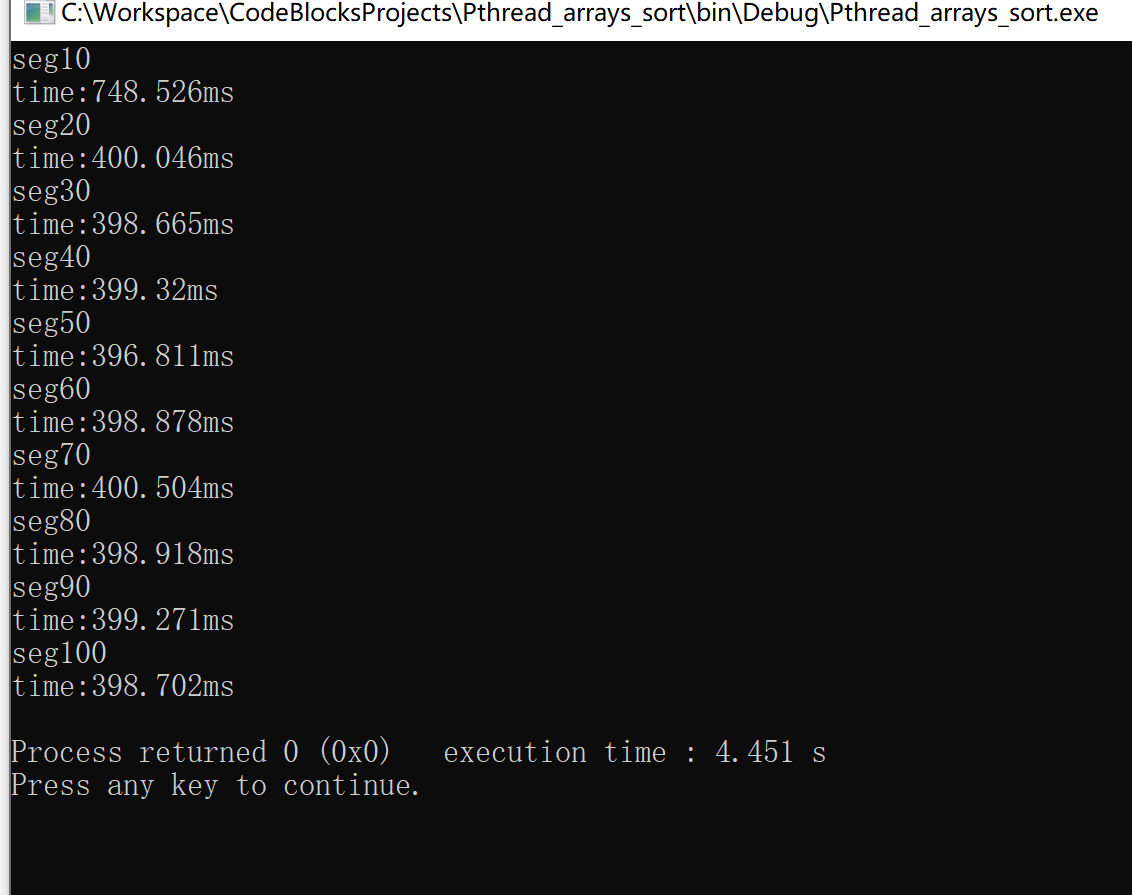
**Phread 编程**

姓名：郁万祥 学号：2013852

1. 对于课件中“多个数组排序”的任务不均衡案例进行复现（规模可自己调整），并探索较优的方案。提示：可从任务分块的大小、线程数的多少、静态动态多线程结合等方面进行尝试，探索规律。

