Socket Programming

CSEE, Handong Univ.

Jong-won Lee

ljw@handong.edu

Network Programming Interface (API)

- API
  - application callable services
  - interfaces and abstractions provided by the system to the application

- Network Programming at different levels
  - send Ethernet, ATM, … packets
  - exchange UDP/TCP packets
  - RPC, Xlib, Corba,...

API for TCP/IP

- TCP/IP does not include an API definition.

- There are a variety of APIs for use with TCP/IP:
  - Sockets
  - TLI, XTI
  - Winsock
  - MacTCP

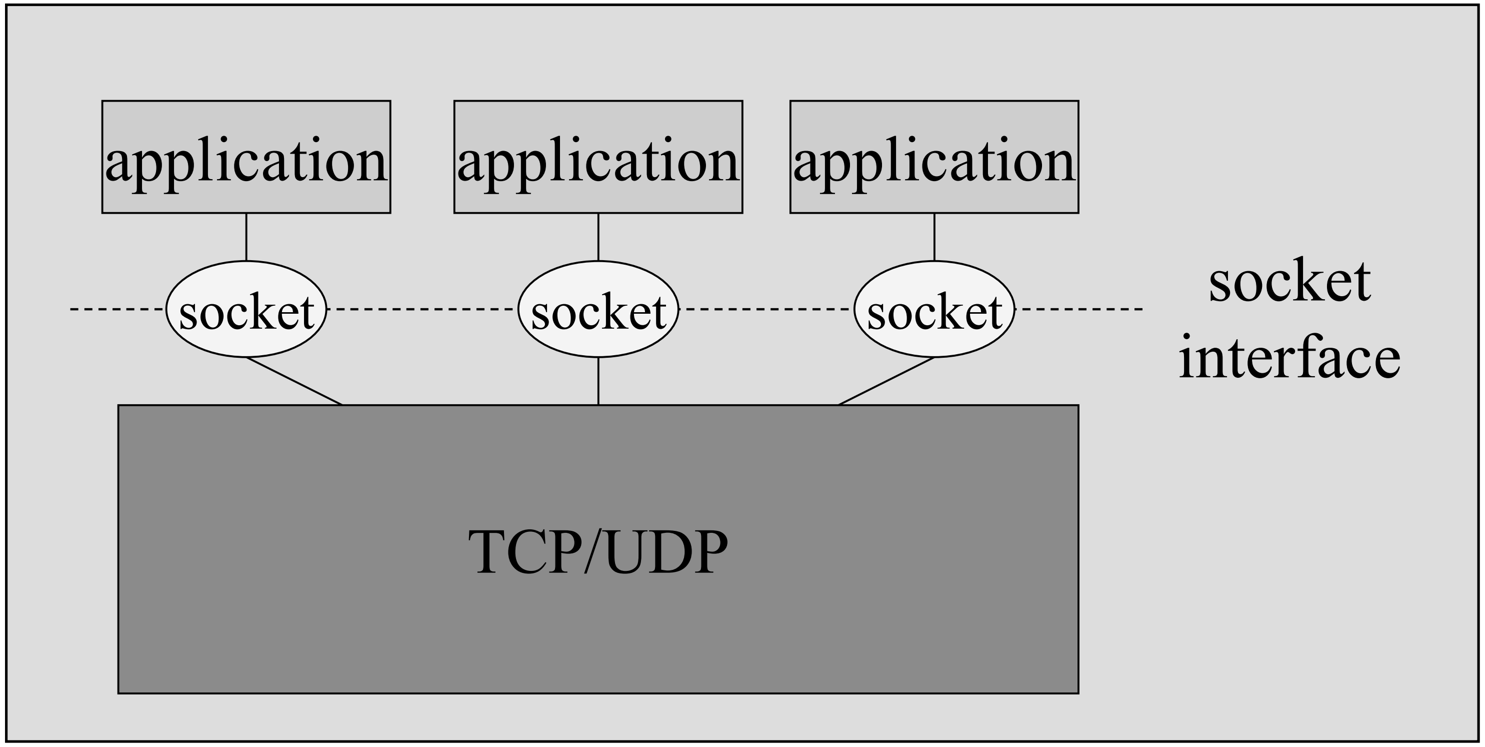
Socket API

- Introduce in 1981 by BSD 4.1
- support for multiple protocol families.
- originally only UNIX (=> winsock)
- implemented as system calls(BSD) or library(SVR4)

