**作业3**

**3.1 pthread函数库可以用来在Linux上创建线程，请调研了解pthread\_create，pthread\_join，pthread\_exit等API的使用方法，然后完成以下任务：**

**（1）写一个C程序，首先创建一个值为1到100万的整数数组，然后对这100万个数求和。请打印最终结果，统计求和操作的耗时并打印。（注：可以使用作业1中用到的gettimeofday和clock\_gettime函数测量耗时）；**

**（2）在（1）所写程序基础上，在创建完1到100万的整数数组后，使用pthread函数库创建N个线程（N可以自行决定, 且N>1），由这N个线程完成100万个数的求和，并打印最终结果。请统计N个线程完成求和所消耗的总时间并打印。和（1）的耗费时间相比，你能否解释（2）的耗时结果？（注意：可以多运行几次看测量结果）**

**（3）在（2）所写程序基础上，增加绑核操作，将所创建线程和某个CPU核绑定后运行，并打印最终结果，以及统计N个线程完成求和所消耗的总时间并打印。和（1）、（2）的耗费时间相比，你能否解释（3）的耗时结果？（注意：可以多运行几次看测量结果）**

**提示： cpu\_set\_t类型，CPU\_ZERO、CPU\_SET宏，以及sched\_setaffinity函数可以用来进行绑核操作，它们的定义在sched.h文件中。请调研了解上述绑核操作。