**实验3：进程跟踪**

**（201600800456董喆）**

1. **实验目的：**

* 走走停停的特性是进程区别于程序的显著特征，也是理解进程概念的一个直观而有效的视角，因此本实验通过对进程运行轨迹的跟踪来让实验者对进程的概念形象化，这是本实验的基本目的。
* 进程在其生命期中会进行多次状态切换，在这些切换中，就绪态和运行态之间的切换是一类非常重要的切换，会直接导致操作系统的表现行为（如批处理、分时、实时等）。而进程调度正是控制这一切换的关键，因此通过追踪进程在就绪态和运行态之间切换可以从侧面了解进程调度算法（操作系统中最重要的核心算法之一）的行为特性，进而可以对进程调度算法进行评价。因此本实验的第二个基本目标是对进程运行轨迹跟踪的基础上进行相应的数据统计，如统计进程的等待时间（在等待队列中时间），从而能对进程调度算法进行实际的量化评价（如算出平均等待时间），更进一步加深对进程调度概念和调度算法的理解，获得能在实际操作系统上对调度算法进行实验数据对比的直接经验。

1. **实验内容：**

* （1）学习实验的预备知识和linux关于进程管理的知识。
* （2）在Linux 0.11上实现上述进程运行轨迹的跟踪，基本任务是在Linux 0.11内核中维护一个日志文件/etc/process.log，要求把从操作系统启动后到操作系统关机的所有的进程的 的运行轨迹（生命轨迹）都记录在这一log文件中，记录的内容可参照上面给出的实例。
* （3）在进程运行轨迹的跟踪基础上，统计所有进程的等待时间、完成时间、运行时间等信息。然后针对现有的Linux 0.11调度算法，计算该算法导致的平均等待时间，平均完成时间，平均吞吐量等数据。
* （4）实验者需要修改现有的Linux 0.11调度算法，然后再统计这些时间，并和现有的Linux 0.11调度算法对比，体会调度算法的实验验证。

1. **实验步骤**
2. **实验准备：**

**①linux的进程管理：**

* **以下为linux操作系统的进程状态和切换过程：**

运行状态（TASK\_RUNNING）:当进程在被CPU执行，或**已经准备就绪随时**可由调度程序执行，则称该进程处于运行状态。

可中断睡眠状态（TASK\_INTERRUPTIBLE）:当进程处于中断睡眠状态时，系统不会调度该进程的执行，当系统产生一个中断或者释放了该进程等待的资源，或者收到一个信号，都可以唤醒转换到就绪态。使用sleep\_on()或interruptible\_sleep\_on()自愿放弃使用CPU，调度程序去执行其他程序。进而进入睡眠状态

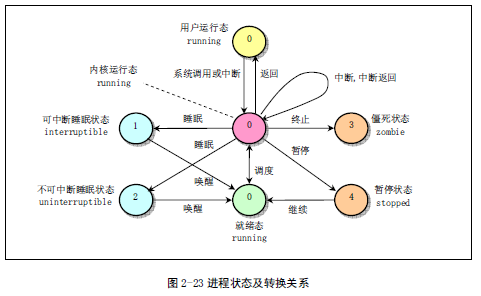
不可中端睡眠状态（TASK\_UNINTERRUPTIBLE},除了不会因为收到信号而被唤醒，该状态与可中断睡眠状态详细。但出于该状态的进程只有使用wake\_up()函数明确唤醒才能转换到可运行的就绪状态。

暂停状态（TASK\_STOPPED）:当进程收到信号SIGSTOP、SIGSTP、SIGTTIN或者STGTTOU时会进入暂停状态，可向其发送SUGCIBT信号让进程进入可运行状态。在Linux-0.11中，没有对该进程的转化处理。

僵死状态（TASK\_ZOMBIE）当进程已经停止运行，但是没进程还没有调用

wait（）询问其状态时，则进程成为僵死状态。为了让父进程能够获取其停止运行的信息，此时子进程的任务数据结构信息还需要保留着，一旦父进程调用

wait（）得到子进程的信息，则处于该状态的进程数据被释放。



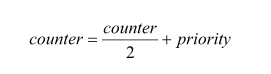
* 进程创建：

linux系统中创建新进程使用fork（）系统调用。所有进程都是通过进程0而得到的（本次实验未涉及到进程初始化所以省略），都是进程0的子进程。细节方面：linux系统最多可以同时拥有64个进程，如果多于这个数量会报错；系统创建子进程会在主内存去申请一页来存放数据，为了在这个过程中它被调度函数选中，进程状态被置为不可中断等待。注：初始时间片数量为15个系统滴答数。

