要的知识点，以及与对应的OS原理中的知识点，并简要说明你对二者的含义，关系，差异等方面的理解（也可能出现实验中的知识点没有对应的原理知识点）

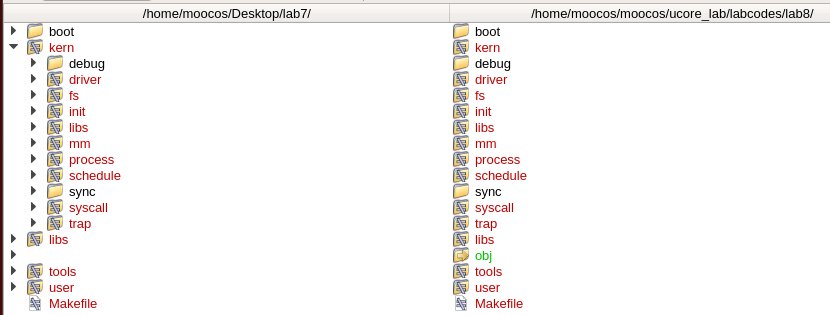
5）列出你认为OS原理中很重要，但在实验中没有对应上的知识点

# 实验过程

## 实验内容1：完成ucore文件系统

### 练习0：填写已有实验

本实验依赖实验1/2/3/4/5/6/7。请把你做的实验1/2/3/4/5/6/7的代码填入本实验中代码中 有“LAB1”/“LAB2”/“LAB3”/“LAB4”/“LAB5”/“LAB6” /“LAB7”的注释相应部分。并确保编译通过。 注意：为了能够正确执行lab8的测试应用程序，可能需对已完成的实验1/2/3/4/5/6/7的代码进行进一步改进。



实验1-7的相关代码通过understand/diff进行比对并填入lab8中的相应注释部分

### 练习1：完成读文件操作的实现

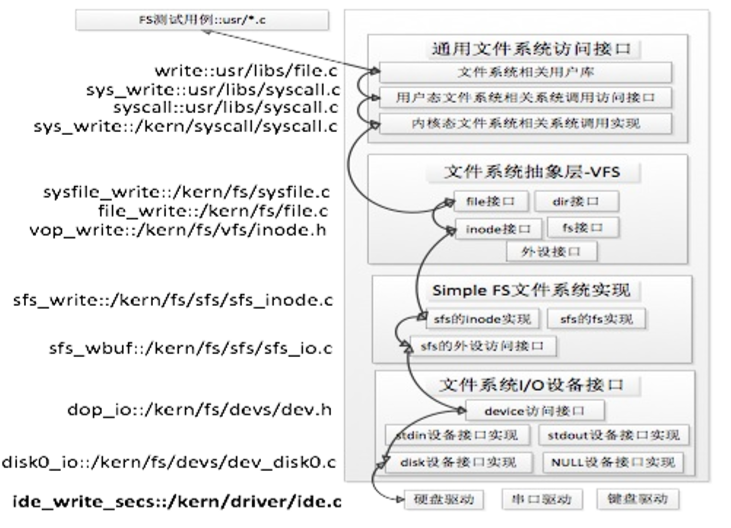
首先了解打开文件的处理流程，然后参考本实验后续的文件读写操作的过程分析，编写在 sfs\_inode.c中sfs\_io\_nolock读文件中数据的实现代码。

ucore文件系统的设计模仿了UNIX, 它的文件系统架构由四部分组成

1. 通用文件系统访问接口层：该层提供了一个从用户空间到文件系统的标准访问接口。这一层访问接口让应用程序能够通过一个简单的接口获得ucore内核的文件系统服务。
2. 文件系统抽象层：向上提供一个一致的接口给内核其他部分（文件系统相关的系统调用实现模块和其他内核功能模块）访问。向下提供一个同样的抽象函数指针列表和数据结 构屏蔽不同文件系统的实现细节。
3. Simple FS文件系统层：一个基于索引方式的简单文件系统实例。向上通过各种具体函数 实现以对应文件系统抽象层提出的抽象函数。向下访问外设接口
4. 外设接口层：向上提供device访问接口屏蔽不同硬件细节。向下实现访问各种具体设备驱动的接口，比如disk设备接口/串口设备接口/键盘设备接口等。

