



HPDS

High Performance Distributed Systems Lab

# Linux Programming socket

報告者：彭皓廷

- ◆ 通訊端是通訊機制，它讓 client/server 系統可以在單一機台上或是在網路上進行開發。
- ◆ 首先。server 應用程式建立一通訊端，它是被指定到 server 程序的作業系統資源
- ◆ 接下來，server 程序給通訊端名稱，對於網路 socket 檔案名稱就是服務的辨識子，客戶端可以藉由它連結到伺服器。

- ◆ socket 的命名必須透過 bind 系統呼叫，伺服器就在這個具名的 socket 上等待客戶端的連結請求，listen 是用來產生一個佇列接應連結請求，而 accept 系統呼叫則讓伺服器接受連結。

# 產生 socket

- ◆ `int socket(int domain , int type , int protocol );`
- ◆ `AF_UNIX`
- ◆ `AF_INET`
- ◆ `AF_ISO`
- ◆ `type`: 指定 socket 類型，可用值為：
- ◆ `SOCK_STREAM`: 連續性可靠性連結性的雙向位元組串流。
- ◆ `SOCK_DGRAM`: 傳送固定大小的訊息，可靠度不佳。



# Socket 位址格式 AF\_UNIX, AF\_INET

◆ struct sockaddr\_un {  
    sa\_family\_t  sun\_family; //AF\_UNIX  
    char         sun\_path[ ]; //path name  
};

struct sockaddr\_in {  
    short int  
    unsigned short int  
    struct in\_addr  
    address  
};

sin\_family; //AF\_INET  
sin\_port; //port num  
sin\_addr; //internet

# 通訊端命名

- ◆ `#include <sys/socket.h>`
- ◆ `int bind(int socket, const struct sockaddr *address, size_t address_len);`
- ◆ `bind` 呼叫參數指定的位址 `address` 到一未命名的通訊端，此通訊端關連於 `socket` 檔案 descriptor，位址結構的長度以 `address_len` 傳遞。

# 建立通訊佇列

- ◆ `#include <sys/socket.h>`
- ◆ `int listen(int socket, int backlog);`
- ◆ 根據上限，限制佇列中延後的連結數目，`listen` 將佇列長度設定為 `backlog`，根據佇列長度限制安排通訊端上被暫緩的連結。

# 接受連結

- ◆ `int accept( int socket, struct sockaddr *addr, size_t socklen_t* len );`
- ◆ 當一個客戶端試圖連結 socket，accept 系統呼叫會回覆，而此客戶端是在佇列中第一個未解決的連結請求，accept 函數建立新的通訊端以便與 client 連接傳回 descriptor。



# 連結請求

- ◆ `#include <sys/socket.h>`
- ◆ `int connect(int socket, const struct sockaddr *address, size_t address_len);`
- ◆ 參數 `socket` 指定的通訊端連接到參數 `address` 指定的 `server` 通訊端，`address` 的長度為 `address_len`
- ◆ `EBADF` 一無效的檔案描述被傳給 `socket`
- ◆ `EALREADY` `socket` 中已有一連結執行中
- ◆ `ETIMEDOUT` 連結發生
- ◆ `ECONNREFUSED` 請求支連結被 `SERVER` 拒絕

# client

## ◆ 建立 client 的通訊端

```
sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
```

## ◆ 命名 server 也同意的通訊端

```
address.sun_family = AF_UNIX;  
strcpy(address.sun_path, "server_socket");  
len = sizeof(address);
```

## ◆ 將我們的通訊端連上 server 通訊端

```
result = connect(sockfd, (struct sockaddr *)&address, len);  
  
if(result == -1) {  
    perror("oops: client1");  
    exit(1);  
}
```

## ◆ 現在透過 sockfd 來讀寫

```
write(sockfd, &ch, 1);  
read(sockfd, &ch, 1);  
printf("char from server = %c\n", ch);  
close(sockfd);  
exit(0);
```

# server

- ◆ 移除舊的通訊端與為 server 建立一未具名的通訊端

```
unlink("server_socket");  
server_sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
```

- ◆ 為通訊端命名

```
server_address.sun_family = AF_UNIX;  
strcpy(server_address.sun_path, "server_socket");  
server_len = sizeof(server_address);  
bind(server_sockfd, (struct sockaddr *)&server_address,  
server_len);
```



## ◆ 建立一連結佇列與等待 client

```
listen(server_sockfd, 5);  
while(1) {  
    char ch;  
  
    printf("server waiting\n");
```

## ◆ 接受連結

```
client_len = sizeof(client_address);  
client_sockfd = accept(server_sockfd,  
    (struct sockaddr *)&client_address, &client_len);
```