* 进程调度：

Linux进程是抢占式的，但是被抢占的进程仍然处于TASK\_RUNNING状态，只是暂时没有被CPU运行。Linux 0.11中采用基于优先级排队的调度策略

调度程序：schedule（）扫描任务数组，根据任务的剩余时间时间片数选择进程。如果所有处于就绪态进程时间片数量为0，则计算心得时间片：



tips:这种情况下，处于睡眠状态的进程会获得较高的时间片。

然后schedule函数重新扫描任务数组中所有处于TASK\_RUNNING状态的进程，重复知道选出一个进程位置，最后调用switch\_to执行实际的就进程切换操作。

注：进程切换的具体过程这里不具体描述。

* 终止进程：

当一个进程结束了运行或者在中途终止了运行，就需要释放其所占用的系统资源。当一个用户程序条用了exit（）系统调用时，就会直接内核函数do\_exit()。该函数会首先是放程序代码段和数据段占用的内存，关闭进程打开这的所有文件。

**②某些代码：**

* **dup（）函数：**

调用dup函数时，内核在进程中创建一个新的文件描述符，此描述符是当前可用文件描述符的最小数值，这个文件描述符指向oldfd所拥有的文件表项。

* **编写的cpuio\_bound(int last, int cpu\_time, int io\_time)函数：**

这个函数用于测试，last参数表示cpu和io共同占用的总时间长度，cpu\_time和io\_time是一个相对的比值，当cpu\_time = 1，io\_time=0，cpu模拟占用last时间，同理反之，io模拟占用last时间，其中使用sleep函数对io进行模拟。

* **jiffies：**

jiffies表示从开机时到现在发生的时钟中断次数

1. **实验进行：**

**①添加必要的变量：**

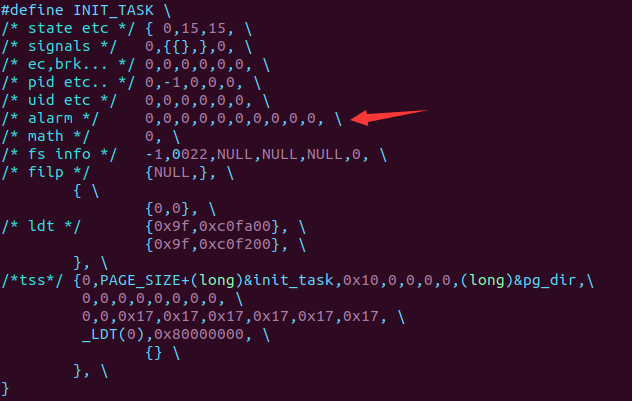
long last\_sleep\_time;

long block\_time;//进入阻塞态的时间点

long JQueue\_time;//进入就绪队列的时间点

long run\_time;//开始运行的时间

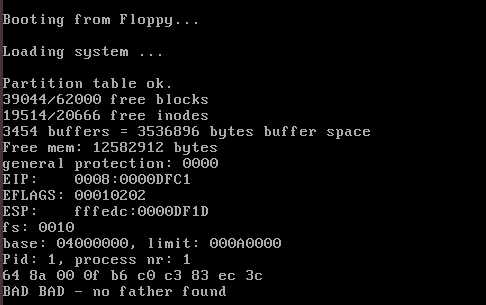
这句话用于统计休眠时间，在添加这句话时你会发现报错的情况，这是因为你没有在宏定义中为这两个变量赋值：



这段代码是一个宏定义，在以后使用的函数中使用它来初始化，在红色箭头指示的地方加入两个0报错情况消失。

**② 如何打印到文件？**

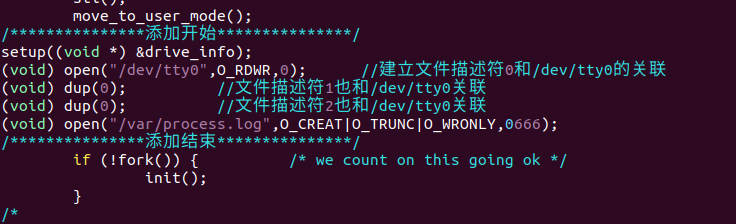
**这里我没有按照实验要求的方法完成，因为在修改main函数中的printf函数时，会出现运行报错，这个错误没有找到相关的资料，只能修改printk函数（kernel/printk.c）**