**操作文件的一般流程：**

当应用程序操作文件（打开/创建/删除/读写）时，首先需要通过文件系统的通用文件系统访问接口层给用户空间提供的访问接口进入文件系统内部，接着由文件系统抽象层把访问请求转发给某一具体文件系统（比如SFS文件系统），具体文件系统（Simple FS文件系统层）把应用程序的访问请求转化为对磁盘上的block的处理请求，并通过外设接口层交给磁盘驱动例程来完成具体的磁盘操作。



**在ucore中，打开文件的具体过程为**

以user/sfs\_filetest1.c为例，首先用户进程会调用在main函数中的如下语句：

int fd1 = safe\_open("sfs\\_filetest1", O\_RDONLY);

如果ucore能够正常查找到这个文件，就会返回一个代表文件的文件描述符fd1

1. 通用文件访问接口层的处理流程

调用用户态函数： open->sys\_open->syscall，从而引起系统调用进入到内核态。进入内核态后，通过中断处理例程，调用sys\_open内核函数，并进一步调用sysfile\_open内核函数。由copy\_path完成把位于用户空间的字符串"sfs\_filetest1"拷贝到内核空间中的字符串path中，进入到文件系统抽象层的处理流程完成进一步的打开文件操作中。

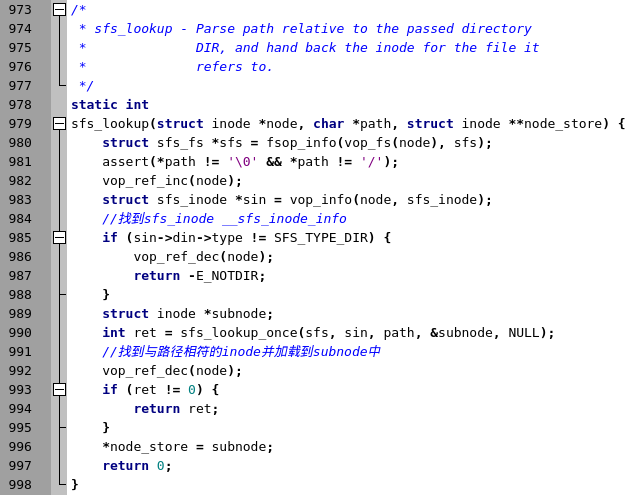
1. 文件系统抽象层的处理流程

首先分配一个空闲的file数据结构变量file，它是当前进程的打开文件数组current->fs\_struct->filemap[]中的一个空闲元素（即还没用于一个打开的文件），它的索引值最终要返回到用户进程并赋值给变量fd1，但此时还没有找出对应的文件索引点。接着调用vfs\_open函数来找到path指出的文件所对应的基于inode数据结构的VFS索引节点node，并把file和node建立联系，设置file的读写权限。完成后，将返回到file\_open函数中，通过执行语句“file->node=node;”，就把当前进程的current->fs\_struct->filemap[fd]（即file所指变量）的成员变量node指针指向了代表sfs\_filetest1文件的索引节点inode。经过重重回退，最终把fd返回赋值给fd1，完成了打开文件操作。

