**操作系统实验报告**

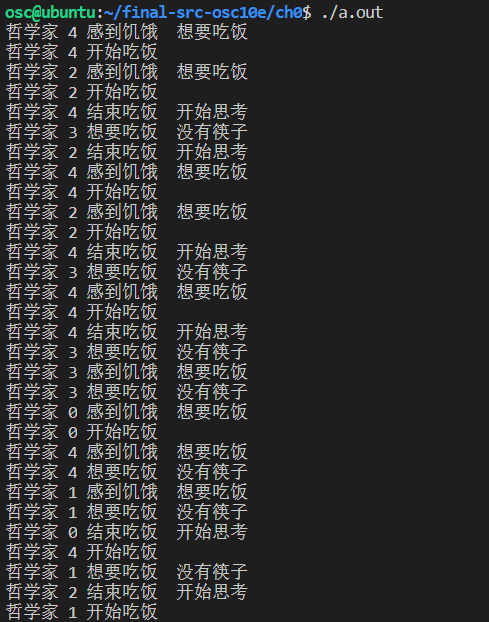
学号：17363041

姓名：李杰

班级：智科一班

1. **就餐问题**

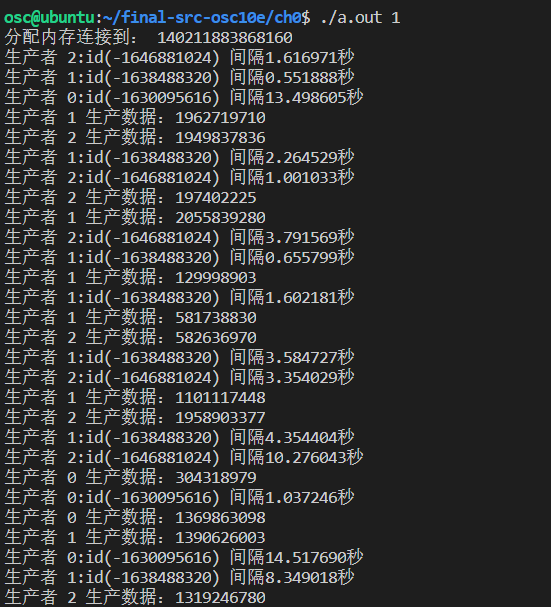
在思考过程与吃饭过程加入了随机sleep，得到如下结果：

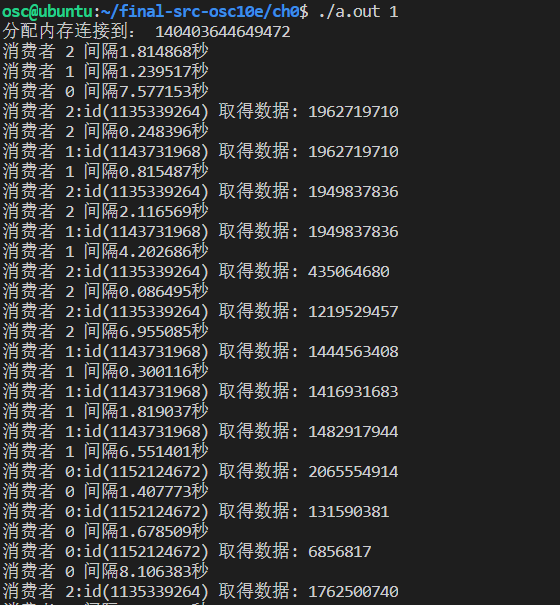
****

1. **生产者消费者问题**