## ◆ 讀取與寫入位在 client\_sockfd 上的 client

```
read(client_sockfd, &ch, 1);  
ch++;  
write(client_sockfd, &ch, 1);  
close(client_sockfd);
```

```
root@cuda04:~/Frank/ex/socket# ./server1 &  
[8] 8246  
root@cuda04:~/Frank/ex/socket# server waiting  
./client1  
server waiting  
char from server = B
```

# 網路 client

## ◆ 建立 client 的通訊端

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

## ◆ 命名通訊端並且與 server 相符

```
address.sin_family = AF_INET;  
address.sin_addr.s_addr = inet_addr("127.0.0.1");  
address.sin_port = 9734;  
len = sizeof(address);
```

# 網路 server

## ◆ 為 server 建立一未命名的通訊端

```
server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

## ◆ 命名通訊端

```
server_address.sin_family = AF_INET;  
server_address.sin_addr.s_addr = inet_addr("127.0.0.1");  
server_address.sin_port = 9734;  
server_len = sizeof(server_address);  
bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
```



```
root@cuda04:~/Frank/ex/socket# ./server2 &  
[10] 8309  
root@cuda04:~/Frank/ex/socket# server waiting  
./client2  
server waiting  
char from server = B
```

# 網路資訊

- ◆ `#include <netdb.h>`
- ◆ `struct hostent *gethostbyaddr(const void *addr, size_t len, int type);`
- ◆ `struct hostent *gethostbyname(const char *name);`
- ◆ `struct hostnet{`
- ◆ `char *h_name;       //name of the host`
- ◆ `char **h_aliases;   //list of aliases`
- ◆ `int h_addrtype;    //address type`
- ◆ `int h_length;     // length in bytes of the address`
- ◆ `char **h_addr_list   //list of address`
- ◆ `};`

- ◆ `struct servent *getservbyname(const char *name, const char *proto);`
- ◆ `struct servent *getservbyport(int port, const char *proto);`
- ◆ `struct servent{`
- ◆ `char *s_name; //name of the service`
- ◆ `char **s_aliases; //list of aliases`
- ◆ `int s_port; // the ip port number`
- ◆ `char *s_proto //the service type , tcp or udp`
- ◆ `};`

# 取得主機的電腦資訊

```
int main(int argc, char *argv[])
{
    char *host, **names, **addrs;
    struct hostent *hostinfo;
```

- ◆ 利用 gethostname 取得 host 名稱，或直接根據使用者傳入的參數

```
if (argc == 1) {
    char myname[256];
    gethostname(myname, 255);
    host = myname;
}
else
    host = argv[1];
```



## ◆ 呼叫 gethostname ，若沒有發現資訊回傳錯誤

```
hostinfo = gethostbyname(host);  
if(!hostinfo) {  
    fprintf(stderr, "cannot get info for host: %s\n", host);  
    exit(1);  
}
```

## ◆ 顯示主機名與其他別名 (alias)

```
printf("results for host %s:\n", host);  
printf("Name: %s\n", hostinfo -> h_name);  
printf("Aliases:");  
names = hostinfo -> h_aliases;  
while(*names) {  
    printf(" %s", *names);  
    names++;  
}  
printf("\n");
```

## ◆ 若有問題的主機非一 IP 主機則離開

```
if(hostinfo -> h_addrtype != AF_INET) {  
    fprintf(stderr, "not an IP host!\n");  
    exit(1);  
}
```

## ◆ 否則顯示 IP 位址

```
addr = hostinfo -> h_addr_list;  
while(*addr) {  
    printf(" %s", inet_ntoa(*(struct in_addr *)*addr));  
    addr++;  
}  
printf("\n");  
exit(0);
```

```
root@cuda04:~/Frank/ex/socket# ./getname cuda04
results for host cuda04:
Name: cuda04
Aliases:
  192.168.1.204
```

# 多重客戶端

- ◆ 呼叫 fork 產生第二個處理程序，socket 被子處理程序繼承，主伺服器繼續接聽其他客戶端的連結請求，子處理程序負責與客戶通訊



```
int server_sockfd, client_sockfd;
int server_len, client_len;
struct sockaddr_in server_address;
struct sockaddr_in client_address;

server_sockfd = socket(AF_INET, SOCK_STREAM, 0);

server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons(9734);
server_len = sizeof(server_address);
bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
```

- ◆ 產生一個連結佇列，忽略子處理程序的離開信號，等待客戶端

```
listen(server_sockfd, 5);
signal(SIGCHLD, SIG_IGN);
while(1) {
    char ch;

    printf("server waiting\n");
```

## ◆ 接受連結

```
client_len = sizeof(client_address);  
client_sockfd = accept(server_sockfd,  
    (struct sockaddr *)&client_address, &client_len);
```

## ◆ 利用 fork 產生一個處理程序，並檢查目前的是父處理程序還是子處理程序

