資料結構 – HW3 報告 *2024 NOV*

姓名：徐崇恆
學號：112503023
系級：通訊二

1 摘要

本作業於 Linux 環境中，以 C 語言開發，為作業一的類 Redis 資料庫支援對串列 (List) 的操作，實現了 LPUSH、LPOP、RPUSH、RPOP、LLEN 和 LRANGE 等功能。透過採用雙向鏈結串列作為資料結構，重構資料庫設計，支援以指令操作資料庫，提升操作效能。

2 需求分析

Redis 中的串列是由字串組成的鏈結串列 (linked list)，屬於有限且有序的集合抽象資料型態，其中每個值可重複出現。串列常用於實現堆疊 (stack)、佇列 (queue)，以及後台工作系統的佇列管理。

在本作業中，需要支援的串列 (list) 操作包括：

1. **LPUSH**
   句法：LPUSH key element [element …]
   將一個或多個元素從串列左側（頭部） 插入，回傳串列長度。
2. **LPOP**
   句法：LPOP key [count]
   將一個或多個元素從串列左側（頭部）彈出，回傳被移除的元素。count 為選擇性參數，默認值為 1 。
3. **RPUSH**
   句法：RPUSH key element [element …]
   將一個或多個元素從串列右側（尾部）插入，回傳串列長度。
4. **RPOP**
   句法：RPOP key [count]
   將一個或多個元素從串列右側（尾部）彈出，回傳被移除的元素。count 為選擇性參數，默認值為 1 。
5. **LLEN**
   句法：LLEN key
   回傳串列長度。若串列不存在則回傳 0 。
6. **LRANGE**
   句法：LRANGE key [start] [stop] （Redis 原句法：LRANGE key start stop）
   回傳指定範圍內的元素。範圍基於索引從 0 開始，並以 -1 表示最後一個元素。回傳範圍包含的位於 start 和 stop 的元素。與原版 Redis 不同的是，在本作業中將 start 和 stop 都作為選擇性參數，它們的默認值分別為 0 和 -1。

3 設計

串列 (List)

本作業使用雙向鏈結串列作為資料庫串列的資料結構。雙向鏈結串列能高效支援左側操作和右側操作，使這些操作僅需更新頭部或尾部的指位器。同時，雙向遍歷也能提高範圍查詢的效能，能使其從最接近的一端開始遍歷。此外，串列的長度屬性將被記錄，並且在每次操作時更新，以使長度查詢同樣為 O(1)，避免了遍歷節點所帶來的額外計算開銷。

但是，雙向鏈結串列也存在一定的空間開銷，每個節點需額外儲存前後指位器，記憶體使用率相對單向鏈結串列較高。此外，操作的複雜性也有所增加，由於需要在插入或刪除節點時同步更新相鄰節點的指位器以維持結構的一致性，因此操作較為繁瑣。

資料庫儲存機制重構

由於作業一的題目並未明確要求資料庫設計需效仿 Redis，最初的資料庫設計參考了其他 NoSQL 資料庫的設計思路，考慮到值 (value) 支援多種資料型別，包括 JSON 格式的物件或陣列等。

在原有的作業一資料庫中，值採用了 cJSON 結構作為儲存格式。然而，若要實現類似 Redis 的 List 操作，並需要更精細的操作控制，顯然以 cJSON 結構儲存資料值已不再適合。

因此，重構了資料庫的存儲邏輯，將資料庫中值的型別簡化為僅支援字串，以更貼近 Redis 的設計模式。在資料庫重構的過程中，重新設計了處理機制，並支援了資料庫指令的使用，以提升整體效能與操作效率。

4 實作

開發環境

硬體：Raspberry Pi 5
作業系統：Raspberry Pi OS Lite (64-bit) (Linux-based OS)
IDE：Visual Studio Code (with SSH Remote)
編譯器：GCC
版本控制：Git, GitHub

雙向鏈結串列 (Doubly linked list)

在本作業中，使用了雙向鏈結串列實現對資料庫串列的支援。串列的結構包含頭部指位器 (head)、尾部指位器 (tail) 和長度屬性 (length)；節點的結構則包含資料指位器 (data) 以及指向前一節點 (prev) 和後一節點 (next) 的指位器。

在實作左側插入和刪除操作 (LPUSH、LPOP) 時，直接更新頭部指位器和相關節點的前後指位器，使操作維持在 O(1) 的時間複雜度。右側的插入和刪除 (RPUSH、RPOP) 則類似，透過操作尾部指位器實現。同時，每次插入或刪除後，及時更新串列的長度屬性，確保長度查詢的效率。

為了最佳化範圍查詢操作 LRANGE，實作中加入了根據索引範圍選擇遍歷方向的機制。當查詢範圍接近串列尾部時，從尾部開始逆序遍歷，減少節點訪問次數，提升查詢效率。

```c
typedef struct DLNode
{
  char *data;
  struct DLNode *prev;
  struct DLNode *next;
} DLNode;

typedef struct DLList
{
  DLNode *head;
  DLNode *tail;
  db_uint_t length;
} DLList;
```

