**实验六, 调度算法**

**李卓 pb19000064**

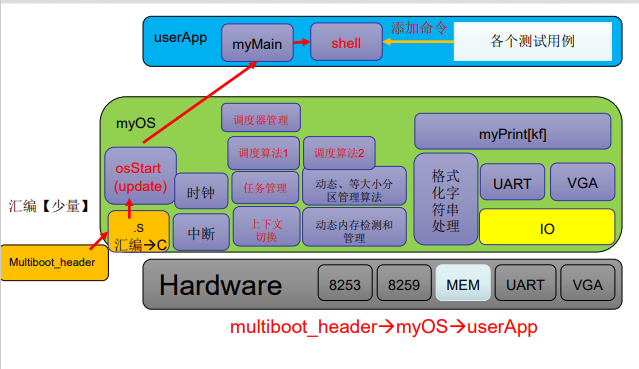
**实验目的**

1. 实现调度算法，至少2种（不含FCFS）
2. 实现支持调度算法的任务管理器
3. 根据调度算法需要修改, 任务数据结构, 任务创建/销毁, 调度器

**实验内容**

1. 实现任务随时钟动态化到达
2. 调度器和任务参数采用统一接口
3. 完成一种抢占式调度算法
4. 完成另外两种任意调度算法

**实验框架**



内核：上下文切换、任务管理和调度

用户：新功能测试

被测功能：任务创建、所实现的调度算法

自测：userApp

**实验流程**



1. 在 multiboot\_header 中完成系统的启动。

2. 在 start32.S 中做好准备，调用 osStart.c 进入 c 程序。

3. 在 osStart.c 中完成初始化 8259A，初始化 8253，清屏及内存初始化等操作，调用 myMain， 进入 userApp 部分。

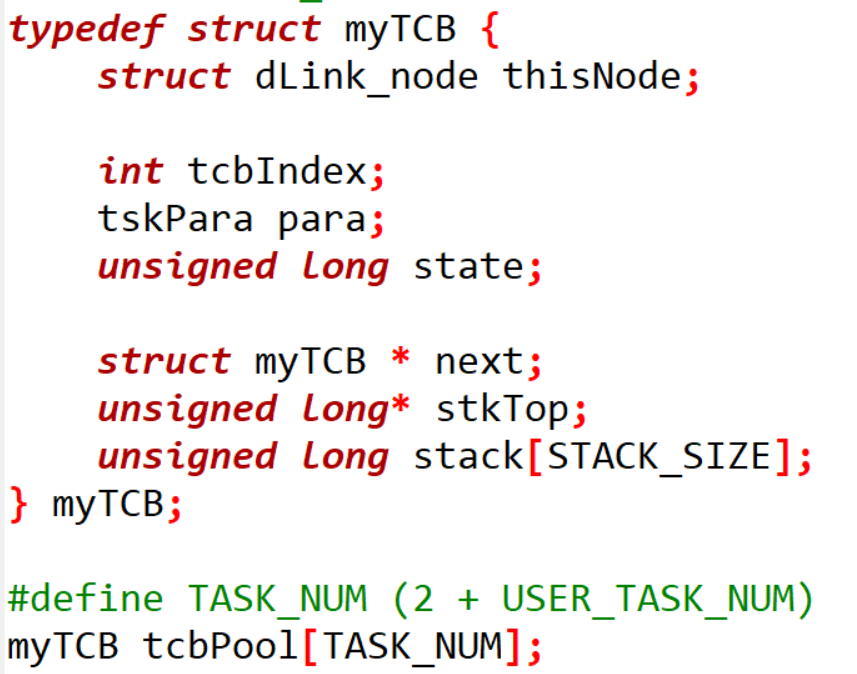
4. 运行 myMain 中的代码，进行时钟设置，shell 初始化，内存测试初始化等操作，启动 shell。

