第二章编程作业实验报告

191220156　计算机科学与技术系　张桓

实现逻辑

蒙特卡洛算法的核心就是不断取随机点，取足够多次(比如本次作业的`1000000`次)，统计落在圆内和圆外的点数经过计算即可得到π。

为了避免各个线程之间因为共享数据而导致统计错误，我为每个线程提供了一个独立的全局变量，即在堆区开辟了一个全局数组，数组的下标和线程`id`一一对应，统计该线程中落入圆内的点的总数。

而另一个更关键的问题在于**随机数如何生成**？按照之前的一贯做法，就是用`srand(time());`以时间作为随机数种子，然后用`rand()`函数得到随机数序列。但是查阅资料后得知：`rand()`函数是非线程安全的，因为每调用一次`rand()`函数，都要改变其状态值，由于同一进程的不同线程之间**共享这些状态**，所以每个线程对`rand()`函数的调用，会影响到其他线程对`rand()`的使用，所以`rand()`函数是**非线程安全的**，在多线程环境中应该使用明确指定`seed`值的`rand_r()`函数。

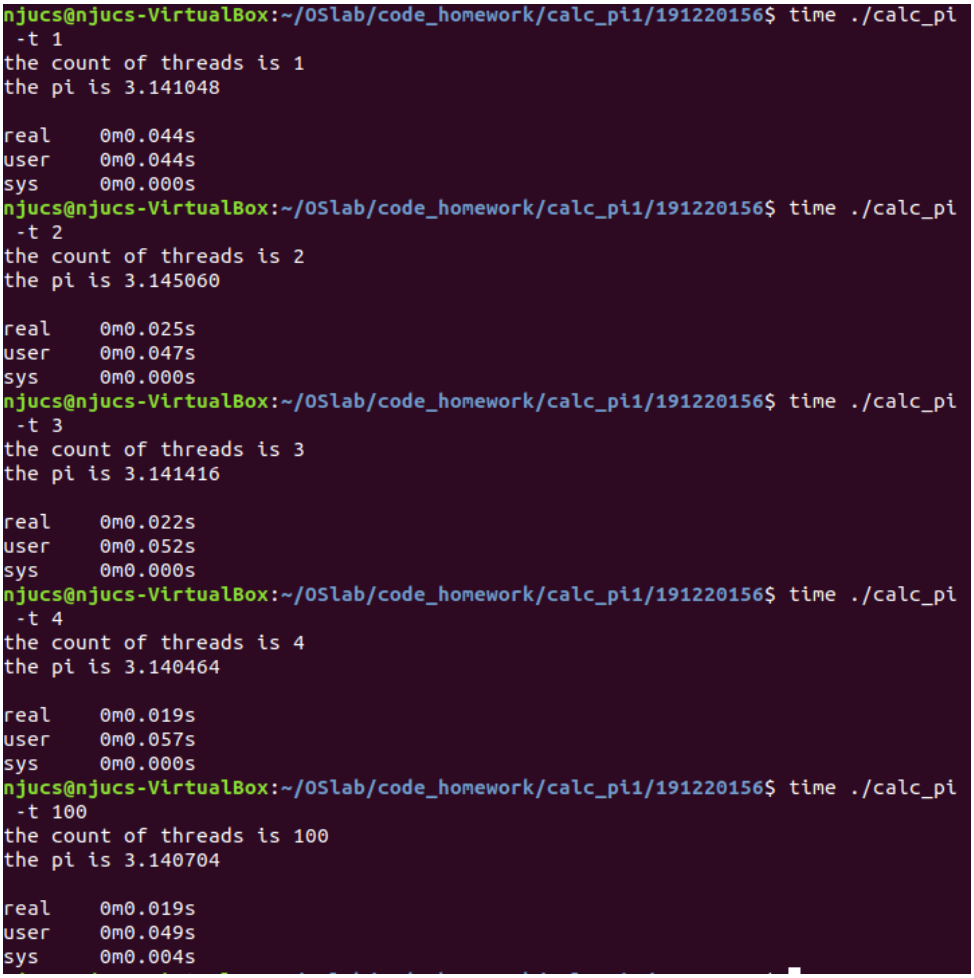
接下来就是确定`rand_r()`函数的种子，虽然随机函数线程安全了，为了使生成数据的随机性更强，我们还需要保证每个线程调用`rand_r()`时有**单独的种子**。可以使用`time()`函数但这显然不够，因为它的精度只有1秒，而很有可能当线程数量较多时，同1秒内有多个线程同时调用`rand_r()`，由于它们的种子一样生成的随机数也一样，随机性大大降低。因此种子尽可能和线程`id`相关联，我的实现是将`seed`设置成`time(NULL) + id`。

内容

- 运行机器的CPU核数
- 因为实验实在虚拟机环境下做的，用命令`cat /proc/cpuinfo| grep "physical id"| sort| uniq| wc -l`查看虚拟机的核数为`4`

```
njucs@njucs-VirtualBox:~/OSlab/code_homework$ cat /proc/cpuinfo| grep "cpu cores
"| uniq
cpu cores       : 4
```

- 不同n值情况下的运行时间和计算结果
    - 为了呈现不同规模的n值情况下的结果，从1个线程开始逐步增大n，通过`time ./cala_pi -t n`命令观察输出的时间信息，结果如下：



    - 可以看到，当只有一个线程的时候，时间开销比较大，而且因为只有一个线程，此时的`real == user`。
    - 当线程数量>=2时，消耗时间比只有一个线程要少，这时应该会有内核进行线程调度的时间，但是由于线程数量很少并且时间的精度只有三位，所以显示不出来`sys`的时间。可以看到由于我的虚拟机内核是`4`个，因此四个线程是消耗的时间是最少的。
    - 当线程数量==100时，这时由于线程数量很多，系统调度消耗的时间变多，因此`sys`时间也大于0了。
- 分析
    - 程序执行消耗的`real`时间包括包括进程在`CPU`上消耗的时间（包括用户态和内核态），进程阻塞的时间和其他进程消耗的时间。`user`时间记录的是所有线程运行在用户态的时间之和，`sys`时间是所有线程运行在内核态的时间之和。当进程数量大于1时，由于我的虚拟机为4核，多线程下程序会利用多核计算的能力，可以并行处理，理论上讲并行执行一个特定进程的几个线程，会使得该程序执行的时间变短。
    - 但是创建线程数量越多，操作系统管理这些线程代价就越大，线程切换的频率越高，因此陷入内核执行调度函数的次数越多，大部分时间都消耗在内核态管理这些线程了，从图中也可以看到`sys`对应的时间变大。
    - 而计算结果并没有太大区别，因为增加线程数量只是改变了程序的执行方式，不会影响到程序本身。本质上程序还是一个`1000000`次随机取点的蒙特卡洛算法，因此计算结果无明显差异。

参考文章

为什么说rand_r()线程安全

rand()函数详解

随机数生成方法