

# Bonus

## Libcli :

### Introduction :

Libcli 提供一個Shared Library，內含類似命令行介面(Command line interface)的功能

可以執行歷史指令，身分驗證和調用自定義的函數

如果要在程式中使用的话，我們就需要它的函式庫版本－libcli

### Compile:

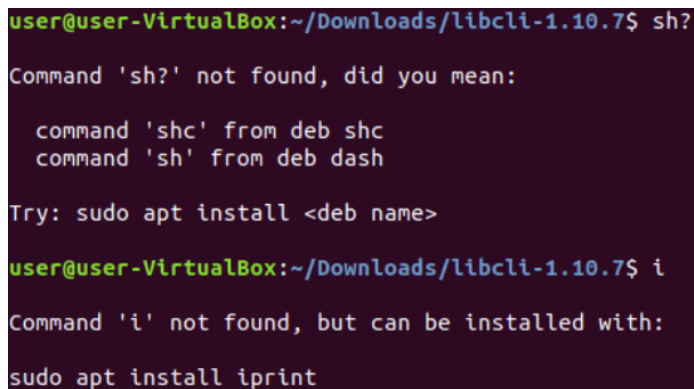
編譯需要的原始檔可以在網站下載

在原始碼所在的目錄下，輸入：

```
$tar zxvf lib-1.10.7.tar.gz //將檔案解壓縮
$make //編譯
$sudo make install //
```

內含搜尋功能，若是打出所要指令的一部分

會顯示出含有搜尋文字的指令



```
user@user-VirtualBox:~/Downloads/libcli-1.10.7$ sh?
Command 'sh?' not found, did you mean:
  command 'shc' from deb shc
  command 'sh' from deb dash
Try: sudo apt install <deb name>

user@user-VirtualBox:~/Downloads/libcli-1.10.7$ i
Command 'i' not found, but can be installed with:
sudo apt install iprint
```

編譯時會利用Compile-Time 判定要使用 `cli_loop()` 裡面的 `select()` 或是 `poll()`

#Compile-Time 會幫你確認清楚 if statement 中會執行到哪一個判斷式對應到的區塊，從而把其他一定不會用到的區塊 "拔掉"

一般情況下會使用 `select()` 函式

如果編譯使用'CFLAGS=-DLIBCLI\_USE\_POLL make'，則會調用 `poll()` 函式

接下來在 `cli_loop()` 中進行一項額外檢查，以確保傳遞的文件描述符在範圍內。

如果不在範圍內，將發送一條錯誤消息，並且 `cli_loop()` 將在子進程中以 `CLI_ERROR` 退出。

一般來說會將Shared Library 檔儲存在 `/usr/local/lib`

要記得在Makefile 裡面導向這個位置

## Commands :

1. help : 印出命令指令列表
2. quit : 結束 `cli_loop()`
3. exit : 關掉終端機
4. logout : 登出
5. history : 紀錄之前下過的指令

## Coding :

開頭要記得加上標頭檔 `#include <libcli.h>`

編譯時後面要加 `-lcli` 指令

使用流程大致是這樣：

初始化 → 身分證明 → 設定 → Socket → 執行 → 關閉

## Example code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <libcli.h>

int32_t cmd_hello(struct cli_def *cli, char *command, char *argv[], int argc)
{
    cli_print(cli, "hello world");
    return 0;
}

int32_t cmd_test(struct cli_def *cli, char *command, char *argv[], int argc)
{
    cli_print(cli, "called %s with %s\r\n", __FUNCTION__, command);
    return CLI_OK;
}

int32_t cmd_set(struct cli_def *cli, char *command, char *argv[], int argc)
{
    if (argc < 2)
    {
        cli_print(cli, "Specify a variable to set\r\n");
        return CLI_OK;
    }
    cli_print(cli, "Setting %s to %s\r\n", argv[0], argv[1]);
    return CLI_OK;
}
```

```

int main(void)
{
    struct cli_def *cli;
    int32_t sockfd, acceptfd;
    struct sockaddr_in saddr, raddr;
    unsigned int on = 1;
    unsigned int rlen = sizeof(struct sockaddr_in);

    /*初始化*/
    cli = cli_init();
    cli_set_hostname(cli, "usr");//名稱決定
    cli_set_banner(cli, "Hello!");//顯示Hello給連線者

    /*身分證明*/
    cli_allow_user(cli, "fred", "nerk");
    cli_allow_user(cli, "foo", "bar");

    /*設定開始*/
    cli_register_command(cli, NULL, "hello", cmd_hello, PRIVILEGE_UNPRIVILEGED, MODE_ANY, NULL);
    cli_register_command(cli, NULL, "test", cmd_test, PRIVILEGE_UNPRIVILEGED, MODE_EXEC, NULL);
    cli_register_command(cli, NULL, "set", cmd_set, PRIVILEGE_UNPRIVILEGED, MODE_EXEC, NULL);

    /*Socket*/
    if((sockfd=socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        perror("socket");
        exit(-1);
    }