5. 进入 shell 程序，等待命令的输入

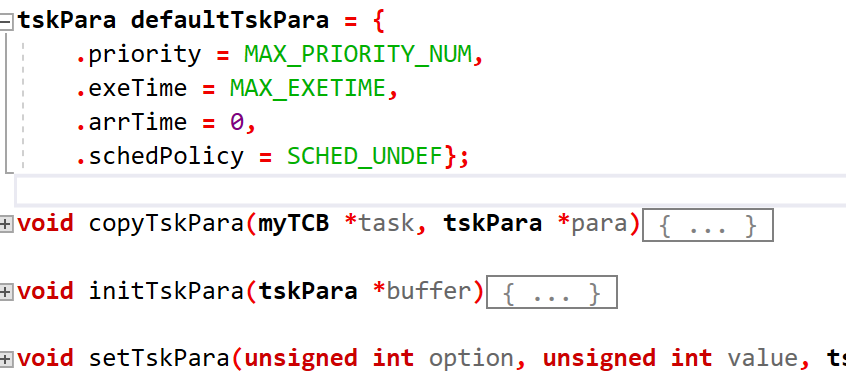
Multiboot\_header 为进入C程序准备好上下文 初始化操作系统各个模块 调用userApp入口 myMain（自测）+shell

**实验原理**

Tcb 结构 以及tcb池

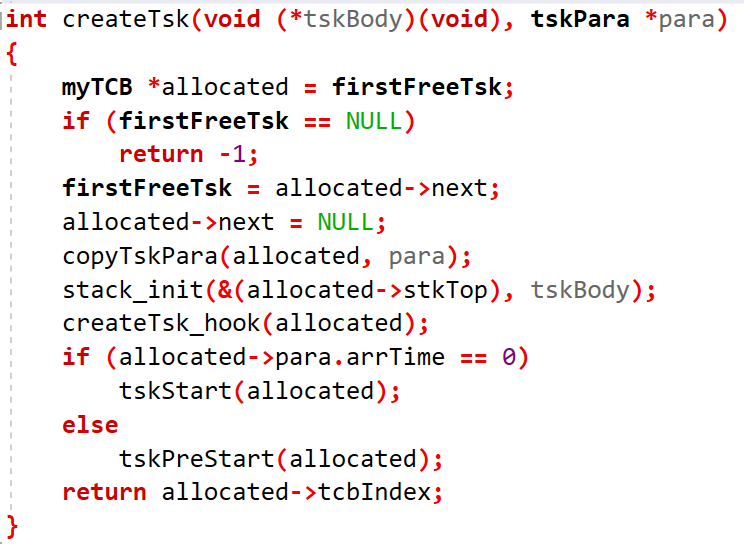


Tcb参数及其操作



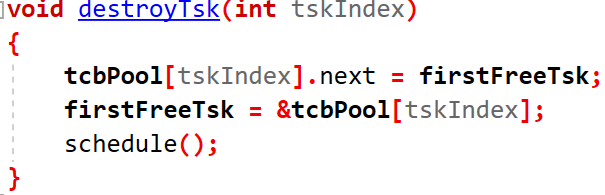
Tsk 使用链表存储

Tsk创建:

createTsk()实现 TCB 分配，对调度参数和栈初始化，对下一空闲 TCB 进行修改，若 此时为到达时间，直接调用 tskStart()启动任务. 否则调用 tskPreStart()函数对tsk放置在合适位置。   