（报错信息）

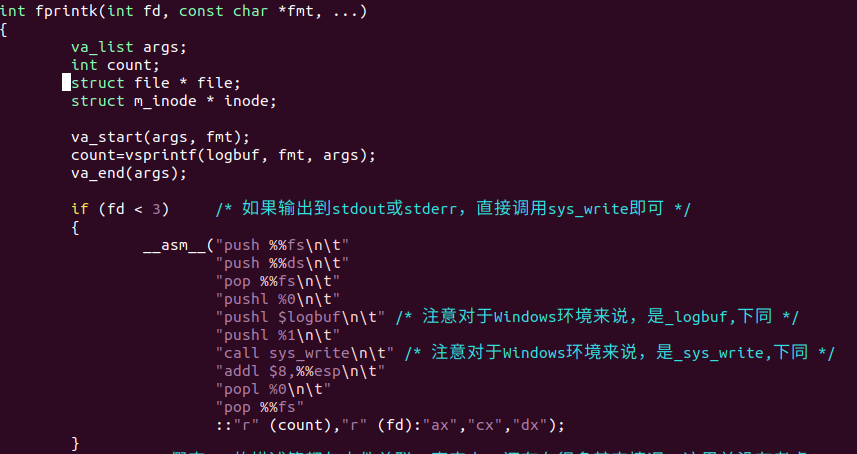
我们进行如下修改：

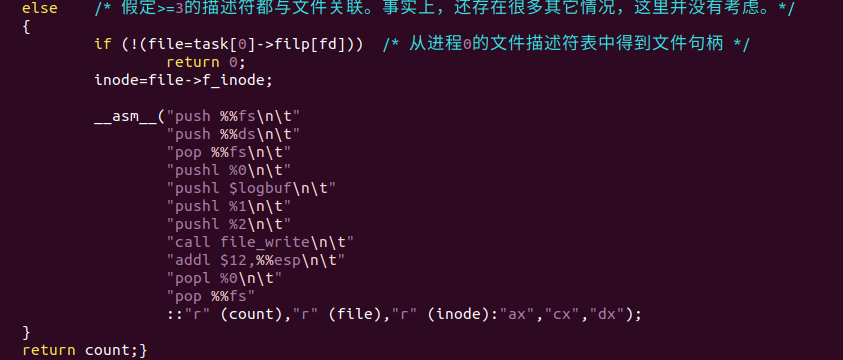
init/main.c,在main函数中添加如下代码：



这句代码的意思是：创建文件描述符，其中0 1 2分别与不用的位置相连，其中dup前面已经说过，我们只需要在dup后加入我们的log文件打开句柄就可以了。

然后定位到kernel/printk.c:，添加如下代码：





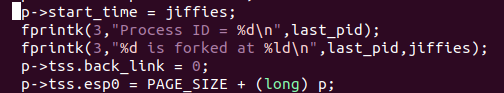
其中logbuf为定义的全局变量 char logbuf[1024]，这个函数是仿照printk函数完成。