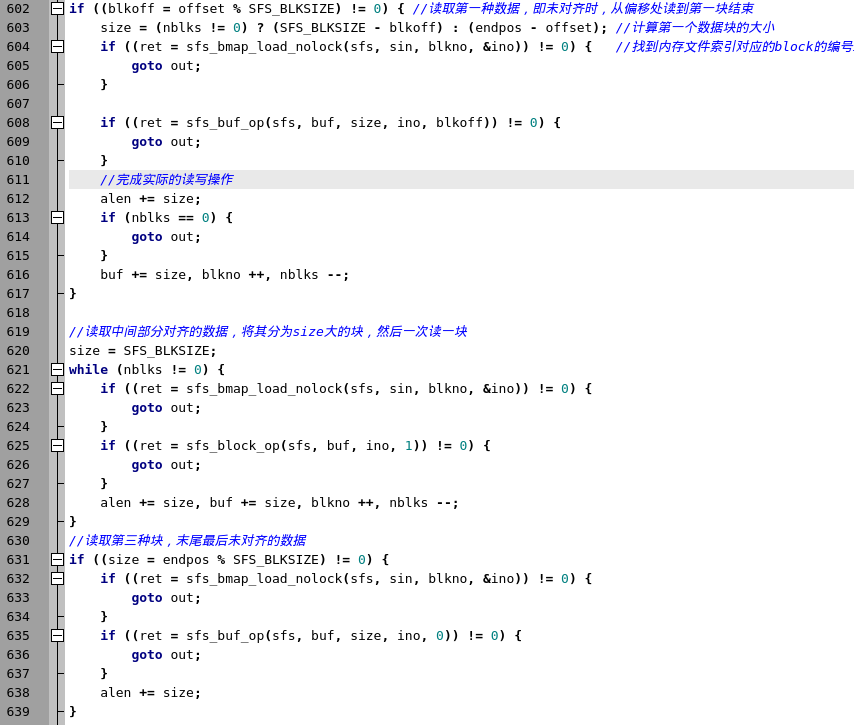
1. SFS文件系统层的处理流程

在sfs\_inode.c中的sfs\_node\_dirops变量定义了“.vop\_lookup = sfs\_lookup”，分析sfs\_lookup的实现。

sfs\_lookup有三个参数：node，path，node\_store。其中node是根目录“/”所对应的inode节点；path是文件sfs\_filetest1的绝对路径/sfs\_filetest1，而node\_store是经过查找获得的sfs\_filetest1所对应的inode节点。 sfs\_lookup函数以“/”为分割符，从左至右逐一分解path获得各个子目录和最终文件对应的inode节点。在本例中是调用sfs\_lookup\_once查找以根目录下的文件sfs\_filetest1所对应的inode节点。当无法分解path后，就意味着找到了sfs\_filetest1对应的inode节点，就可顺利返回了