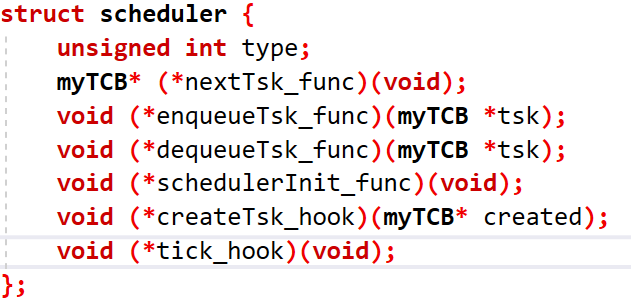
Tsk销毁

destroyTsk()实现 TCB 回收，修改TCB链表，同时调 度新任务。

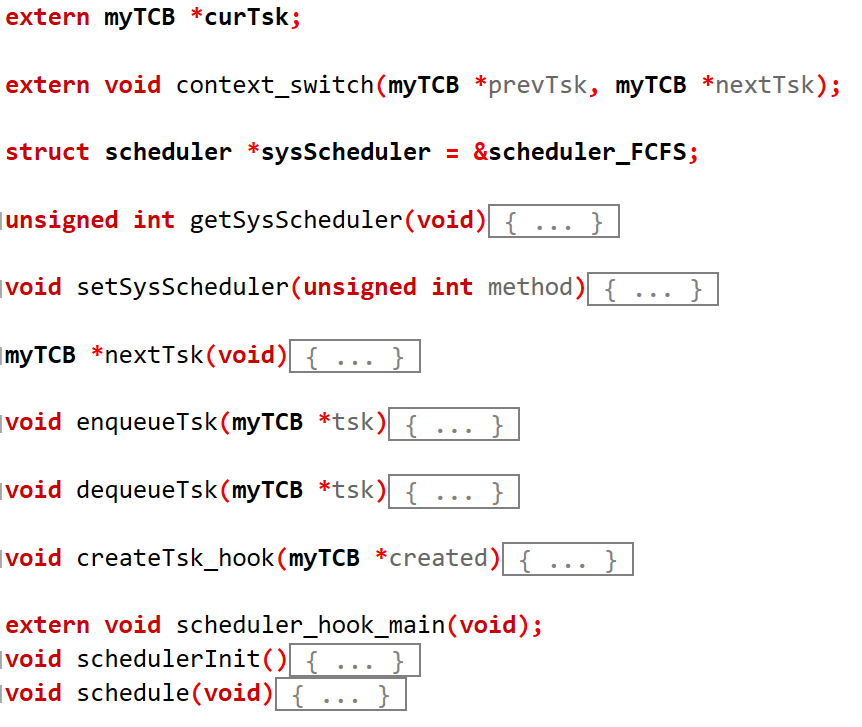


调度算法:

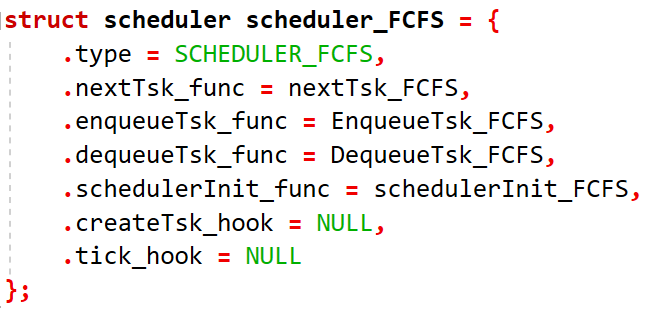
Scheduler结构



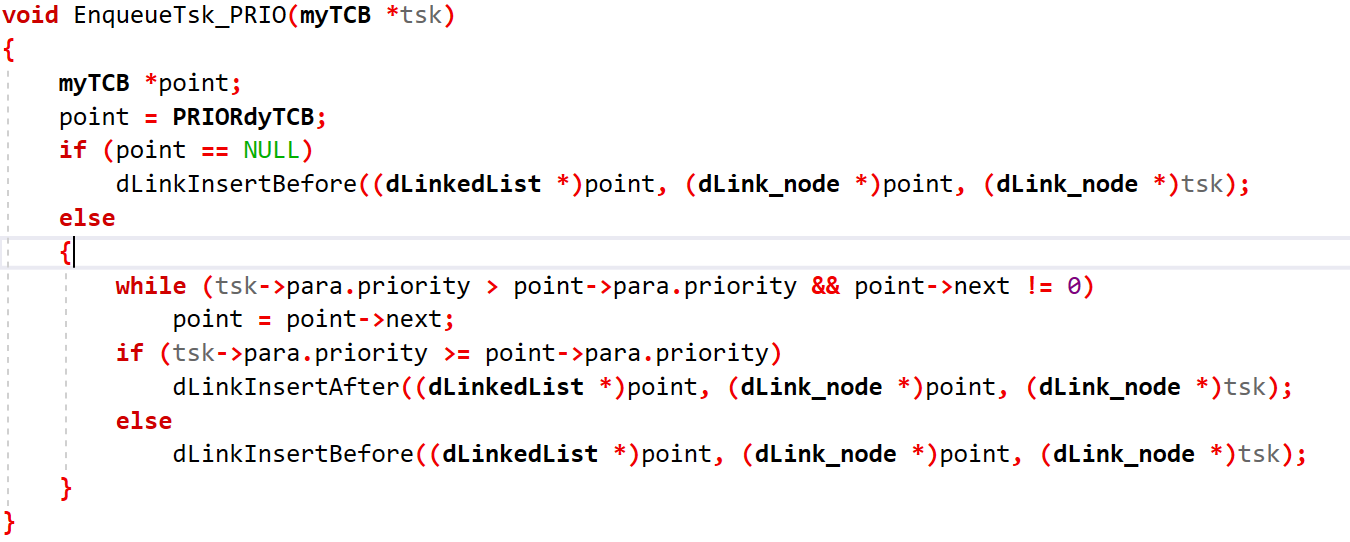
实现了统一的调度接口



利用hook机制配置相应算法

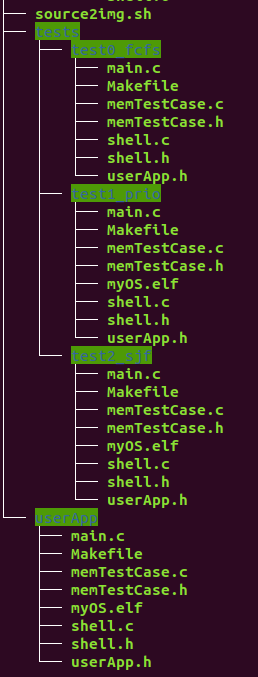
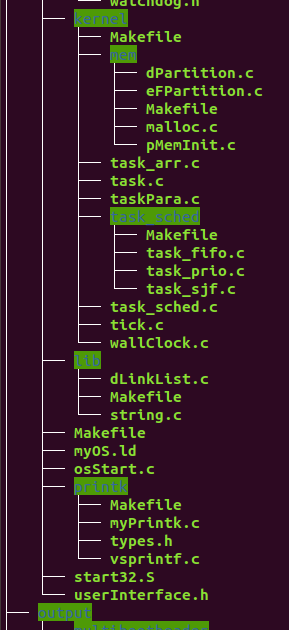


实现了Prio, FCFS, SJF算法

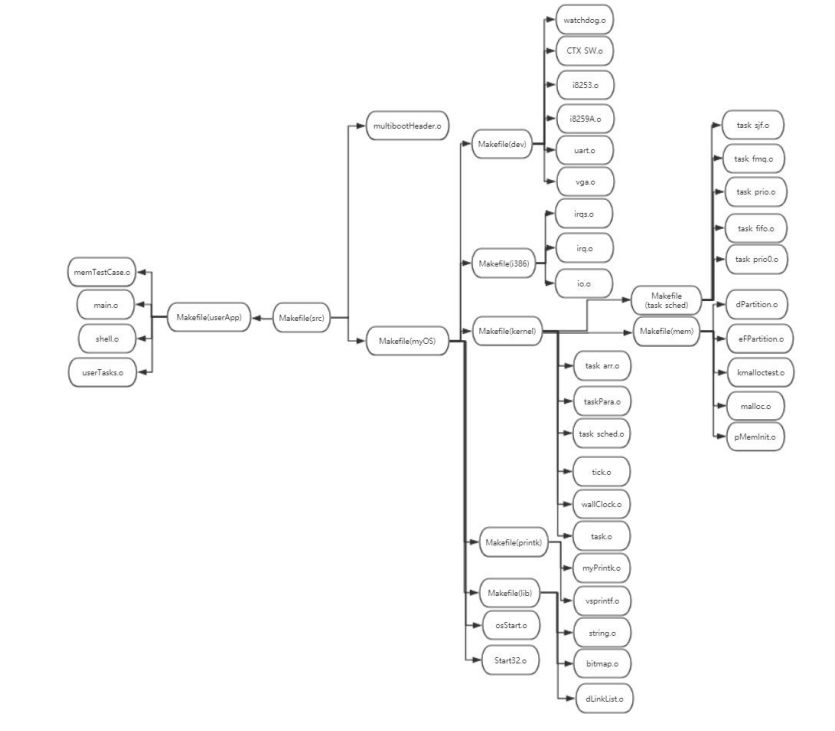
Prio算法:  