- for TCP/IP, three socket types:
  - stream-oriented: TCP
  - datagram: UDP
  - raw IP: IP, ICMP

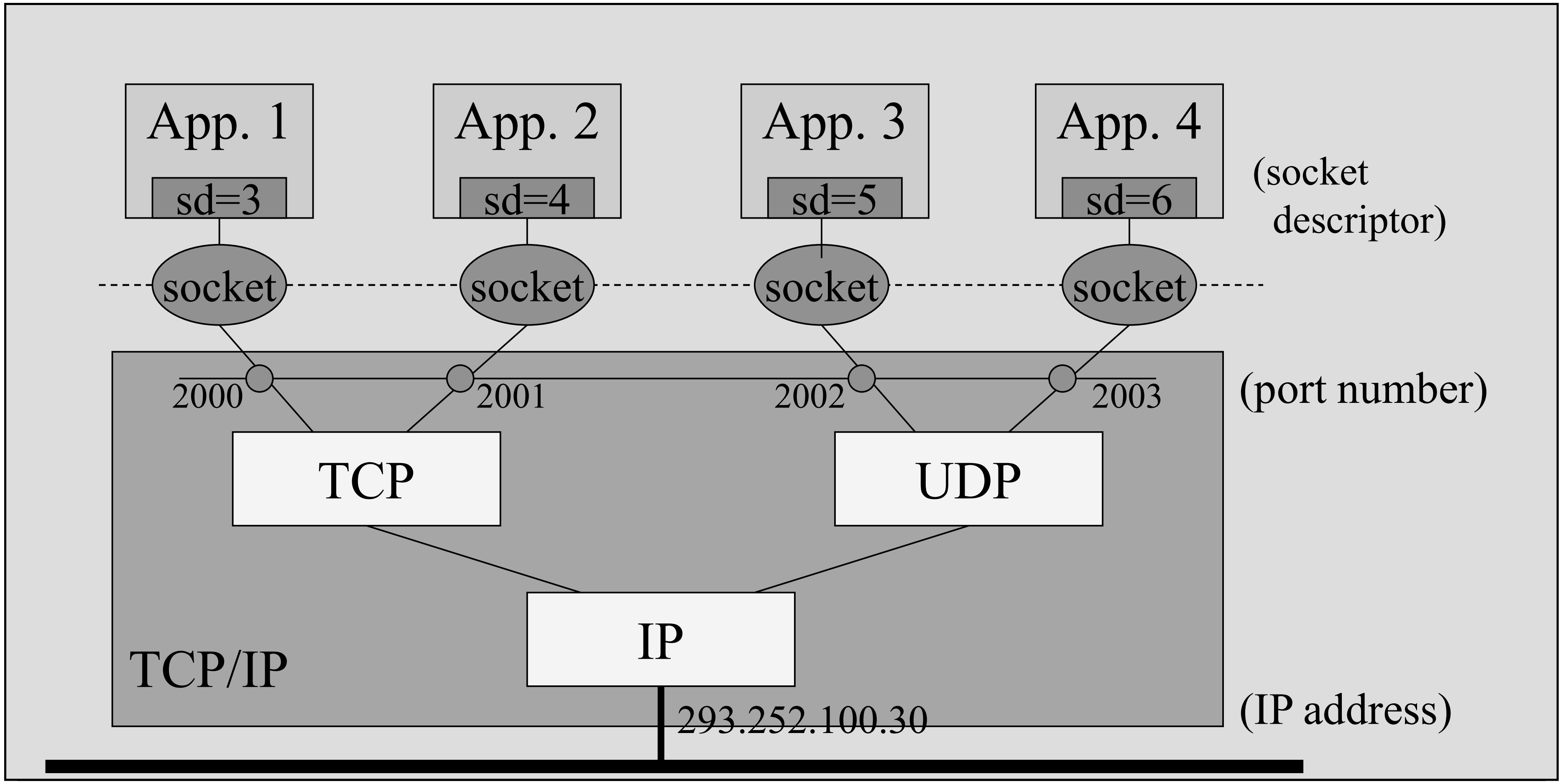
Socket

- **socket interface:**
  - **located between application and TCP, UDP and other protocol stacks (common interface)**



