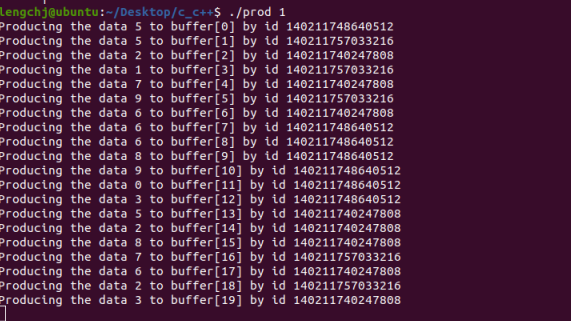
**操作系统第一次大作业实验报告**

18364039 冷春江

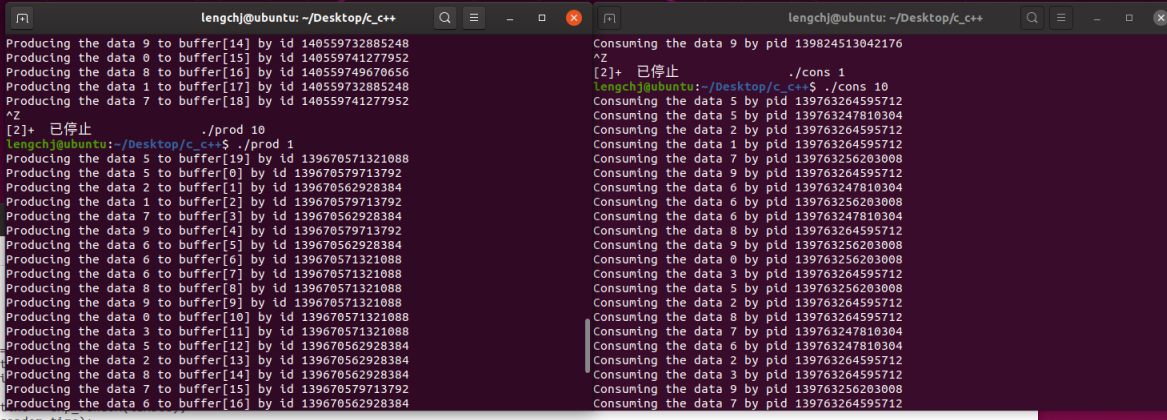
1. **生产者消费者问题**



可以看到prod队列被阻塞了

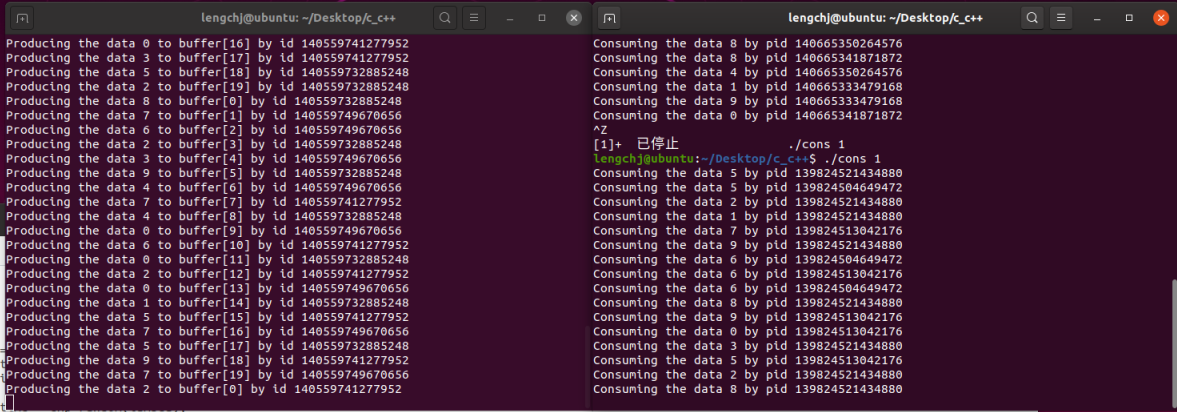
情况 1：

./prod 1 ./cons 10

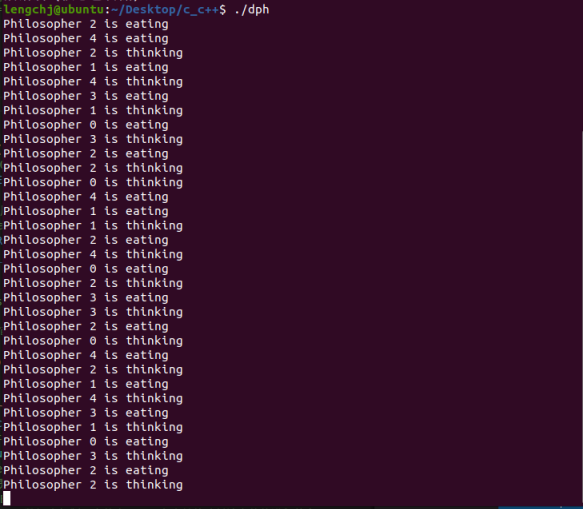