1.1从任务分块的大小入手：

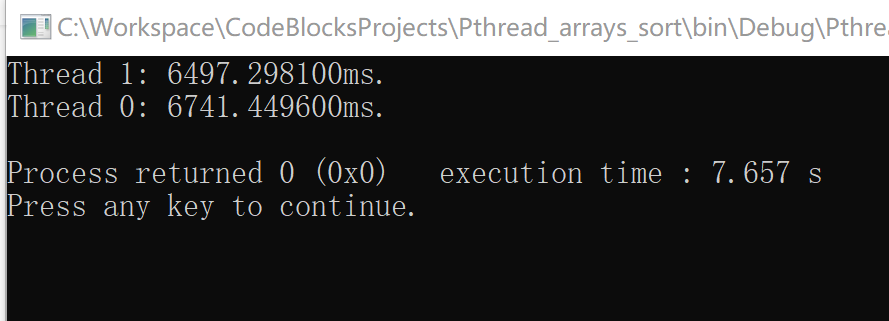
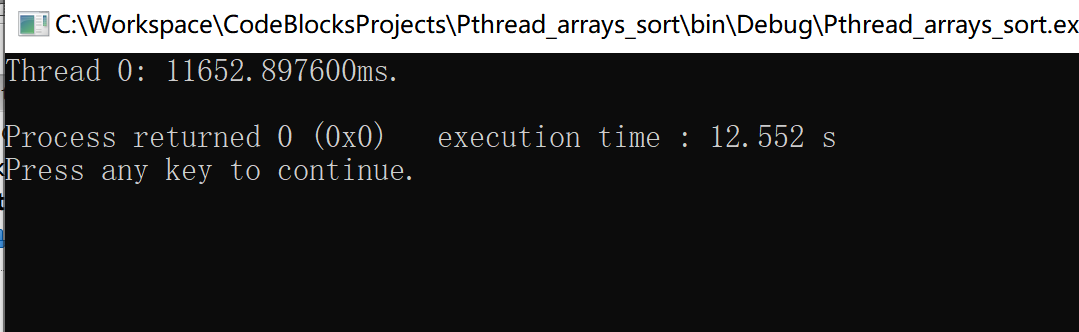
我们保持线程数为4不变，将任务块从10到100每次增加10，观察运行效率，下面是实验结果：

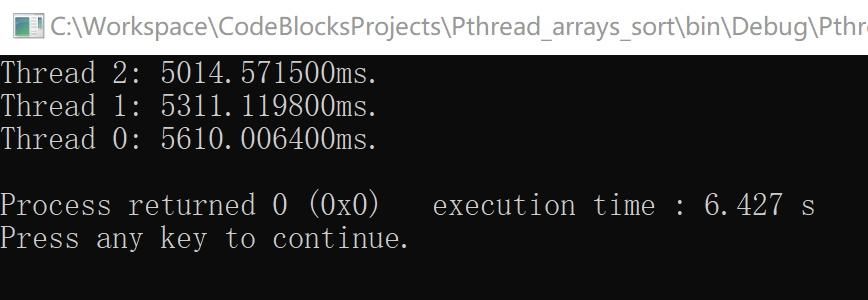


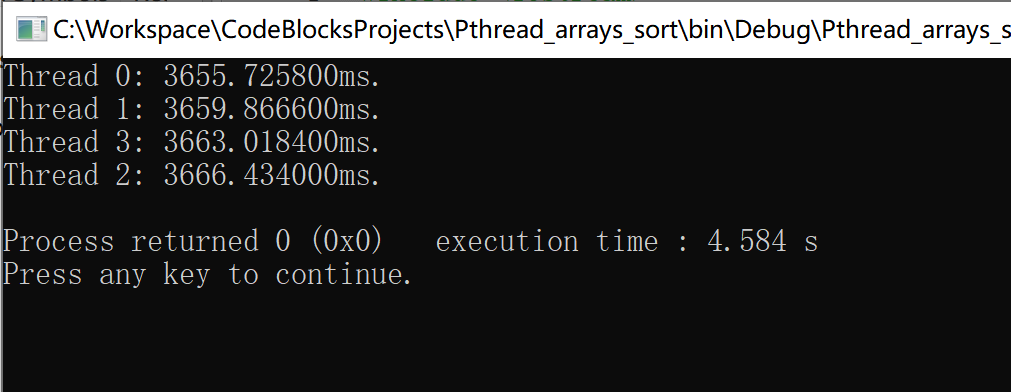
可能是由于数据加载缓冲等影响，第一次的时间显然更长一些，出去第一次的结果，我们发现，seg的改变对并行效率影响并不大，在本次问题当中，seg=50时表现较好一些。

