Bilkent University

Department of Computer Engineering

CS342 Operating Systems

Project 2 Report

Authors:	*ID-s:*
Naisila Puka	21600336
Kunduz Efronova	21600469

March 30, 2019

Experiments

Set up

We have written a program called `text.c`, to which we give the size of the hash table, the number of locks, the number of threads and the number of operations per threads as arguments. Based on the arguments provided, this program does the same number of `insert` operations on the specified hash table inside each of the threads, and measures the time taken to do all operations. This report presents results based on this program, by using time as a performance metric. The following notations will be used:

N: The size of the hash table

K: The number of locks

T: The number of threads

W: The number of `insert` operations per thread

For example, for $N = 100$, $K = 10$, $T = 5$ and $W = 150$, we execute the following command:

```
./test 100 10 5 150
```

This means that we have created a hash table of size 100, protected by 10 locks. Then 5 different threads perform 150 `insert` operations with distinct keys each. So in total, $5 \cdot 150 = 750$ `insert` operations will be done in the table. The program will measure the amount of time (in milliseconds) to finish this job. Sample output of the executed command is:

```
Hash Table successfully created!

-------------------------------------------

Doing insert operations ...

For N = 100, K = 10, W = 150, T = 5:

Required time is 446 milliseconds.

-------------------------------------------
```

We have designed some experiments to analyze the effect of each of the above mentioned parameters, and we show the results for each of them in the following subsections.

Number of Locks (K)

In this subsection we will show results for the effect of the number of locks. To measure the effect of (K), we have kept the value of N, T and W the same, and we have increased K, as shown in Figure 1. Here, $N = 100$, $T = 5$ and $W = 150$. Although the size of the table is small and we are not doing many operations, we can still see performance improvement when we increase K. This happens because more locks are given for more and smaller regions of the table, so not all threads will be stuck in the same lock.

N, T, W	K	Time (ms)
100, 5, 150	1	916
100, 5, 150	2	738
100, 5, 150	4	672
100, 5, 150	5	664
100, 5, 150	10	577

Figure 1: Results when changing K

This performance improvement can be better seen for a larger table with much more `insert` operations, as in Figure 2. Here, $N = 1000$, $T = 10$ and $W = 150$. The time to finish the operations by using 1 lock is more than twice the time needed to finish the operations by using 100 locks. With fewer locks, there is really high contention for them from multiple threads trying to access the hash table. This is why more locks show performance increase.

N, T, W	K	Time (ms)
1000, 10, 1500	1	8901
1000, 10, 1500	10	6419
1000, 10, 1500	25	4410
1000, 10, 1500	50	4029
1000, 10, 1500	100	3597

Figure 2: Results when changing K

Number of Threads (T)

In this subsection we demonstrate the impact of number of threads used to perform a specific number of `insert` operations. In order to measure this, we change T and W concurrently. For example, if we want to perform 1000 `insert` operations, for $T = 1$ we use $W = 1000$, whereas for $T = 2$ we use $W = 500$ (each thread does 500 operations, the workload is distributed evenly among the threads).

Normally, since we are distributing the work among different threads, we expect better performance. However, by using more threads, we provide room for contention, since the threads are going to access a critical section: the hash table! Therefore, we observed that the time to finish the specified number of operations was larger as we increased the number of threads, as shown in Figure 4. With only 1 thread, the time was better since there was no contention.

N, K	T, W	Time (ms)
100, 10	1, 1000	484
100, 10	5, 200	670
100, 10	10, 100	1044
100, 10	50, 20	3697
100, 10	100, 10	6247

Figure 3: Results when changing T

In the table shown in 3, we created a hash table of size 100 with 10 locks. The results didn't change when we used a hash table of size 1000 with 50 locks, as shown in 4.

N, K	T, W	Time (ms)
1000, 50	1, 2000	634
1000, 50	10, 200	1157
1000, 50	50, 40	4239
1000, 50	100, 20	6310
1000, 50	200, 10	20715

Figure 4: Results when changing T

Size of the Hash Table (N)

This subsection demonstrates the impact of size of hash-table. In order to measure the of N, we have kept the values of K, T and W the same and increased N as shown in Figure 5. Here, $K = 10$, $T = 10$ and $W = 1500$. We observe from time spent on each size of hash-tables that with the increase of size, we get better performance time. This happens because hash-tables are created with more buckets, therefore, there are less collisions, which means more distribution between different regions of the table. This means there will be less contention for a specific bucket.

K, T, W	N	Time (ms)
10, 10, 1500	100	106354
10, 10, 1500	200	67091
10, 10, 1500	500	38879
10, 10, 1500	700	17750
10, 10, 1500	1000	4355

Figure 5: Results when changing N

Number of Operations per Thread (W)

This subsection demonstrates the impact of number of operations per thread. In order to measure the impact of (W), we have kept the values of K, N and T the same and increased W, as shown in Figure 6. Here, K=10, N=1000 and T=10. We observe from time spent on each size of hash-tables that with the increase of number of operations per thread, we get deteriorating performance time. This happens due to the obvious reason of every thread getting more and more operations to perform, and, moreover, being stuck more in locks.

N, K, T	W	Time (ms)
1000, 10, 10	100	732
1000, 10, 10	500	1125
1000, 10, 10	1000	1841
1000, 10, 10	1500	3734
1000, 10, 10	2000	6072

Figure 6: Results when changing W