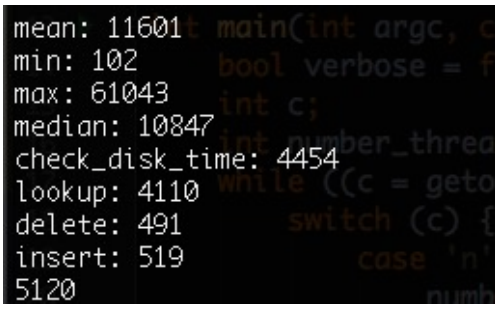
Multicore lab4

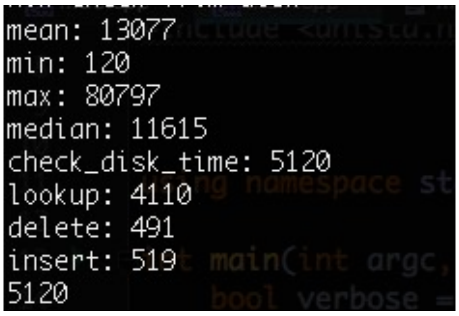
In this lab, I implement an LRU cache using ThreadSafeKVStore and a List. When there is an insert, the item will be pushed into the front of list, and store it's key and position in list into KVstore. If the list stores more than 128 items, it will automatically pop_back the last element. If an item has been looked up and exists(no matter in disk or KVstore), this item will be push to the front of list(and put into KVstore if needed). If the item being looked or deleted not in Cache, the disk will be checked, and costs long time.

To run the server without cache, pass the argument -v.
Here is the result without cache:



```
mean: 11601
min: 102
max: 61043
median: 10847
check_disk_time: 4454
lookup: 4110
delete: 491
insert: 519
5120
```

and it's the result with using cache:



```
mean: 13077
min: 120
max: 80797
median: 11615
check_disk_time: 5120
lookup: 4110
delete: 491
insert: 519
5120
```

When compare the mean time, the time cost has been increased about 12% per request, and there is slight difference between min time, the reason might be checking a file with few lines costs not much long time than search in memory.
The max time for a request might come from checking the whole file and rewriting it, so there is no meaning comparing the max time.
It can be easily inferred that if the cache is large enough, we don't need to find in disk thus the efficiency should be increased as the cache size increase (ignore the time cost to hit when the cache size increased).
Actually the network environment might be changed, so the check_disk_time might be better to measure the performance of the cache. By using the cache, we reduce the times of checking from disk about 13%.
There are several cache implementations like LFU, which can also increase the efficiency in some degrees. We should choose different implementation of cache according to real situations.