Figure 1. DLList (Doubly Linked List) 及其節點的 struct 定義

5 測試與結果

本作業中所有的測試結果均符合預期，程式的功能都能夠正常執行，沒有明顯的錯誤。

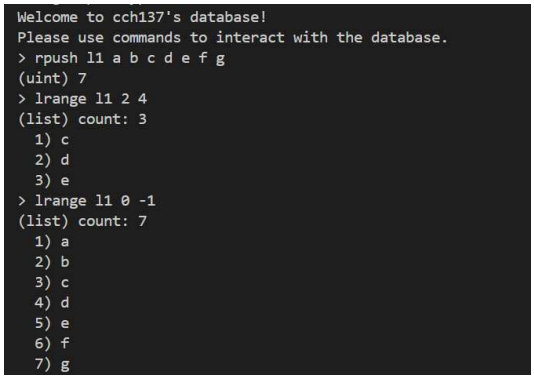

Figure 2. 測試 `rpush` 與 `lrange`

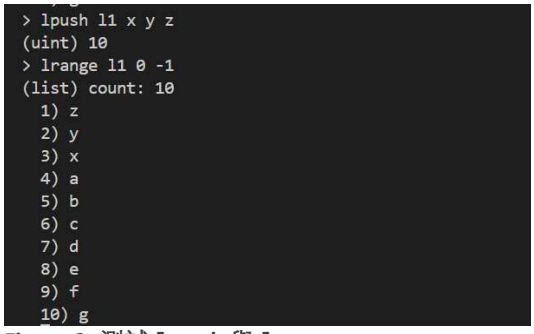

Figure 3. 測試 `lpush` 與 `lrange`

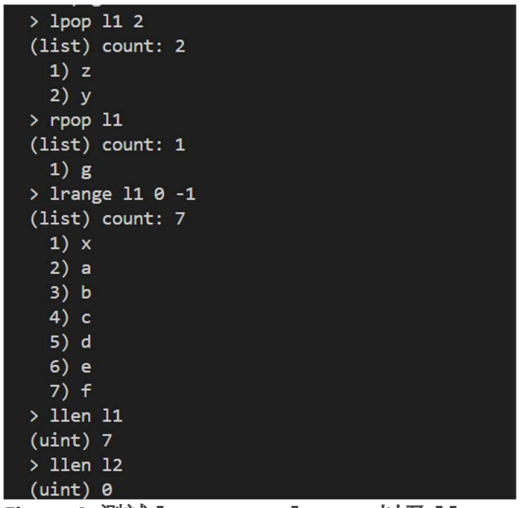

Figure 4. 測試 `lpop`, `rpop`, `lrange` 以及 `llen`

6 討論

Redis 的多執行緒機制

在查閱資料的過程中，發現 Redis 的核心邏輯採用單執行緒設計，透過簡化架構，避免多執行緒可能帶來的資料競爭與鎖機制開銷，並結合非阻塞 I/O 和事件驅動機制，高效處理大量請求。由於 Redis 的操作基於記憶體，資料讀寫速度極快，其效能瓶頸主要來自網路傳輸而非內部運算，因此單執行緒即可應對大多數應用場景的需求。在本次作業中，對資料庫程式進行了重構，並參考了此設計機制。

Jemalloc

在作業二中，觀察到 Redis 的記憶體消耗遠低於自行開發的資料庫。經查閱相關資料，發現其中一個原因是 Redis 採用了 jemalloc 編譯。jemalloc 是一種高效的記憶體分配器，專注於降低碎片化並提升高併發效能，廣泛應用於資料庫程式等需頻繁記憶體操作的高效能場景。因此，在本次作業中，於編譯時採用 jemalloc 取代傳統的 malloc。

休眠機制

為降低資料庫在閒置狀態下的資源消耗，本作業中實作了一種漸進式休眠機制。當資料庫在 100 毫秒內未接收到任何請求時，啟動休眠計時器，透過逐步增加的睡眠時間段，減少 CPU 資源的使用。即使進入休眠狀態，資料庫仍定期執行維護操作，如檢查是否需要進行 rehashing 等。此設計在閒置和高負載場景下平衡了系統的運行效能，避免無效的忙等待，並確保能即時處理新請求。

7 結論

本次作業成功地以雙向鏈結串列為基礎，實現了高效的串列操作支援，並重構資料庫結構，使其更貼近 Redis 的運作邏輯，同時支援以指令操作資料庫。此外，在編譯過程中引入 jemalloc 替代傳統記憶體分配器，顯著提升了記憶體管理效率和操作效能。本次作業亦參考了 Redis 的單執行緒設計理念，簡化併發處理的複雜度，進一步強化系統的穩定性與效能表現。

8 參考文獻與資料

1. Redis - Docs
   https://redis.io/docs/latest/
2. Redis lists | Docs
   https://redis.io/docs/data-types/lists/
3. Doubly Linked List Tutorial - GeeksforGeeks
   https://www.geeksforgeeks.org/doubly-linked-list/

9 附錄

1. 函式庫 cJSON GitHub 倉庫 - DaveGamble/cJSON: Ultralightweight JSON parser in ANSI C
   https://github.com/DaveGamble/cJSON
2. 函式庫 jemalloc GitHub 倉庫 - jemalloc/jemalloc - GitHub
   https://github.com/jemalloc/jemalloc