**补充sfs\_inode.c中sfs\_io\_nolock读文件中数据的实现代码：**



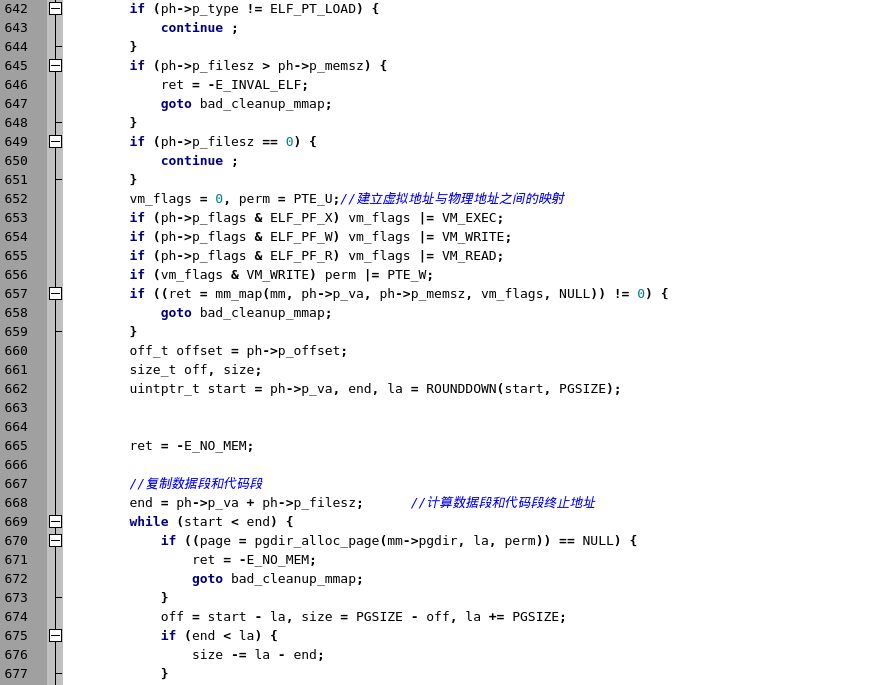
### 练习2：完成基于文件系统的执行程序机制的实现

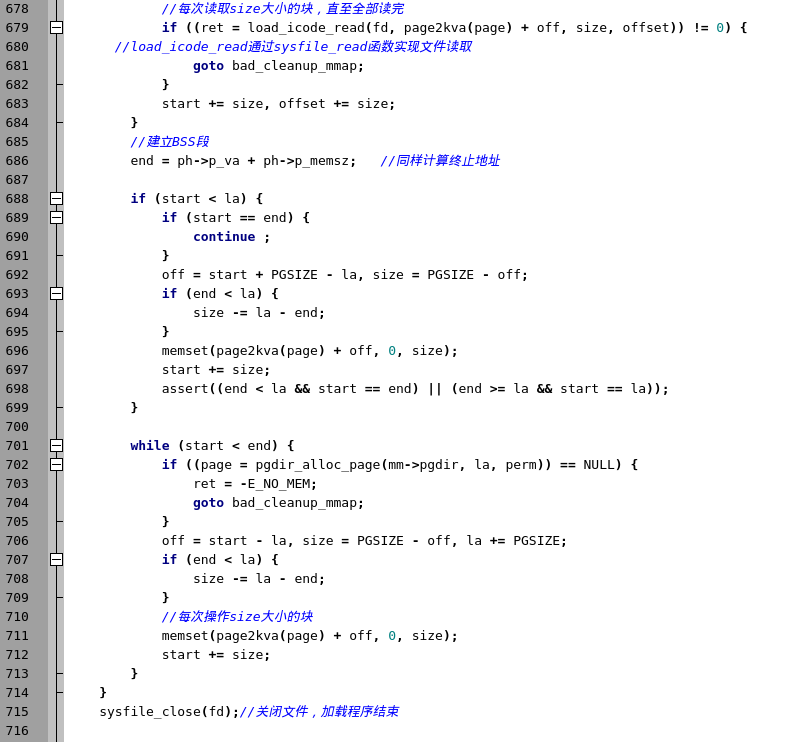
改写proc.c中的load\_icode函数和其他相关函数，实现基于文件系统的执行程序机制。

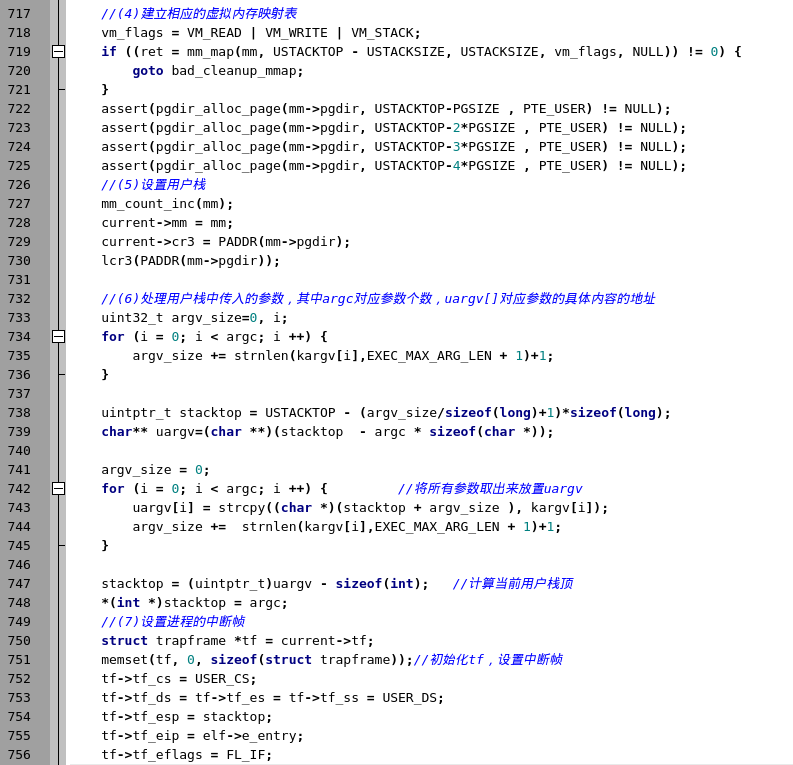
通过修改，读elf文件由直接从内存中读改为了从磁盘上读

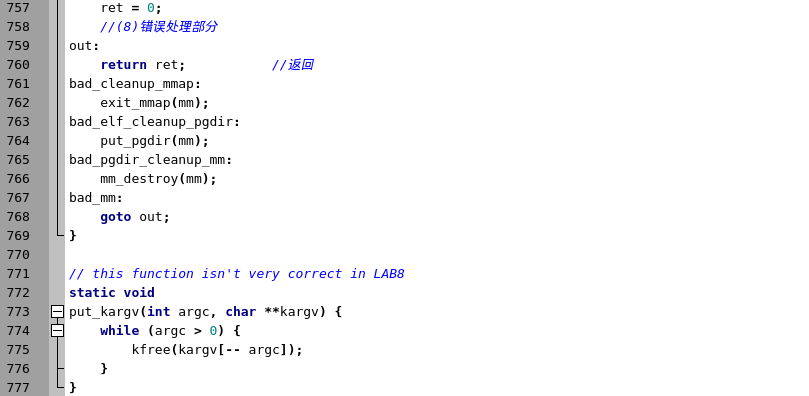
Load\_icode作用为将文件加载到内存中执行，一共分为7步，根据注释填写proc.c中的load\_icode代码





