1.We'll start by redoing the measurements within this chapter. Use the call gettimeofday() to measure time within your program. How accurate is this timer? What is the smallest interval it can measure? Gain confidence in its workings, as we will need it in all subsequent questions. You can also look into other timers, such as the cycle counter available on x86 via the rdtsc instruction.

I use gettimeofday() to measure, and it is pretty accurate.

2. Now, build a simple concurrent counter and measure how long it takes to increment the counter many times as the number of threads increases. How many CPUs are available on the system you are using? Does this number impact your measurements at all?

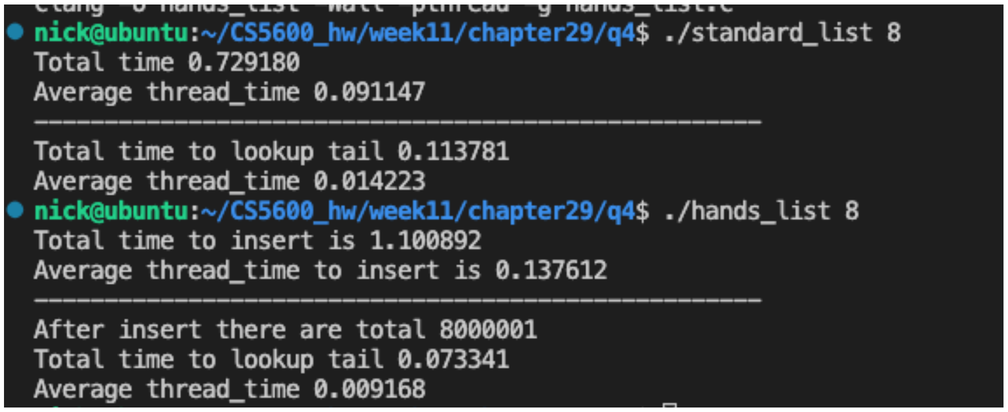
./simple_count_lock [num_thread]
There are 8 CPU are available on my system. So each time increase the number of the thread, per thread to increment 1 million count increase around 20%.

3. Next, build a version of the sloppy counter. Once again, measure its performance as the number of threads varies, as well as the threshold. Do the numbers match what you see in the chapter?

./approximate_counter [threshold]
It actually not match what I see in the chapter. My best guess is I didn't use condition variable, thus, cause the threads spinning and waste cpu time.

4. Build a version of a linked list that uses hand-over-hand locking [MS04], as cited in the chapter. You should read the paper first to understand how it works, and then implement it. Measure its performance. When does a hand-over-hand list work better than a standard list as shown in the chapter?

