Socket Programming(1/2)

2020/03/23

Outline

- 1. Introduction to Network Programming
- 2. Network Architecture – Client/Server Model
  - getaddrinfo (demo)
- 3. TCP Socket Programming
  - TCP client/server (demo)
- 4. UDP Socket Programming
  - UDP client/server (demo)
- 5. IPv4/IPv6 Programming Migration
  - IPv6 TCP client/server (demo)

Introduction to Network Programming

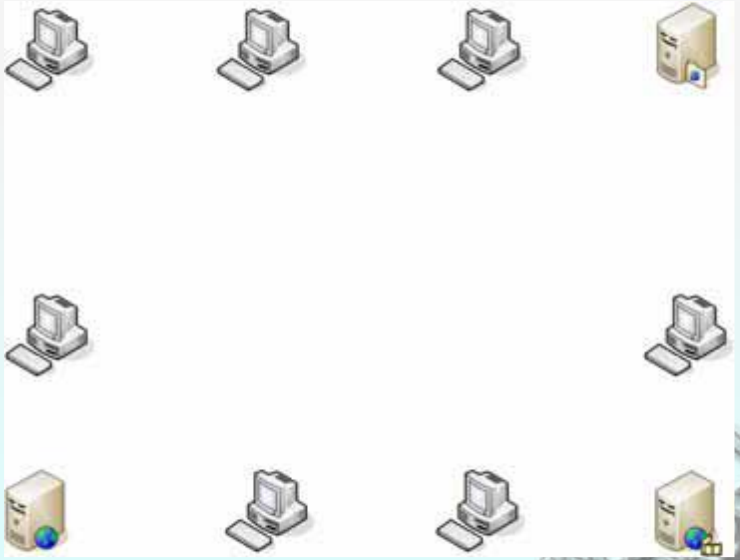
Introduction to Network Programming

- 1-1. What is Computer Networks?
- 1-2. How to use Computer Network to exchange information?
- 1-3. How to build network applications?

1-1. What is Computer Networks?

What is Computer Networks?

- Communicating Hosts and Network Equipments
- Communicating Links
- Communicating Protocols



1-2. How to use Computer Network to exchange information?

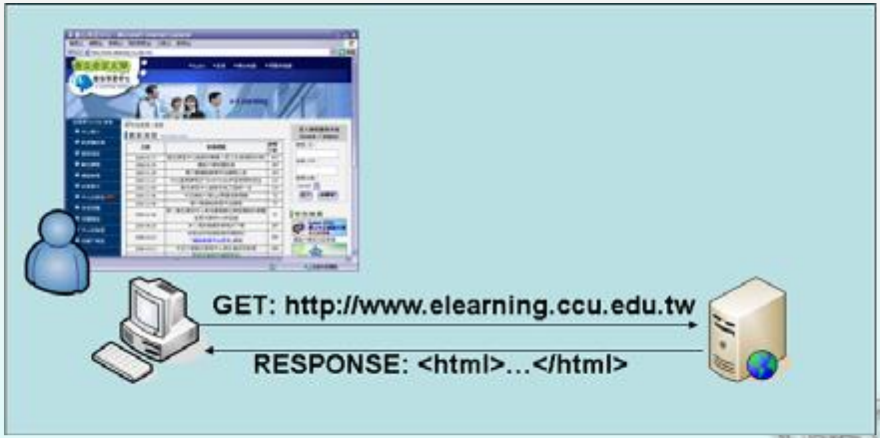
How to use Computer Network to exchange information?

Network Applications

- Protocol Instance

Examples

- World Wide Web (HTTP)
- File Transferring (FTP)
- E-Mail (SMTP)
- VoIP (SIP, H323)


GET: http://www.elearning.ccu.edu.tw
RESPONSE: <html>...</html>

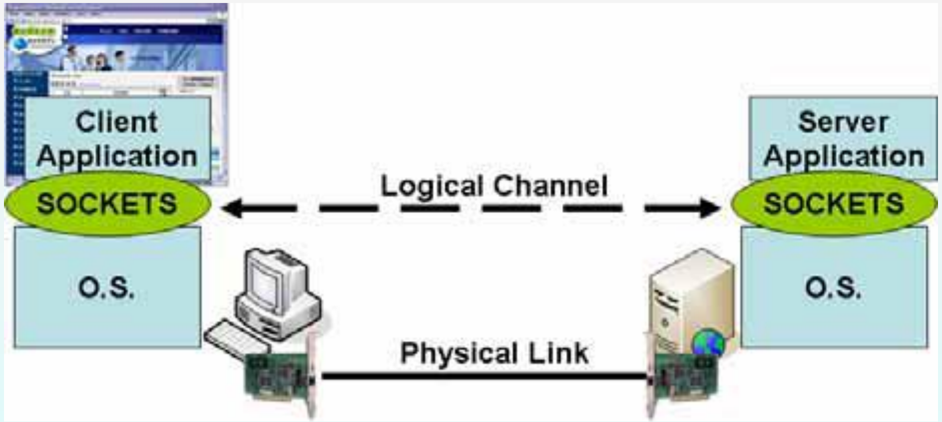
1-3. How to build network applications?

How to build network applications?

- SOCKETS - Network Programming Libraries (Interfaces)

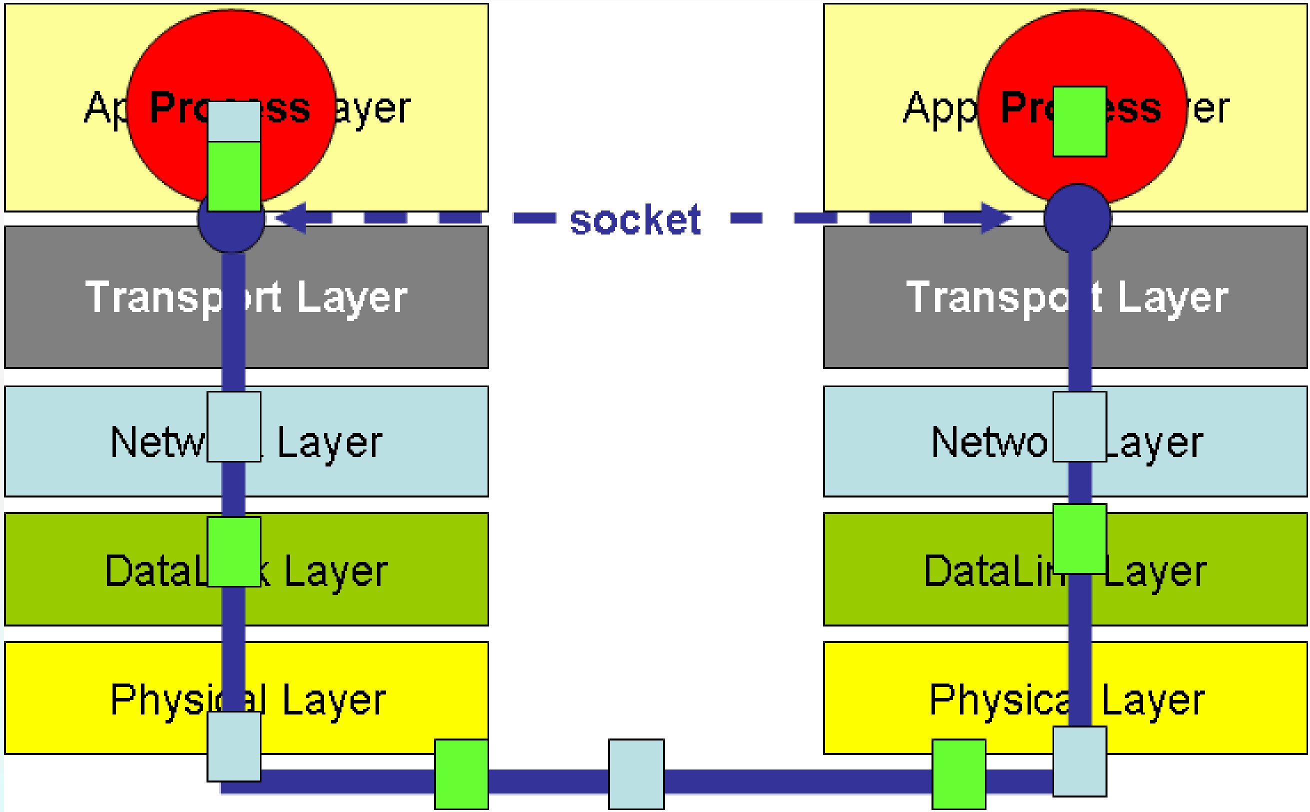
Examples:

- Linux – BSD Sockets
- Windows – WinSock
- JAVA, …



2. Socket Overview

Socket Overview



Network Programming Interfaces - Sockets

- Socket is an interface between application layer and transport layer in programming view
- Network applications can use socket libraries to build a networked communicating channel and transmit information

Socket Data Structure

- Socket Addressing Information

Internet Protocol Address – Network Layer

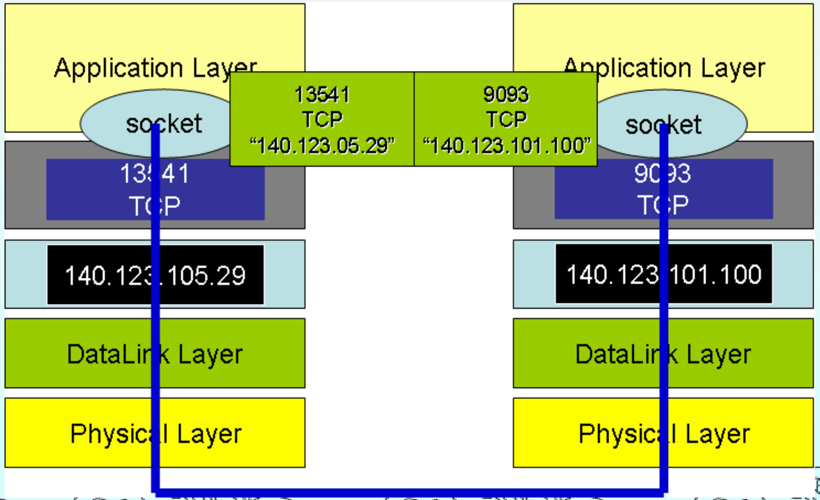
- “140.123.101.100” / “140.123.105.25”

Transport Service Type – Transport Layer

- TCP / UDP/ SCTP

Transport Service Port – Transport Layer

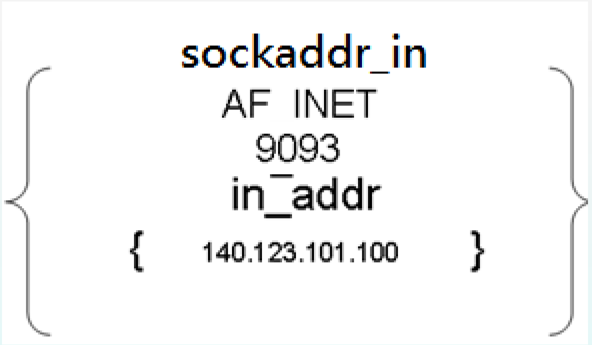
- 80 (http) / 21 (ftp) / 9093 (user defined)



Socket Data Structure

Two important Data Structures (in Ipv4)

- Internet Address Structure : struct in_addr
- Socket Address Structure : struct sockaddr_in



Internet Address Structure

IPv4 Address Length : 32 bits (4 bytes)

“140.123.101.100” – Dotted-Decimal String

Network Byte Ordered IPv4 Internet Address Structure

```
struct in_addr {
            in_addr_t    s_addr ;
    }
```

in_addr_t : uint32_t (Unsigned 32-bit integer)

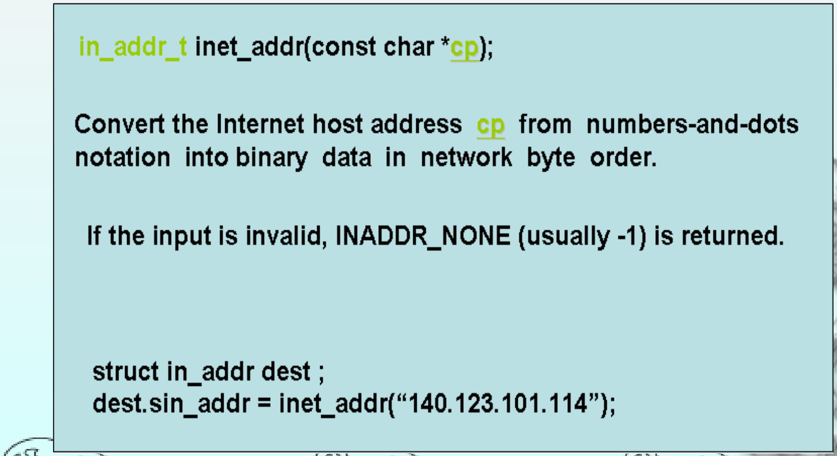
Conversion of Network Byte Address and Address String

Byte Manipulation Functions

int inet_aton()

in_addr_t inet_addr()

char *inet_ntoa()



```
in_addr_t inet_addr(const char *cp);
```

Convert the Internet host address cp from numbers-and-dots notation into binary data in network byte order.

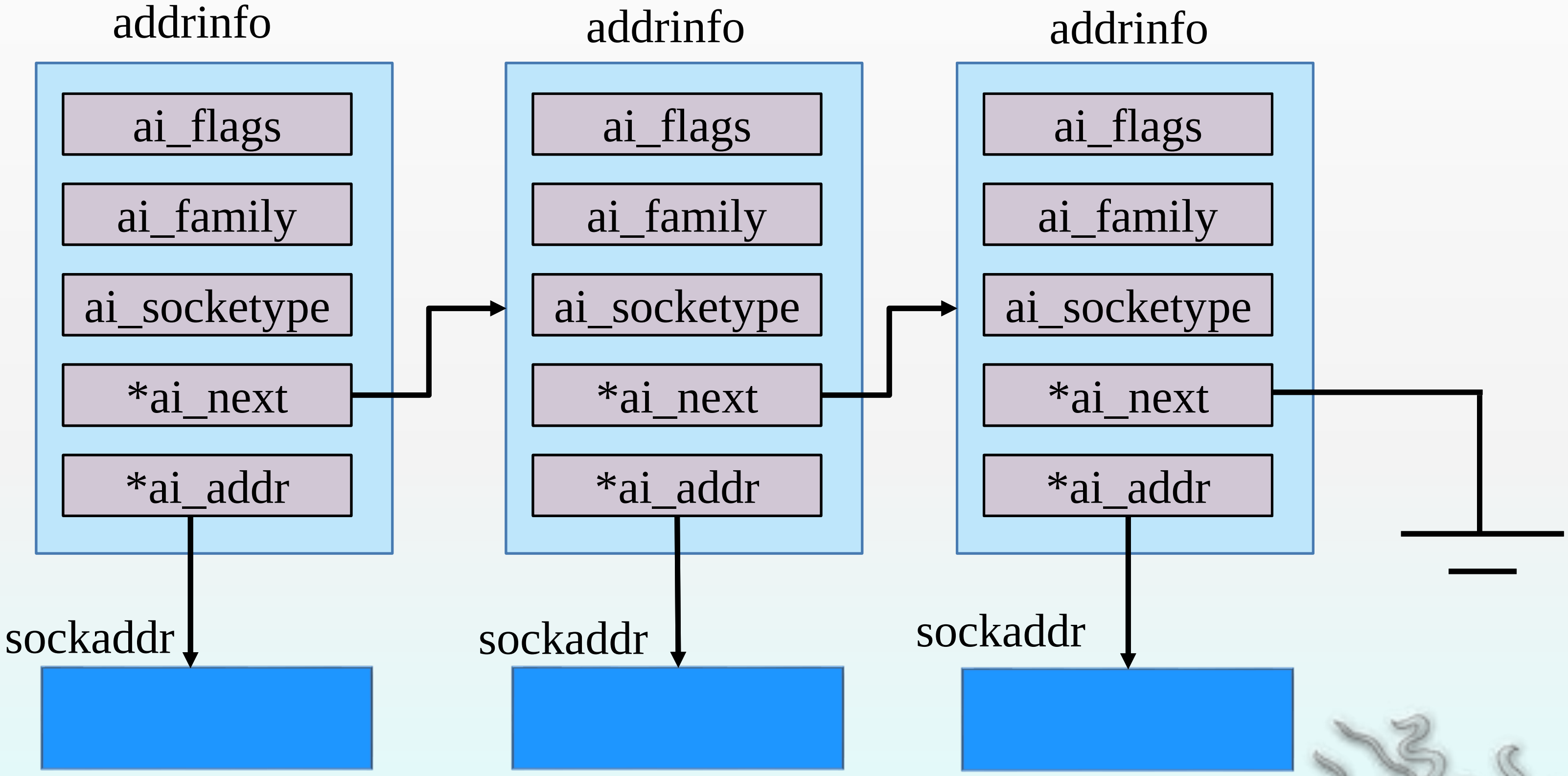
If the input is invalid, INADDR_NONE (usually -1) is returned.

```
struct in_addr dest ;
dest.sin_addr = inet_addr(“140.123.101.114”);
```