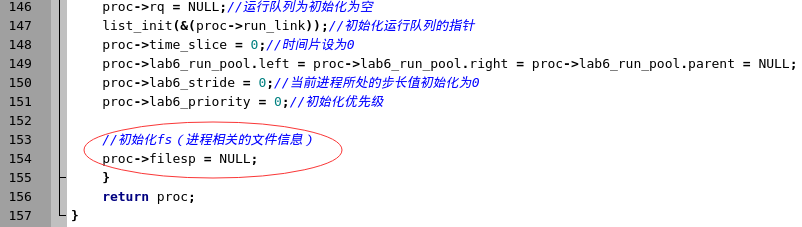




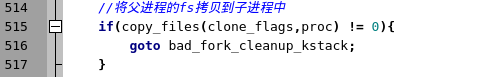


对proc.c文件中其他注释有lab8的部分进行填写

alloc\_proc()函数，添加对proc中fs的初始化

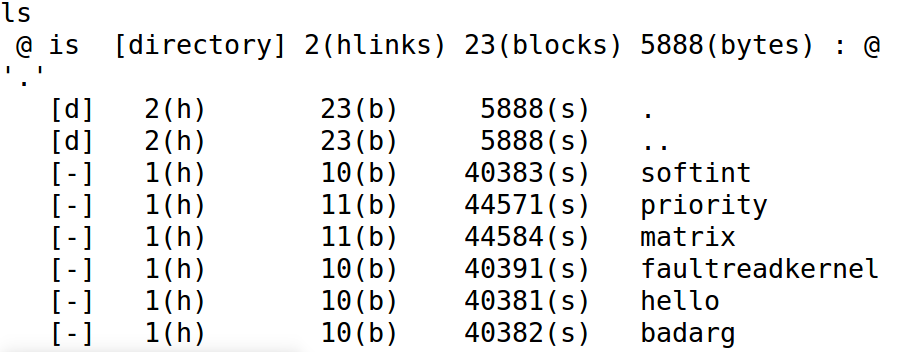


do\_fork()函数中添加

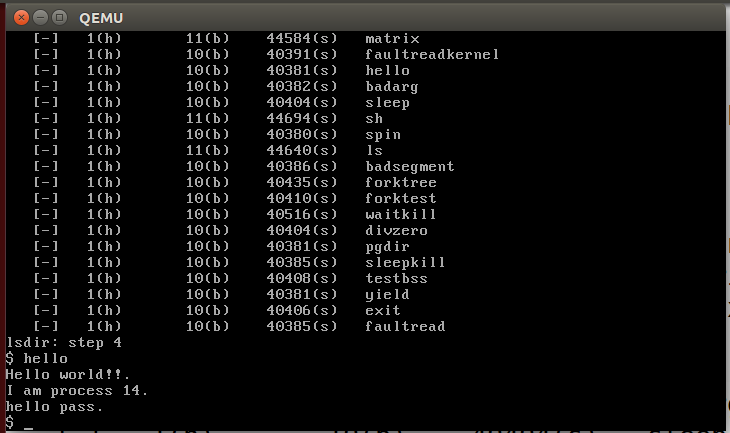


### 运行结果

在终端运行make qemu, 在界面上执行ls，列出了文件目录



输入hello，成功执行



### 实验总结

1. 列出你认为本实验中重要的知识点，以及与对应的OS原理中的知识点，并简要说明你对二者的含义，关系，差异等方面的理解（也可能出现实验中的知识点没有对应的原理知识点）

* VFS:提供一个面向对象实现文件系统的方式，使得一个系统的API（interface）可以被不同类型的文件系统使用，在ucore中的SFS也可以换为其他的文件系统实例化VFS接口
* 索引节点（inode）: 在SFS文件系统中，需要记录文件内容的存储位置以及文件名与文件内容的对应关系。sfs\_disk\_inode记录了文件或目录的内容存储的索引信息，sfs\_disk\_entry表示一个目录中的一个文件或目录，包含该项所对应inode的位置和文件名，均存储在硬盘中，需要时读入内存。
* 文件目录也是文件，普通文件保存文件中的数据，而目录保存目录下的所有文件名。
* 文件系统挂载（mounting）:sfs\_init中调用了sfs\_mount->vfs\_mount将一个文件系统挂载到系统上，在本实验中为Simple FS，从而让ucore的其他部分能够通过虚拟文件系统的接口来进一步访问SFS实例文件系统
* 文件的基本操作：open,close,read,write
* 文件和目录访问相关系统调用:与文件相关的open、close、read、write用户库函数对应的是sys\_open、sys\_close、sys\_read、sys\_write四个系统调用接口。与目录相关的readdir用户库函数对应的是sys\_getdirentry系统调用。这些系统调用函数接口将通过syscall函数来获得ucore的内核服务。

