CSE Hands-on-2 20161029

Name: ChenJiayang ID: 5140379036

Question1：Study helgrind's output and the ph.c program. Clearly something...

- Because this put() operation is a kind of writing operation, which means that it can easily result to race condition when two threads try to write the same field at the same time.Since in the ph.c, there is no locks to ensure this atomic operation, it will definitely cause incorrectness of the program.
- However, when it comes to only onre thread, there is no mistake because there is no other thread wirting things together.
- If there are two threads named A and B, when A finished setting key and not changing the inuse value, then at this moment, B came to set its own key which has occupied the original A's key, this situation will cause "missing key".

Question2：Describe your changes and argue why these changes ensure...

- First using instruction

```
pthread_mutex_t plock;
```

to declare a lock for put() then

```
assert(pthread_mutex_init(&plock, NULL) == 0);
```

to initialize the lock. Next, modify the whole put() like below:

```
static
void put(int key, int value)
{
assert(pthread_mutex_lock(&plock) == 0);
int b = key % NBUCKET;
int i;
// Loop up through the entries in the bucket to find an unused one:

for (i = 0; i < NENTRY; i++) {
  if (!table[b][i].inuse) {
    table[b][i].key = key;
    table[b][i].value = value;
    table[b][i].inuse = 1;
    assert(pthread_mutex_unlock(&plock) == 0);
    return;
  }
}
assert(0);
}
```

- The reason why this kind of modify works is that after adding lock, it can ensure that there is always one writer to modify the process of adding entry so that key will not be changed by another thread.

Question3：Is the two-threaded version faster than the single-...

- Unfortunately, two-threaded version is not faster than the single-threaded version in the put phase.
- The time of single-threaded version is 3.247758 and the time of two-threaded version is 4.673060.

Question4：Most likely (depending on your exact changes) ph won't achieve...

- Because this exact change just guarantee that at one time there is only one thread to exceute writing operation, so it doesn't make two threads implemented parallel processing, but adding the time cost of thread switching and lock operation.

Question5：You should observe speedup on several cores for the get phase...

- Because get() is a kind of reading operation, if several threads just read the same content, not to modify it, it will not cause race condition.
- After we have removed lock from get(), then several threads can get the specific content at the same time, which can undoubtedly increase the efficiency.

Question6：Why does valgrind report no errors for get()? Can you imagine a e...

- Because in main() function, when all put() operation are done, then it executes get(), so it will not report errors since no other threads wound modify the entries.
- However, if you modify the main() function and change the sequence to make put() and get() interleaved with each other, it may report errors.

Question7：Can you think of a way of modifying ph.c so that you get speedup for...

- Firstly, using a smaller granularity lock to ensure parallel execution is a good idea when it comes to get() and put()
- Another method which I can give an example is that if now we want to get A, we should avoid other put() operation from another thread, however we can still allow get() and put() operated on any other variable, only to guarantee that the put() will not cause race condition to the variable it modified, so we can introduce a new judgment strategy to implement this method.