getaddrinfo
(network address and service)

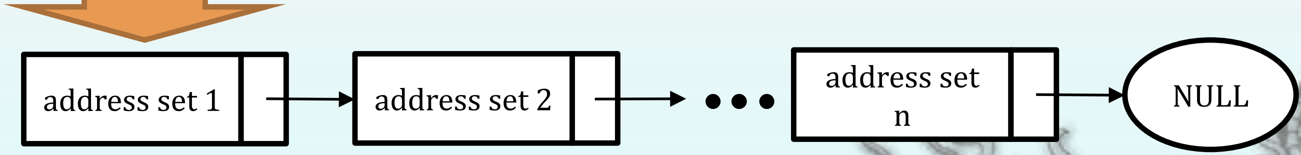
struct addrinfo

```
struct addrinfo {
   int     ai_flags;
   int     ai_family;
   int     ai_socktype;
   int     ai_protocol;
   size_t  ai_addrlen;
   struct  sockaddr *ai_addr;
   char    *ai_canonname;     /* canonical name */
   struct  addrinfo *ai_next; /* this struct can form a linked list */
};
```


addrinfo
ai_flags
ai_family
ai_socketype
*ai_next
*ai_addr
sockaddr
addrinfo
ai_flags
ai_family
ai_socketype
*ai_next
*ai_addr
sockaddr
addrinfo
ai_flags
ai_family
ai_socketype
*ai_next
*ai_addr
sockaddr

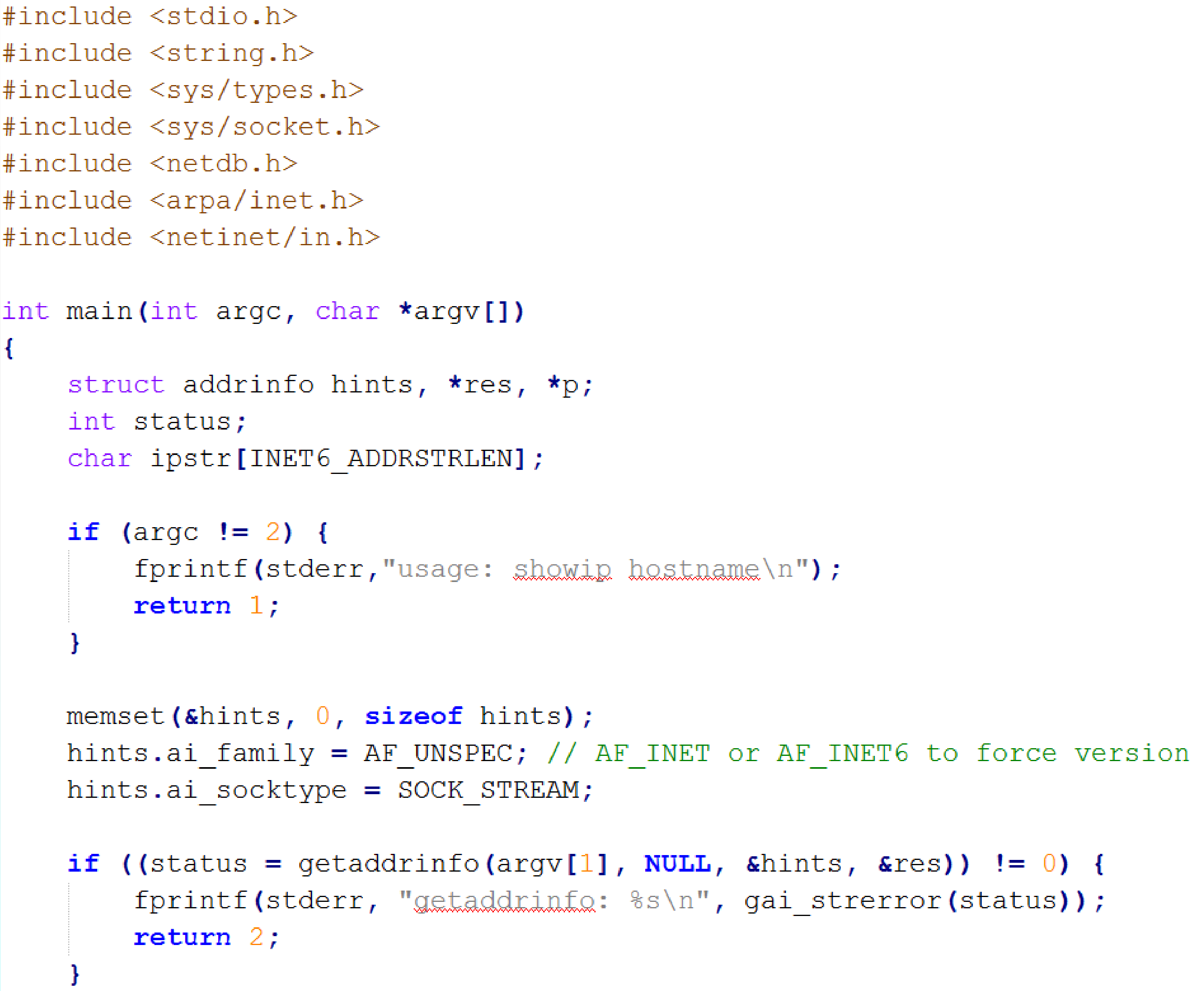
getaddrinfo()