Socket

❑socket and TCP/UDP relationship



Socket

- Each socket is associated with five components.
    - Protocol
        - protocol family and protocol
    - source address, source port
    - destination address, destination port

- Where to define components
    - protocol: `socket()`
    - source address and port: `bind()`
    - destination address and port: `connect()`
      `sendto()`

Creating a socket

- Synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol)
```

- **The socket() system call returns a socket descriptor (small integer) or a -1 on error.**

- **socket() allocates resources needed for a communication endpoint**

Creating a socket

- Synopsis
  - `int socket(`int *family*, int *type*, int *protocol*)
- `family` specifies the protocol family
  - PF_INET : Internet protocol (TCP/IP)
  - PF_INET6 : IPv6
  - PF_LOCAL : for local communication
  - PF_UNIX : UNIX system internal protocol
- `type` specifies the type of service
  - SOCK_STREAM :
  - SOCK_DGRAM :
  - SOCK_RAW : raw IP
- `protocol` specifies the specific protocol
  - (usually 0 which means the default).

Creating a socket

Example

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

int sockfd;

if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      /* print “socket error + the error message */
      perror(“socket error”); exit(1);
}
```

Print error : perror()

```
#include <stdio.h>

void perror(const char *s)
```

Bind the local address

- **Synopsis**

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

int bind(int  sockfd,  struct sockaddr *addr, int  addr_len)
```

- **The bind() system call is used to assign an address to an existing socket.**

- **Bind() returns 0 if successful or -1 on error.**

Address Structures

- Defined in <netinet/in.h>

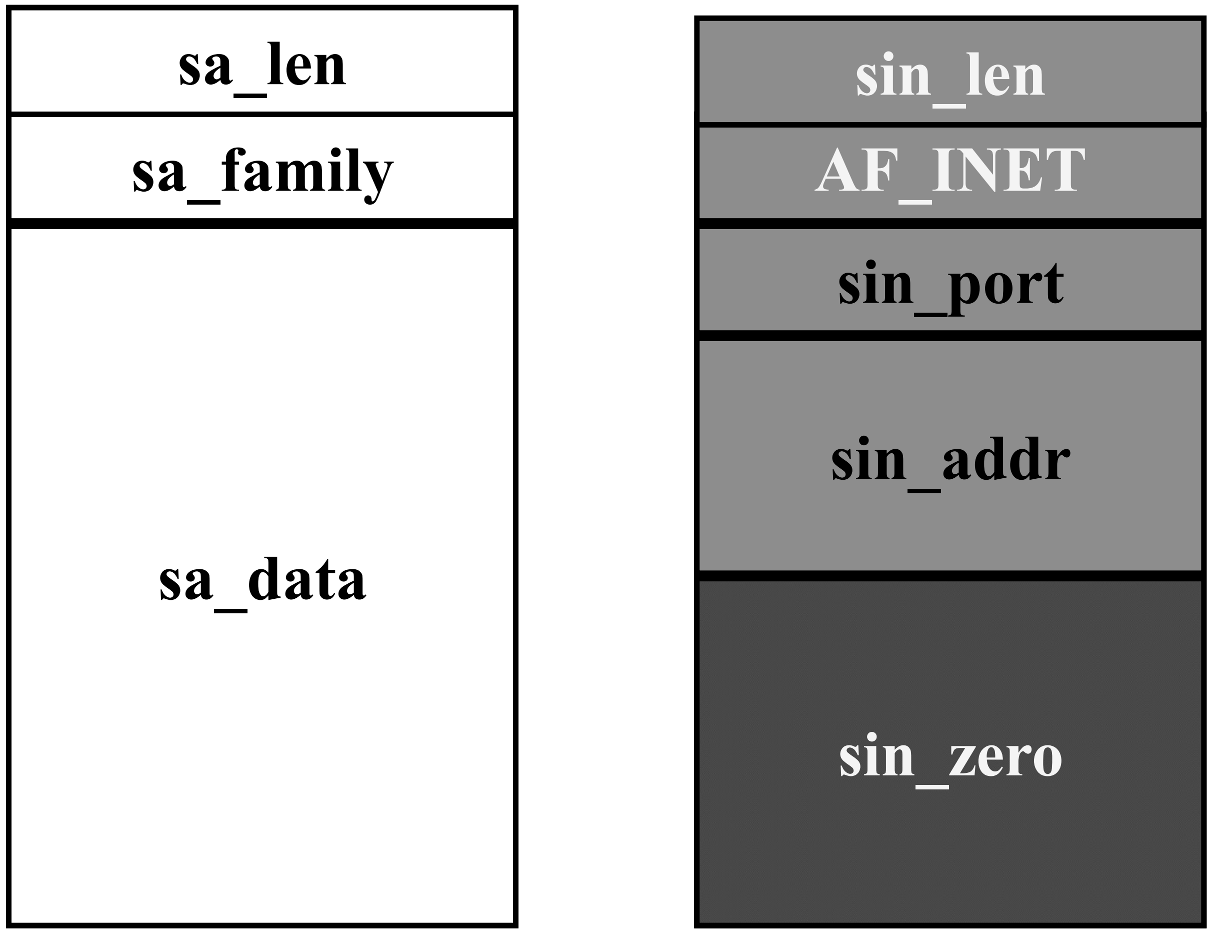
```
struct  sockaddr {
        u_char              sa_len;            /* length : used in kernel */
        u_short             sa_family;         /* address family */
        char                sa_data[14];       /* address */
}