FCFS, SJF算法与PRIO本质相同, 都是将tsk序列排序, 只不过FCFS优先级是到达时间, SJF优先级是剩余运行时间.

**文件目录组织**



makefile组织



**地址空间布局**

|  |  |  |
| --- | --- | --- |
| Section | Offset (Base = 1M) | align |
| .multiboot\_header | 0 | 8 |
| .text(代码段) | 16 | 8 |
| .data(数据段) | 16+.text section | 16 |
| .bss | 当前 | 16 |
| 堆栈(动态内存空间) | 当前 |  |

**编译过程说明**

默认方式, 链接生成 myOS.elf 文件

chmod 777 source2run.sh

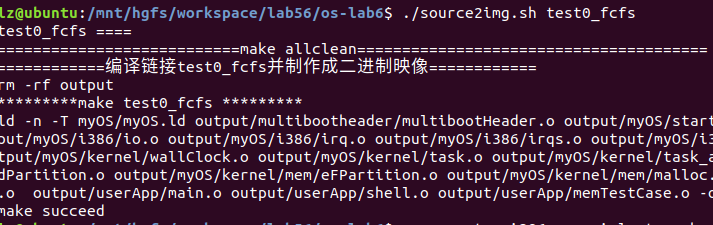
./source2run.sh test0\_fcfs

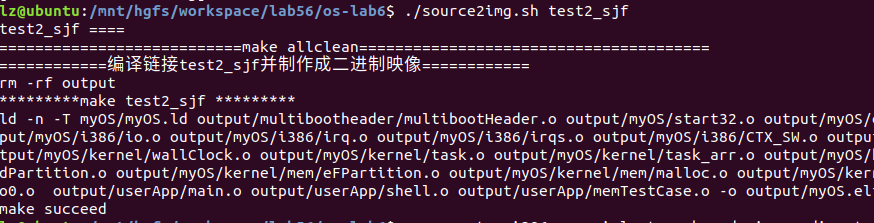
./source2run.sh test1\_prio

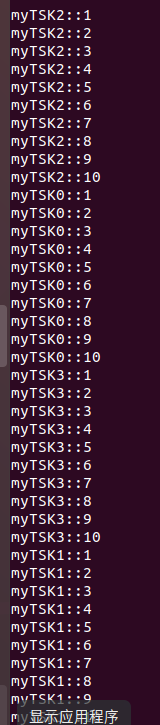
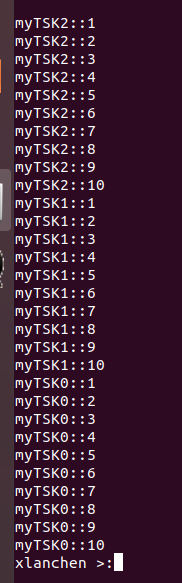
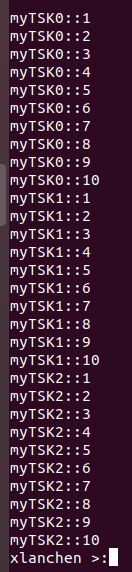
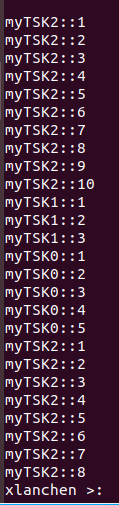
./source2run.sh test2\_fjs

