**操作系统实验5报告**

**软件框图**

**主流程及其实现**

在上次实验的基础上，增加下列内容。

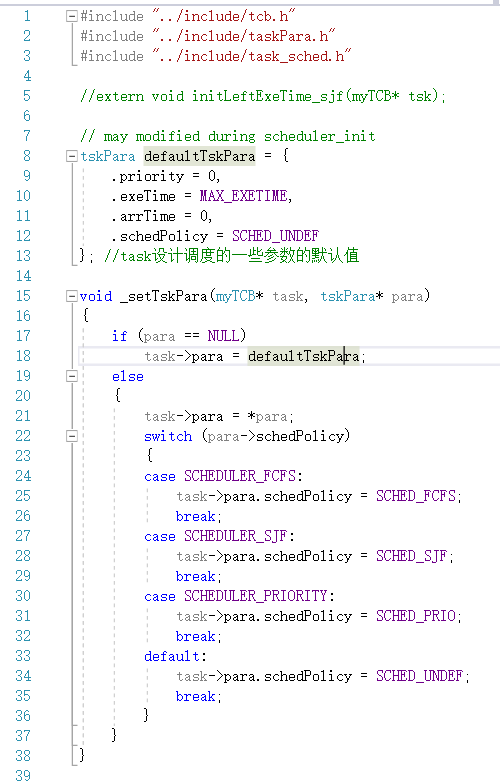
**主要功能模块及其实现**

本次实验中要实现的功能模块分别按五个部分实现

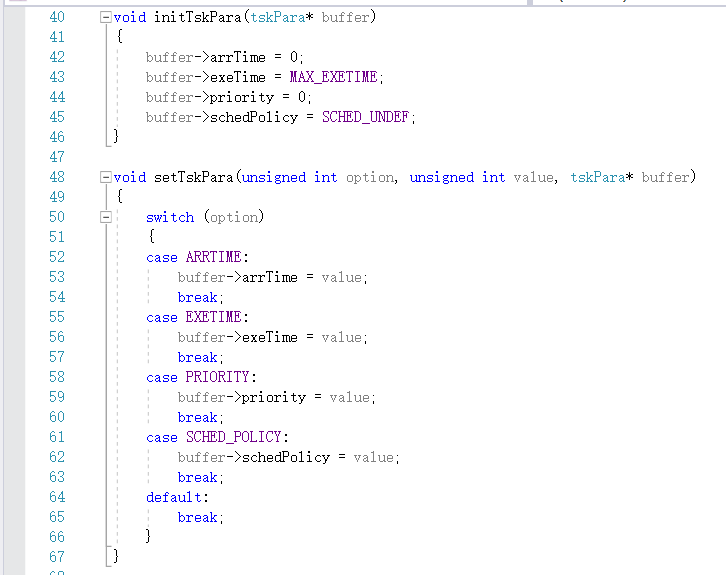
**源代码说明**

在本次实验的根目录下，除了配置了makefile与DS文件，还有4个文件夹：multibootheader、myOS、output、userApp 。multibootheader中放置了有关multibootheader协议的S文件。在output中，原本为空文件夹，用来输出编译后生成的文件。在该文件夹中的目录结构与根目录相似，主要是为了对每个文件输出时不产生混淆。在userApp中存放的是main文件、memTestCase文件、userTasks文件与该文件夹下的makefile文件，本次实验中的userTasks是提供的测试Task相关功能文件。在myOS文件夹中存放了本次实验的主要源文件。直接存放在此该目录下的文件有DS文件、该文件夹下的makefile文件、链接器文件、一个配置地址的S文件和一个包含了main函数的与multibootheader中的S文件相关联的C文件。这个文件也是从汇编到C的转变。在该目录下则有6个实现相关功能的文件夹：dev、i386、printk、kernel、lib、include。每个文件夹下都有一个相应的makefile文件以及实现功能的C语言源文件。本次实验中在include中添加了task\_arr.h、task\_sched.h、taskPara.h文件，主要功能是对应的C文件中的函数声明与相关宏定义。此外，我在include中自行添加了颜色宏的头文件，方便用宏定义写入myPrint的颜色参数。kernel文件夹中增加task\_arr.c、task\_sched.c、taskPara.c源代码文件以实现任务调度算法的相关功能，以及task\_sched文件夹存放FCFS、SJF、PRIO等调度算法的源代码文件。每个子目录下的makefile文件的输出目录都是在output中的同名目录。下面展示相关源代码。