struct  sockaddr_in {
        u_char              sin_len;           /* length */
        u_short             sin_family;        /* AF_INET */
        u_short             sin_port;          /* port number */
        struct in_addr      sin_addr;          /* IP addess */
        char                sin_zero[8];       /* unused */
}

struct  in_addr {
        u_long              s_addr;  /* 32 bit IP address */
}
```

Address Structures

- sockaddr vs. sockaddr_in



Bind() Example

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

#define MYPORT 50000

int sockfd;
struct sockaddr_in my_addr;

if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      perror(“socket error”); exit (1); }
memset(&my_addr, 0, sizeof(my_addr));
/* bzero( (char *)&my_addr, sizeof(my_addr)); */
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYPORT);
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) < 0) {
   perror(“bind error” ); exit(1);}
```

Byte Order Conversion

- **Byte ordering**
  - **little endian: least significant byte first (Intel)**
  - **big endian: most significant byte first (Motorola, Sun)**
- ***network byte order* = big endian**
- **Synopsis**

```
#include <sys/types.h>
#include <netinet/in.h>

/* byte swapping host byte order <-> network */
/* for long and short integer */
u_long   htonl(u_long hostlong);

u_long ntohl(u_long netlong);

u_short htons(u_short hostshort);

u_short ntohs(u_short netshort)
```

Name-to-Address Conversion

Synopsis

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

/* return host information by taking host name */
struct hostent *gethostbyname(const char *name);

/* return host information by taking network byte order
   address */
struct hostent *gethostbyaddr(const char *addr, int
   length, int type);
```

Looking up A Domain Name

Defined in <netdb.h>

```
#define h_addr haddr_list[0]
struct hostent {
    char    *h_name;         /* host name */
    char    **h_alias;       /* list of alternate names */
    int     h_addrtype;      /* type of address = 2 (=AF_INET) */
    int     h_length;        /* address length = 4 for IPv4 */
    char    **h_addr_list;;  /* address list */
  };
```

Gethostbyname()

Example

```
struct hostent *phost;
struct in_addr **addr_list;

if ((phost = gethostbyname(“www.handong.edu”)) == NULL) {
    perror("gethostbyname");
    return 1;
}
// print information about this host:
printf("Official name is: %s\n", phost->h_name);
printf("    IP addresses: ");
addr_list = (struct in_addr **)phost->h_addr_list;
for(i = 0; addr_list[i] != NULL; i++) {
    printf("%s ", inet_ntoa(*addr_list[i]));
}
printf("\n");
```

Gethostbyaddr()

Example