****

在运行的过程中，可以发现消费者消费的速度快然后进程阻塞了。

./prod 10 ./cons 1

****

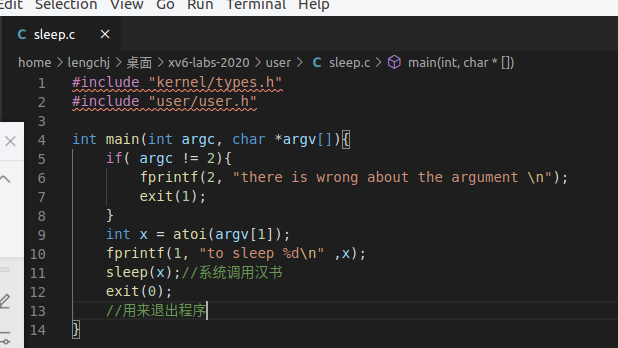
**二，哲学家就餐问题**



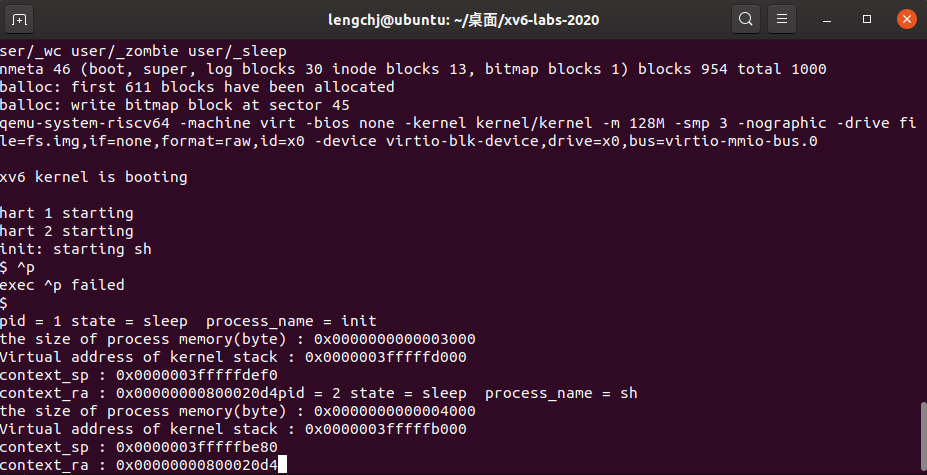
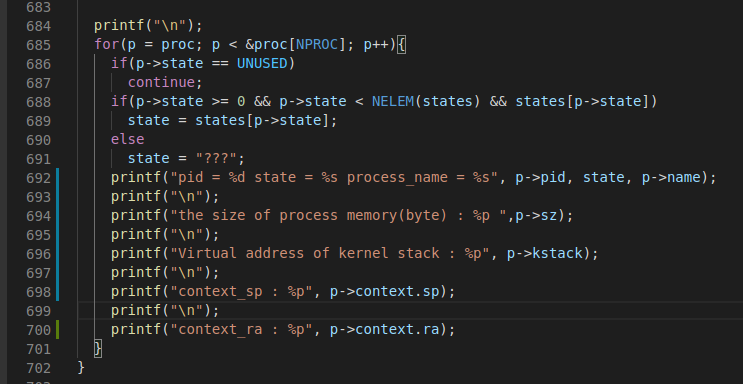
**第三题：MIT 6.S081 课程实验**