**当生产者输入参数**=**1，消费者输入参数为**c=1**时：**

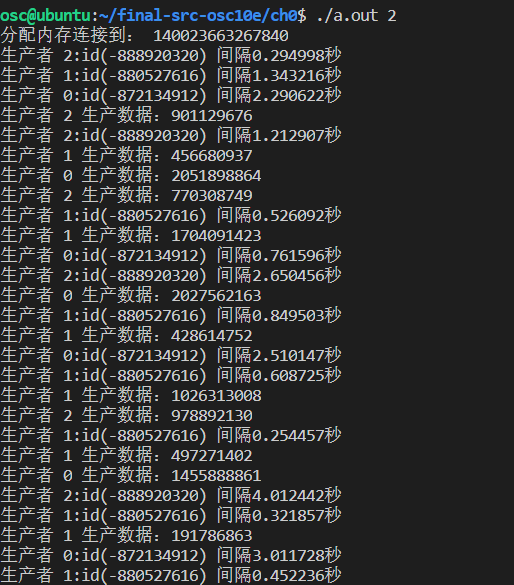
**生产与消费速率大致相同：**

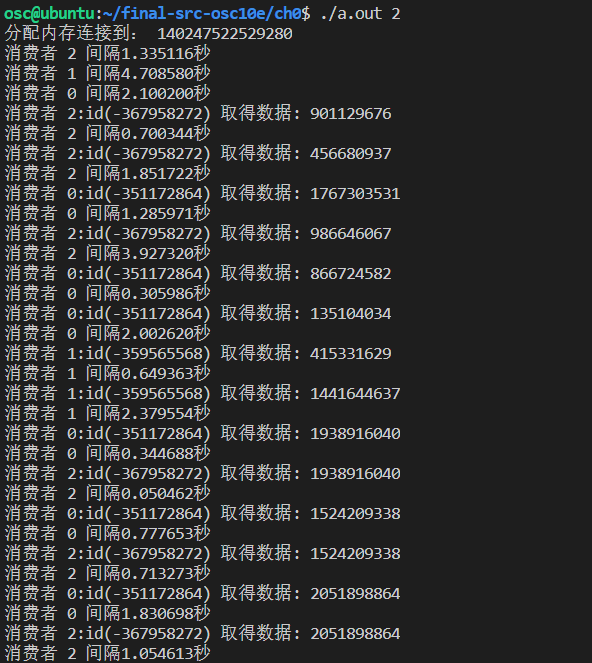




**当生产者输入参数**=**2，消费者输入参数为**c=2**时：**

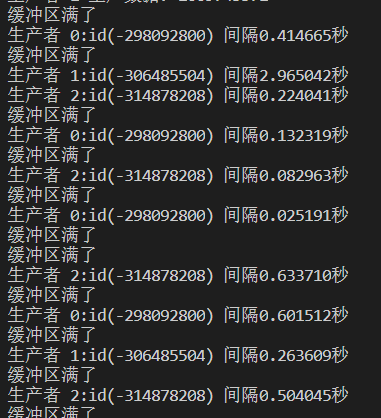
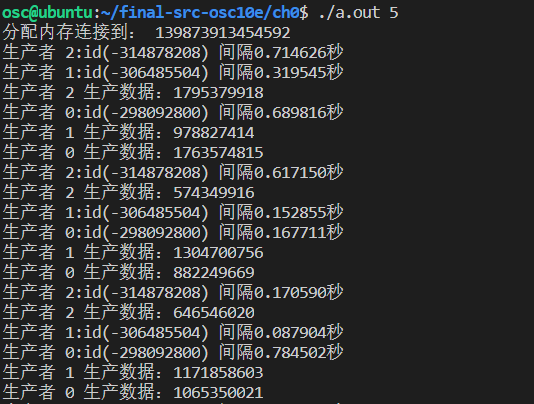
**生产与消费速率大致相同：**