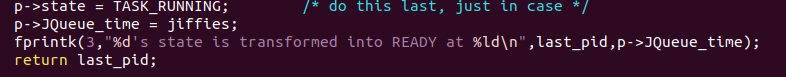
这段代码经过测试可以顺利完成输出功能。

**③修改fork.c**

fork.c的作用是复制进程并进入就绪状态：

定位到kernel/fork.c，在copy\_process函数中加入以下代码：





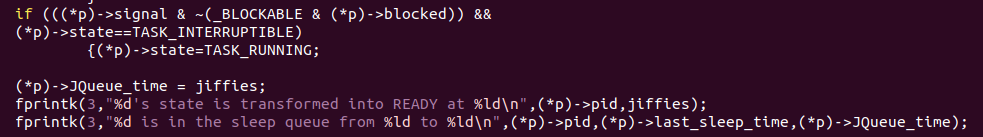
这两段代码的作用是输出当前进程id，输出进程创建的时间和进程进入就绪态的时间。

**④修改sched.c函数：**

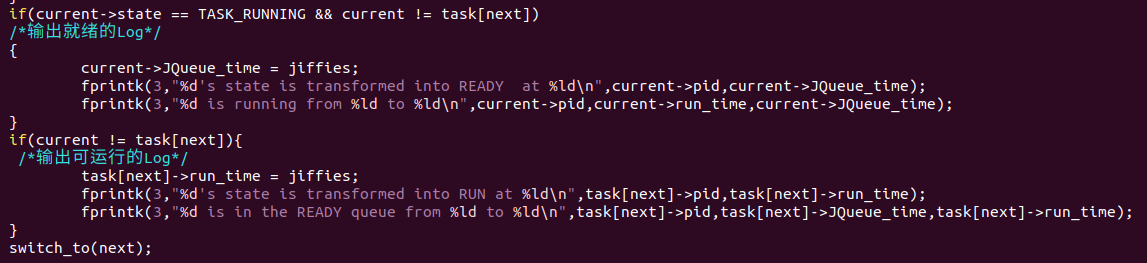
这段修改是比较困难的，需要对linux的系统进程状态切换有深入的了解

定位到kernel/sched.c：

在schedule函数下添加：

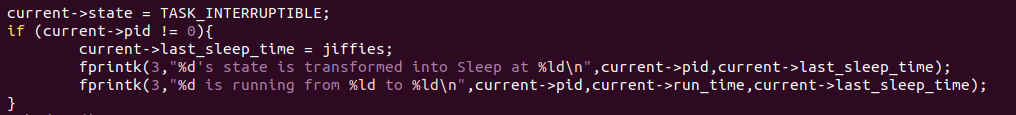


这一段函数显示进程从可中断睡眠到就绪态的时间点和睡眠时间



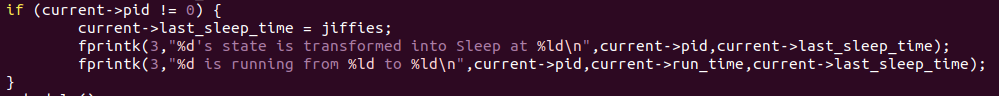
这一段函数显示时间片用完的自动进入就绪态的时间点和运行时间

在sys\_pause函数添加如下：



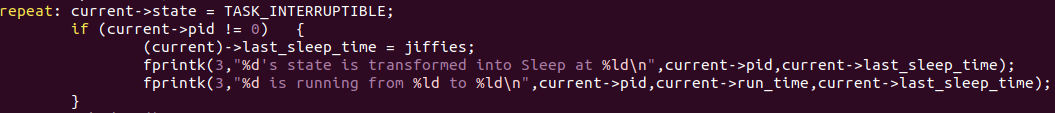
这段函数显示通过sys\_pause进入可中断睡眠状态进程的信息