```
 struct hostent *phost;
 struct in_addr addr;

 inet_aton(“203.252.97.12", &addr);
phost = gethostbyaddr(&addr, sizeof(addr), AF_INET);
printf("Host name: %s\n", phost->h_name);
```

Looking up A Well-known Port by Name

Defined in <netdb.h>

```
struct servent {
      char     *s_name;         /* official service name */
      char     **s_alias;       /* list of alternate names */
      int      s_port;          /* port for this service */
      int      *s_proto;        /* protocol to use */
   };

struct servent *pService;

if (pService = getservbyname("smtp", "tcp")) {
   // port number is now in pService->s_port;
} else {
   /* error */
}
```

New Functions: name-to-addr conversion

getaddrinfo(): converts human-readable text strings representing hostname into a dynamically allocated linked list of struct addrinfo structures.

getnameinfo(): looks up the host name and service name information for a given struct sockaddr.

Synopsis

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

int getaddrinfo(const char *name, const char *servname,
     const struct addrinfo *hints, struct addrinfo **res);

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
             char *host, size_t hostlen,
             char *serv, size_t servlen, int flags)
```

New Functions: name-to-addr conversion

Struct addrinfo

```
struct addrinfo {
  int    ai_flags;                   // AI_PASSIVE, AI_CANONNAME, ...
  int    ai_family;                  // AF_xxx
  int    ai_socktype;                // SOCK_xxx
  int    ai_protocol;       // 0 (auto) or IPPROTO_TCP, IPPROTO_UDP
  socklen_t  ai_addrlen;      // length of ai_addr
  char   *ai_canonname;       // canonical name for nodename
  struct sockaddr  *ai_addr; // binary address
  struct addrinfo  *ai_next;   // next structure in linked list
};
```

Ge

E

```
int sockfd;
struct addrinfo  hints,  *servinfo,  *p;
int rv;
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;        // use AF_INET6 to force IPv6
hints.ai_socktype = SOCK_STREAM;

if ((rv = getaddrinfo("www.example.com", "http", &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    exit(1);
}

for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,  p->ai_protocol)) == -1) {
        perror("socket");
        continue;
    }
    if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("connect");
        continue;
    }
    break; // if we get here, we must have connected successfully
}
…

freeaddrinfo(servinfo); // all done with this structure
```

Getnameinfo()

Example

```
struct sockaddr_in sa;
char host[1024];
char service[20];

//pretend sa is full of good information about the host and port...

getnameinfo(&sa, sizeof sa, host, sizeof (host), service, sizeof (service), 0);

printf("   host: %s\n", host);
printf("service: %s\n", service);
```

IP Address Manupulation

Dotted decimal vs. IP address

Synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet.h/in.h>

/* IP address to dotted decimal */
char *inet_ntoa(struct in_addr address);

/* dotted decimal to IP address */
u_long inet_addr(char *dottedAddress);
                // return -1 if error
Int  inet_aton(const char *cp, struct in_addr *inp);
                // return 0 if error
```

IP Address Manupulation

Synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet.h/in.h>
#include <arpa.h>

struct  sockaddr_in   my_addr;

memset(&my_addr, 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYPORT);
inet_aton(“10.12.110.57”, &(my_addr.sin_addr));
/* my_addr.sin_addr.s_addr = inet_addr(“10.12.110.57”); */
printf(“%s”, inet_ntoa(my_addr.sin_addr);
```

Address Format Conversion Summary

$$\begin{tikzcd}[column sep=huge]
\textbf{Domain name :} \arrow[r, shift left=2, "\texttt{gethostbyname()}"] & \textbf{IP address (binary) :} \arrow[l, shift left=2, "\texttt{gethostbyaddr()}"'] \arrow[r, shift left=2, "\texttt{inet\_ntoa()}"] & \textbf{Dotted decimal} \arrow[l, shift left=2, "\texttt{inet\_addr()}"', "\texttt{inet\_aton()}"]
\end{tikzcd}$$