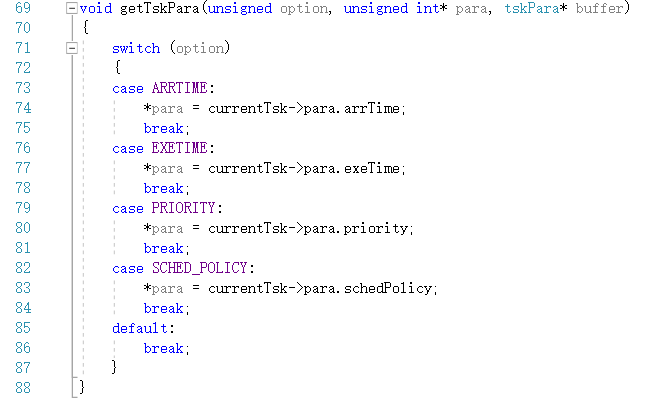
下面展示本次实验的框架中所要求实现的部分。



task\_para.c代码部分1，关于参数的默认值以及对参数的赋值实现



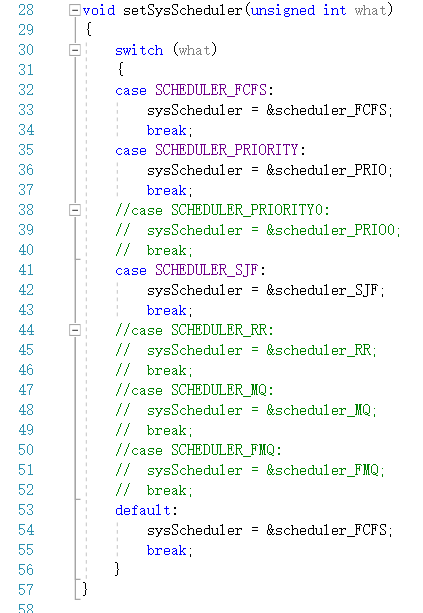
task\_para.c代码部分2，主要实现初始化参数和对参数成员的指定赋值。



task\_para.c代码部分3，获得指定参数功能的实现



task\_arr.c代码，实现创建节点并按到达时间加入队列



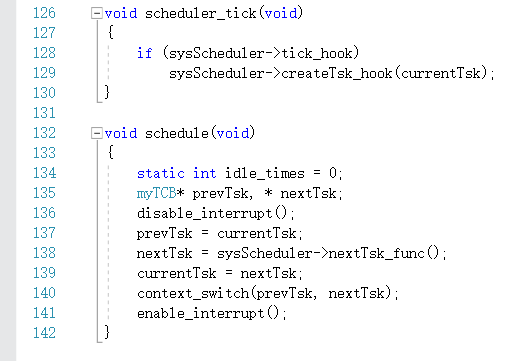
task\_sched.c代码部分1，主要实现调度器的选择。未实现的调度算法已经被注释。



