**SOCKET và LẬP TRÌNH SOCKET**

* Khái niệm socket

     Socket tiếng Anh là cái ổ cắm . Nhưng trong tin học khi người ta nói đến giao tiếp socket thì nó lại có ý nghĩa là một phương pháp dùng để giao tiếp giữa các tiến trình khác nhau trong mạng máy tính hay là trên một máy đơn . Socket là một trong những phương thức giao tiếp giữa các quá trình (inter-process communication) như pipe , message queue .... Trong các ứng dụng được viết ngày nay người ta sử dụng công cụ socket này rất nhiều như các Web , FTP .... Trong phần này chúng ta sẽ xem xét vấn đề trên môi trường Windows .  
     Bây giờ chúng ta sẽ nói rõ hơn về socket . Giả sử ta muốn thiết kế một hệ thống trao đổi file . Hệ thống này hoạt động như sau : Tất cả dữ liệu được lưu trữ trên một máy chủ mạnh để bảo đảm an toàn và dễ quản lý . Nhưng những người làm IT thường hay di chuyển nên ít khi nào họ ở gần server đó , với lại họ cũng bị hạn chế lại gần server và có mức độ giới hạn truy cập . Thế thì làm sao mà truy cập khi không ở ngay tại máy server đó . Giải pháp đưa ra là ta sẽ nối cái máy chủ đó ra mạng (LAN , WAN hay internet ) và mọi người truy cập thông qua mạng . Như vậy ta có thể hình dung mô hình truy cập sau : máy chủ có chương trình chạy trên đó tiếp nhận các yêu cầu (server) . Những người nào muốn lấy dữ liệu (client) thì cứ kết nối với máy chủ đó thông qua môi trường mạng để lấy dữ liệu về trên máy của mình . Đây chính là mô hình client / server .  
     Giao tiếp socket giúp chúng ta hiện thực mô hình client / server dễ dàng hơn . VD A muốn tới nhà B chơi thì trước hết A phải mở cửa để bước ra ngoài đường .Sau đó thì A có thể đón taxi mà đi không cần quan tâm đến việc phải đi như thế nào để đến nhà B , Khi tới nhà B thì A phải bấm chuông để B ra mở cửa . Quá trình kết nối giữa 2 ứng dụng trên mạng cũng giống như vậy , client mở một socket vô danh ( khái niệm trừu tượng để chỉ mở "cái cửa" ) , sau đó nó sẽ nhờ các lớp mạng tạo ra kết nối tới server (không cần quan tâm đến nó kết nối như thế nào cả ) . Song song đó thì server cũng tạo một socket khác có đia chỉ hẳn hoi ( một cái cửa thứ hai , nhưng có chuông , số nhà .... ) Khi có kết nối tới đúng server dựa vào thông tin trên socket của server ( giống như client tìm đúng số nhà , bấm chuông ...) thì server sẽ mở cổng ra và client / server sẽ bắt đầu quá trình truyền dữ liệu .  
     Bây gìơ ta sẽ nói đến địa chỉ IP và port . IP là địa chỉ trên mạng của thiết bị như PC , router .... . Port là một khái niệm ảo được xây dựng trên lớp Application trong mô hình OSI để phân biệt một gói tin được nhận thì của process nào . VD như gói tin A ( khi lên tới lớp application )là cho ứng dụng FTP thì hệ điều hành lại cho nó vào ứng dụng web Vì vậy ở lớp Application cần có sự phân biệt gói tin nào là của ứng dụng nào . Port chính là thông tin để biết được gói tin nào là cuả ứng dụng nào . Khi ta nói Web server chạy ở port 80 tức là khi các gói tin được nhận thì hệ điều hành sẽ lấy các gói tin gửi tới port 80 truyền cho Web server .