```
if(fork() == 0) {
```

- ◆ 如果是子處理程序，利用 `client_sockfd` 讀寫客戶端。途中五秒的延遲只是為了展示目的

```
read(client_sockfd, &ch, 1);  
sleep(5);  
ch++;  
write(client_sockfd, &ch, 1);  
close(client_sockfd);  
exit(0);
```

- ◆ 如果父處理程序，對客戶端工作已經完成

```
else {  
    close(client_sockfd);  
}
```

```
root@cuda04:~/Frank/ex/socket# ./server4 &
[17] 9855
root@cuda04:~/Frank/ex/socket# server waiting
./client3
server waiting
char from server = B
root@cuda04:~/Frank/ex/socket# vim server4.c
root@cuda04:~/Frank/ex/socket# ./server4 &
[18] 9930
root@cuda04:~/Frank/ex/socket# server waiting
./client3 & ./client3 & ./client3 &
[19] 9931
[20] 9932
[21] 9933
root@cuda04:~/Frank/ex/socket# server waiting
server waiting
server waiting
char from server = B
char from server = B
char from server = B
```

# slect

- ◆ 讓一個程式可以同時針對很多低階的檔案描述子，等待輸入到達（或輸出完成）這代表終端模擬程式可以被擱置，直到有事情發生。
- ◆ select 函數在 fd\_set 資料結構上運作
- ◆ #include<sys/types.h>
- ◆ #include<sys/time.h>
- ◆ void FD\_ZERO(fd\_set \*fdset);
- ◆ void FD\_CLR(int fd, fd\_set \*fdset);
- ◆ void FD\_SET(int fd, fd\_set \*fdset);
- ◆ int FD\_ISSET(int fd, fd\_set \*fdset);



- ◆ FD\_ZERO 初始 fd\_set ，將它設為空集合。
- ◆ FD\_CLR ， FD\_SET 設定和清除集合中的 fd 檔案描述子。
- ◆ 如果 fd 是 fdset 中的一個項目 FD\_ISSET 就會回傳一個非零值。
- ◆ FD\_SETSIZE 定義 fd\_set 結構中，可以容納的檔案描述子數量。

- ◆ select 函數也可以定義一個逾時時間值，來防止無限制地被擱置。
- ◆ struct timeval{
- ◆ Time\_t tv\_sec; // seconds
- ◆ long tv\_usec; // microseconds
- ◆ }

- ◆ `int select (int nfds ,fd_set *readfds,fd_set *writefds,fd_set *errorfds, struct timeval * timeout);`
- ◆ `nfds` 代表要檢驗的檔案描述子數量，`readfds`，`writefds`，`errorfds`，可以為空指標
- ◆ `readfds` ：用於檢查可讀性
- ◆ `writefds` ：用於檢查可寫性
- ◆ `errorfds` ：檢查是否有錯誤
- ◆ `timeout` 時間過後沒有情況發生 `select` 會回覆
- ◆ 如果 `timeout` 參數是空指標也沒情況發生，函數將被擱置

# 實作 select，讀取鍵盤，逾時 2.5 秒

## ◆ 宣告，初始 input 管理鍵盤輸入

```
int main()
{
    char buffer[128];
    int result, nread;

    fd_set inputs, testfds;
    struct timeval timeout;

    FD_ZERO(&inputs);
    FD_SET(0, &inputs);
```

## ◆ 在 stdin 上等待輸入

```
while(1) {
    testfds = inputs;
    timeout.tv_sec = 2;
    timeout.tv_usec = 500000;

    result = select(FD_SETSIZE, &testfds, (fd_set *)0, (fd_set *)0, &timeout);
```

- ◆ 檢驗 result，如果沒有輸入，程式會繼續迴圈，如果有錯誤程式會離開

```
while(1) {  
    testfds = inputs;  
    timeout.tv_sec = 2;  
    timeout.tv_usec = 500000;  
  
    result = select(FD_SETSIZE, &testfds, (fd_set *)0, (fd_set *)0, &timeout);
```



- ◆ 程式等待的過程中，如果有動作，就會從 stdin 讀取輸入，並印出，直到輸入，ctrl+d

```
default:
    if(FD_ISSET(0, &testfds)) {
        ioctl(0, FIONREAD, &nread);
        if(nread == 0) {
            printf("keyboard done\n");
            exit(0);
        }
        nread = read(0, buffer, nread);
        buffer[nread] = 0;
        printf("read %d from keyboard: %s", nread, buffer);
    }
    break;
}
```

```
root@cuda04:~/Frank/ex/socket# ./select
timeout
qwtimeout

read 4 from keyboard: qwe
timeout
qw
read 3 from keyboard: qw
timeout
qwe
read 4 from keyboard: qwe
```



# Thanks for your listening !!