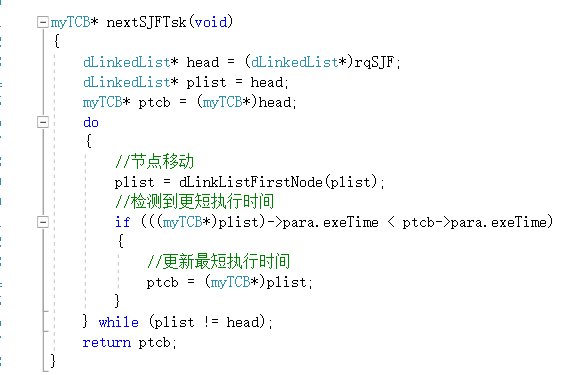
task\_sched.c代码部分2，主要实现了调度器参数获取和赋值



task\_sched.c代码部分3，主要实现调度器的内部函数的封装。这有利于对不同算法的统一管理



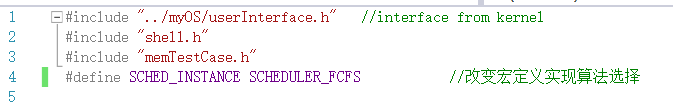
task\_sched.c代码部分4，主要实现调度过程。



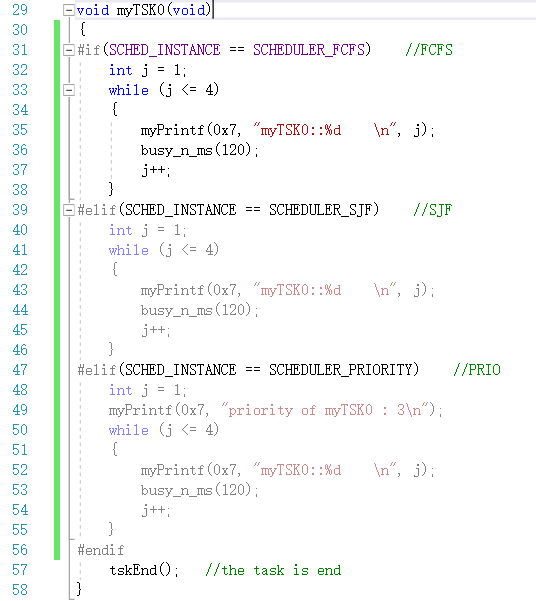
task\_SJF代码部分，由于大多数代码与FCFS代码文件所给相同，故主要展示不同的next函数的实现，PRIO算法同。不同调度算法主要是对next函数的不同实现。主要是检测当前任务队列并依次执行时间并进行比较，最终找到最短执行时间的任务并返回



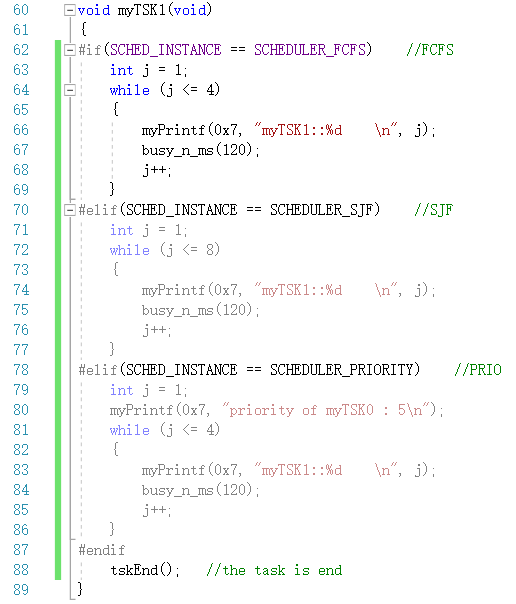
原理同上一算法，检测的成员不同。但是此算法需要避开initsk和idletsk的优先级对判断的干扰，故需要绕开此节点的prio检测。



