

# 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話，我們就需要它的函式庫版本—libcli

### Compile:

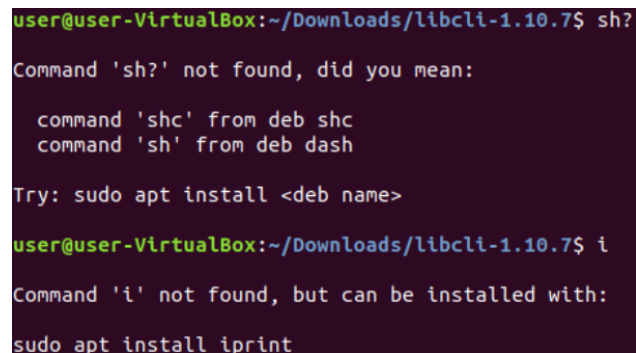
編譯需要的原始檔可以在網站下載

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 → 執行 → 關閉

## Example code :

```
#include <libcli.h>

int main(int argc, char *argv[])
{
    struct sockaddr_in info;
    struct cli_command *c;
    struct cli_def *cli;
    int on = 1, x, s;
    /*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, "ex1", cmd_test, PRIVILEGE_UNPRIVILEGED, MODE_EXEC, NULL);
    cli_register_command(cli, NULL, "ex2", NULL, PRIVILEGE_UNPRIVILEGED, MODE_EXEC, NULL);
    cli_register_command(cli, NULL, "ex3", cmd_set, PRIVILEGE_PRIVILEGED, MODE_EXEC, NULL);
    // 會形成兩命令 ex4ex5 , ex4ex6
    c = cli_register_command(cli, NULL, "ex4", NULL, PRIVILEGE_UNPRIVILEGED, MODE_EXEC, NULL);
    cli_register_command(cli, c, "ex5", cmd_test, PRIVILEGE_UNPRIVILEGED, MODE_EXEC, NULL);
    cli_register_command(cli, c, "ex6", cmd_test, PRIVILEGE_UNPRIVILEGED, MODE_EXEC, "Show the counters that the system uses");
    //Socket部分
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
    bzero(&info, sizeof(info));
    info.sin_family = AF_INET;
    info.sin_addr.s_addr = htonl(INADDR_ANY);
    info.sin_port = htons(12345);
    bind(s, (struct sockaddr *)&info, sizeof(info));

    listen(s, 50);
    while ((x = accept(s, NULL, 0))) //若x符合套接字，則迴圈持續
    {
        /*執行*/
        cli_loop(cli, x);
        close(x);
    }
    /*關閉*/
    cli_done(cli);
    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

<1> `cli_allow_user(username, password)`

選擇username 或 password 連接

<2> `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(parent, cli_register_command(...), command, callback, privilege, mode, help)`

1. `parent` : 為一個 `cli_command *` 結構，可以拿來建構命令
  2. `cli_register_command(...)` 若不為 `NULL`，存儲這個命令 `cli_register_command(...)` 並將其用作整個命令的父級。
  3. `command` : 指令名稱
  4. `callback` : 代入需要的函式
  5. `privilege` : 若 `callback` 函式需要 `argv[]` 代入 `PRILEGE_PRIVILEGED`，不需要則代入 `PRILEGE_UNPRIVILEGED`
  6. `mode` : 代入 `MODE_EXEC`
  7. `help` :
- 

## `callback` 常會用到的函式：

```
int 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;
}

int 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;
}
```

`cmd_test()` :

只是將輸入的命令回顯給客戶端

`cmd_set()` :

處理 `argv[]` 上給出的參數

---

使用 `socket(domain, type, 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`

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

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

`struct sockaddr_in` :

用來儲存伺服端的IP及port

`bzero()` :

初始化 `struct sockaddr_in` (清零)

設置 `struct sockaddr_in` :

`info.sin_family`

指定address family時一般設定為AF\_INET

```
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 = htons(...);
```

htons() 將主機的無符號短整形數轉換成網絡字節順序

```
bind(s, (struct sockaddr*)&info, sizeof(info));
```

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

---

```
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