以下是一个参考示例。**

**假设你的电脑有两个核core 0和core1,同时你创建了两个线程thread1和thread2，则可以用以下代码在线程执行的函数中进行绑核操作。**

**示例代码：**

**//需要引入的头文件和宏定义**

**#define \_\_USE\_GNU**

**#include <sched.h>**

**#include <pthread.h>**

**//线程执行的函数**

**void \*worker(void \*arg){**

**cpu\_set\_t cpuset; //CPU核的位图**

**CPU\_ZERO(&cpuset); //图将位图清零**

**CPU\_SET(*N*, &cpuset); //设置位第*N*位为1，表示与core *N*绑定。*N*从0开始计数**

**sched\_setaffinity(0, sizeof(cpuset), &cpuset); //将当前线程和cpuset位图中指定的核绑定运行**

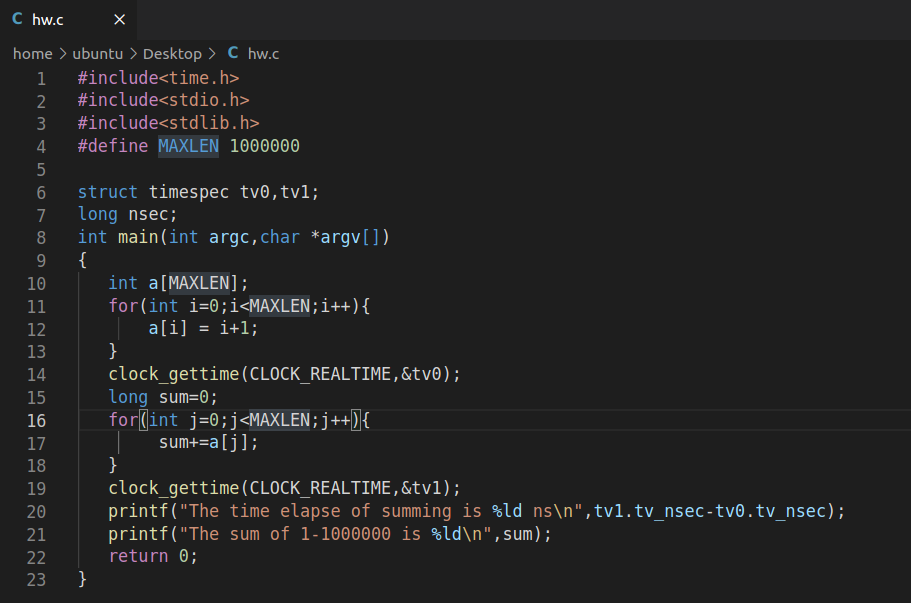
**//其他操作**

**}**

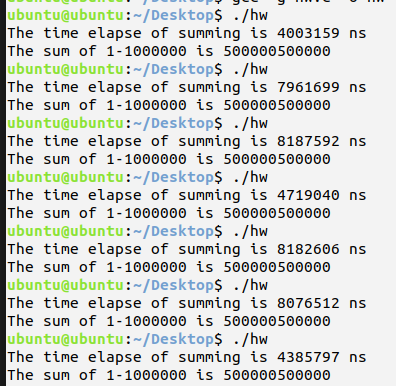
**提交内容：**

1. **所写C程序，打印结果截图等**
2. **所写C程序，打印结果截图，分析说明等**