IP Address Manupulation

Synopsis

```
#include <sys/socket.h>
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *src,
                    char *dst, socklen_t size);
               // return value: null if error

int inet_pton(int af, const char *src, void *dst)
               //return value: -1 if error,
               //                       0 for invalid IP address
```

Example

```
struct sockaddr_in sa;
char str[INET_ADDRSTRLEN];

// store this IP address in sa:
inet_pton(AF_INET, "192.0.2.33", &(sa.sin_addr));

// now get it back and print it
inet_ntop(AF_INET, &(sa.sin_addr), str, INET_ADDRSTRLEN);
printf("%s\n", str); // prints "192.0.2.33"
```

Address Format Conversion Summary

$$\begin{tikzcd}
\text{Domain name} \arrow[r, shift left, "\texttt{getaddrinfo()}"] & \text{IP address (binary)} \arrow[l, shift left, "\texttt{getnameinfo()}"] \arrow[r, shift left, "\texttt{inet\_ntop()}"] & \text{Dotted decimal} \arrow[l, shift left, "\texttt{inet\_pton()}"]
\end{tikzcd}$$

System Call : connect()

Initiate a connection on a socket (client only)

- Returns 0 on success; -1 on failure
- For a TCP socket, it establishes a connection to the server
- For a UDP socket, it simply stores the server's address so that the client can use a socket description

Synopsis

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

int connect(int sd, struct sockaddr *addr, int addr_len);
/* addr = server address */
```

Connect() Example

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define DEST_IP  “10.12.110.57”
#define MYPORT 23

int sockfd;
struct sockaddr_in dest_addr;

if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
      /* error */ }
memset(&dest_addr, 0, sizeof(dest_addr));
dest_addr.sin_family = AF_INET;
dest_addr.sin_port = htons(DEST_PORT);
dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
If (connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(dest_addr)) !=0 ) {
    close(sockfd);
    return -1;
}
```

System Call : listen()

Tell OS to receive and queue SYN packets

Specify backlog size for the pending connection requests on a socket (TCP server only)

- **Returns 0 on success; -1 on failure**

Synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
int listen(int sd, int backlog);
```

Parameter

- **bakclog : specifies the max. number of connection requests that system can queue while it waits for the server to accept them (usually 5)**

System Call : accept()

Accept a connection on a socket

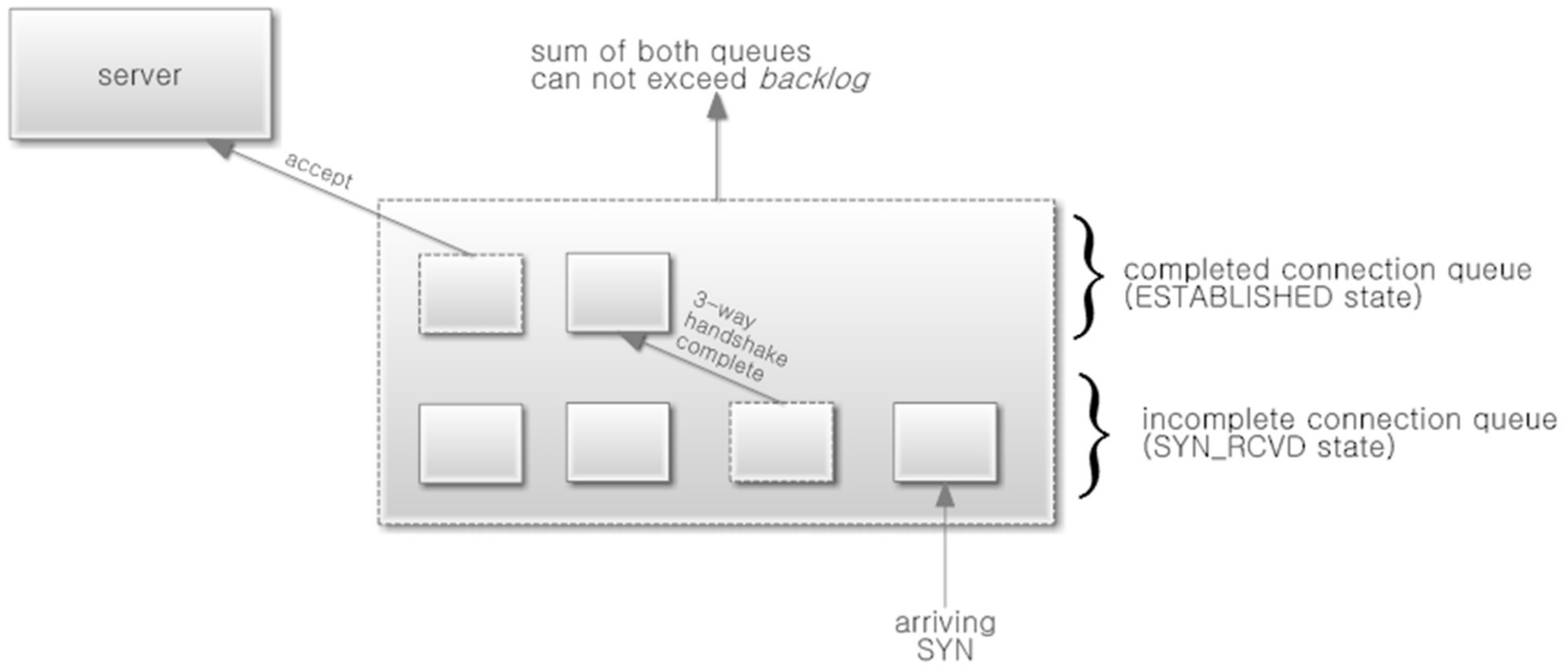
- TCP server only
- Returns a new socket descriptor (>0) on success; -1 on failure
- Block until a connection request arrives

Synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sd, struct sockaddr *addr, int *addrlen);
```

