# Socket Programming報告

王若琳 111502001 資工三A

### Prompt 設計

本次優化的目標是提升 `client.c` 和 `server.c` 程式碼的品質。設計 prompt 時，將重點放在以下方面：

1. **可讀性**：請提升 `client.c` 和 `server.c` 的可讀性，包含變數命名、函數命名與註解補充，讓程式碼易於理解。
2. **結構優化**：請針對 `client.c` 和 `server.c` 重構，將邏輯模組化並減少重複代碼，優化結構設計。
3. **去耦合**：請幫我將 `server.c` 的廣播邏輯與客戶端管理分離成獨立函數，減少耦合度，讓程式模組化。
4. **可維護性**：請幫我提升 `client.c` 和 `server.c` 的可維護性，包含一致的錯誤碼處理與必要的日誌記錄。
5. 錯誤處理：請優化 `client.c` 和 `server.c` 的錯誤處理機制，確保資源正確釋放，並加強錯誤記錄。

### 使用模型

使用 OpenAI GPT-4 (或類似高效能 LLM) 進行程式碼重構。模型的強項在於語意理解與提供高質量的程式建議。

---

## 優化後程式碼與原始程式碼比較

## 客戶端程式 `client.c` 的改進

### 主要優化方向

1. **結構化**：將核心邏輯拆分成獨立函數，避免主函數過長。
2. **錯誤處理**：統一錯誤處理，避免程式崩潰。
3. **命名一致性**：變數與函數命名更具描述性。

### 優化範例

在連線管理部分，重構前後的比較：

- 原始程式碼：

```
if (connect(*sock, (struct sockaddr*)&server, sizeof(serve
r)) < 0) {
    perror("Reconnection failed");
    close(*sock);
    return 0;
}
```

- 優化後：

```
if (connect(*sock, (struct sockaddr*)&server, sizeof(serve
r)) < 0) {
    perror("Failed to reconnect to the server");
    close(*sock);
    return CONNECTION_ERROR;
}
```

**改進摘要**

- 增加狀態碼 `CONNECTION_ERROR`，明確函數返回值意義。
- 改善錯誤訊息，方便除錯。

---

## 伺服器端程式 `server.c` 的改進

**主要優化方向**

1. **去耦合**：獨立客戶端處理邏輯，例如將廣播、客戶端管理拆分為模組化函數。
2. **同步優化**：強化多線程的同步機制。
3. **效能提升**：針對客戶端超時情況進行改良，減少不必要的資源浪費。

**優化範例**

針對廣播訊息部分的優化：

- 原始程式碼：

```
for (int i = 0; i < client_count; i++) {
    if (clients[i].socket != exclude_sock) {
        if (send(clients[i].socket, message, strlen(messag
e), 0) < 0) {
            perror("Broadcast failed");
        }
    }
}
```

- 優化後：

```
for (int i = 0; i < client_count; i++) {
    if (clients[i].socket == exclude_sock) continue;

    if (send(clients[i].socket, message, strlen(message),
0) < 0) {
        fprintf(stderr, "Failed to send message to client
%d: %s\\n", i, strerror(errno));
    }
}
```

**改進摘要**

- 加入 `strerror` 提供具體錯誤訊息。
- 使用 `fprintf` 將訊息導向標準錯誤輸出，便於診斷。

---

## 評估 LLM 的有效性與局限性

**有效性**

- **自動化結構優化**：LLM 善於識別模式並進行重構，減少重複代碼。
- **語意一致性**：透過自然語言理解，優化變數與函數命名。
- **高效性**：生成的優化建議能快速實現功能改進。

**局限性**

1. **上下文限制**：模型對大型專案可能無法全面理解，需補充人工檢視。
2. **專案特性不足**：對於特定領域的程式碼，可能缺乏必要的背景知識。
3. **調整成本**：模型的建議有時過於通用，需開發者額外調整。

---

## 除提升程式碼品質外的應用探討

1. **程式碼文件化**：生成程式碼的完整文檔，包括函數描述、使用說明。
2. **測試案例生成**：透過分析程式邏輯，自動生成單元測試。
3. **錯誤模擬與修復**：模擬程式可能出現的問題並提供修復建議。
4. **性能分析**：透過代碼結構建議提升執行效能。