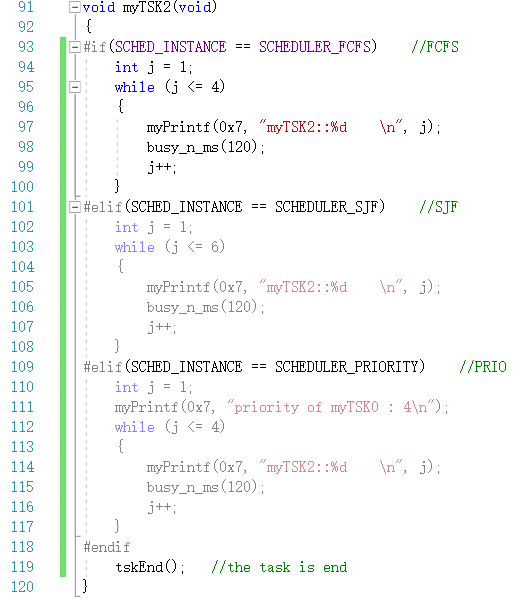
main代码部分1，本次实验为了任意在测试用例间的切换，我选择用宏定义的方法来产生对应不同的编译内容。



main代码部分2，此部分为task0在不同调度算法下的内容



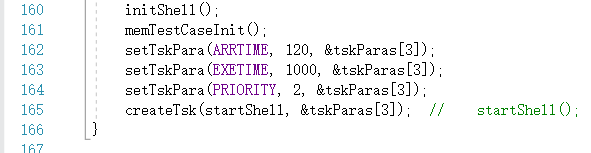
main代码部分3，此部分为task1在不同调度算法下的内容



main代码部分4，此部分为task2在不同调度算法下的内容



main代码部分5，此部分为不同调度算法下的各个task参数设置



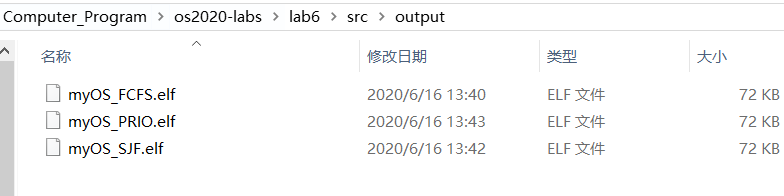
main代码部分4，此部分为Shell的参数设置与任务创建

**代码布局说明**

所有的引导模块将按页（4KB）边界对齐，物理内存地址从1M处开始。

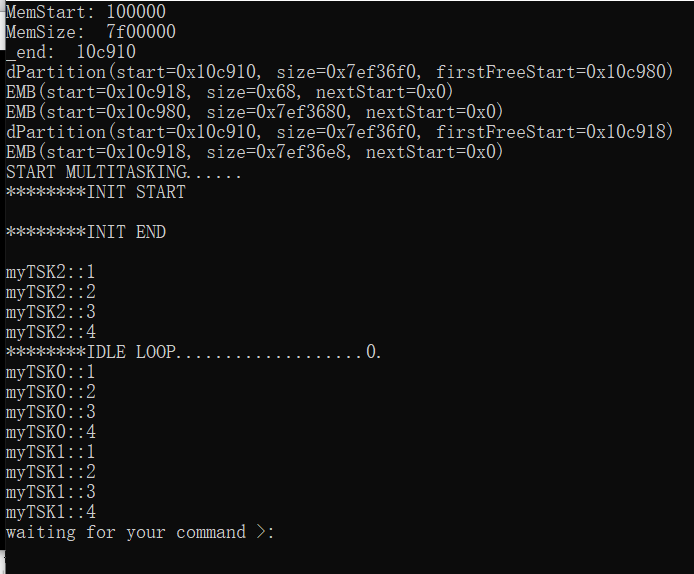
**编译过程说明：**

在Ubuntu中先搜索到lab6的目录，然后通过指令make完成编译，在output目录中的对应目录中分别输出了与根目录下对应文件相同文件。随后我将三种调度算法下编译的可执行文件分别再次命名，并且删除了中间文件，得到了三个elf文件。