3. **所写C程序，打印结果截图，分析说明等**

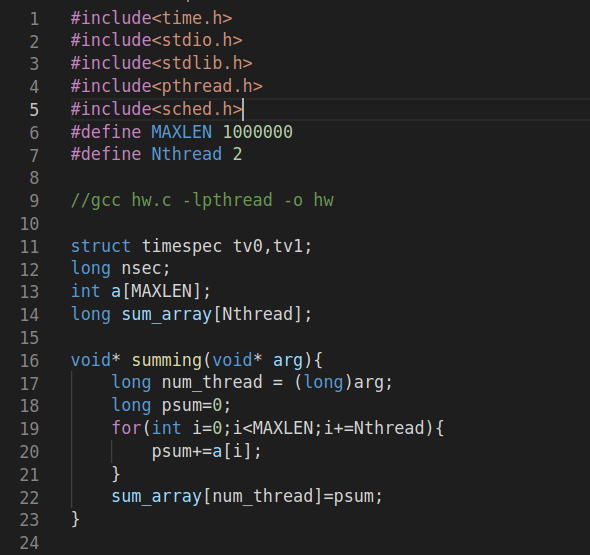
答：（1）**C程序如下：**

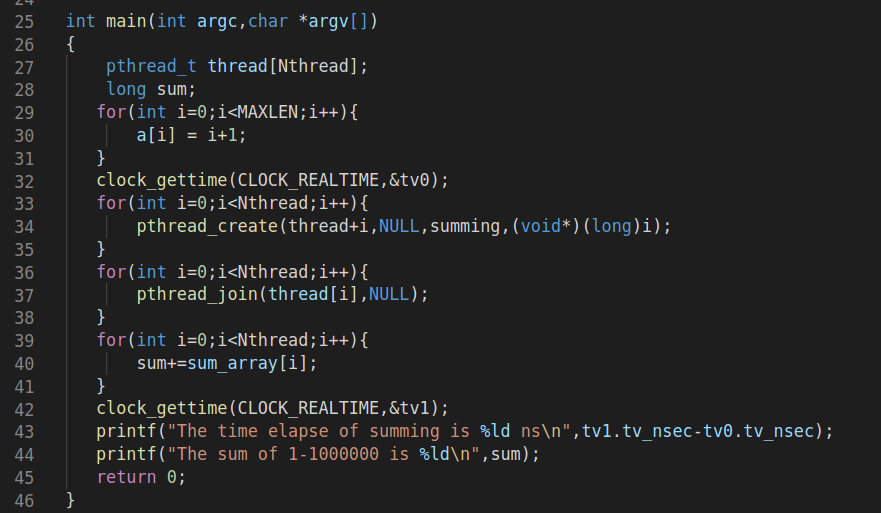


**运行结果如下**：

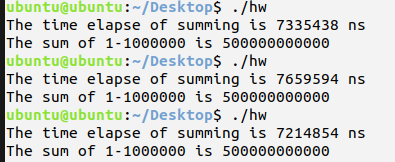


（2）答：**C程序如下：**



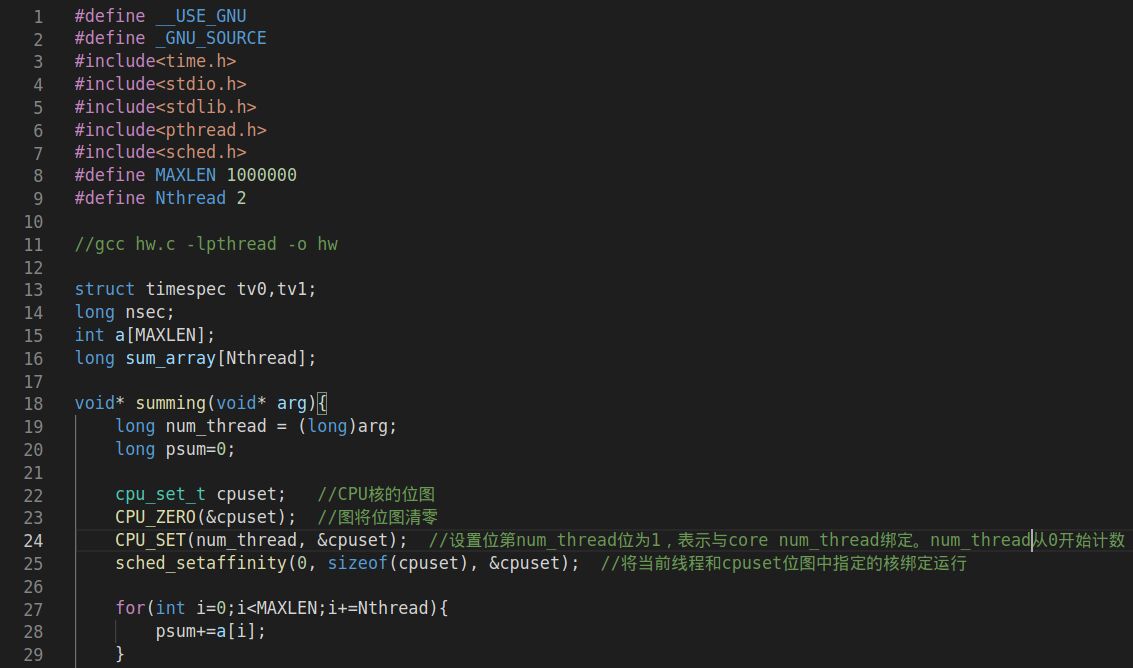


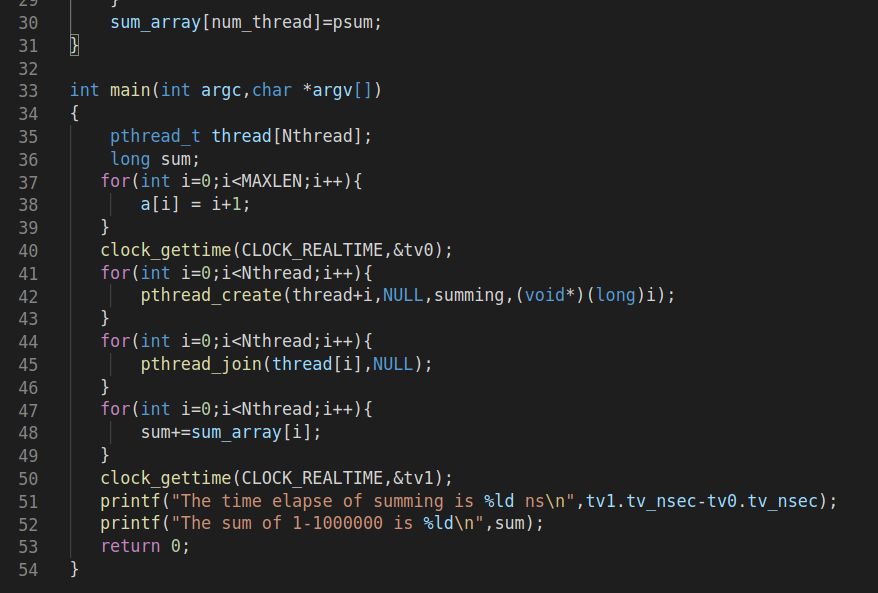
**运行3次结果如下**：



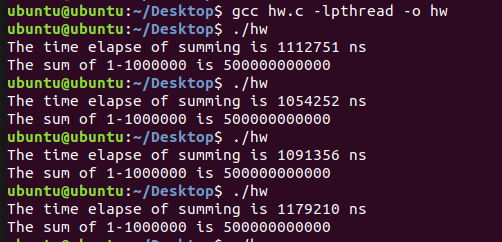
**说明**：我们可以看到开2线程之后运行时间有缩短，但是加速比并不高（不到20%），理论上双线程应当有较高加速比，猜测应当是与线程之间上下文切换、CPU核资源调度所耗费资源太多有关。

（3）**答：C程序如下**：





**运行结果如下：**



**分析：**相比单纯开双线程而言，运行时间显著缩短，缩短为1/7，性能加速比极高！至于为什么会出现这样的结果，推断是因为绑核让线程单独占用CPU资源，硬件资源利用率提高，至于单独占用CPU资源竟能获得如此之高的加速比，也侧面印证了（2）中性能受限很大程度是源于CPU资源调度和分配。