**1、**

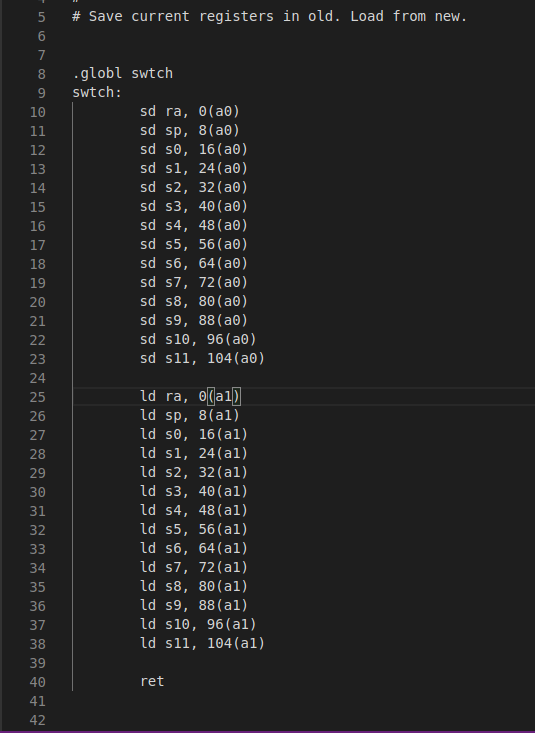
Sleep.c 的代码如下：





2.1

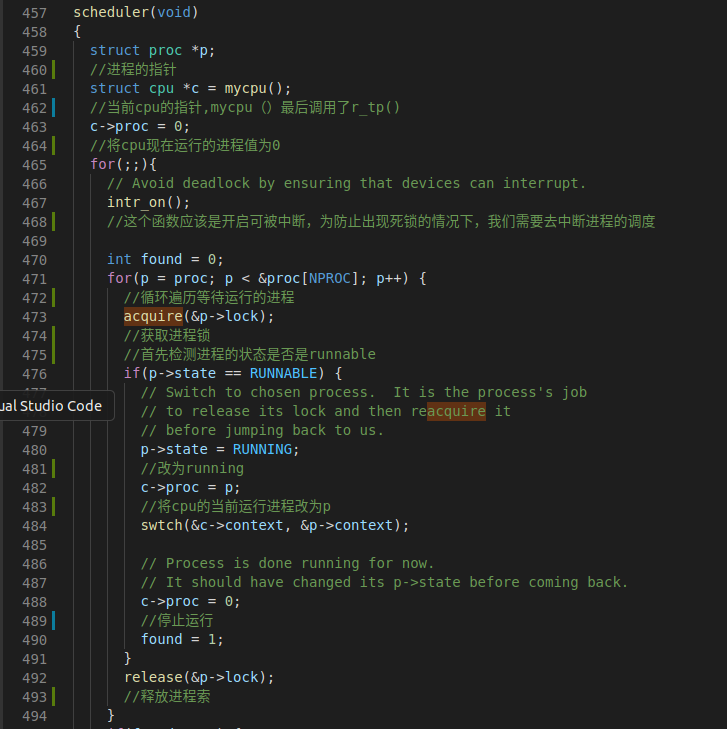
2.2

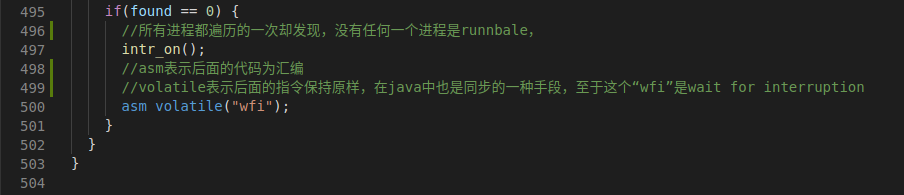


可以看到我们将新旧进程的寄存器进行了一个替换。

接下来对 proc.c 中的 scheduler(void),sched(void),yield(void)等函数核心部分进行解释。

Scheduler函数





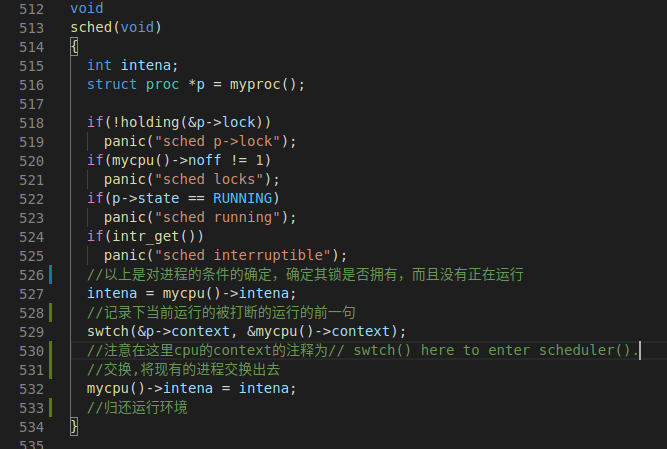
可以看到scheduler函数的主要内容是，不断的进行遍历整个进程队列，将其中的处于runnable状态的进程不断的进行替换，运行。当进程结束时，将控制权通过swtch()移交给scheduler。