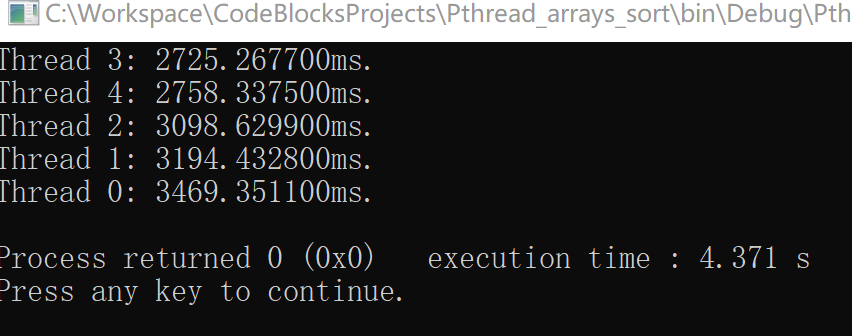
1.2从线程数目少入手：

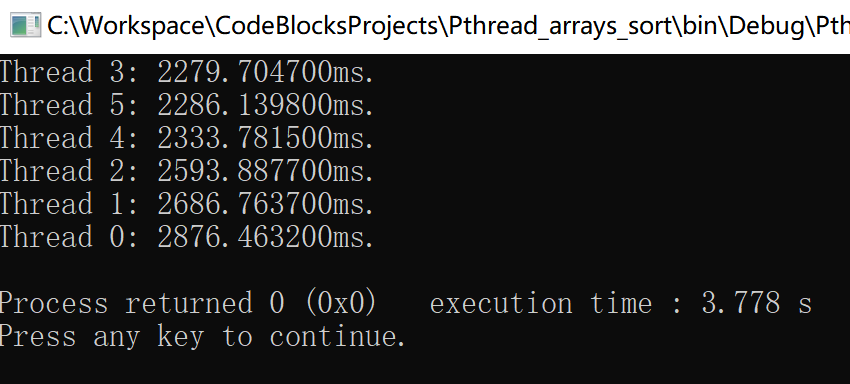
根据上述实验，我们保持seg=50不变，实验线程数目设置为单线程到8个线程，每个线程数目下运行10次，取平均程序运行总时间代价，下面是实验结果：