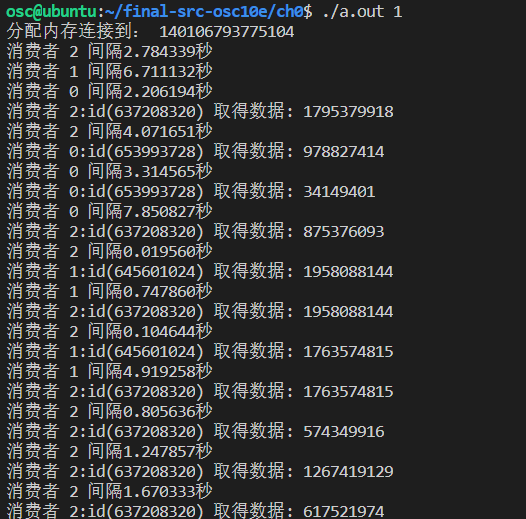




**当生产者输入参数**=**5，消费者输入参数为**c=1**时：**

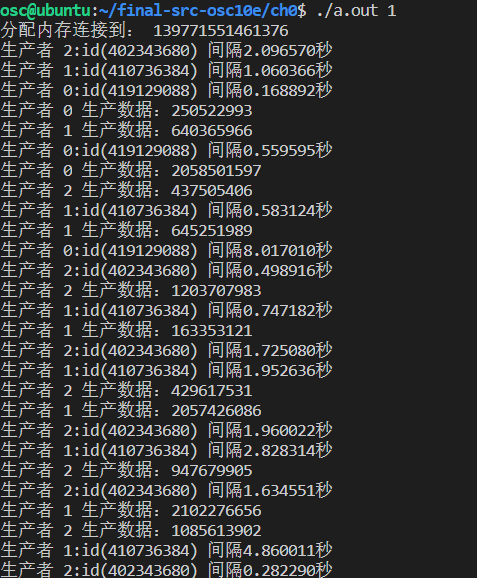
**生产速率大于消费速率，一段时间后缓冲区会填满，然后消费者取用后，生产者再进行生产。**

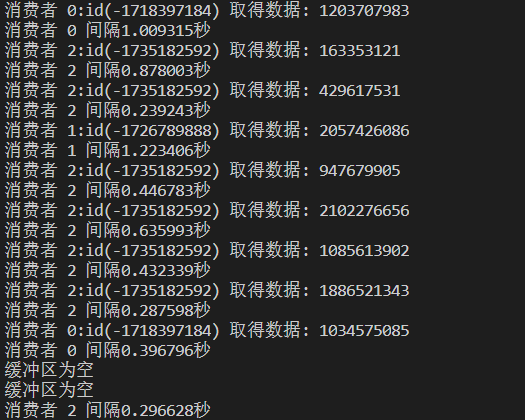




**当生产者输入参数**=**1，消费者输入参数为**c=5**时：**

**生产速率小于于消费速率，一段时间后缓冲区会清空，生产者生产后，然后消费者再取用。**





1. **Linux内核实验（Linux 4.0或以上）**

**A题**

**Linux进程的基本结构：**

Linux系统是一个多进程的系统，它的进程之间具有并行性、互不干扰等特点。也就是说，每个进程都是一个独立的运行单位，拥有各自的权利和责任。其中，各个进程都运行在独立的虚拟地址空间。因此，即使一个进程发生异常，它也不会影响到系统中的其他进程。

每个进程在内核中都有一个进程控制块（PCB）来维护进程相关的信息，Linux中的进程包含3个段，分别为“数据段”、“代码段”、“堆栈段”。

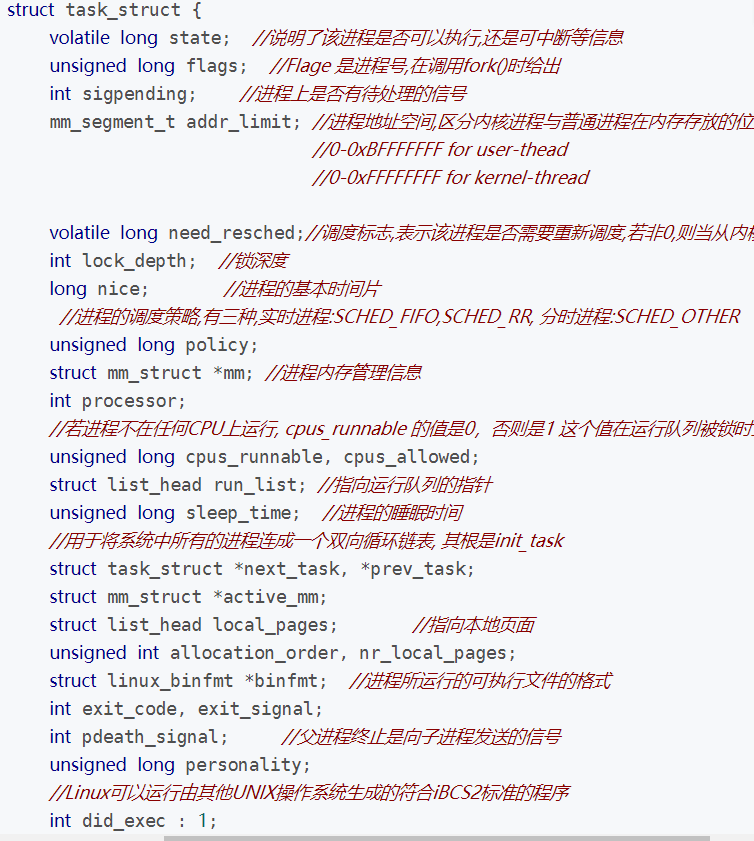
（1）数据段：存放的是全局变量、常数以及动态数据分配的数据空间。根据存放的数据，数据段又可以分为普通数据段（包括可读可写/只读数据段，存放静态初始化的全局变量或常量）、BSS数据段（存放未初始化的全局变量）、堆（存放动态分配的数据）。

