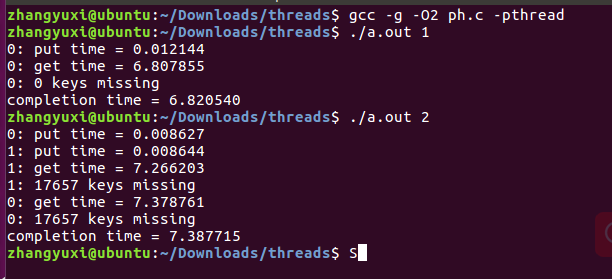
在使用gcc -g -O2 ph.c -pthread成功编译后，按照实验指导书体会一下单线程和双线程的区别



每个线程分两个阶段(phases)运行。在第一个阶段，每个线程将NKEYS/nthread keys放入the hash table。在第二阶段，每个线程从散列表中获取NKEYS。print语句告诉你每个阶段为每个线程花费了多长时间。倒数第二行会告诉你有多少个key丢失了，底部的completion time告诉你应用程序的总运行时间。

ph.c程序主要工作是使用随机数生成10000个keys，然后利用线性hash将其插入5个哈希槽中，最后再取出每个key。

两个发现：

（1）通过以上单个线程和双线程我们可以分析得到，双线程完成的工作是单线程的二倍，但是时间上很接近，

（2）在单线程运行时key的丢失为0，而双线程时对于key的丢失时很严重的.

key的丢失是因为，在用户没有设定线程间的调度策略时，系统默认采取基于时间片轮转的调度策略。所以当NKEYS=100000时，会出现两个线程的多次轮转，也才有可能导致key的丢失。具体过程是当thread1刚进入insert()还未进行插入时，轮转到另一个线程2进行了插入。回到线程1，此时的n还是为空，再进行连接时，就会丢失掉线程2插入的表项。

说白了是当第一个thread还未插入key时，另一个thread的key覆盖了原来key,所以导致了第一个线程key的丢失。

当我们把NKEYS改成10的时候，就算不用put里也不加锁，也并没有丢失任何key，说明了这一点。

通过以上我们了解到为了避免双线程丢失大量key的发生，在put和get中插入lock和unlock语句，使得丢失的键数总是0。

修改内容如下

//首先在开始处定义一个五个互斥锁

pthread\_mutex\_t bucket\_locks[NBUCKET]; //五个是因为有五个存放key的桶，每一个都需要在使用时上锁

//然后再在put函数中加锁和解锁：

static

void put(int key, int value)

{

int i = key % NBUCKET;

pthread\_mutex\_lock(&bucket\_locks[i]);

//上锁

insert(key, value, &table[i], table[i]);

pthread\_mutex\_unlock(&bucket\_locks[i]);

//解锁

}

修改完成后编译运行，结果如下：