如果一次遍历下来没一个可以运行，那么我们进入等待中断状态。

Sched

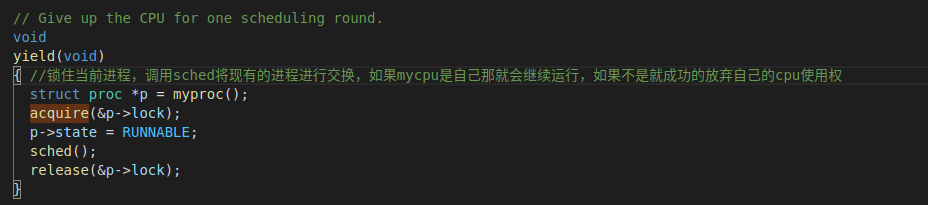
代码的核心是检查进程的条件

检测对应进程的状态是否满足情况，然后我们就记录下他的运行到的代码，进入到scheduler的循环中。



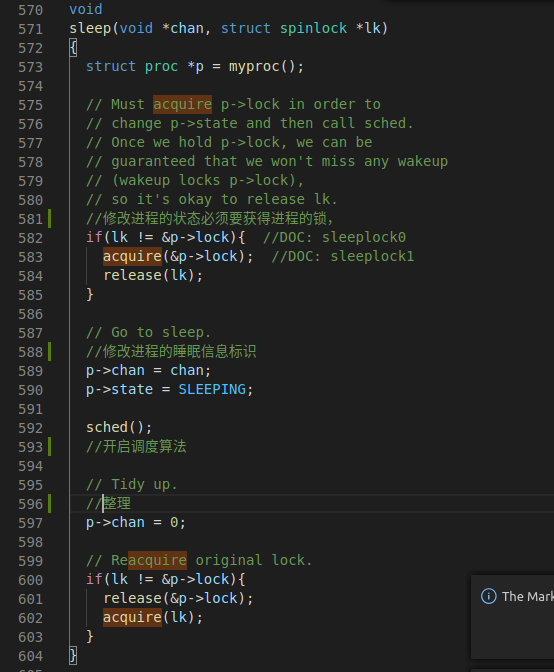
Yield

yield()函数将CPU主动让出一个调度周期(scheduling round)，放弃cpu的使用权，进行时间片调度算法



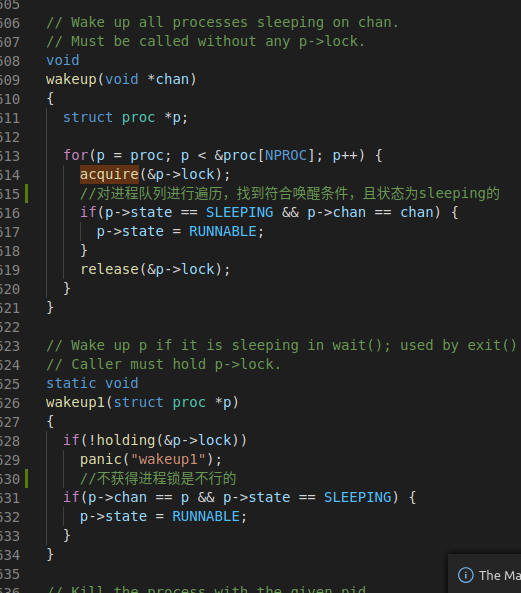
Sleep

将修改对应进程的睡眠信息标识，并且改变其进程状态，绝对不会被cpu再次使用



Wakeup

唤醒指定进程或者符合条件的进程



xv6的进程调度算法还是相当的原始，使用时间调度算法，不断的遍历进程队列，只要进程的状态是runnable，那么就立刻进行switch线程切换，进行运行。

同时，从整体上这种算法采用轮询遍历的方式，解决了进程饥饿的问题。但是也有很大的问题，就是无法解决不同线程的优先级不同的问题。

然后通过wait，sleep和wakeup函数进行进程的状态的切换。

3.

CFS概括

cfs定义了一种新的[模型](http://www.07net01.com/tags-%E6%A8%A1%E5%9E%8B-0.html" \t "https://www.cnblogs.com/tianguiyu/articles/_blank)，它给运行队列中的每一个进程安排一个虚拟时钟，vruntime。如果一个进程得以执行，随着时间的增长（也就是一个个tick的到来），其vruntime将不断增大。没有得到执行的进程vruntime不变。  
    而调度器总是选择vruntime较小的那个进程来执行。这就是所谓的“完全公平”。为了区别不同[优先级](http://www.07net01.com/tags-%E4%BC%98%E5%85%88%E7%BA%A7-0.html" \t "https://www.cnblogs.com/tianguiyu/articles/_blank)的进程，优先级高的进程vruntime增长得慢，以至于它可能得到更多的运行机会。

那么我们还需要一个数据结构来更方便的获得较小值。也就是优先队列，在此场景下，我们可以采用红黑树来满足要求。