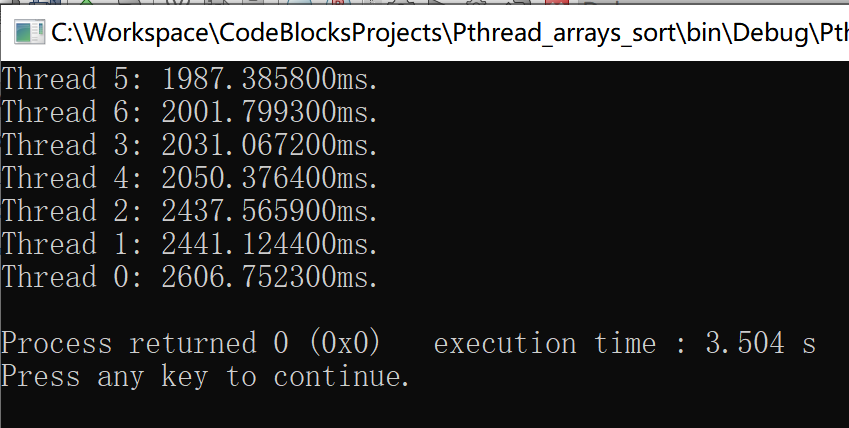


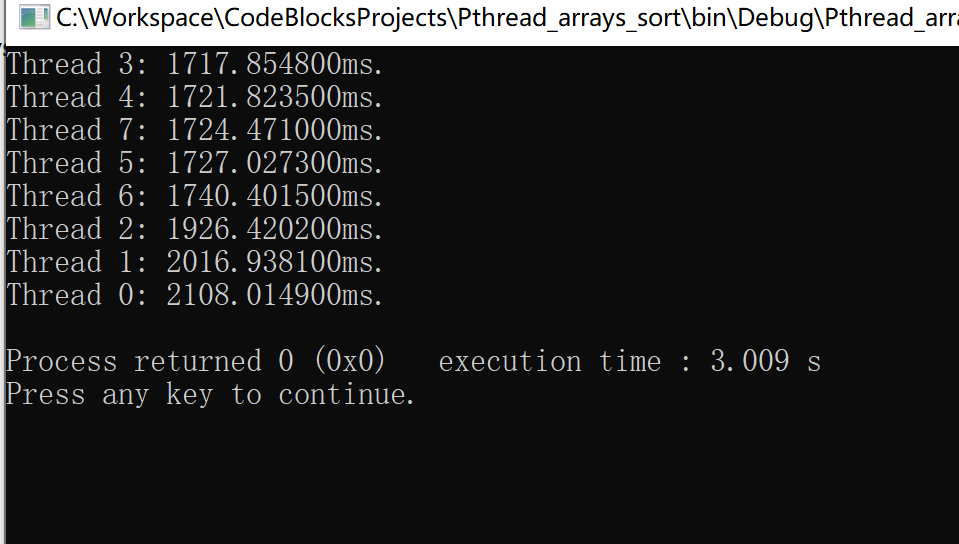










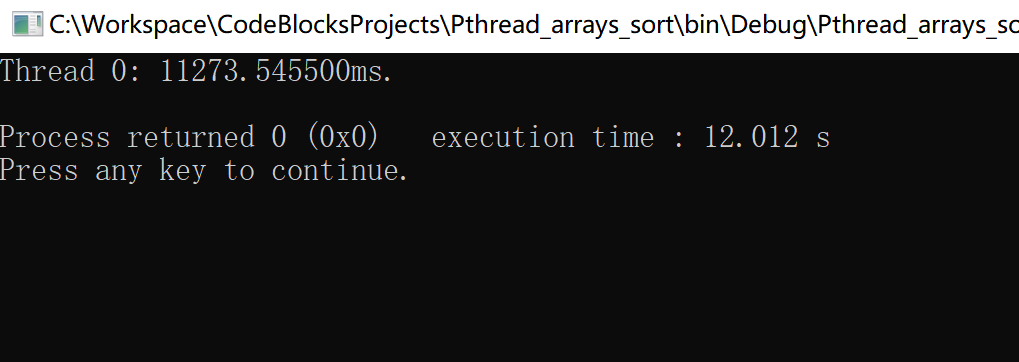


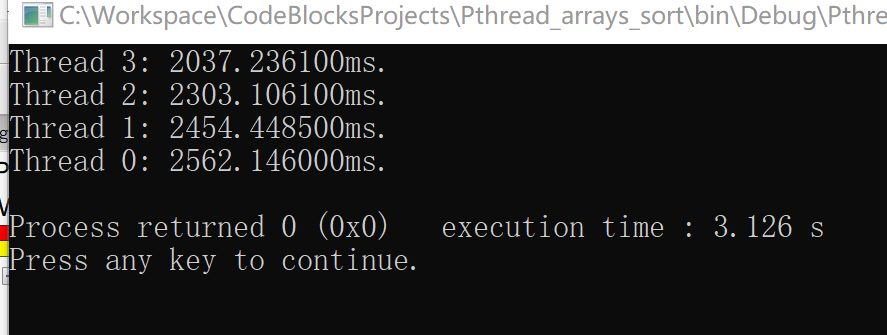
结果分析：

经过计算，我们发现随着线程数目增加，所消耗的总时间，反而会增加，这可能是由于创建线程的本身比较费时，同时静态划分的方式，并不适合创建多线程，所以效率并不高。

1.3从静态动态多线程结合入手：

对于多线程，反而效率不高的问题，我们改进动态任务划分的方式，进行多线程的操作，下面是实验结果：





结果分析：

我们发现通过动态划分任务的方式，四个线程的效率会比单线程的效率高。因此，最后发现，在目前的环境下，设置seg为50，线程数为4，动态任务划分的方式是效率比较高的方式。

