shell

- 姓名：陈姝宇
- 学号：3017218119
- 班级：软工3班

实验内容

在本作业中，您将探索使用哈希表进行线程和锁的并行编程。你应该在一台有多个内核的真正的计算机（不是xv6，不是qemu）上做这个作业。

实验步骤

在本作业中，您将使用`a hash table`(哈希表)研究`threads and locks`的并行编程。您应该在具有多个核心的真实计算机(不是xv6，也不是qemu)上完成这项作业。

在自己的操作系统上使用下面命令

```
➜  thread gcc -g -O2 ph.c -pthread
➜  thread ./a.out 2
```

2指定在哈希表上执行put和get操作的线程数。结果如下：

```
0: put time = 0.004365
1: put time = 0.004453
0: get time = 4.395119
0: 16487 keys missing
1: get time = 4.413804
1: 16487 keys missing
completion time = 4.418394
```

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语句告诉您每个阶段为每个线程花费了多长时间。底部的completion time告诉您应用程序的总运行时间。在上面的输出中，应用程序的完成时间约为4.4秒。每个线程计算大约4.4秒(put ~0.00 + get ~4.4)。

要查看使用两个线程是否提高了性能，让我们与单个线程进行比较$ ./a.out 1:

```
chenshuyu[illegible] ./a.out 1
0: put time = 0.007418
0: get time = 4.193720
0: 0 keys missing
completion time = 4.201333
```

为什么有两个或多个线程丢失键，而一个线程时却不会?识别可能导致两个线程丢失键的事件序列。 可能线程1在线程0执行完insert函数中的e->next = n;后执行了insert操作，此时新增的两个Entry都指向n，并且线程1的Entry作为链表头放入了bucket，然后线程0的Entry也会放入，这样就覆盖了线程1的那个，就是丢失了一个key。(尽管线程1的key1与线程0的key2不相等，但当key1%NBUCKET == key2%NBUCKET == i时，就有可能同时想插入bucket[i]中)

为了避免这个事件序列，在`put`和`get`中插入`lock`和`unlock`语句，使得丢失的键数总是0。相关的pthread调用是：

```
  pthread_mutex_t lock;      // declare a lock
  pthread_mutex_init(&lock, NULL);   // initialize the lock
  pthread_mutex_lock(&lock);  // acquire lock
  pthread_mutex_unlock(&lock);  // release lock
```

修改 ph.c，先声明一把锁，在get和put操作的前后加锁：

```
pthread_mutex_t lock;     // declare a lock
```

```
static
void put(int key, int value)
{
  pthread_mutex_lock(&lock);  // acquire lock
  int i = key % NBUCKET;
  insert(key, value, &table[i], table[i]);
  pthread_mutex_unlock(&lock);  // release lock
}
```

```
static struct entry*
get(int key)
{
  pthread_mutex_lock(&lock);    // acquire lock
  struct entry *e = 0;
  for (e = table[key % NBUCKET]; e != 0; e = e->next) {
    if (e->key == key) break;
  }
  pthread_mutex_unlock(&lock);  // release lock
  return e;
}
```

首先用二个线程测试代码，然后用一个线程测试它。

```
➜ thread gcc -g -O2 ph.c -pthread
➜ thread ./a.out 2
0: put time = 0.003766
1: put time = 0.007942
0: get time = 15.645645
0: 0 keys missing
1: get time = 15.656982
1: 0 keys missing
completion time = 15.674018
➜ thread ./a.out 1
0: put time = 0.008124
0: get time = 8.225299
0: 0 keys missing
completion time = 8.233742
```

确实没有再miss keys了，但是双线程时间变长了。说明双线程并不比单线程版本快，所以有时候多线程并不总是好的。

修改代码，以便在保持正确性的同时并行运行get操作。 答：get只是遍历哈希表，并不会改变哈希表，所以加不加锁都不会导致key丢失。所以单纯只讨论key会不会丢失时get里不用加locks。

修改代码，使某些put操作并行运行，同时保持正确性。

```
static struct entry*
get(int key)
{
  struct entry *e = 0;
  for (e = table[key % NBUCKET]; e != 0; e = e->next) {
    if (e->key == key) break;
  }
  return e;
}
```

实验结果

没有miss key

```
➜ thread gcc -g -O2 ph.c -pthread
➜ thread ./a.out 1
0: put time = 0.008321
0: get time = 7.835665
0: 0 keys missing
completion time = 7.844308
➜ thread ./a.out 2
1: put time = 0.008011
0: put time = 0.014955
0: get time = 16.729821
0: 0 keys missing
1: get time = 16.748570
1: 0 keys missing
completion time = 16.770260
```