* **Tìm hiểu các thuộc tính của socket**

Socket có 3 thuộc tính sau :  
**+ Domain**: vùng giao tiếp của socket . Nếu socket hoạt động trên môi trường internet thì nó sẽ có giá trị là AF\_INET (tức là hoạt động dựa theo chuẩn internet và dùng IP để xác định địa chỉ vật lý . ) Nếu socket hoạt động trên môi trường mạng cục bộ thì ta có thể dùng giao tiếp theo chuẩn : AF\_UNIX (dùng đường dẫn và hệ thống file để đặt tên và xác định kết nối giữa hai hay nhiều ứng dụng . Trong môi trường Linux thì các thiết bị cũng được coi như hệ thống file đặc biệt và socket cũng được coi như vậy )  
  
+ **Kiểu socket type** :  
SOCK\_STREAM : tạo ra kết nối bền vững (kết nối dạng TCP ) . Nói nôm na thì kết nối này đảm bảo gói tin này đến đích an toàn và không bị mất mát , nhưng nó có nhược điểm là chậm và tốn nhiều tài nguyên ,  
SOCK\_DGRAM : cách gửi nhận dữ liệu một chiều (kết nối UDP) , client gửi dữ liệu đi mà không cần biết là server có nhân đựơc dữ liệu hay không . Server khi nhận được dữ liệu cũng không cần phải xác nhận với client là đã nhận dữ liệu . Tuy không an toàn nhưng nó nhanh và được sử dụng nhiều trong môi trường multimeida  
(còn một loại socket nữa là raw -socket nhưng ít dùng )  
+**Giao thức của socket (protocol ):**  
Giao thức là cách quy ước gửi nhận dữ liệu trong quá trình trao đổi thông tin (VD ai gửi trước , ai nhận , gửi nhận theo thứ tự nào ....) Có nhiều giao thức đối với mỗi loại socket , nhưng dùng nhiều là TCP và UDP .  
  
**MỘT SỐ HÀM THÔNG DỤNG CỦA SOCKET**  
**+ Hàm tạo socket**:  
SOCKET socket( int af, int type, int protocol);  
   af : domain . Có thể lấy các giá trị sau : AF\_INET , AF\_UNIX , AF\_IPX , AF\_ISO , AF\_NS ...  
type : SOCK\_STREAM , SOCK\_DGRAM  
protocol : lấy giá trị 0 để sử protocol mặc định cho mỗi loại socket  
Giá trị trả về là sokcet discriptor , là một con số dùng để mô tả socket mình vừa tạo ra .  
  
**+ Hàm đóng socket** :  
closesocket(SOCKET sk ) ;  
  
**+ Hàm bind:**  
int bind(SOCKET s, const struct sockaddr\* name, int namelen);  
    Trước khi gọi hàm bind bạn phải tạo một socket trước bằng hàm socket() . Sau khi tạo socket là tới lúc mình sẽ dùng hàm bind gán cho nó các thông tin như port và địa chỉ  
Thông tin này đựơc lưu trữ trong struct sockaddr\* name  
  
**+ Hàm listen:**  
int listen( SOCKET s, int backlog);  
   Hàm này dùng để tạo hàng đợi các kết nối . backlog là số kết nối tối đa trong hàng đợi . Do một server thì một lúc có thể có nhiều kết nối tới do đó cần phải có một hàng đợi để lưu trữ các kết nối này . Socket s sẽ đóng vai trò lắng nghe các kết nối và đưa nó vào hàng đợi  
  
**+ Hàm accept:**  
SOCKET accept( SOCKET s, struct sockaddr\* addr, int\* addrlen);  
Cách hoạt động của hàm accept là như sau:  
   Khi ta gọi hàm accept nó sẽ đưa thread mà hàm accept đang hoạt động vào trạng thái blocking . Khi có một kết nối ở trong hàng đợi thì hàm accept sẽ kết thúc và thread sẽ thoát khỏi trạng thái blocking . Khi này hàm accept đã tạo ra một socket mới ( thông tin về socket này chúng ta không thể biết ) và đặt các địa chỉ của client vào struct sockaddr\* addr . Socket mới này chính là kết quả trả về của hàm accept . Ta sẽ dùng socket mới này để liên lạc với client chứ không dùng socket cũ là s .  
  