Parameters

- addr: a pointer to an address structures to be filled in
- addrlen: a pointer to an integer that should be set to sizeof(struct sockaddr_in)


server
accept
sum of both queues can not exceed *backlog*
3-way handshake complete
completed connection queue (ESTABLISHED state)
incomplete connection queue (SYN_RCVD state)
arriving SYN

Example of TCP Server

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MYPORT 50000
main()
{
    int sockfd, new_fd;
    struct sockaddr_in my_addr, client_addr;
    int sin_size;

    sockfd = socket(PF_INET, SOCK_STREAM, 0);
    memset(&my_addr, 0, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT);
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    listen(sockfd, 5);
   sin_size = sizeof(client_addr);
   new_fd = accept(sockfd, (struct sockaddr *)&client_addr, &sin_size);
```

Sending and Receiving data

Function calls

- read(), write() : any device, single buffering
- readv(), writev() : any device, from/to several buffers
- recv(), send() : socket device, single buffer
- recvfrom(), sendto() : socket device, single buffer, specifying peer

System calls : read() and write()

read()

- int read(int fd, char *buf, int buflen)
- Returns the number of bytes received on success; -1 on failure
  - When 0 is returned, it means that the remote side has closed the connection.
- Block until data received

write()

- int write(int fd, char *buf, int buflen)
- Returns the number of bytes transmitted on success; -1 on failure

System calls : send()

send()

#include <sys/types.h>
#include <sys/socket.h>
int send(int sockfd, const void *msg, int len, int flag)

- **Returns the number of bytes transmitted on success; -1 on failure**
- **msg: the pointer to the data to send**
- **flag: 0 for normal data**

- **Example:**

```
char *msg = “Hi, Beej!”;
int len, bytes_sent;
len = strlen(msg);
byte_sent = send(sockfd, msg, len, 0);
```

System calls : recv()

synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
int recv(int sockfd, void *buf, int len, unsigned int flag)
```

- **Returns the number of bytes received on success; -1 on failure**
  - **When 0 is returned, it means that the remote side has closed the connection.**
- **Block until data received**
- **buf: the buffer to read the information into**
- **len: the max. length of the buffer**
- **flag: 0 for regular data**

System Call : sendto()

Synopsis

```
 #include <sys/types.h>
#include <sys/socket.h>
int sendto(int sockfd, const void *msg, int len, int flag,
               struct sockaddr *dstaddr, int addrlen)
```