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(  const char *hostname,
            const char *service,
            const struct addrinfo *hints,
            struct addrinfo **res);
```

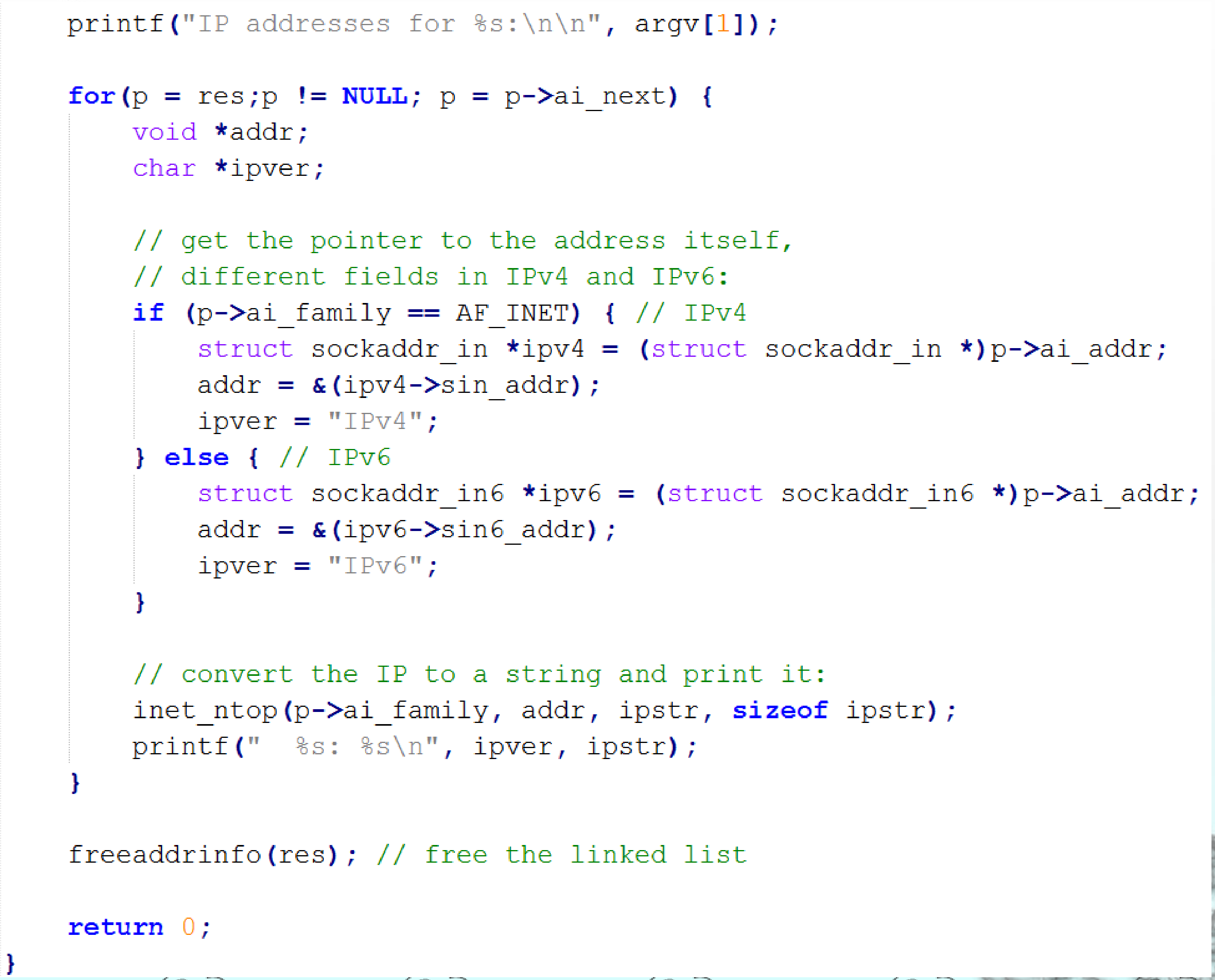


freeaddrinfo()

```
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
```



```c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
    struct addrinfo hints, *res, *p;
    int status;
    char ipstr[INET6_ADDRSTRLEN];

    if (argc != 2) {
        fprintf(stderr,"usage: showip hostname\n");
        return 1;
    }

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC; // AF_INET or AF_INET6 to force version
    hints.ai_socktype = SOCK_STREAM;

    if ((status = getaddrinfo(argv[1], NULL, &hints, &res)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(status));
        return 2;
    }
```



```
    printf("IP addresses for %s:\n\n", argv[1]);

    for(p = res;p != NULL; p = p->ai_next) {
        void *addr;
        char *ipver;

        // get the pointer to the address itself,
        // different fields in IPv4 and IPv6:
        if (p->ai_family == AF_INET) { // IPv4
            struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
            addr = &(ipv4->sin_addr);
            ipver = "IPv4";
        } else { // IPv6
            struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p->ai_addr;
            addr = &(ipv6->sin6_addr);
            ipver = "IPv6";
        }

        // convert the IP to a string and print it:
        inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
        printf("  %s: %s\n", ipver, ipstr);
    }

    freeaddrinfo(res); // free the linked list

    return 0;
}
```

Socket Address Structure

Socket Address Structure

Three Addressing Information

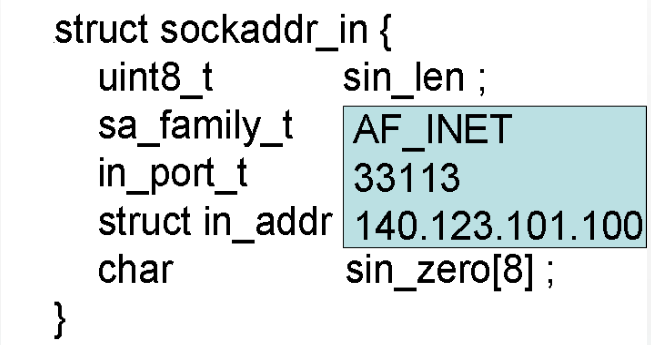
- Transport Layer : Port Number
- Transport Layer : Protocol Type
- Network Layer : IPv4 Address

Socket Address Structure

```
struct sockaddr_in {
        uint8_t      sin_len ;
        sa_family_t  sin_family ;
        in_port_t    sin_port ;
        struct in_addr sin_addr ;
        char         sin_zero[8] ;
    }
```

sa_family_t : unsigned short

in_port_t : uint16_t (unsigned 16-bit integer)