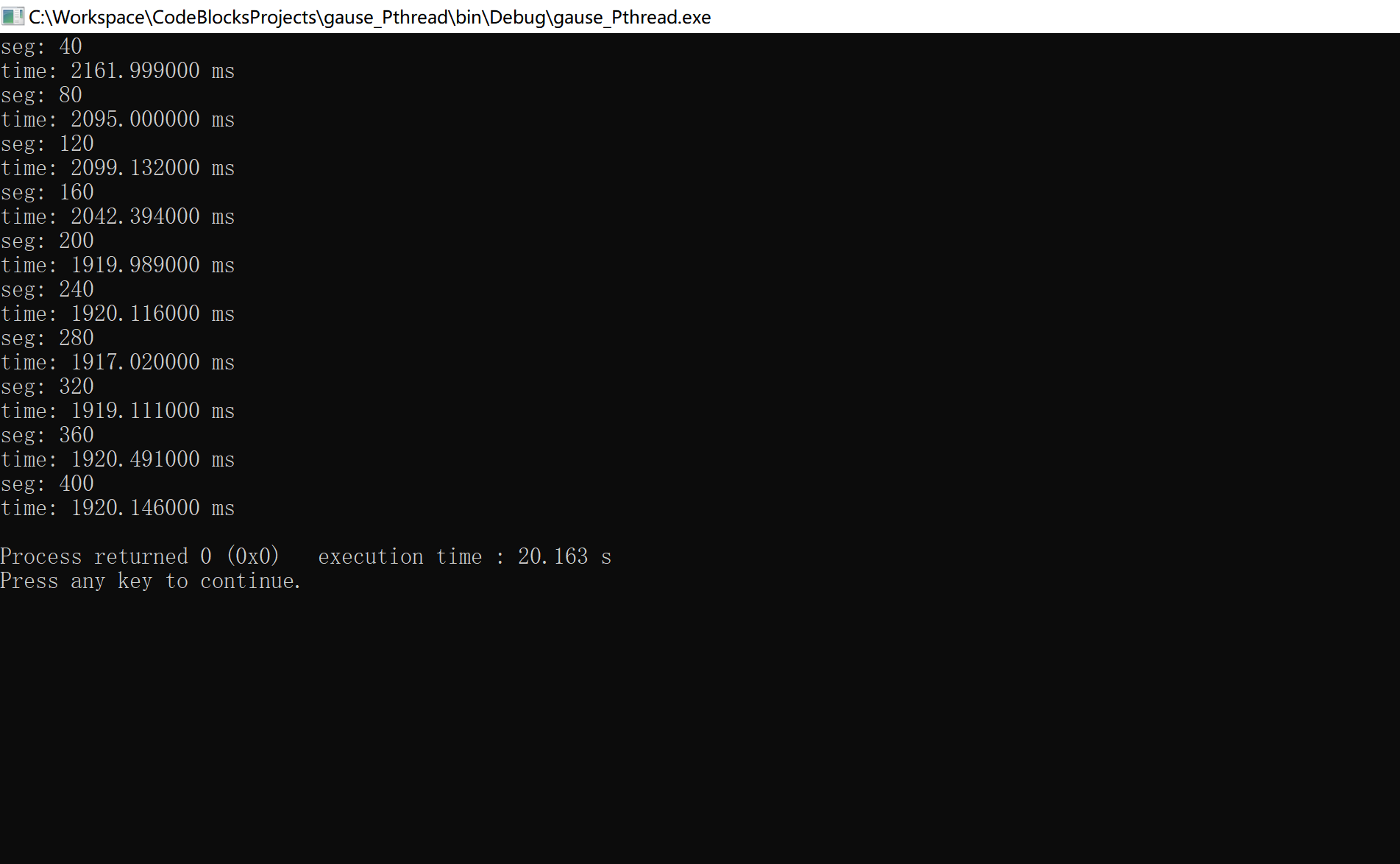
1. 实现高斯消去法解线性方程组的Pthread多线程编程，可与SSE/AVX编程结合，并探索优化任务分配方法。

实验过程：

首先，我们通过第一个实验发现，其实对于较小的矩阵规模，多线程的效果并不明显，会导致，无论任务划分如何，效率都还不错，因此对于任务划分的探索也比较局限，所以在进行高斯并行化实验的过程中，增加矩阵整体的大小，同时相应的，对于任务的分配也应该有所增加，因此实验数据如下：

矩阵规模固定为2048\*2048，线程数固定为4，对于任务的划分，我们设定seg从40到400，每次增加40，来实现任务划分从每次40到每次400的目的，探索高斯消去中任务划分的优化。