2）列出你认为OS原理中很重要，但在实验中没有对应上的知识点

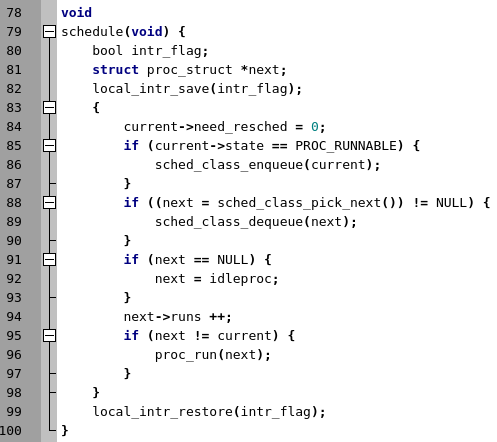
* 强制锁和建议锁
* 多用户文件共享
* FAT文件及文件目录结构
* 空闲空间管理方法
* 一致性检查及恢复方法

等在本实验中未涉及

## 实验内容2：修改ucore调度器为采用多级反馈队列调度算法

要求：队列共设6个优先级（6个队列），最高级的时间片为q(使用原RR算法中的时间片），并且每降低1级，其时间片为上一级时间片乘2

在实验7了解到，调度框架由sched\_class\_enqueue ， sched\_class\_dequeue ， sched\_class\_pick\_next ， sched\_class\_proc\_tick等函数构成，实现进程入队、出队、挑选进程、时间片判断等功能，它们在调度器中的应用如下



在实现多级反馈队列调度算法中我们通过分析该算法原理，完成入队、出队、挑选进程、时间片判断的具体实现并修改相应的结构，基本上就可以实现该算法

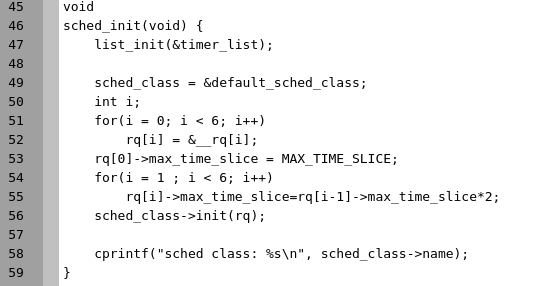
**数据结构调整：**

1. 修改等待队列：由原来的一个等待队列改为6个不同优先级的队列，队列编号越大优先级越低，每降低1级，其时间片为上一级时间片乘2

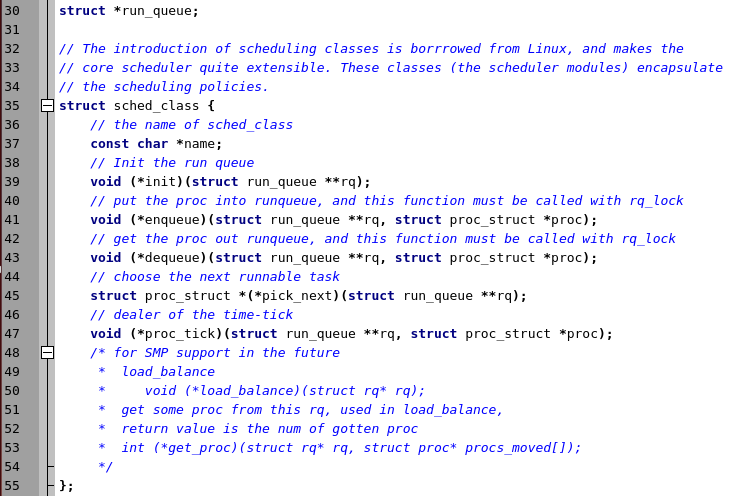


…





1. 修改调度器接口sched\_class:将sched.h中调度器接口使用的函数上的队列参数由一位指针转为二维指针并相应修改调用sched\_class的函数之处



1. PCB: proc\_struct 中需要新增属性queueIndex来记录进程所处的队列优先级，并且进程初始化时 PCB 中的所处队列优先级要设为最高等级

Proc.h:



Proc.c:



分析：

新进程进入内存后，先投入队列0的末尾，按FCFS算法调度

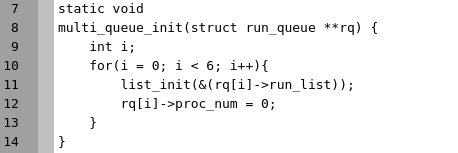
队列在一个时间片内未执行完，则降低优先级投入下一级队列末尾，按FCFS算法调度若队列处于队列5，则仍加入队列5末尾，按照时间片轮转算法调度

当一个进程执行时，如果有进程进入较高优先级队列，则抢占执行新进程，被抢占进程加入原队列末尾

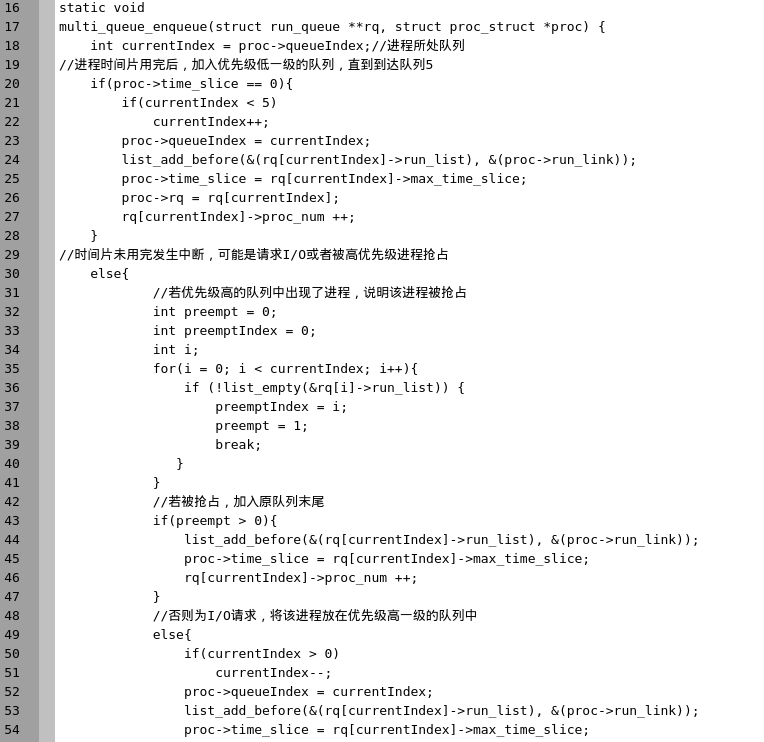
当一个进程因请求I/O让出了CPU，那么在I/O完成后，进程将进入比优先级比I/0请求时离开的队列高一级的队列中

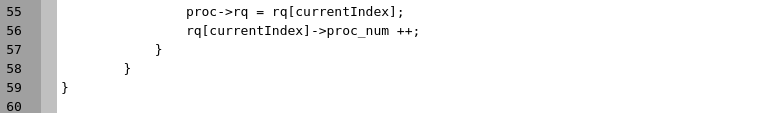
**主要函数实现：**

1. 调度器初始化：初始化6条队列的链表和对应的进程数

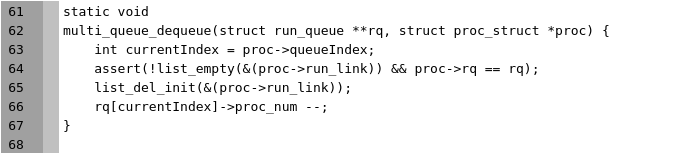


1. 进程入队：正常情况下，当进程时间片用完后，就加入优先级低一级的队列的末尾；否则当时间片没有用完发生调度时，可以根据优先级更高的队列中有无进程来判断该进程是被抢占还是发生了I/O调度

