

tcp_echo_server.c

```
//bind the protocol to socket
//cast the ptr to generic socket address structure
if(bind(listen_fd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0){
    perror("bind :");
    exit(EXIT_FAILURE);
}

//listening, and define the action when catching SIGCHLD signal
listen(listen_fd, MAX_CONNECTION);
signal(SIGCHLD, sigchld_handler);
```

用 signal 定義一下收到 SIGCHLD 訊號要幹嘛，定義在 sigchld_handler 裡面。

```
while(1){
    cli_len = sizeof(cli_addr);
    connect_fd = accept(listen_fd, (struct sockaddr*)&cli_addr, &cli_len);

    if((child_pid = fork()) == 0){
        close(listen_fd);           //child closes listen socket
        str_echo(connect_fd);       //child does his/her task
        close(connect_fd);         //child finished its task
        exit(EXIT_SUCCESS);
    }

    close(connect_fd);             //parent close connected socket
}
```

main 函式裡面的 while 迴圈，利用 fork() 建立子行程，子行程要關掉 listen_fd，因為他不需要去聽通道是否有新來的連線，str_echo() 用來處理連線建立後要做的事情。

如果 child 從 str_echo 返回之後，就立刻關閉 connect_fd。

```
void str_echo(int sockfd){
    char rcv_buf[MAXLINE];

    for(;;){
        memset(rcv_buf, 0, sizeof(rcv_buf));           //initail rcv_buf
        recv(sockfd, rcv_buf, sizeof(rcv_buf), 0);     //rcv the data from the other end

        printf("Hi client, I received %s", rcv_buf);
        //for(int i = 0; i < 128; i++)
        // printf("rcv[%d] = %c(ascii %d)\n", i, rcv_buf[i], (int)rcv_buf[i]);

        send(sockfd, rcv_buf, sizeof(rcv_buf), 0);     //send the data to the other end
        printf("OK, I send you back %s", rcv_buf);
    }
}
```

str_echo() 就是處理資料傳遞主要就是利用 send 跟 recv 來處理資料，

每次都會用 memset 清空 buffer，確保 buffer 內沒有雜亂的東西。

```
/*handle child process terminated*/
void sigchld_handler(int signo){
    pid_t pid;
    int stat;

    while((pid = waitpid(-1, &stat, WNOHANG)) > 0) //wait child process to prevent zombies
        printf("child %d terminated\n", pid);
    return;
}
```

收到 SIGCHLD 信號，利用 waitpid 來防止殭屍進程產生。

tcp_echo_client.c

```
//convert IPv4 address from text to binary
//take argv[1] in here ,argv[1]:address of server
check = inet_pton(AF_INET ,argv[1] ,&serv_addr.sin_addr.s_addr);
assert(check == 1); //using assert to check program
printf("connection success\n");

//check connection status
if(connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr)) < 0){
    perror("connect :");
    exit(EXIT_FAILURE);
}

str_cli(sockfd); //Client does what he/she wants.
exit(EXIT_SUCCESS);
```

main 裡面會利用輸入的 argv[1] 當作 server address 建立連線

每個 client 都呼叫 str_cli

```
void str_cli(int sockfd){

    char send_buf[MAXLINE];
    char recv_buf[MAXLINE];

    for(;;){
        memset(send_buf,0,sizeof(send_buf)); //initail send buffer to zero
        if(fgets(send_buf ,MAXLINE ,stdin) == NULL) //read from stdin
            break;
        send(sockfd ,send_buf ,sizeof(send_buf) ,0); //send the data to the other end

        //for(int i = 0 ; i < MAXLINE ;i++ )
        // printf("send[%d] = %c(ascii%d)\n",i,send_buf[i],(int)send_buf[i]);

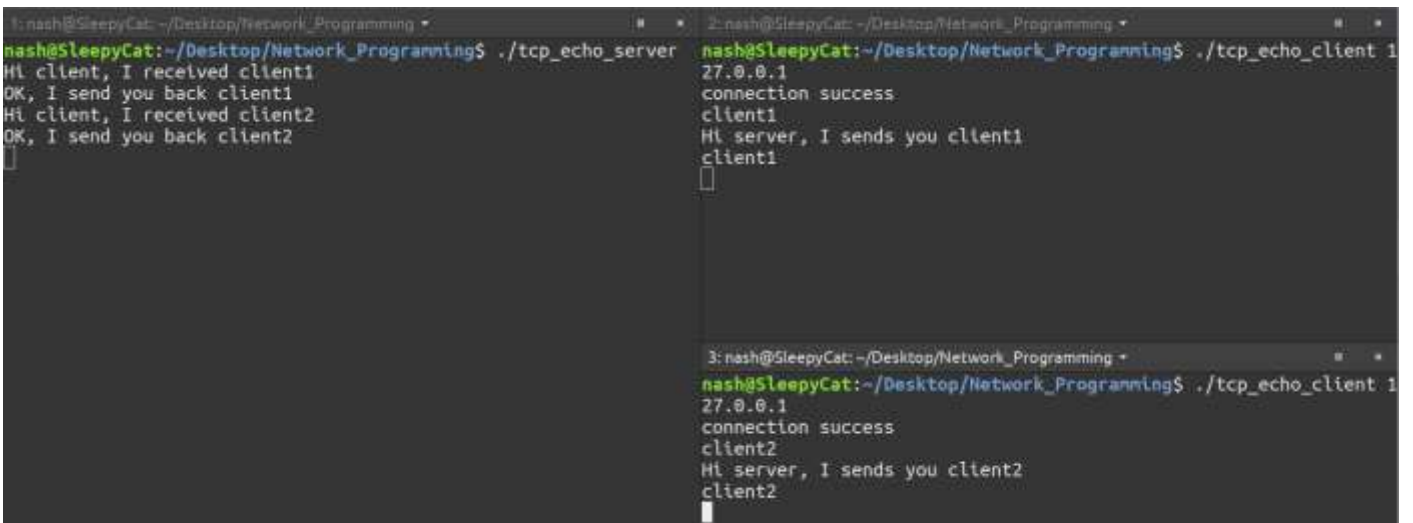
        printf("Hi server, I sends you %s",send_buf);

        memset(recv_buf ,0 ,sizeof(recv_buf)); //initail rcv buffer to zero
        recv(sockfd ,recv_buf ,sizeof(send_buf) ,0); //rcv the data from the other end
        fputs(recv_buf ,stdout); //print the data to stdout
    }
}
```