实验结果：

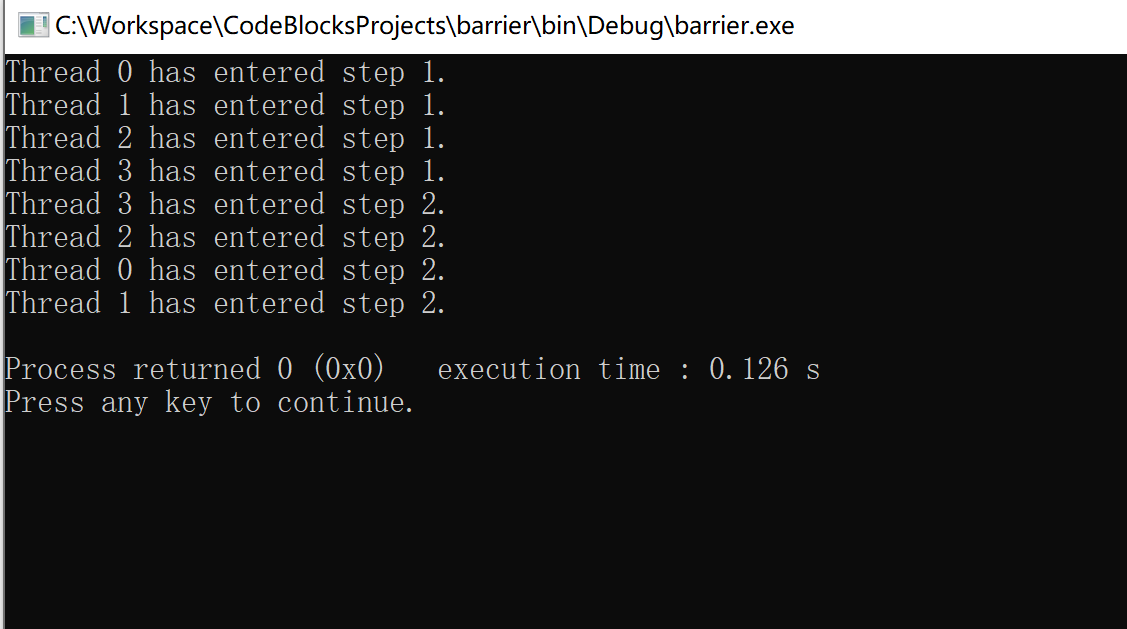


结果分析：

当前情况下，我们发现，当任务划分比较大的时候，运行的效率比较高，其中当任务划分为280时，本次实验的效率最高，所以，其实任务的合适划分需要根据问题的规模进行敲定，问题规模越大，任务划分的优化效果更加明显，达到效率最高所需要的任务划分一般也尽量比较大。

**3、附加题：**使用其他方式（如忙等待、互斥量、信号量等），自行实现不少于2种路障Barrier的功能，分析与Pthread\_barrier相关接口功能的异同。提示：可采用课件上路障部分的案例，用其他2种方式实现相同功能；也可自行设定场景，实现2种或以上barrier的功能，并进行效率、功能等方面的展示比较。

实现效果：



3.1 使用忙等待和互斥量实现路障

使用一个由互斥量保护的共享计数器。当计数器的值表明每个线程都已经进入临界区，所有线程就可以离开忙等待的状态了。

相同： 1、都是线程在执行一个任务后等待其他线程完成。

2、都是使得所有线程在某个地方同步。

不同：接口使用的是wait函数，每个线程完成任务主动调用wait，然后等待同步，忙等待使用的是while循环，使得线程处于忙等待阶段，从而进行同步。

缺点： 1、线程处于忙等待循环时浪费了很多CPU周期，并且当程序中的线程 数过多时，程序的性能有可能会下降的非常厉害。

2、通过这这种方式实现的路障，有多少个路障就要有多少个不同的共享计数器变量来进行计数，非常繁琐。

3.2信号量实现路障

相同： 1、都是线程在执行一个任务后等待其他线程完成。

2、都是使得所有线程在某个地方同步。

在忙等待实现的路障中，使用了一个计数器counter来判断有多少线程抵达了路障。在这里，采用了两个信号量：count\_sem，用于保护计数器；barrier\_sem，用于阻塞已经进入路障的线程。

线程被阻塞在sem\_wait不会消耗CPU周期，所以用信号量实现路障的方法比用忙等待实现的路障性能更佳。

如果想执行第二个路障，counter和count\_sem可以重用，但是重用barrier\_sem会导致竞争条件。