問題描述

於 Linux 環境，將作業一的程式碼變成函式或函式庫，讓其他程式呼叫使用。以 C 語言寫一個性能測試程式 (benchmark)。
需有以下三個功能:

甲、產生大量假資料（十萬筆(以上) key-value）(註: 假資料格式自定義，key 需不同)，新增至作業一的 NoSQL DB 與 開源 Redis 的函式庫 (見 eeclass 參考資料 HiRedis)。測試延遲 (time complexity) 與使用的記憶體總量（space complexity)，跟開源的 Redis 比較差距為何。

乙、測試存取十萬筆(以上)平均延遲: 新增 (Create) 與 任意讀取 (Read) 分別量測。

丙、分別觀察作業一 與 Redis 使用的記憶體容量，可以使用任何方法

假資料產生

假資料定義為一個 3 個 bits 的字串，每個 bits 可以由 a~z 與 A~Z 組成，透過 54 進位的方法把數字轉換成字串，而 value 設為隨機 1~10 中的數字

```
char* getKey(int digit)
{
    char key[4] = { 0 };
    char* keyptr=key;
    int i = 0;
    char table[52] = { 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x
    ,'E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z' };
    while (1) {
        int r = digit % 52;
        key[i] = table[r];
        i++;
        digit = digit / 52;
        if (digit == 0)
            break;
    }
    return keyptr;
}
```

作業一延遲

新增:寫一個 100000 次的迴圈，每次先用 getkey()拿出字串之後，呼叫作業一中的 CREATE()函式，並使用 clock start 與 end 量每次 CREATE()跑的時間，最後加總起來取平均。
讀取: 寫一個 100000 次的迴圈，每次先用 getkey()拿出字串之後，呼叫作業一中的 READ()函式，並使用 clock start 與 end 量每次 READ()跑的時間，最後加總

起來取平均。

```
    end = 0;
    for(int i=0;i<100000;i++)
    {
        int value=rand()%10+1;
        keyptr = getKey(i);
        start = clock();
        CREATE(&d, keyptr, value);
        end = clock();
        diff = diff + ((double) end - (double)start);
    }
    printf("average create time: %f  ms\n", diff/100000);
    start = 0;
    end = 0;
    for (int i = 0; i < 100000; i++)
    {
        keyptr = getKey(i);
        start = clock();
        READ(&d, keyptr);
        end = clock();
        diff = diff + ((double)end - (double)start);
    }
    printf("average read time: %f  ms\n", diff/100000);
```

Hiredis 延遲

新增:先透過 redisConnect("127.0.0.1", 6379)連接到 redis，寫一個 100000 次的迴圈，每次先用 getkey()拿出字串之後，並使用 clock start 與 end 量每次 redisCommand(context, "SET %s %s", key, val)跑的時間，最後加總起來取平均。
讀取: 寫一個 100000 次的迴圈，每次先用 getkey()拿出字串之後，並使用 clock start 與 end 量每次 reply = redisCommand(context, "GET %s", key);
跑的時間，最後加總起來取平均。

```
printf("----------------connect success-----------------\n");
for(int i=0;i<100000;i++)
{
    char *key;
    int value;
    key=getKey(i);
    value=rand()%10+1;
    start = clock();
    redisReply *reply = redisCommand(context, "SET %s %c", key,'0'+value);
    end=clock();
    diff = diff + ((double) end - (double)start);
    freeReplyObject(reply);
}
printf("hiredis average create time: %f  ms\n", diff/100000);
diff=0;
start=0;
end=0;
for(int i=0;i<100000;i++)
{
    char *key;
    key=getKey(i);
    start = clock();
    redisReply *  reply = redisCommand(context, "GET %s", key);
    end=clock();
    diff = diff + ((double) end - (double)start);
```

延遲測量結果

```
james@LAPTOP-CPSCMM3H:~/databaseHW$ ./a
my average create time: 0.238380  ms
my average read time: 0.426950  ms
-----------------connect success--------------------
hiredis average create time: 0.859330  ms
hiredis average read time: 0.676300  ms
james@LAPTOP-CPSCMM3H:~/databaseHW$
```

記憶體使用

利用 valgrind 分別對自己與 hiredis 程式測量

自己記憶體用量

```
= HEAP SUMMARY:
=     in use at exit: 2,400,786 bytes in 100,005 blocks
=   total heap usage: 100,006 allocs, 1 frees, 2,400,850 bytes allocated
=
= LEAK SUMMARY:
=    definitely lost: 232 bytes in 2 blocks
=    indirectly lost: 2,400,554 bytes in 100,003 blocks
=      possibly lost: 0 bytes in 0 blocks
=    still reachable: 0 bytes in 0 blocks
=         suppressed: 0 bytes in 0 blocks
```

Hiredis 記憶體用量

```
HEAP SUMMARY:
    in use at exit: 2,058 bytes in 5 blocks
  total heap usage: 1,500,014 allocs, 1,500,009 frees, 24,876,818 bytes allocated

LEAK SUMMARY:
   definitely lost: 208 bytes in 1 blocks
   indirectly lost: 1,850 bytes in 4 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 0 bytes in 0 blocks
        suppressed: 0 bytes in 0 blocks
Rerun with --leak-check=full to see details of leaked memory
```