str_cli 副函式利用 fgets 從 stdin 讀入資料放入 send_buf 裡面，再利用 send 寄出去

最後用 fputs 將收到的資料印到 stdout。

執行結果



```
1: nash@SleepyCat: ~/Desktop/Network_Programming
nash@SleepyCat:~/Desktop/Network_Programming$ ./tcp_echo_server
Hi client, I received client1
OK, I send you back client1
Hi client, I received client2
OK, I send you back client2
[]

2: nash@SleepyCat: ~/Desktop/Network_Programming
nash@SleepyCat:~/Desktop/Network_Programming$ ./tcp_echo_client 1
27.0.0.1
connection success
client1
Hi server, I sends you client1
client1
[]

3: nash@SleepyCat: ~/Desktop/Network_Programming
nash@SleepyCat:~/Desktop/Network_Programming$ ./tcp_echo_client 1
27.0.0.1
connection success
client2
Hi server, I sends you client2
client2
[]
```

可以看見兩個 client 同時連進 server，並傳入 client1 和 client2 字樣

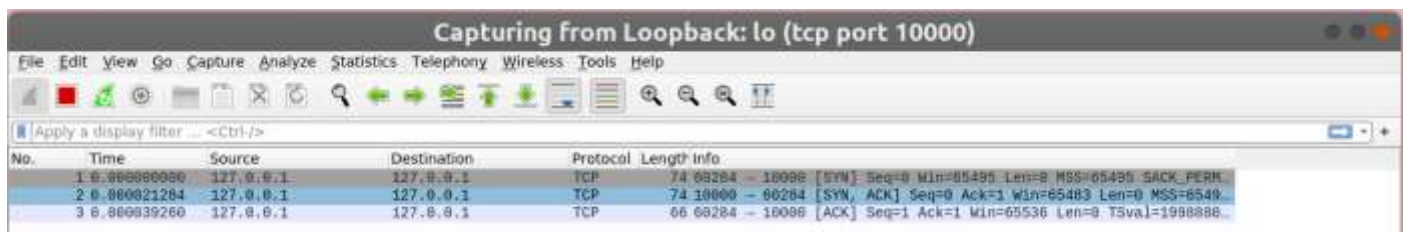
```
nash@SleepyCat:~/Desktop/Network_Programming$ ./tcp_echo_server
Hi client, I received client1
OK, I send you back client1
Hi client, I received client2
OK, I send you back client2
child 8212 terminated
child 8215 terminated

nash@SleepyCat:~/Desktop/Network_Programming$ ./tcp_echo_client 1
27.0.0.1
connection success
client1
Hi server, I sends you client1
client1
^C
nash@SleepyCat:~/Desktop/Network_Programming$

3: nash@SleepyCat: ~/Desktop/Network_Programming *
nash@SleepyCat:~/Desktop/Network_Programming$ ./tcp_echo_client 1
27.0.0.1
connection success
client2
Hi server, I sends you client2
client2
^C
nash@SleepyCat:~/Desktop/Network_Programming$
```

中止連線時可以看見 server 印出對應的 child process 被中止的字樣

Wireshark



抓 TCP 的三向交握，server 的 port 有指定為 10000，client 的 port 則沒有指定，所以是由系統自行選擇，在我的抓到的封包上，client port number 是 60284。