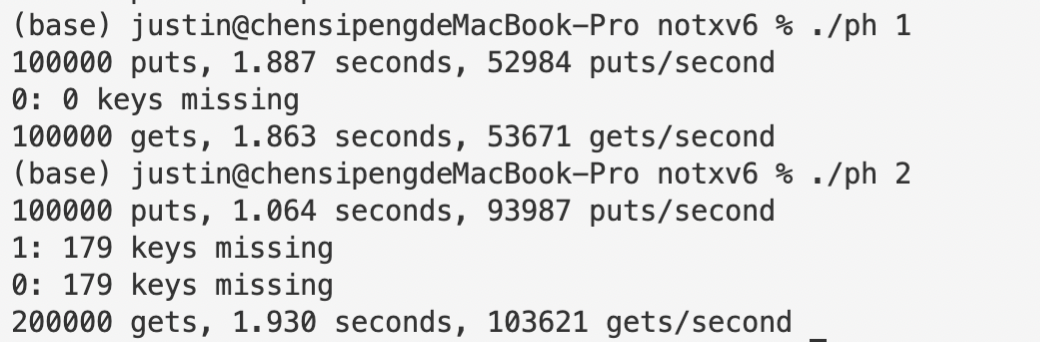
The essence of the problem of missing keys derives from the fact that two separate threads that are executing concurrently both have access to the same address space. In this sense, imagine, for instance, both thread A and B are trying to put two different new keys at the end of the same bucket. Suppose A is executed first, but before the execution could finish, which is when pointer e has already pointed at the end of the linked list but hasn’t yet inserted a new key, thread B cuts in and carries itself out all the way till the end. In this case, what B saw as the end of the list is the same as A, though they are supposed to be different, i.e. the one that B saw should be what A has inserted, and this results in a race condition between A and B where B obliterates the new key that A has inserted. Hence, when consequent threads are trying to get the corresponding keys, they can only see the key that B inserted, which is why there will always be missing keys.



To tackle this problem, if regardless of the efficiency, what we’ll have to do is to simply make every put execution atomic, as this problem could be almost completely attributed to puts being not atomic. Note that it’s fine if get executions are not atomic. The following are the revised part of code and its output.

static

void **put**(int key, int value)

{

**pthread\_mutex\_lock**(&lock);

int i = key % **NBUCKET**;

*// is the key already present?*

struct **entry** \*e = 0;

for (e = table[i]; e != 0; e = e->next) {

if (e->key == key)

break;

}

if(e){

*// update the existing key.*

e->value = value;

} else {

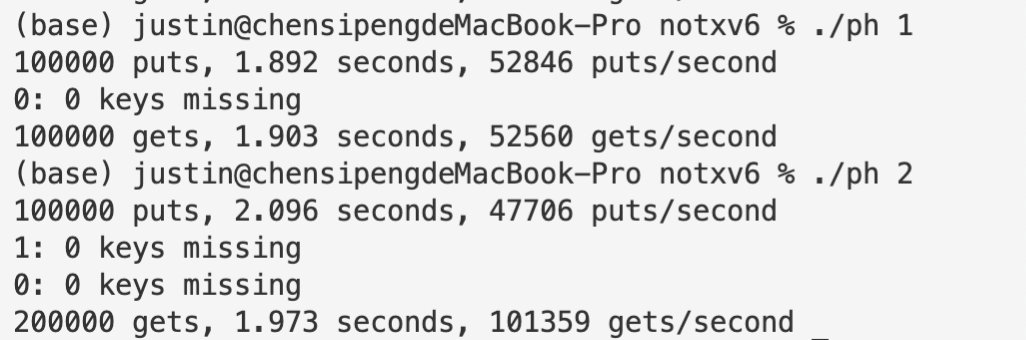
*// the new is new.*

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

}

**pthread\_mutex\_unlock**(&lock);

}



To make the code more efficient, we need to pin down the exact part of code that has to be atomic if we don’t expect any missing. We can do this by allocating different locks for different hash buckets, which will allow threads that are accessing different buckets and hence are immune from race condition to execute simultaneously. We can see that the speed of puts almost doubled after this change.