（2）代码段：存放的是程序代码的数据。

（3）堆栈段：存放的是子程序的返回地址、子程序的参数以及程序的局部变量等。

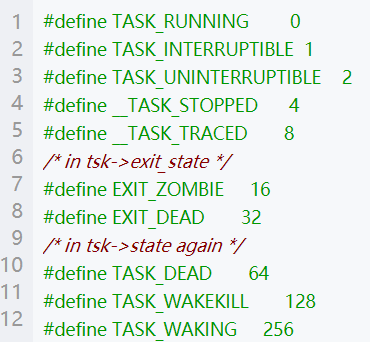
保存进程信息的数据结构叫做task\_struct，并且可以在include/linux/sched.h里找到它。所以运行在系统里的进程都以task\_struct链表的形式存在于内核中。

部分代码如下：



**状态设置：**

进程运行状态state可能有如下取值：



系统中的进程必然处于上述进程状态的一种，下面就各状态做简要说明：



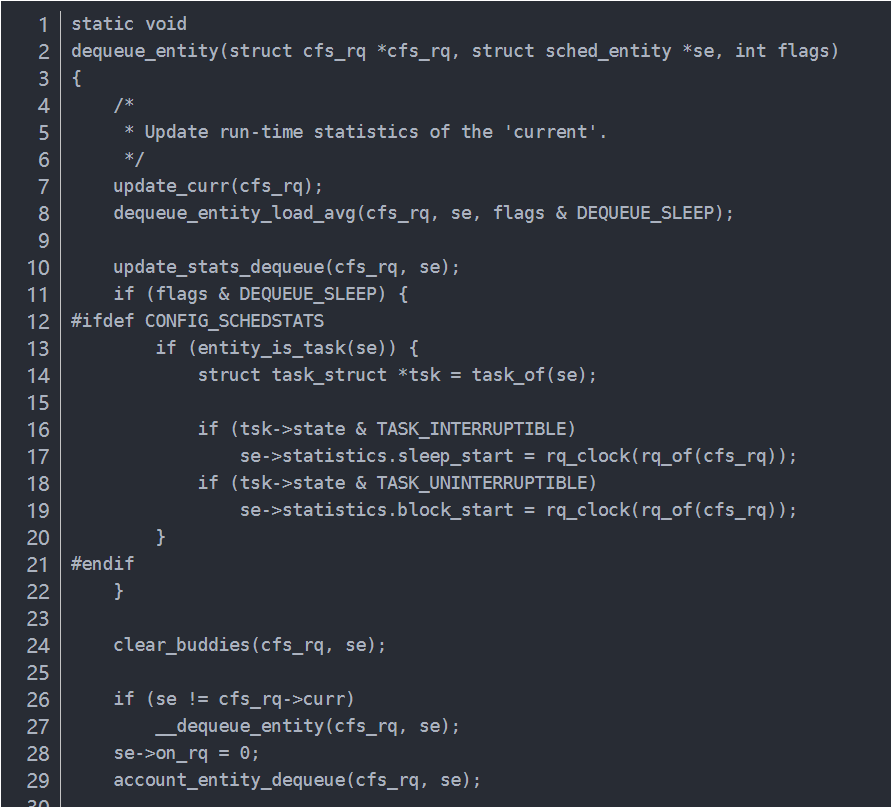
**CPU的调度基本架构：**

在多CPU的系统上，不同的CPU的负载不一样，有的CPU更忙一些，而每个CPU都有自己的运行队列，每个队列中的进程的vruntime也走得有快有慢，

当进程从一个CPU的运行队列中出来 (dequeue\_entity) 的时候， 它的vruntime要减去队列的min\_vruntime值；