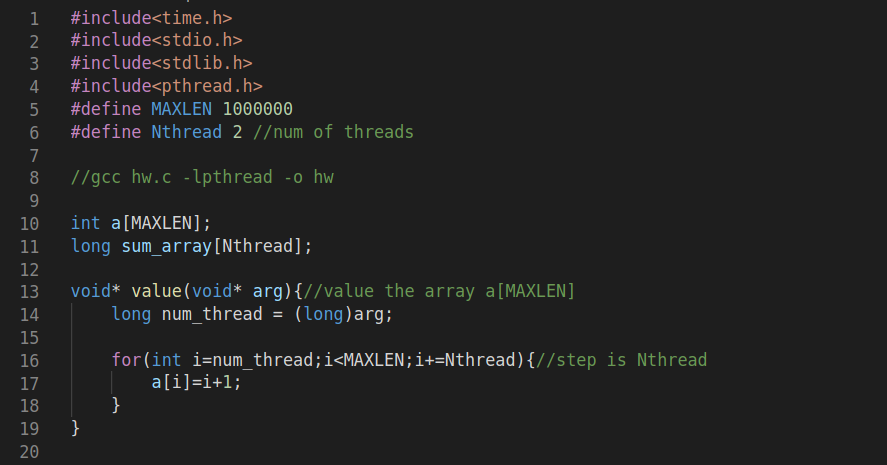
**3.2 请调研了解pthread\_create，pthread\_join，pthread\_exit等API的使用方法后，完成以下任务：**

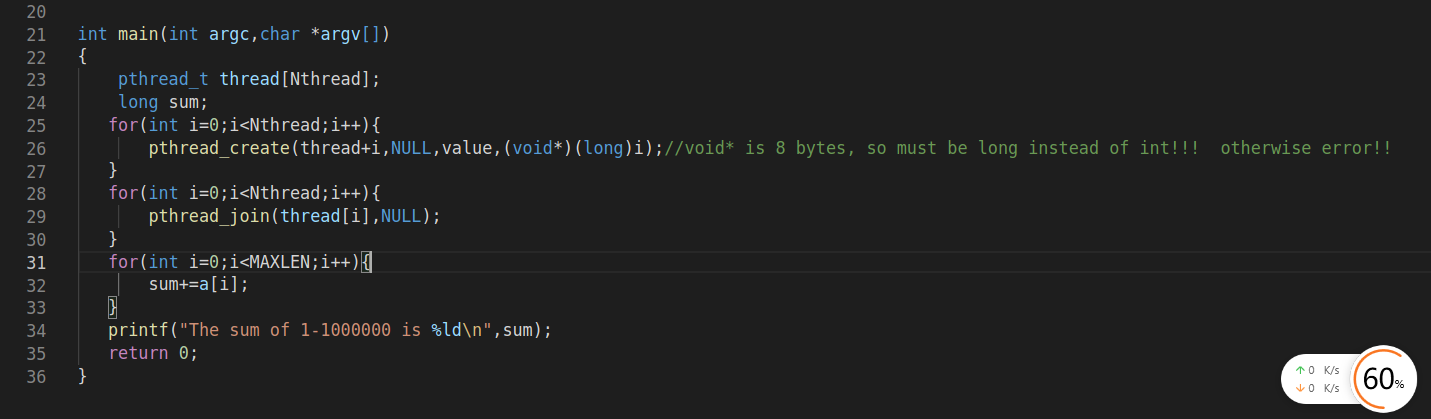
**（1）写一个C程序，首先创建一个有100万个元素的整数型空数组，然后使用pthread创建N个线程（N可以自行决定, 且N>1），由这N个线程完成前述100万个元素数组的赋值(注意:赋值时第i个元素的值为i) 。最后由主进程对该数组的100万个元素求和,并打印结果,验证线程已写入数据。**

**提交内容：**

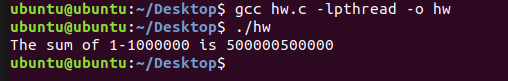
1. **所写C程序，打印结果截图,关键代码注释等**

答：C程序如下：





打印结果截图：



验证线程成功写入数据！