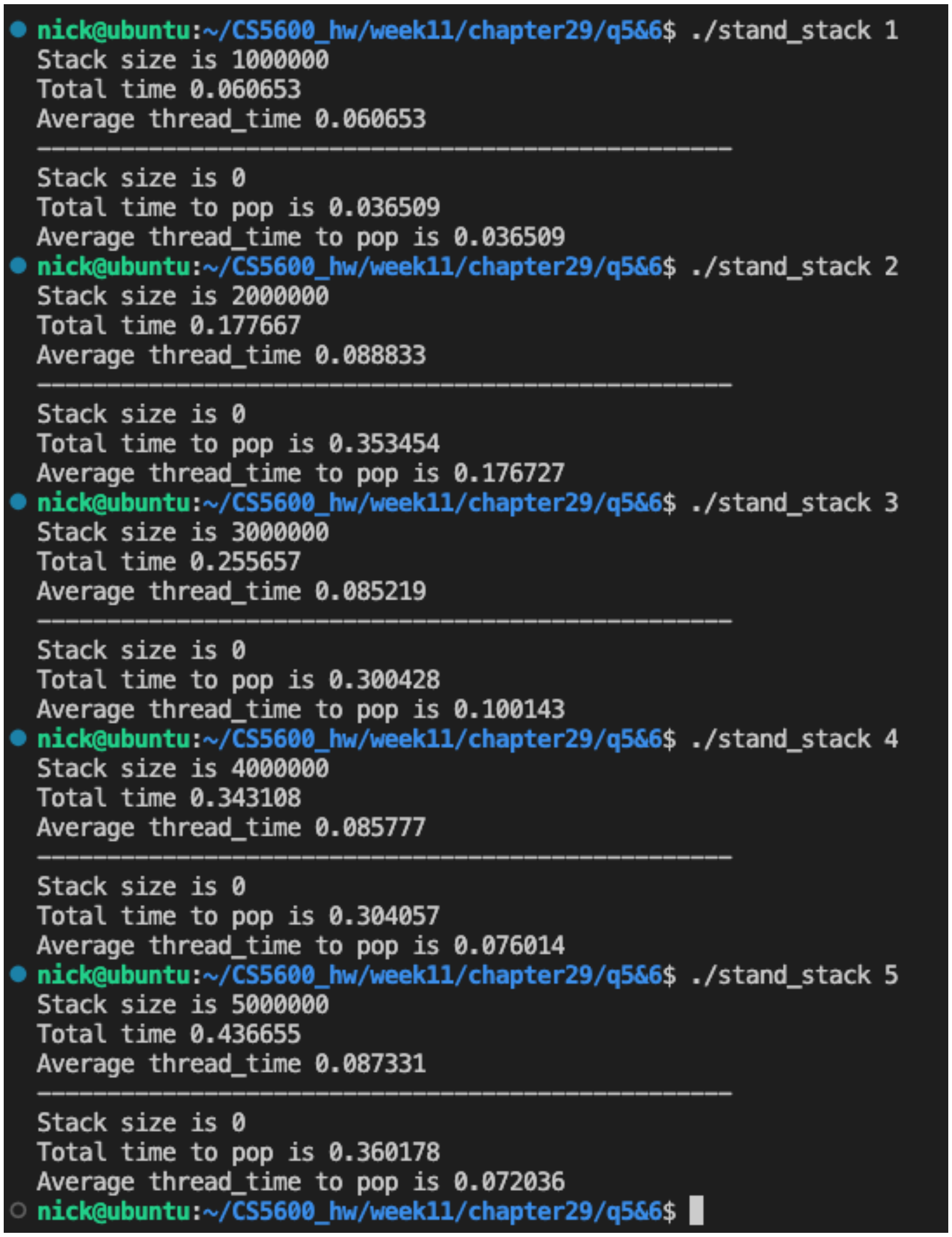
The lookup time doing much better than the standard one.



```
nick@ubuntu:~/CS5600_hw/week11/chapter29/q4$ ./standard_list 8
Total time 0.729180
Average thread_time 0.091147
--------------------------------------------------
Total time to lookup tail 0.113781
Average thread_time 0.014223
nick@ubuntu:~/CS5600_hw/week11/chapter29/q4$ ./hands_list 8
Total time to insert is 1.100892
Average thread_time to insert is 0.137612
--------------------------------------------------
After insert there are total 8000001
Total time to lookup tail 0.073341
Average thread_time 0.009168
```