    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = INADDR_ANY;
    saddr.sin_port = htons(12345);

    memset(&raddr, 0, sizeof(raddr));
    raddr.sin_family=AF_INET;
    if(setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) != 0)
    {
        perror("setsockopt");
        exit(-1);
    }
    if(bind(sockfd, (struct sockaddr*)&saddr, sizeof(saddr)) == -1)
    {
        perror("bind");
        exit(-1);
    }
    if(listen(sockfd, 12) != 0)
    {
        perror("listen");
        exit(-1);
    }
    /*執行*/
    while((acceptfd=accept(sockfd, (struct sockaddr*)&raddr, &rlen)) > 0)
    {
        while(cli_loop(cli, acceptfd) == 0);
    }
    /*關閉*/
    cli_done()
    return 0;
}

```

## 初始化：

`cli_init()`

這隻函式會提供所需的結構

`struct cli_command` 和 `struct cli_def`

`cli_set_hostname(cli, "名稱")` 可以設置名稱

用戶連接後，如果有設置 `cli_set_banner(cli, "問候")` 會顯示問候

## 身分驗證：

接著，有兩種身分驗證的方法

- `cli_allow_user(char *username, char *password)`

選擇username 或 password 連接

- `cli_set_auth_callback(callback)`

將username和password作為字串傳遞，如果用戶具有訪問權限，則必須返回 `CLI_OK`，否則返回 `CLI_ERROR`

## 設定：

`cli_register_command(struct cli_def *cli, struct cli_command parent, char *command, int (callback)(struct cli_def *, char *, char **, int), int privilege, int mode, char *help)`

**\*cli：**

代入 `struct cli_def` 結構

```
struct cli_def
{
    int completion_callback;
    struct cli_command *commands;
    int (*auth_callback)(char *, char *);
    int (*regular_callback)(struct cli_def *cli);
    int (*enable_callback)(char *);
    char *banner;
    struct unip *users;
    char *enable_password;
    char *history[MAX_HISTORY];
    char showprompt;
    char *promptchar;
    char *hostname;
    char *modestring;
    char *newline;
    int privilege;
    int mode;
    int state;
    struct cli_filter *filters;
    void (*print_callback)(struct cli_def *cli, char *string);
    FILE *client;
};
```

**parent：**

若不為 `NULL`，存儲這個命令，作為整個命令的父級。

**\*command：**

命令名稱

**(callback)(struct cli\_def \*, char \*, char \*\*, int) :**

代入需要的函式

**privilege :**

- `PRILEGE_PRIVILEGED` : `callback` 函式需要 `*argv[]` 代入
- `PRILEGE_UNPRIVILEGED` : `callback` 函式不需要 `*argv[]` 代入

**mode :**

- `MODE_EXEC` : 需配置模式
- `MODE_CONFIG` : 不需配置模式

**help :**

用戶若按?會跳出

**SOCKET :**

`struct sockaddr_in` 結構 :

用來儲存伺服端的IP及port

```
struct sockaddr_in
{
    short          sin_family;    //使用IPv4地址
    struct in_addr sin_addr;      //具体的IP地址
    unsigned short sin_port;      //端口
    char           sin_zero[8];   //為0
};

struct in_addr
{
    unsigned long s_addr;
};
```

使用 `socket(int domain, int type, int protocol)` , 它能幫助我們在kernel中建立一個socket, 並傳回對該socket的檔案描述符

**domain :**

socket要在哪個領域溝通AF\_UNIX/AF\_LOCAL

- `AF_UNIX/AF_LOCAL` : 用在本機程序間的傳輸, 讓兩個程序共享一個檔案系統
- `AF_INET` : 讓兩台主機透過網路進行資料傳輸, IPv4協定
- `AF_INET6` : 讓兩台主機透過網路進行資料傳輸, IPv6協定

**type :**

傳輸的手段 SOCK\_STREAM

- `SOCK_STREAM` : 提供一個序列化的連接導向位元流, 可以做位元流傳輸。對應的protocol為TCP

- `SOCK_DGRAM`：提供的是一個一個的資料包，對應的protocol為UDP

`protocol`：

設定socket的協定標準，一般來說都會設為0，讓kernel選擇type對應的默認協議。

### Return Value：

成功產生socket時，會返回該socket的檔案描述符(socket file descriptor)

若socket創建失敗則會回傳-1，也就是 `INVALID_SOCKET`

### 設置 `struct sockaddr_in`：

初始化：`memset(struct sockaddr_in, int c, size_t n);`

- `info.sin_family`：指定address family時一般設定為AF\_INET
- `info.sin_addr.s_addr`：網絡字節順序

`info.sin_addr.s_addr = htonl(INADDR_ANY);`

`htonl()`：將32位的主機字節順序轉化為32位的網絡字節順序

`INADDR_ANY`：所有地址”、“任意地址”

- `info.sin_port`：端口

`info.sin_port = htons(...);`

`htons()`：將主機的無符號短整形數轉換成網絡字節順序

`bind(int sock, struct sockaddr *addr, socklen_t addrlen);`

將套接字和IP、端口綁定

`listen(int sock, int backlog);`

- `sock`：為需要進入監聽狀態的套接字
- `backlog`：請求隊列的最大長度

`accept(int sock, struct sockaddr *addr, socklen_t *addrlen);`

返回一個新的套接字來和客戶端通信

- `sock` 是服務器端的套接字
- `addr` 保存了客戶端的IP地址和端口號

---

`cli_unregister_command(command)`

若是不想使用某個命令，可以使用上面的函數

```
cli_set_context(cli, context)
```

```
cli_get_context(cli)
```

可以將用戶定義的上下文帶到callback

## 執行：

```
cli_loop(cli, sock)
```

做好傳輸控制協定(Transmission Control Protocol)連接後

執行函式，與用戶連接

如果使用的是 `cli_loop()` 裡面的 `select()`

若是sock超過範圍(FD\_SETSIZE) ， `cli_loop()` 會向自己與用戶端報錯

離開 `cli_loop()` 後，會將結果回傳

## 關閉：

宣告 `cli_done()` 釋放結構

學號：40947013S

姓名：孫韻婷