而当进程加入另一个CPU的运行队列 ( enqueue\_entiry) 时，它的vruntime要加上该队列的min\_vruntime值。

其框架只截取了部分。



**理解CFS调度算法的基本流程和主要数据结构：**

**基本流程：**

CFS算法

进程调度的时机

（1）进程状态转换：进程终止、进程睡眠

（2）当前进程的“时间片”用完

（3）主动让出处理器，用户调用sleep()或者内核调用schedule()

（4）从中断、系统调用或异常返回时

CFS抛弃了时间片的概念，而是分配给进程一个处理器使用比重

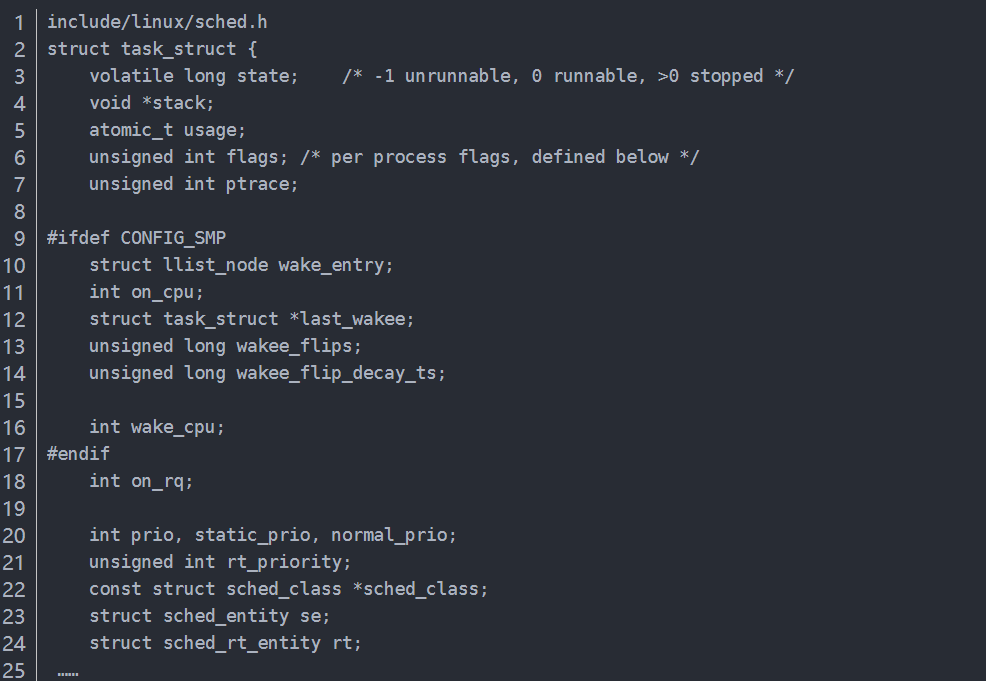
其出发点在于：进程调度的效果应如同系统具备一个理想中的完美多任务处理器，每个进程获得1/n的处理器时间。理想情况下，完美的多任务处理器模型应该是两个进程同时运行，并且各自使用处理器一半的能力

而实际上是做不到的，因此CFS的做法是允许每个进程运行一段时间，循环轮转，选择运行最少的进程作为下一个运行进程，不再采用分配给每个进程时间片的做法，CFS在所有可运行进程总数基础上计算出一个进程应该运行多久，而不是依靠nice值来计算时间片。