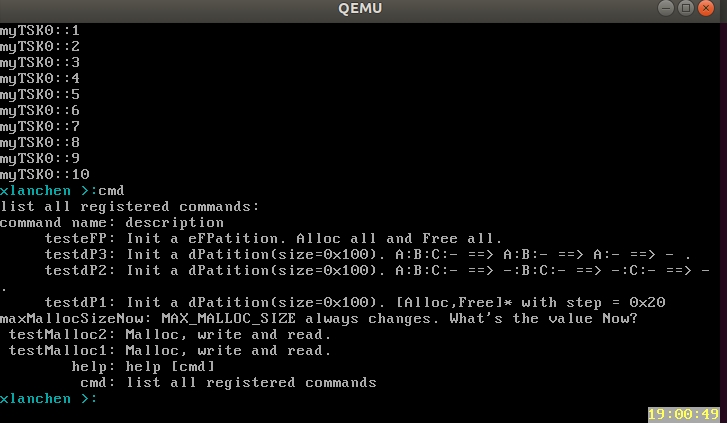
sudo screen /dev/pts/1

**运行结果**

****

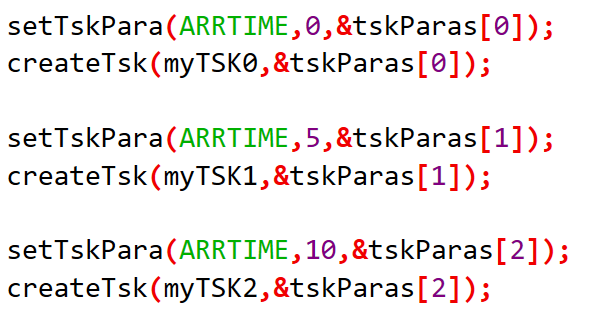
****

****

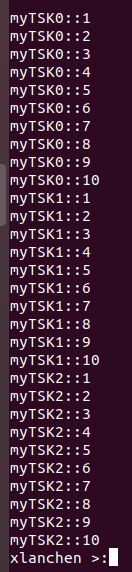


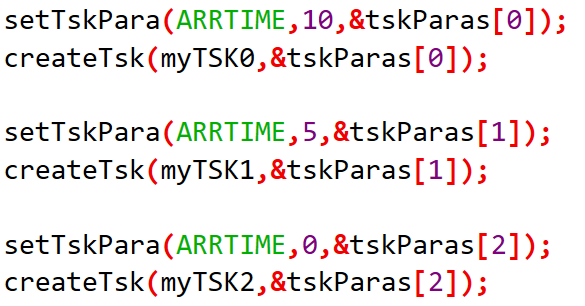
**运行结果解释**

Fcfs:

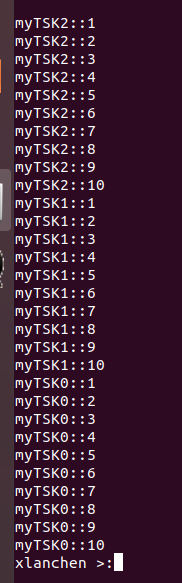


可以明显看到三个task 按照到来时间依次执行

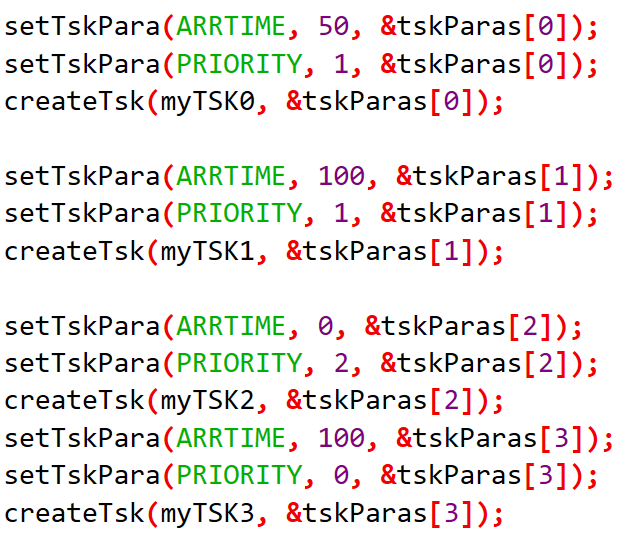




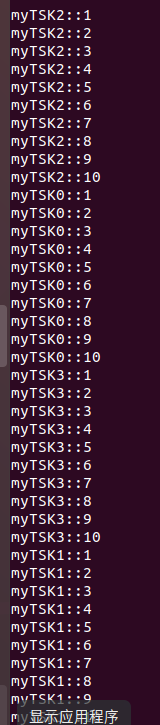
修改到来时间反转, 可以看到倒序执行



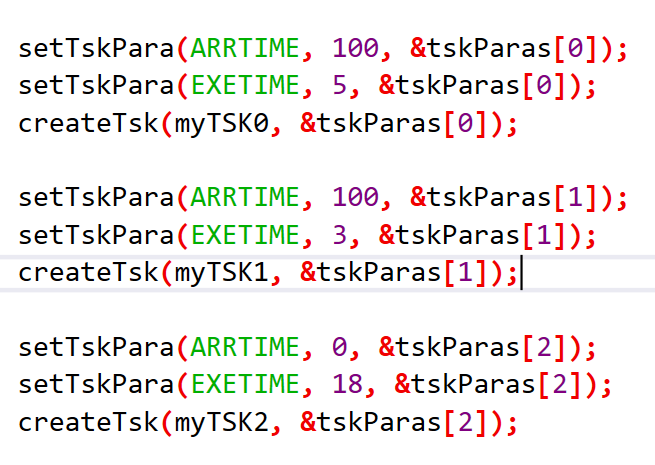
非抢占prio:



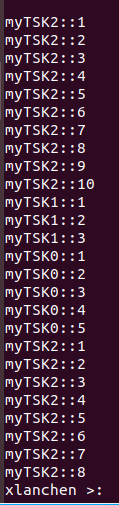
Task2 task3 同时到达, 但是task2 优先级高, 先执行task2. 然后task0到达, 然后task2结束, 优先选择后来的task0, 直到task0结束, task3才得以执行

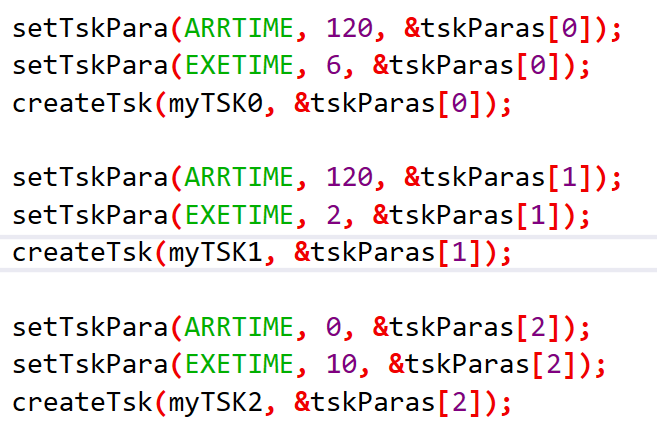


抢占式sjf:

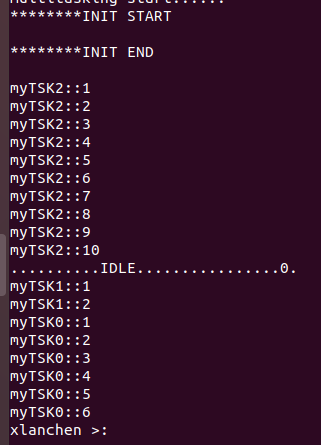


Task2最先到达并开始执行, 执行到第100tick时, task0 和task1 到达, 算法开始调度. 此时task2只执行到第10步还剩余8部分, 由于算法为抢占式, 运行时间更短的task1开始执行, 然后是task2, 都结束最后才轮到task2的剩余部分

****

****

在120ticks时, task0 和task1同时到达, 选择了时间更短的task1

****

**实验中遇到的问题**

1. 没理清文件结构, 对全局变量重定义
2. 使用指针前,忘记判断是否为空指针