```
struct sockaddr_in {
    uint8_t          sin_len ;
    sa_family_t      AF_INET
    in_port_t        33113
    struct in_addr   140.123.101.100
    char             sin_zero[8] ;
}
```

Socket Address Structure

- Conversion of Network Byte Address and Address String
- Byte Manipulation Functions
  - int inet_aton()
  - in_addr_t inet_addr()
  - char *inet_ntoa()
  - int inet_pton()
  - const char *inet_ntop()
- Byte Ordering Functions – Port Number
  - uint16_t htons() – Value in Network Byte Order
  - uint32_t htonl() – Value in Network Byte Order
  - uint16_t ntohs() –Value in Host Byte Order
  - uint32_t ntohl() – Value in Host Byte Order

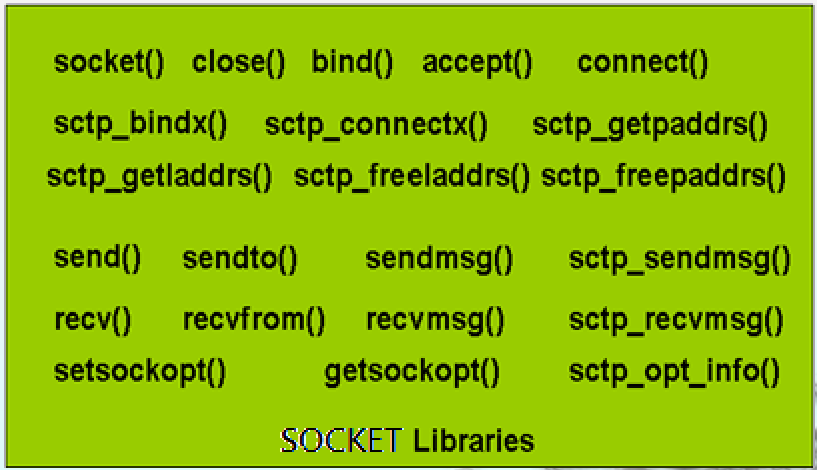
Host Name and Address Translation

- Translation of Host Name and IP Address – DNS
  - struct hostent {}
- IPv4 Host and Address Translation Functions
  - gethostbyname()

    http://www.logix.cz/michal/devel/various/gethostbyname.c.xp
  - gethostbyaddr()

Socket Libraries

- Transport Layer Services
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
  - Stream Control Transmission Protocol (SCTP)
- Socket Libraries
  - Socket Creation/Closing
  - Socket Data Transmission
  - Socket Options



socket() close() bind() accept() connect()

sctp_bindx() sctp_connectx() sctp_getpaddrs()

sctp_getladdrs() sctp_freeladdrs() sctp_freepaddrs()

send() sendto() sendmsg() sctp_sendmsg()

recv() recvfrom() recvmsg() sctp_recvmsg()

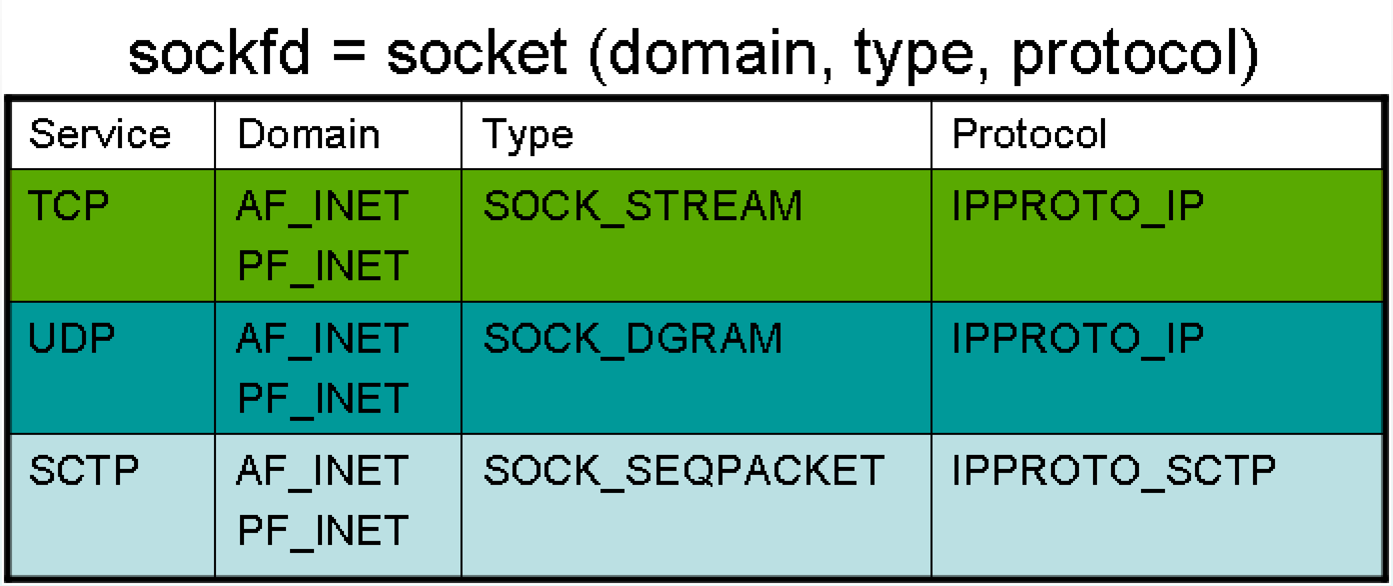
setsockopt() getsockopt() sctp_opt_info()

SOCKET Libraries

Operation of Socket Libraries

- 1. Creating Socket Handle
    - Protocol Type
    - Protocol Family
- 2. At Server - Binding Port Number and specific IP address
- 3. At Client - Connecting Server's Socket with specific Port Number and IP Address
- 4. Data Transmission
    - Sending and Receiving Data
- 5. Closing Socket Handle

Socket Categories



sockfd = socket (domain, type, protocol)

Service	Domain	Type	Protocol
TCP	AF_INET PF_INET	SOCK_STREAM	IPPROTO_IP
UDP	AF_INET PF_INET	SOCK_DGRAM	IPPROTO_IP
SCTP	AF_INET PF_INET	SOCK_SEQPACKET	IPPROTO_SCTP

Categories of Socket Libraries

Category	Functions

Common	socket(), close(), setsockopt(), getsockopt()
	gethostbyname(), htons(), htonl(), bind()

TCP	connect(), listen(), accept(), send(), recv()
UDP	sendto(), recvfrom()