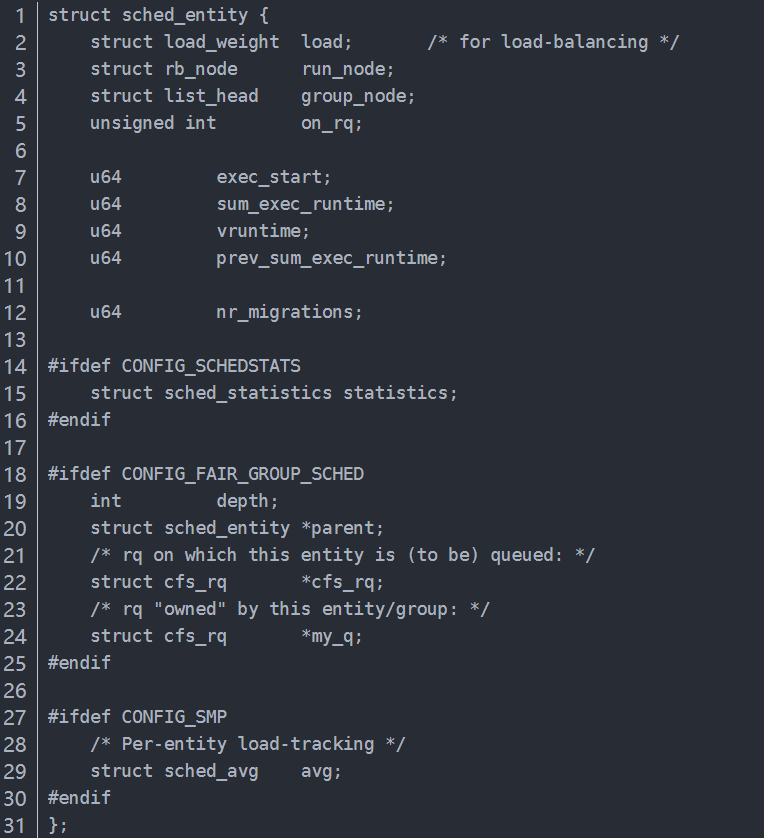
通过优先级和系统负载等加权过的时间，不是物理时钟时间

**主要数据结构**：

每个进程描述符都有一个调度实体se



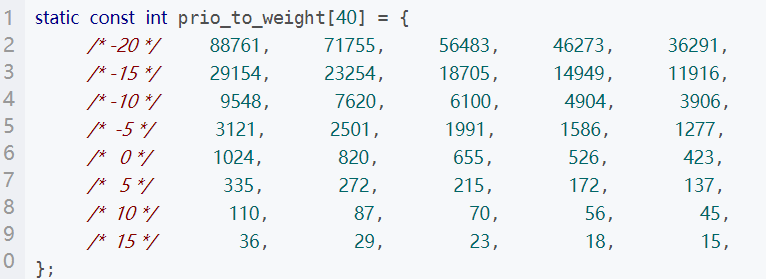
se中即保存着进程的虚拟运行时间，其记录一个程序到底运行了多长时间以及应该再运行多久



CFS为每个CPU维护一个调度队列cfs\_rq

**第一问**

每个进程都有一个nice值, 表示其静态优先级, nice值和进程的权重存在如下关系:



可以看到, nice值越小, 进程的权重越大：CFS调度器的一个调度周期值是固定的, 由sysctl\_sched\_latency变量保存.

进程在创建时被赋予不同的优先级值，而如前面所说，nice的值是表示进程优先级值可被修正数据值，因此，每个进程都在其计划执行时被赋予一个nice值，这样系统就可以根据系统的资源以及具体进程的各类资源消耗情况，主动干预进程的优先级值。在通常情况下，子进程会继承父进程的nice值，比如在系统启动的过程中，init进程会被赋予0，其他所有进程继承了这个nice值。

**第二问：**

一个进程在一个调度周期中的运行时间为:

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

可以看到, 进程的权重越大, 分到的运行时间越多。

一个进程的实际运行时间和虚拟运行时间之间的关系为:

vruntime = 实际运行时间 \* NICE\_0\_LOAD / 进程权重

= 实际运行时间 \* 1024 / 进程权重

NICE\_0\_LOAD = 1024, 表示nice值为0的进程权重

可以看到, 进程权重越大, 运行同样的实际时间, vruntime增长的越慢

一个进程在一个调度周期内的虚拟运行时间大小为:

vruntime = 进程在一个调度周期内的实际运行时间 \* 1024 / 进程权重

= (调度周期 \* 进程权重 / 所有进程总权重) \* 1024 / 进程权重

= 调度周期 \* 1024 / 所有进程总权重

可以看到, 一个进程在一个调度周期内的vruntime值大小是不和该进程自己的权重相关的, 所以所有进程的vruntime值大小都是一样的

总的来说：

进程的nice值越小, 权重越大, 所分到的运行时间也越多.。

两个权重不相同的进程, 运行相同的实际时间, 权重大的进程vruntime值增加的要少一些。

但是在一个调度周期内, 所有进程的vruntime值都是一样大的, 当每个 进程都把分配给自己的运行时间运行完了时, 它们的vruntime值是一样大的. 所以一个进程的vruntime值越大, 表示进程已运行的时间占调度器分配给它的运行时间的比重也越大。