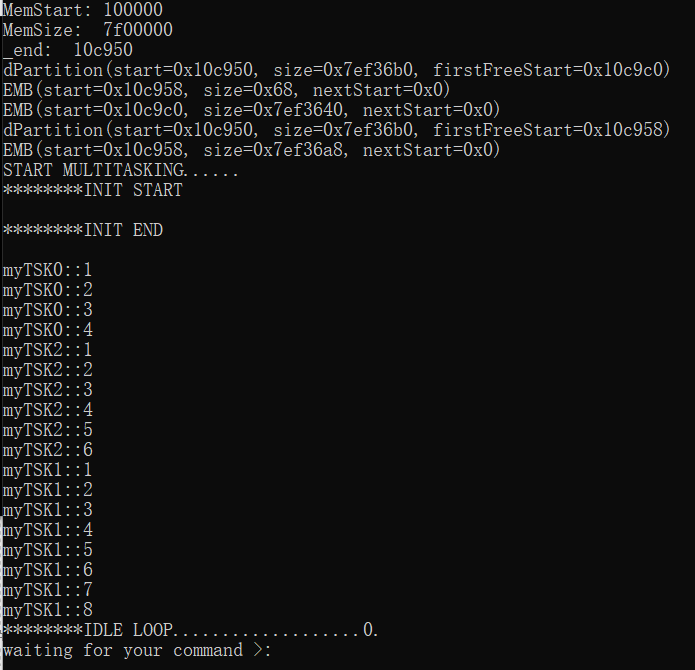


**运行和运行结果说明：**

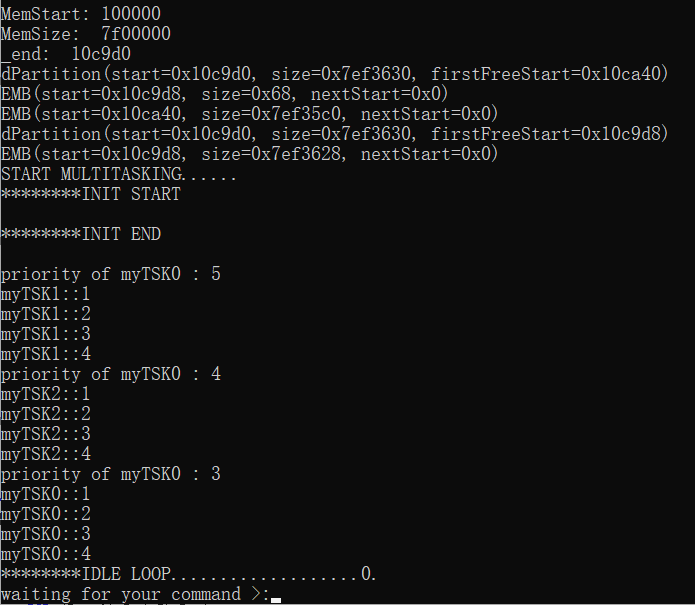
在Ubuntu中通过QEMU启动已经编译生成的bin文件，得到Linux的图形化界面运行结果，然后再通过Ubuntu启动一个交互界面，用于输入与输出。



如图是FCFS算法在screen上的运行结果。为了便于其他算法的实现，我先实现了FCFS算法验证非调度器文件的正确性。在上面的main代码中我已经修改了tsk的相关内容以避免过长的内容不便于查看完整结果。可以看到执行顺序是优先执行已经就绪的任务2，然后执行完毕时判断任务0与人物1的到达顺序，便优先执行任务0，符合目标。



如图是SJF算法在screen上的运行结果。在main中，为了显著体现出next的实现，我将其他算法中的任务到达时间统一修改为了0，以便同时比较。除此之外，每个任务的执行时间也与其执行打印循环次数成正相关，可以看到，依次执行了打印次数递增的任务。



如图是PRIO算法在screen上的运行结果。每个任务的优先级在任务开头先打印一次，可以看到，依次执行了打印优先级递减的任务。

**遇到的问题和解决方案：**

1. 不能理解调度器的具体实现方式

咨询助教并了解相关信息。

1. 不理解链表头文件的用法

咨询助教后自行了解了指针的强行转换原理并成功应用。