分配给进程的运行时间 = 调度周期 \* 进程权重 / 所有进程权重之和

vruntime = 实际运行时间 \* X / 进程权重 。

X为nice值为0的进程的运行权重。B的权重是A的2倍，那么B的vruntime增加速度只有A的一半。

cfs很好的解决了不能够优先处理进程的问题，并且也相对的公平。

相比于xv6的时钟算法就更加的完善。

设计一个更好的进程框架。

public class cpu{

public PriorityQueue tree;

public process nowProcess;

//tree 的key值应该是进程id，防止重复，比较器应该是对应进程的vruntime

//理论上来讲我们假设只有一个cpu，cpu有自己对应的队列

}

public class process {

public int nice;

public int id;

public int context;

public int vruntime;

}

public void new\_process(int nice,int id){

//初始的vurntime可以自己去设定

process new\_process = new process(nice,id);

process.vruntime = 0;

tree.add(new\_process);

}

public void schedule(){

cpu cur\_cpu = get\_cpu();

while (true){

**//从优先队列中取出vurntime最小的进程**

**process curProcess = cur\_cpu.tree.poll();**

**switch (cpu.nowProcess;,curProcess);**

**//进程的切换**

**cur\_cpu.nowProcess = curProcess;**

**//计算运行时间**

**int time = get\_time(id,nice,vruntime);**

**cpu.run(time,process\_id);**

**//运行结束之后，计算新的vruntime并将其放回去**

**vruntime = get\_vruntime(run\_time,id,nice);**

**cur\_cpu.tree.add(cur\_cpu.nowProcess);**

**//然后进行不断的循环**

}

}