所以, vruntime值可以反映出该进程当前已经运行的时间多少, 可以让各个进程公平的竞争CPU, 而且让优先级大的进程实际的运行时间也都多一些。

min\_vruntime是CFS红黑树中最小的虚拟时间（运行最少）。其在进程更新虚拟时间(vruntime)时一起被更新，当一个进程进入运行对列或者出队列时，会调用update\_curr，其结构如下：

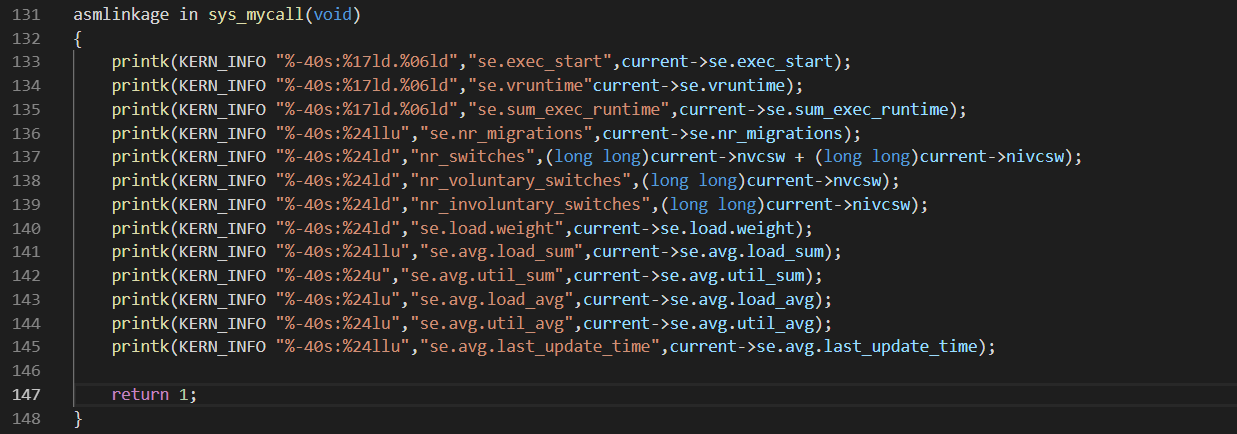


**B题：**

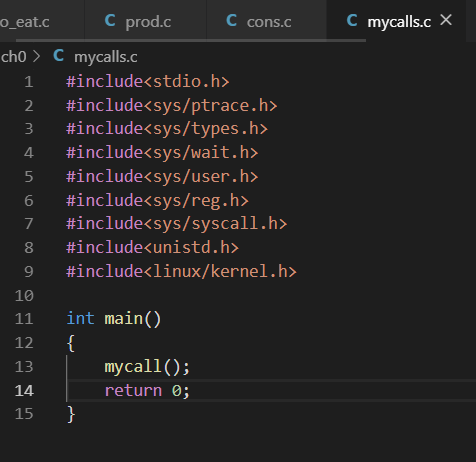
电脑的数据：



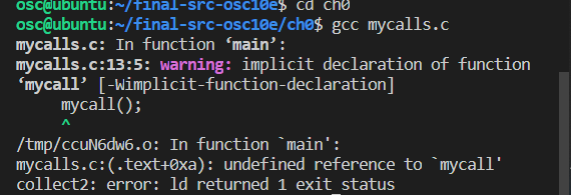
编写到内核里的函数：



由于再编写内核函数时将内核函数名将为mycall，为避免错误，这里将.c文件命名为mycalls。c



由于内核重载过程失败好几次，



所以出现以下错误。