**+ Hàm select:**  
int select( int nfds, fd\_set\* readfds, fd\_set\* writefds, fd\_set\* exceptfds, const struct timeval\* timeout);  
    Đây là một hàm quan trọng . Hàm này dùng để thay đổi trạng thái của socket từ blocking sang non-blocking và ngược lại . Bình thường thì các socket được tạo ra từ các hàm trên sẽ tạo ra một socket dạng blocking , tức là nó sẽ đưa thread vào trạng thái blocking khi chưa có dữ liệu hoặc chưa có kết nối . Ta có thể dùng hàm select để thay đổi các trạng thái đó ,  
Hàm select sẽ trả về nếu như có một socket nào đó trong danh sách các socket readfds và writefds phát sinh tín hiệu là sẵn sàng hoạt động. Nếu như chưa có tín hiệu nào thì hàm listen sẽ đưa thread vào trạng thái ngủ ( hổng có gì làm thì ngủ là khỏe nhất ) Ta có thể lợi dụng điều này để có thể viết chương trình mà không cần phải tạo nhiều thread để quản lý từng kết nối

* Các macro phụ trợ cho hàm select là:

FD\_CLR(s, \*set)  
Removes the descriptor s from set.  
FD\_ISSET(s, \*set)  
Nonzero if s is a member of the set. Otherwise, zero.  
FD\_SET(s, \*set)  
Adds descriptor s to set.  
FD\_ZERO(\*set)  
Initializes the set to the null set.  
  
\*set ứng với các biến thuộc kiểu fd\_set\*  
  
typedef struct  
fd\_set {  
u\_int fd\_count;  
SOCKET fd\_array[FD\_SETSIZE];  
} fd\_set;  
  
chúng ta có thể xem xét nó kĩ hơn trong phần VD của nó  
  
**+ Các hàm send và recv:**  
int send( SOCKET s, const char\* buf, int len, int flags);  
int recv( SOCKET s, char\* buf, int len, int flags);  
  
  Đây là các hàm dùng để trao đổi thông tin giữa client và server (dạng kết nối TCP )  
int recvfrom( SOCKET s, char\* buf, int len, int flags, struct sockaddr\* from, int\* fromlen);  
int sendto( SOCKET s, const char\* buf, int len, int flags, const struct sockaddr\* to, int tolen);  
Đây là các hàm dùng để trao đổi thông tin giữa client và server (dạng kết nối UDP )  
  
Các bạn tham khảo thêm trong MSDN . Mình dùng quote để giữ định dạng .  
  
**Phần Server**  
int main()  
{  
SOCKET sk = socket(AF\_INET,SOCK\_STREAM,0) ;  
if ( sk == SOCKET\_ERROR )  
{  
MessageBox("Không thể tạo socket") ;  
}  
  
int server\_port = 1000 ; //port của server  
char server\_ip[] = "127.0.0.1" ; //địa chỉ IP dạng chuỗi  
  
////Tạo thông tin cho socket  
  
SOCKADDR\_IN addr , client\_add;  
addr.sin\_family = AF\_INET ;  
addr.sin\_port = htons(server\_port) ;  
addr.sin\_addr.s\_addr = inet\_addr(server\_ip);  
if(bind(sk,(LPSOCKADDR)&addr,sizeof(addr))==SOCKET\_ERROR)  
MessageBox("Không thể bind socket") ;  
if(listen(sk,SOMAXCONN)==SOCKET\_ERROR)  
MessageBox("Server không thể lắng nghe kết nối " );  
  
SOCKET newsk ;  
while(true)  
{  
newsk = accept(sk,(struct sockaddr\*)&client\_add,&len);  
if(newsk== INVALID\_SOCKET) MessageBox("Không nhận biết được kết nối");  
else {  
DWORD threadid ;  
HANDLE hthread ;  
hthread = CreateThread(NULL,0,Xu\_Ly\_Ket\_Noi,&newsk ,0,&threadid) ;  
if(hthread== NULL) MessageBox("Lỗi trong việc tạo thread");  
}

}  
  
DWORD WINAPI Xu\_Ly\_Ket\_Noi(LPVOID lpParam)  
{  
//////////Xử lý các dữ liệu truyền qua lại giữa server và client thông qua kết nối này  
.............................  
////VD : Server nhận từ client chuỗi dạng chữ thường thì sẽ reply chuỗi dạng chữ hoa  
/// char buf[80 ] ;  
/// SOCKET \* sk = (SOCKET\*)lpParam ;  
/// int byte\_read = recv(\*sk , buf,sizeof(buf),0) ;  
/// if ( byte\_read == SOCKET\_ERROR)  
/// {  
/// MessageBox("Không nhận được dữ liệu từ client ") ;  
/// }  
/// sprintf(buf , UPPERSTRING(buf) ) ;  
/// send(\*sk, buf , strlen(buf) ,0) ;  
return 0 ;  
}  
  