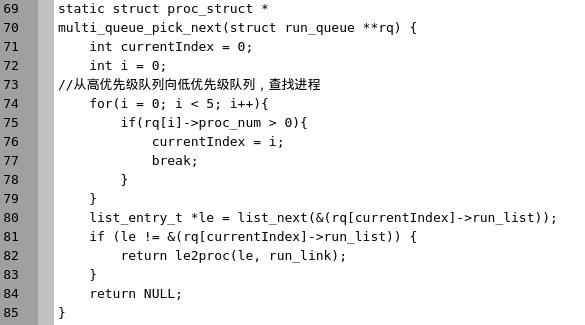




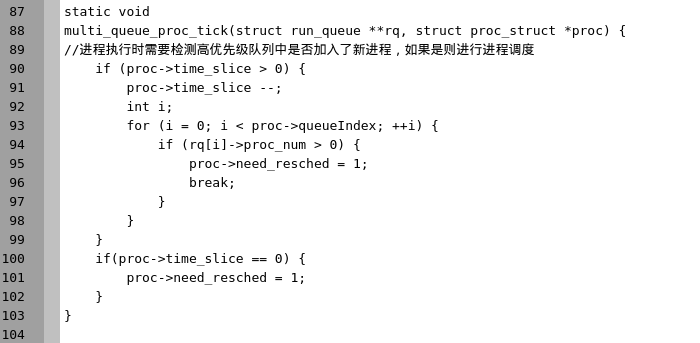
1. 进程出队：基本不变，将队头元素从所在队列删除并相应减少队列中进程数



1. 挑选下一个进程：根据优先级从高到低扫描，找到第一个进程



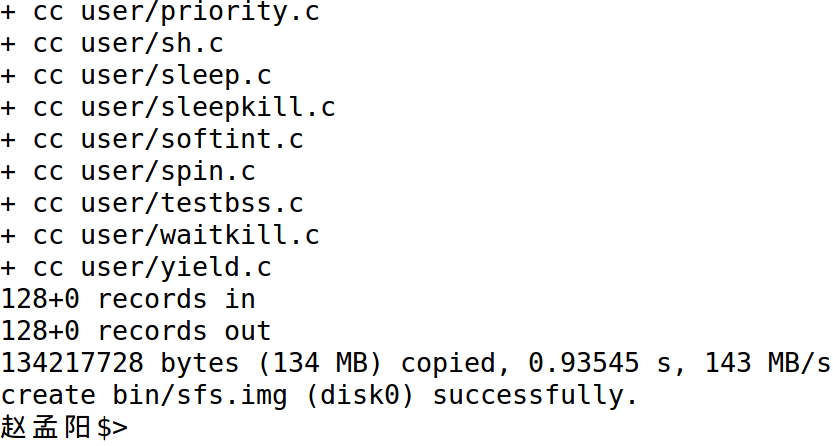
1. 时钟中断：每次时间片减一，当时间片不为0时，每次都检查是否有新进程进入更高优先级的队列，若有进行调度。当时间片用完时，也发生调度。



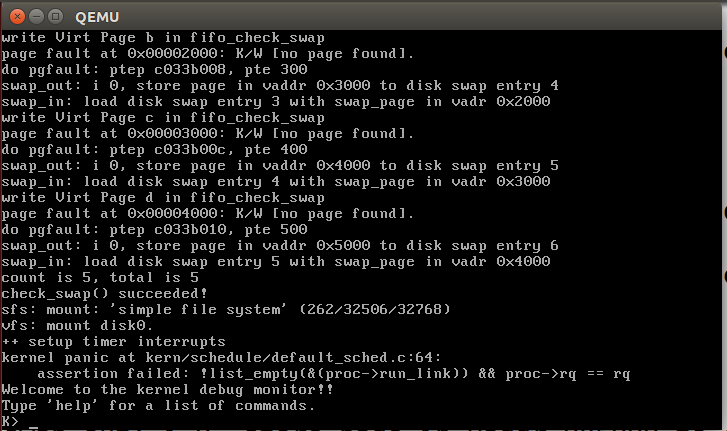
**实验结果：**

编译成功





Make qemu结果如下：



# 实验心得

通过完成实验内容一，我对ucore的文件系统整体结构有了一定了解，对打开磁盘中的文件过程以及索引节点的使用更加熟悉。本实验中的文件系统用到了很多理论课中讲到的知识，通过填写函数，在系统的抽象结构和物理实现之间体会到了一些关联。

多级反馈队列算法的实现主要是6个等待队列的设置和处理引起的变化以及分析原理来完成进程调度框架中的函数实现。进程由于请求I/O让出CPU，进入队列时优先级提高的情况容易被忽略，在写程序时根据思路寻找要改的地方，写起来比之前想的要容易一些。通过实验，对多级反馈队列算法的原理以及初始化，入队，出队等操作的细节有了更新的认识。

# 附件

Lab8.1(实验内容1完成后压缩包，文件操作正常)

Lab8.2（实验内容2完成后压缩包）