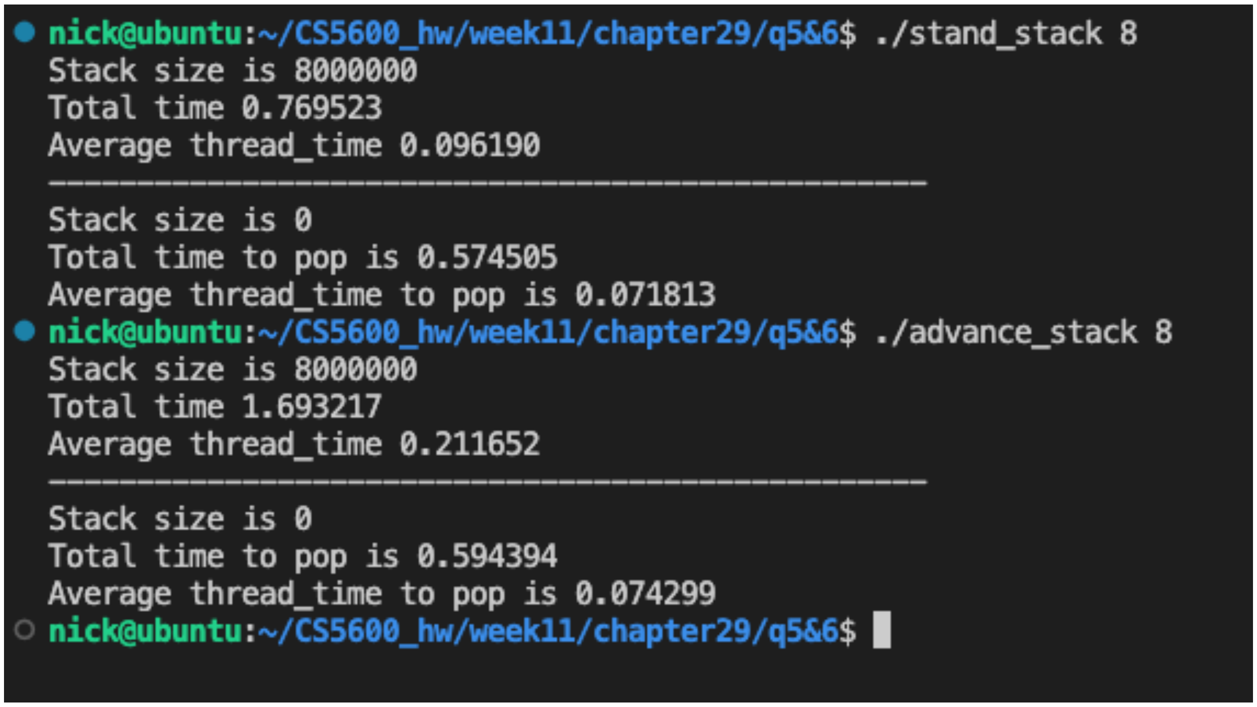
5. Pick your favorite data structure, such as a B-tree or other slightly more interesting structure. Implement it, and start with a simple locking strategy such as a single lock. Measure its performance as the number of concurrent threads increases.

I'm going to pick the stack.



```
● nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ ./stand_stack 1
  Stack size is 1000000
  Total time 0.060653
  Average thread_time 0.060653
  --------------------------------------------------
  Stack size is 0
  Total time to pop is 0.036509
  Average thread_time to pop is 0.036509
● nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ ./stand_stack 2
  Stack size is 2000000
  Total time 0.177667
  Average thread_time 0.088833
  --------------------------------------------------
  Stack size is 0
  Total time to pop is 0.353454
  Average thread_time to pop is 0.176727
● nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ ./stand_stack 3
  Stack size is 3000000
  Total time 0.255657
  Average thread_time 0.085219
  --------------------------------------------------
  Stack size is 0
  Total time to pop is 0.300428
  Average thread_time to pop is 0.100143
● nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ ./stand_stack 4
  Stack size is 4000000
  Total time 0.343108
  Average thread_time 0.085777
  --------------------------------------------------
  Stack size is 0
  Total time to pop is 0.304057
  Average thread_time to pop is 0.076014
● nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ ./stand_stack 5
  Stack size is 5000000
  Total time 0.436655
  Average thread_time 0.087331
  --------------------------------------------------
  Stack size is 0
  Total time to pop is 0.360178
  Average thread_time to pop is 0.072036
○ nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ █
```

6. Finally, think of a more interesting locking strategy for this favorite data structure of yours. Implement it, and measure its performance. How does it compare to the straightforward locking approach?



```
● nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ ./stand_stack 8
  Stack size is 8000000
  Total time 0.769523
  Average thread_time 0.096190
  --------------------------------------------------
  Stack size is 0
  Total time to pop is 0.574505
  Average thread_time to pop is 0.071813
● nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ ./advance_stack 8
  Stack size is 8000000
  Total time 1.693217
  Average thread_time 0.211652
  --------------------------------------------------
  Stack size is 0
  Total time to pop is 0.594394
  Average thread_time to pop is 0.074299
○ nick@ubuntu:~/CS5600_hw/week11/chapter29/q5&6$ 
```

When I put more lock on the stack, the actual performance to push decrease. The pop performance remain the same.