**Phần client**  
int main()  
{  
char server\_ip[] = "127.0.0.1" ;  
SOCKADD\_IN server\_add ;  
server\_add.sin\_family = AF\_INET ;  
server\_add.sin\_port = htons(2000) ;  
server\_add.sin\_addr.s\_add = inet\_add(server\_ip) ;  
  
SOCKET Client\_sk = socket(AF\_INET, SOCK\_STREAM , 0) ;  
if ( connect(Client\_sk , (LPSOCKADDR)&server\_add,sizeof(server\_add)) ==SOCKET\_ERROR)  
{  
MessageBox("Không thể kết nối tới server") ;  
}  
  
////Đoạn code tiếp theo là phần gửi và nhận dữ liệu , xử lý thông tin ;  
////VD : client gửi thông điệp chữ thường --> server sẽ chuyển thành chữ hoa  
/// char buf[80] ;  
/// sprintf(buf , "hello word") ;  
/// if( send( Client\_sk , buf , strlen(buf) ,0) == SOCKET\_ERROR)  
/// {  
/// Message("Không thể gửi dữ liệu") ;  
//// return ;  
///}  
/// int byte\_read = recv(Client\_sk , buf , sizeof(buf) , 0 ) ;  
/// if( byte\_read == SOCKET\_ERROR)  
/// {  
//// MessageBox("Không thể nhận dữ liệu  
//// return ;  
////}  
//// sprintf(buf , "Chuỗi nhận được từ client là %s " , buf) ;  
//// MessageBox( buf ) ;  
}  
  
**Giải thuật cho chương trình client  
Xác định địa chỉ server  
Tạo socket  
Kết nối đến server  
Gởi / nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế  
Đóng kết nối**  
  
  
**- Giải thuật cho chương trình server**  
  
+Chương trình server có hai loại đơn giản : lặp và đồng thời  
+Hai dạng giao thức mà chương trình server có thể sử dụng là connection-oriented (TCP ) và connectionless (UDP)  
  
Giải thuật cho dạng lặp  
Đặc điểm : tại một thời điểm chỉ xử lý một kết nối từ server  
+ Tạo socket , đăng kí địa chỉ socket với hệ thống  
+ Đặt socket ở trạng thái lắng nghe , chờ và sẵn sàng cho việc kết nối từ client  
+ Chấp nhận kết nối từ client , gới nhận dữ liệu theo giao thức lớp ứng dụng đã thiết kế  
+ Đóng kết nối sau khi hoàn thành , trở lại trạng thái lắng nghe và chờ kết nối mới  
  
Giải thuật cho chương trình server dạng đồng thời ( concurrent) Đặc điểm : tại một thời điểm có thể xử lý nhièu kết nối từ client  
Tạo socket , đăng kí nó với hệ thống ( hàm bind)  
Đặt socket ở chế độ chờ , lắng nghe kết nối  
Khi có request từ client , chấp nhận kết nối , tạo một process con để xử lý . quay lại trạng thái chờ lắng nghe kết nối mới .  
Công việc của process mới gồm  
+ Nhận thông tin kết nối của client  
+ Giao tiếp với client theo giao thức lớp ứng dụng đã thiết kế  
+ Đóng kết nối và kết thúc process con  
  
Thiết kế chương trình server dạng multi-protocol server ( gồm cả TCP và UDP )  
Dùng một chương trình , mở một master socket cho cả TCP và UDP  
Dùng một hàm hệ thống (select để chọn lựa TCP hay UDP socket )  
Tùy vào protocol mà xử lý gởi nhận thông điệp giao thức của lớp ứng dụng  
Tham khảo RFC 1060  
  
Thiết kế chương trình server dạng multi-service server ( gồm cả TCP và UDP )  
Tạo một điểm giao tiếp chung  
Với mỗi request , xem loại dịch vụ cần xử lý  
Với mỗi loại dịch vụ , xử lý riêng biệt  
Có thể kết hợp multi-protocol và multi-service cho chương trình server