SCTP	sctp_sendto(), sctp_bindx()
	sctp_sendmsg(), sctp_recvmsg()

TCP Socket Programming

TCP Socket Programming

- 3-1. General Issues
- 3-2. Elementary TCP Socket Functions
- 3-3. TCP Client/Server Example

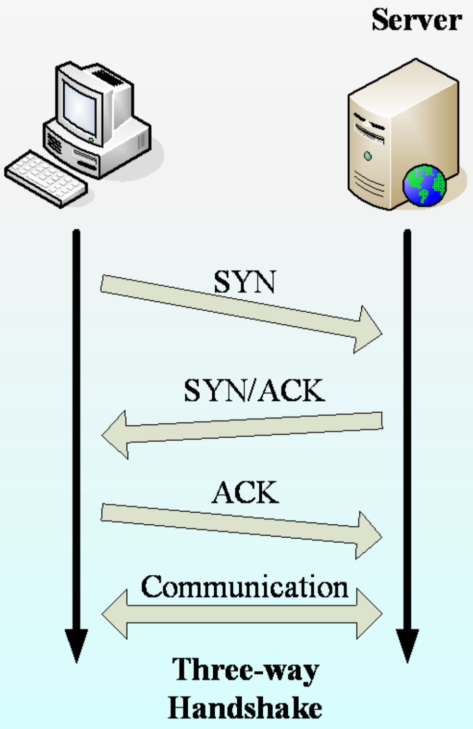
3-1. General Issues

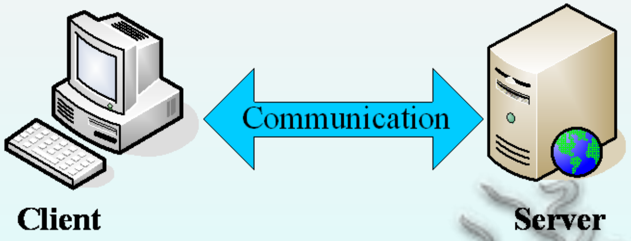
General Issues

- Connection-Oriented
- Point-to-Point
- Reliable Data Transfer
- Flow Control

Connection-Oriented

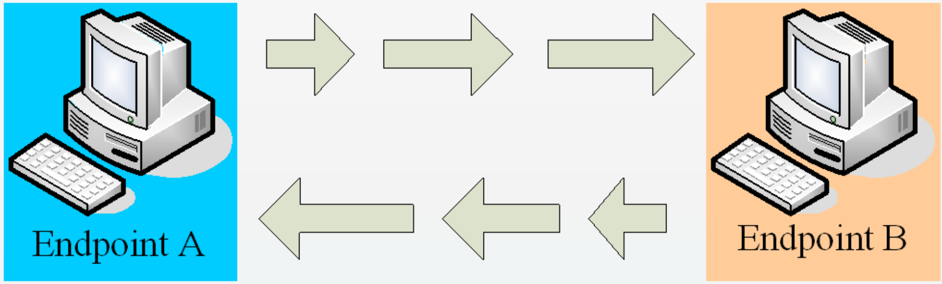
- Before two communicating TCPs can exchange data, they must first agree upon the willingness to communicate.
  - IP does not uses "connections" - each datagram is sent independently.





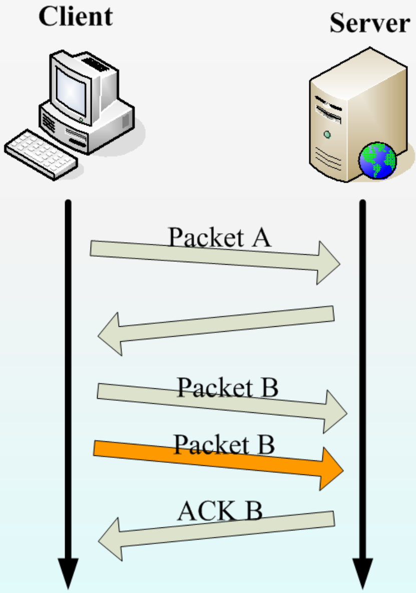
Point-to-Point

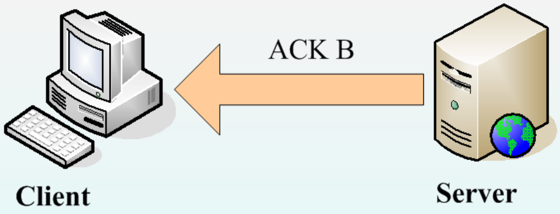
- A TCP connection has two endpoints.
  - No broadcast/multicast



Reliable Data Transfer

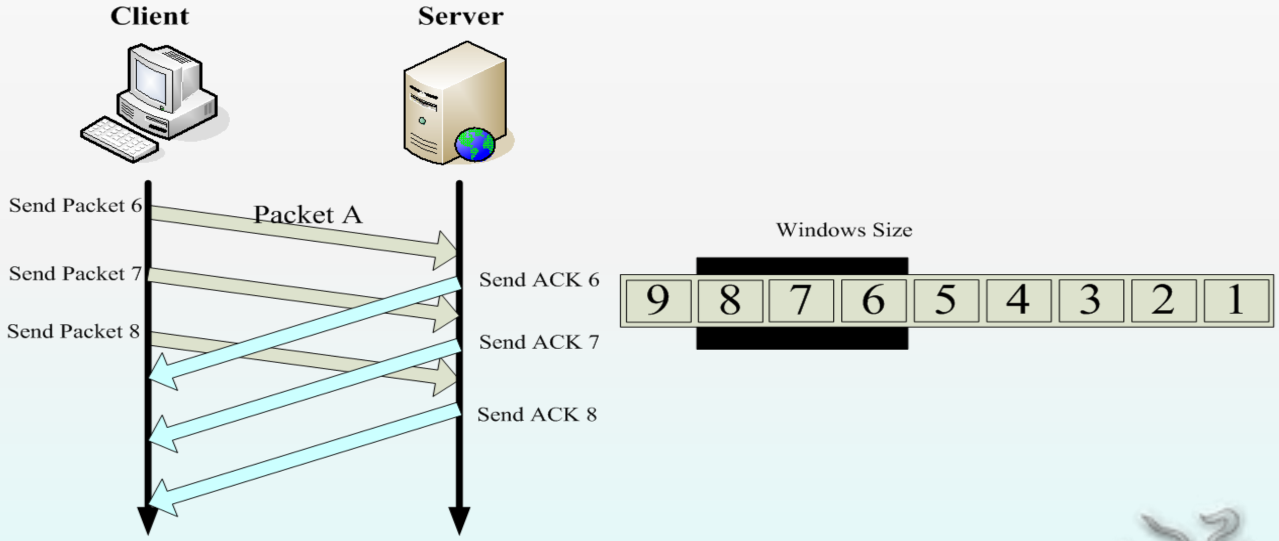
- TCP guarantees that data will be delivered without loss, out of order, duplication or transmission errors.





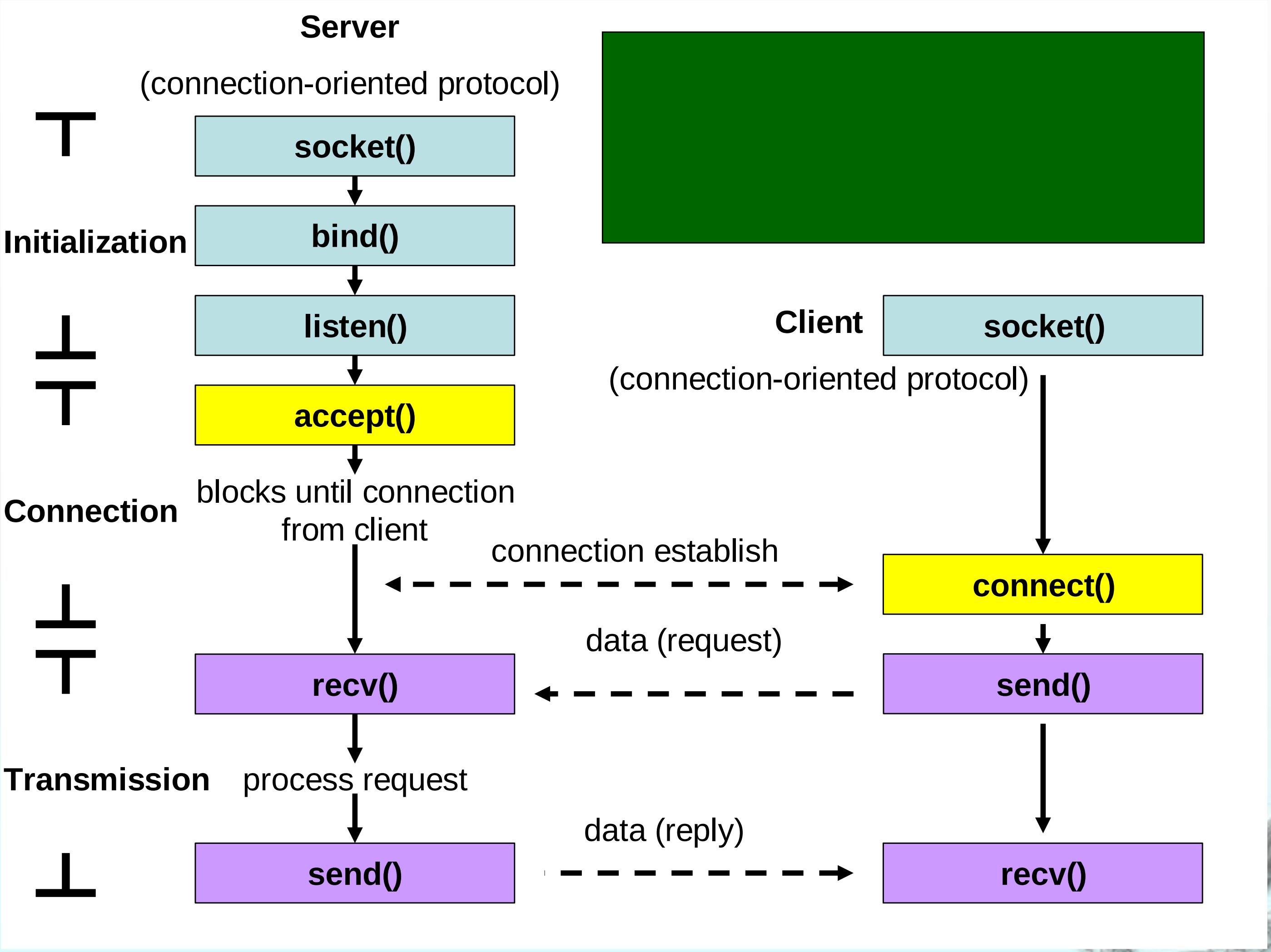
Flow Control

- TCP uses the ACK packets together with the sliding window mechanism.



3-2. Elementary TCP Socket Functions

Elementary of TCP Socket System Calls



Server

(connection-oriented protocol)

socket()

Initialization

bind()

listen()

accept()

Connection

blocks until connection from client

connection establish

Client

socket()

(connection-oriented protocol)

connect()

data (request)

recv()

send()

Transmission

process request

data (reply)

send()

recv()

Elementary of TCP Socket System Calls

- **socket()**

  Create an endpoint for communication.
- **bind()**

  Assign a local protocol address to a socket.
- **listen()**

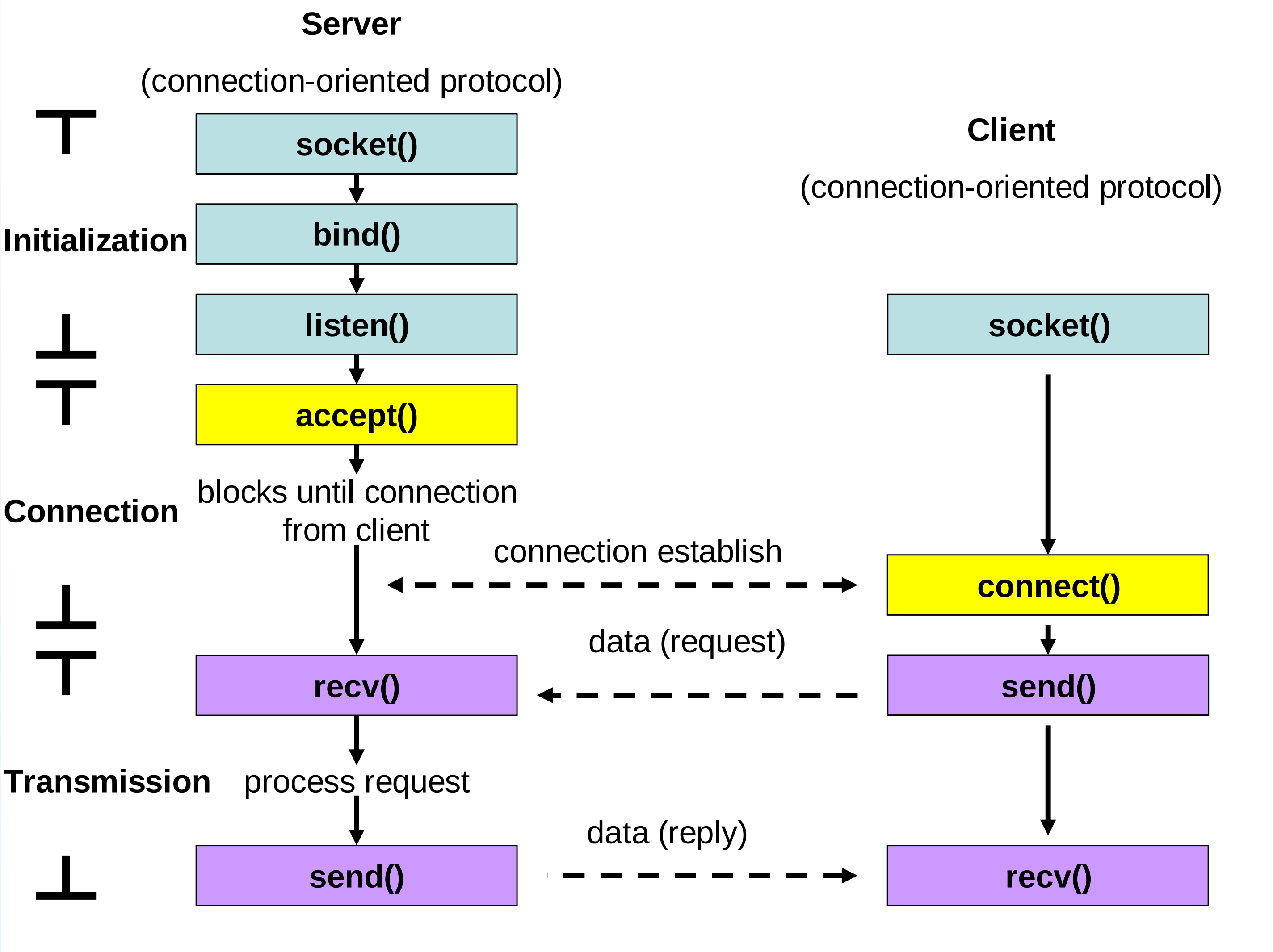
  Listen for connections on a socket.
- **accept()**

  Accept actual connection from some client process.
- **connect()**

  Initiate a connection on a socket.
- **send() & recv()**

  Send/ Receive a message from a socket.

TCP Program Operation Flow



TCP Functions Discussion

- **Socket()**

format：
```
SOCKET socket(  int domain,
                int type,
                int protocol );
```

argument： domain   PF_INET(AF_INET)

type   Socket type (SOCK_STREAM 、SOCK_DGRAM)

protocol   default 0

return value：success   Socket number

fail   INVALID_SOCKET

- example：