- Returns the number of bytes sent on success;   -1 on failure

Parameters

- Transmit the data in ***msg*** upto ***len*** bytes
- **flags : to control transmission behavior (0 for normal)**
- **The dstaddr includes the information of destination address**

System Call : recvfrom()

synopsis

```
#include <sys/types.h>
#include <sys/socket.h>
int recvfrom(int sockfd, void *buf, int len, int flags,
                    struct sockaddr *srcaddr, int *addrlen)
```

- Returns the number of bytes received on success; -1 on failure
- Blocks until data received (normal operation)

Parameters

- Receives up to *len* bytes into *buf*
- flags : to control transmission behavior (0 for normal)
- The srcaddr is filled in with the address of the sender.

System Call : close()

Synopsis

```
#include <unistd.h>
int close(int sockfd)
```

- Prevent any more reads and writes to the socket.
  - if the remote side calls recv(), it will return 0.
  - if the remote side calls send(), it'll receive a signal SIGPIPE and send() will return -1 and errno will be set to EPIPE.

System Call : shutdown()

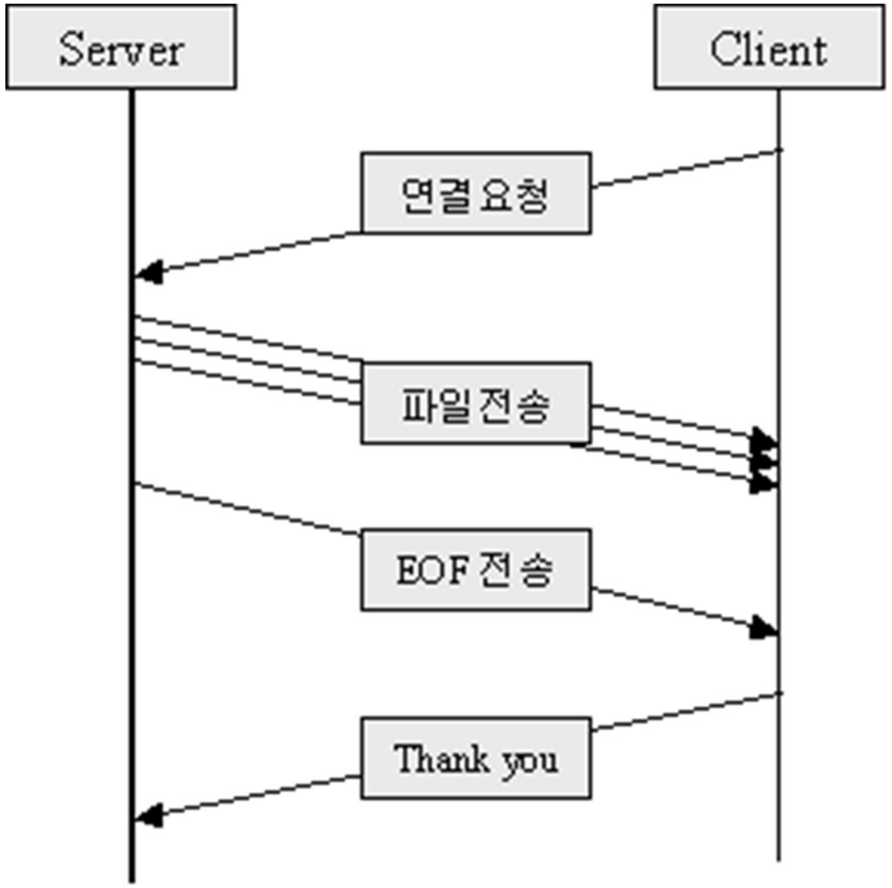
Synopsis

```
#include <sys/socket.h>
Int shutdown(int sockfd, int how)
```

- Returns 0 on success, and -1 on error
- The operation depends on the value of how
  - 0 (SHUT_RD): further receives are disallowed
  - 1 (SHUT_WR): further sends are disallowed
  - 2 (SHUT_RDWR): further sends and receives are disallowed
- Note that shutdown() does not actually free up the socket descriptor. To free the descriptor, use close().

Half-close: shutdown()

Can notify the end of file transmission.


Server
Client
연결요청
파일전송
EOF 전송
Thank you