**在sleep\_on函数添加如下：**

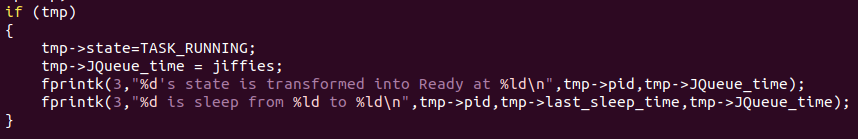


添加因为因资源不满足或者程序不在内存中而进入睡眠状态进程的信息

在interruptible\_sleep\_on函数中添加如下：

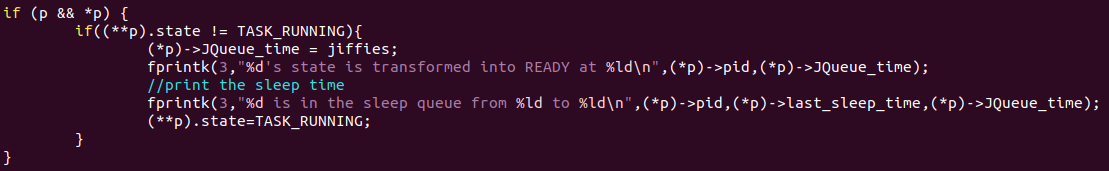


显示因资源竞争而进入等待队列的进程时间信息



显示因资源满足而被唤醒的进程的时间信息

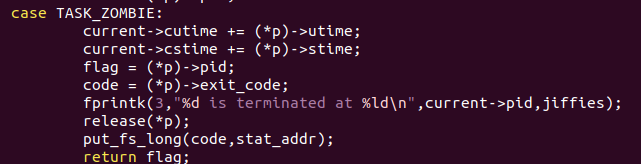
**在wake\_up函数中添加：**



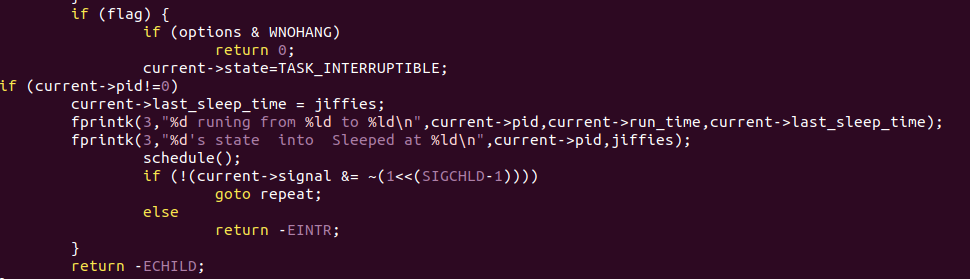
显示显式的唤醒一个进程的时间信息

**⑤修改exit.c：**

**在sys\_exit函数下：**



显示一个进程资源被释放时的进程事件信息：



显示一个进程进入因等待资源(Sleep)而进入等待状态的进程时间信息

**至此完成了所有文件的修改。**

**⑥添加process.c**

在oslab目录下，执行：

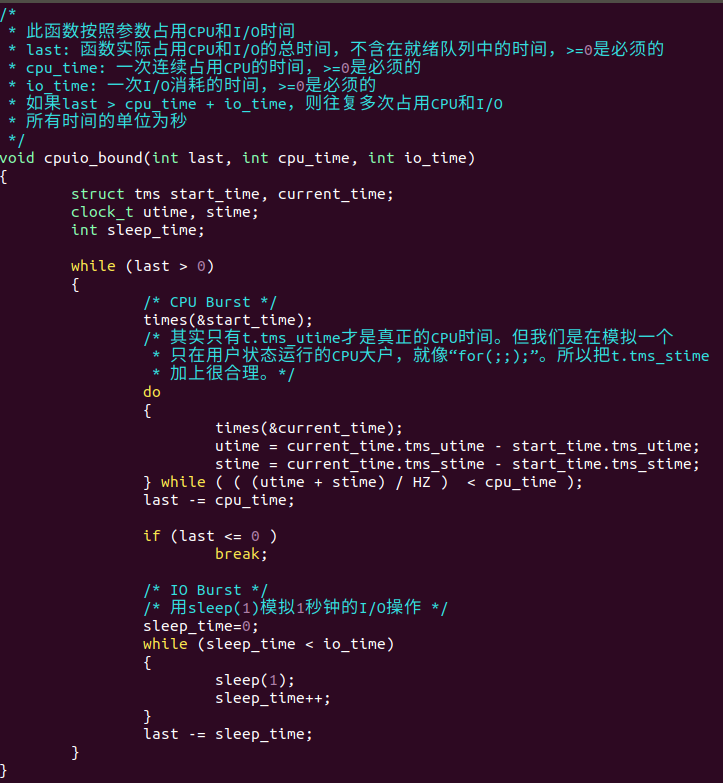
sudo ./mount-hdc

cd hdc/usr/root

touch process.c

process.c中有之前我们介绍的cpuio\_bound，其具体代码如下:





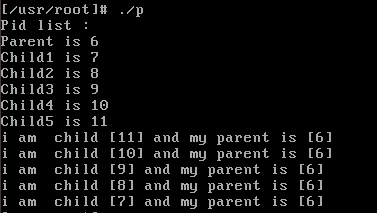
完成之后，我们就可以进行测试。

**⑦第一次测试：**

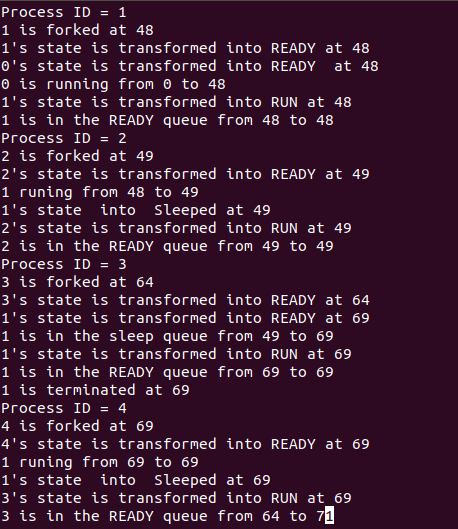
依旧是make clean&&make all&&./run

gcc -o process process

./process



然后进入hdc/var 中的process.log文件，显示：

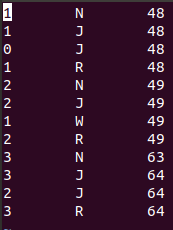


这就显示了4个进程的进程切换跟踪信息。

**⑧测试2：**

为了更加直观方便的完成平均等待时间、睡眠时间等的统计，我们使用网络上提供的stat\_log.py,他可以统计详细的进程切换信息，但是必须要保证process.log的格式正确，而网络上的格式与我们的实验不同，我们直接下载覆盖原来的文件即可。

新的格式文件为：



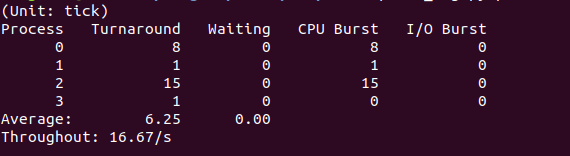
注：要保证process.log与stat\_log.py在同一文件夹下：



注：如果没有python，需要sudo apt-get install python

我们运行如下命令：

./stat\_log.py proecss.log 0 1 2 3 4 5 -g



从这我们可以清楚地看到进程的等待时间、CPU占用时间，IO占用时间、平均等待时间等等。

**⑨测试3：**

**修改时间片来测试调度算法对于进程调度的影响，这里不演示各种指令。**

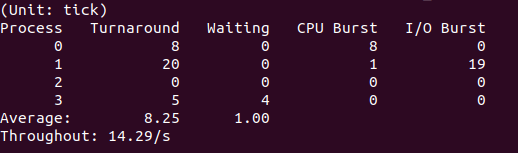
修改时间片：

时间片定义在include/linux/sched.h中:



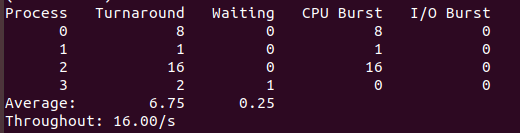
其中第三个值为时间片大小

* 修改为30



可以看出，此处平均等待时间变长

* 修改为8



可以看出此处平均等待时间变化不大

**至此本实验步骤结束。**

1. **实验感悟：**

①只流于书本上对于操作系统进程管理的总体感知，无法对操作系统进行更加深入的了解，只有真正的去了解，深入系统的内核才能对其有更深的理解。

②Linux大牛的编程功力令人感叹，代码中的各种trick令人叹为观止。在完成实验的过程中，改写代码的过程很困难，深刻的感觉到自己的编程功力不足和代码阅读能力不足。

③通过《Linux内核完全剖析》一书，对Linux系统的进程管理有了更深的理解，对一些调度算法实现有了更深的了解。了解了一个进程从创建、运行、等待和结束等各个过程的执行流程。

④创新很难，网络上基本没有关于本实验的具体操作的解释，只有另一个版本的linux实验，自己逼迫自己完全由自己来完成整个要求进程跟踪，其中遇到了很多问题，比如exit.c出错、sched.c进程切换出现问题等，但是最后经过几天的努力终于完成了本次实验，感叹努力你一定行。在更困难的数据统计上，因为代码功力的匮乏，依然选择了stat\_log.py来完成本次试验。

⑤逐渐习惯了使用命令行，脱离鼠标的舒束缚。在完成实验的过程中，频繁的使用鼠标点击给编码和操作带来不变，强迫自己使用命令行增加了对linux系统操作的熟练程度。在以后研究生的阶段（机器学习）离不开对linux系统的使用，这是一条必经之路。