```
tcp_sock = socket(PF_INET, SOCK_STREAM, 0);
udp_sock = socket(PF_INET, SOCK_DGRAM, 0);
```

- **Bind()**

format： int bind(SOCKET s,
const struct sockaddr *name,
int namelen);

argument： s Socket number
name Socket address

return value： success 0
fail SOCKET_ERROR

- example：

```
struct sockaddr_in sa;
sa.sin_family = AF_INET;
sa.sin_port = htons(5001);   /* host to network for short int */

sa.sin_addr.s_addr = INADDR_ANY;
bind(sock, (struct sockaddr *)&sa, sizeof(sa));
```

- **Listen()**

format： int listen(SOCKET s,
int backlog);

argument： s Socket number
backlog maxima listen connection number

return value： success 0
fail SOCKET_ERROR

- example：

listen(sock, 1)

- **Connect()**

format： int connect(SOCKET s,
const struct sockaddr *name,
int namelen);

argument：	s	Socket number
	name	Socket address
	namelen	name length
return value：	success	0
	fail	SOCKET_ERROR

- example：

```
struct sockaddr_in sa;
sa.sin_family = AF_INET;
sa.sin_port = htons(5001);   /* server's port number*/
sa.sin_addr.s_addr = htonl(serverip);
connect( sock, (struct sockaddr *)&sa, sizeof(sa));
```

- **Accept()**

```
format： SOCKET accept( SOCKET s,
                        struct sockaddr *addr,
                        int *addrlen );
```

argument：	s	Socket number
	addr	Socket address
	addrlen	addr length
return value：	success	0
	fail	SOCKET_ERROR

- example：

```
struct sockaddr_in sa;
int sa_len = sizeof(sa);
new_sock = accept(sock, (struct sockaddr far *)&sa, &sa_len)
```

- **Send()**

```
format： int send( SOCKET s,
                   const void *buf,
                   int len,
                   int flags );
```

argument：

s	Socket number
buf	the data buffer for transmission
len	the data buffer length
flags	the way function is called

return value：

success	the sent data length
fail	SOCKET_ERROR

Recv()

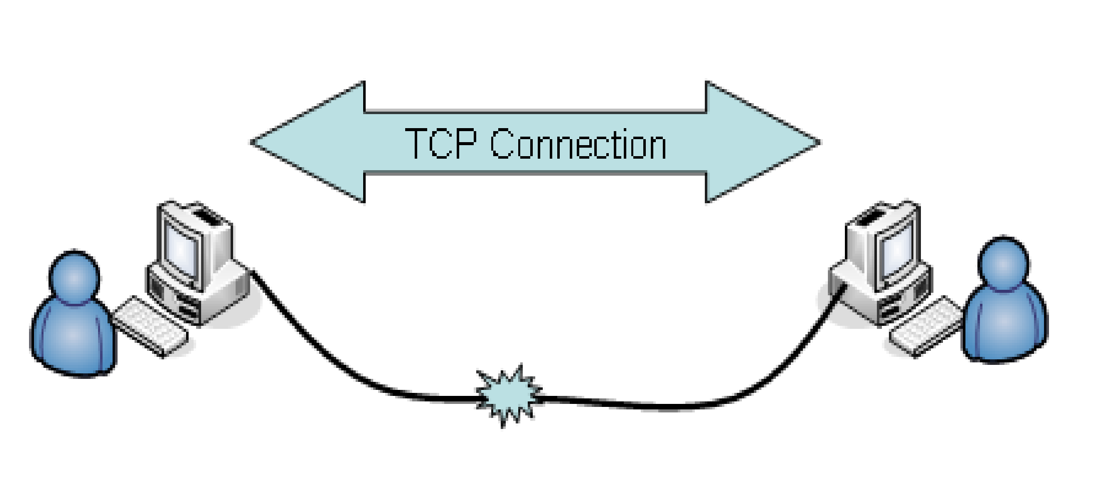
format： int recv(SOCKET s,
void *buf,
int len,
int flags);

argument：		
	s	Socket number
	buf	reveiving data buffer
	len	buffer length
	flags	the way function is called

return value：		
	success	received data length (it is 0 if socket on the other side is closed)
	fail	SOCKET_ERROR

3-3. TCP Client/Server Example

TCP Client/Server Example



Server Initialization

```
/* tcpserver.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h> /* INADDR_ANY */
#include <ctype.h> /* toupper() */

#define PORT 10000          /* server port value */

int main()
{
        int accept_sock;
        int client_sock;
        struct sockaddr_in serv_addr;
        char ch;

        /* create INTERNET,TCP socket */
        accept_sock = socket(PF_INET, SOCK_STREAM, 0);
```

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(PORT);      /* specific port */
/* bind protocol to socket */
if(bind(accept_sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr))
< 0)
{
  perror("bind");
  exit(1);
}
listen(accept_sock,5);
```

Server Connection

```

  for(;;)
  {
    /* accept one connection, will block here.  */
    client_sock = accept(accept_sock, 0, 0);
```

Server Transmission

```
    for(;;)
    {
      if(recv(client_sock, &ch, sizeof ch, 0) <= 0)
        break;
      ch = toupper(ch);
      send(client_sock, &ch, sizeof ch, 0);
      if(ch == '\0')                    // end of string
        break;
    }
    close(client_sock);                 // close a client socket
  }
//    close(accept_sock);                // unreachable
}
```

Client Initialization

```
/* tcpclient.c */

#include <stdio.h>
#include <sys/types.h>  /* basic system data types */
#include <sys/socket.h> /* basic socket definitions */
#include <netinet/in.h> /* sockaddr_in{} and other Internet defns */

#define PORT 10000          /* server port value */

int main()
{
 int                connect_sock;
 char                input[100], output[100], *s = output;
 struct sockaddr_in    serv_addr;
 int                n;

 connect_sock = socket(PF_INET, SOCK_STREAM, 0);

```

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_port   = htons(PORT);
serv_addr.sin_addr.s_addr = inet_addr(“127.0.0.1");
```

Client Connection

```

if (connect(connect_sock, (struct sockaddr *) &serv_addr, sizeof serv_addr) <0)
{
  perror("connect");
  exit(1);
}

```

Client Transmission

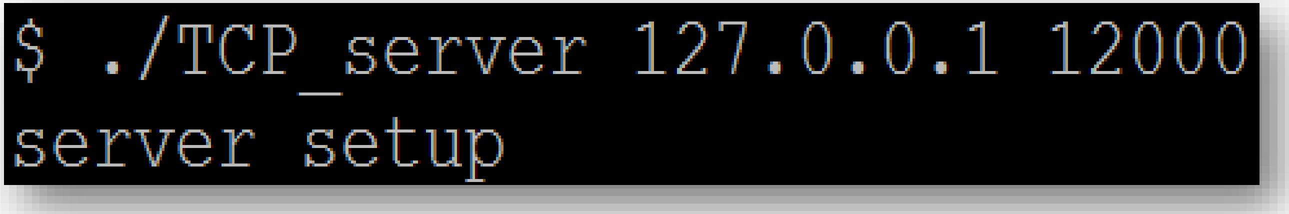
```
printf("Input:");
scanf("%s", input);
send(connect_sock, input, strlen(input) + 1 ,0);  // including ending '\0'
while((n = recv(connect_sock, s, sizeof *s, 0)) > 0) {
  s += n;                                  // move pointer for recv()
}
printf("Output:%s\n", output);
close(connect_sock);
}
```

課堂作業 1 (描述)

- 寫一個會標記「已讀」訊息的echo server，並寫一支client程式跟echo server對話。
  - 使用TCP溝通
  - Server和client皆需把對方送來的訊息印出

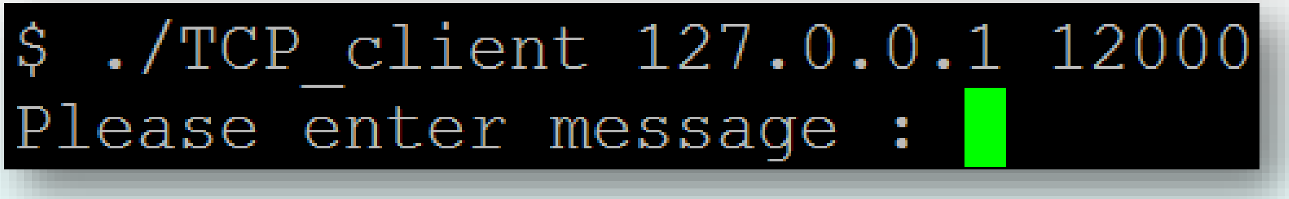
課堂作業 1 (執行結果1/2)

1. 開啟server

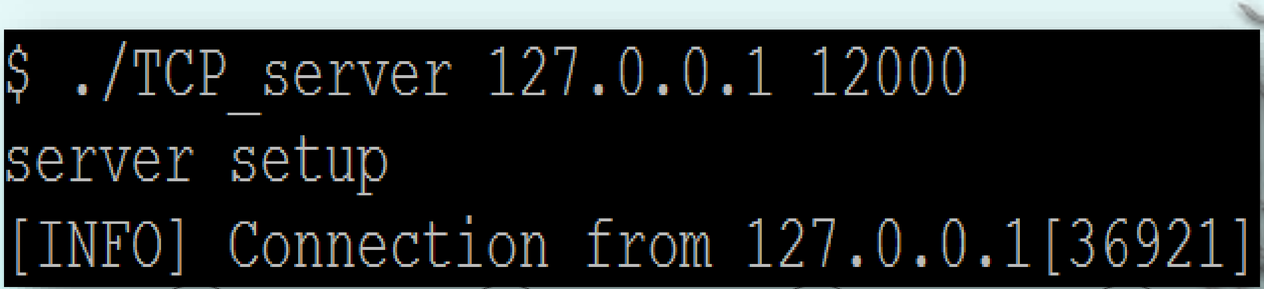


```
$ ./TCP_server 127.0.0.1 12000
server setup
```

2. client 連上 server



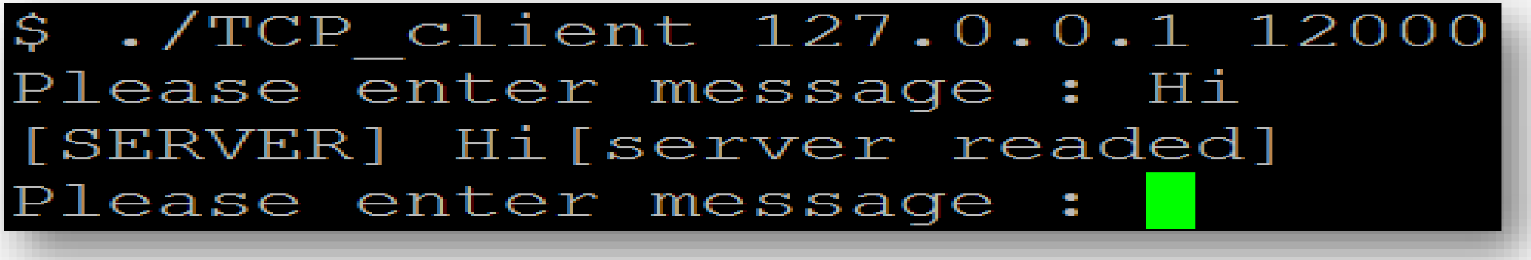
```
$ ./TCP_client 127.0.0.1 12000
Please enter message :
```



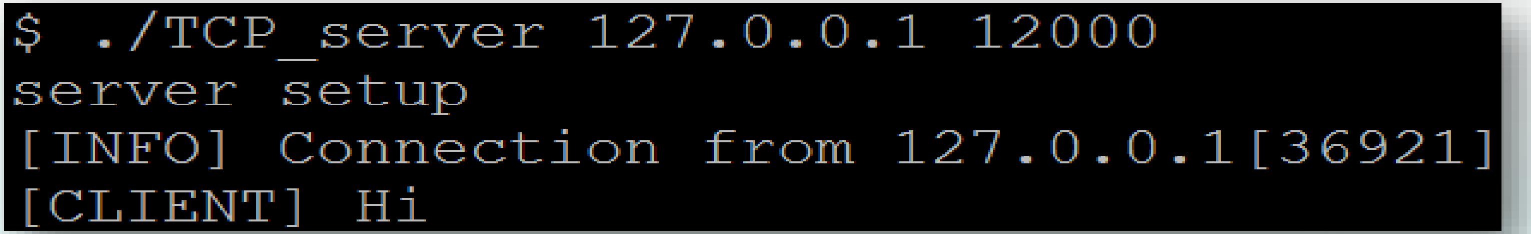
```
$ ./TCP_server 127.0.0.1 12000
server setup
[INFO] Connection from 127.0.0.1[36921]
```

課堂作業 1 (執行結果2/2)

3. client 送訊息給 server後，server 回覆「已讀」



```
$ ./TCP_client 127.0.0.1 12000
Please enter message : Hi
[SERVER] Hi[server readed]
Please enter message :
```



```
$ ./TCP_server 127.0.0.1 12000
server setup
[INFO] Connection from 127.0.0.1[36921]
[CLIENT] Hi
```