Sockets

Hyo-bong Son (proshb@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Sockets (1)

- **Sockets interface**
  - Introduced in BSD4.1 UNIX, 1981.
  - Provides a user-level interface to the network.
  - Explicitly created, used, released by applications.
  - Based on client/server paradigm
  - Two types of transport service
    - Unreliable datagram
    - Reliable, connection-oriented byte stream
  - Underlying basis for all Internet applications

Socket Address Structure(1)

- **Generic socket address**
  - For address arguments to **connect(), bind(),** and **accept()**

```
struct sockaddr {
  unsigned short  sa_family;    /* protocol family */
  char            sa_data[14];  /* address data.  */
};
```

- **Internet-specific socket address**
  - Must cast **(sockaddr_in *)** to **(sockaddr *)** for **connect(), bind(),** and **accept()**

```
struct sockaddr_in  {
  unsigned short  sin_family;  /* address family (always AF_INET) */
  unsigned short  sin_port;    /* port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

IP Addresses (1)

- **Storing IP addresses**
  - IP addresses (and other integer values such as port number) are always stored in memory in network byte order (big endian)

```
/* Internet address structure */
struct in_addr {
    unsigned int s_addr; /* network byte order (big-endian) */
};
```

  - Handy network byte-order conversion functions:
    - **htonl()**: long int from host to network byte order
    - **htons()**: short int from host to network byte order
    - **ntohl():** long int from network to host byte order
    - **ntohs()**: short int from network to host byte order

IP Addresses (2)

- **Dotted decimal notation**
  - By convention, each byte in a 32-bit IP address is represented by its decimal value and separated by a period.
    - IP address 0x739198B5 = 115.145.152.181
  - Converting functions
    - **inet_aton():** a dotted decimal string to an IP address in network byte order
    - **inet_ntoa():** an IP address in network byte order to its corresponding dotted decimal string
    - "n" denotes network representation. "a" denotes application representation.

Socket Address Structure(2)

- **How to fill a socket address structure?**

```
int inet_pton(int af, const char *src, void *dst);
```

  - Returns 1 on success
  - int af – address family (AF_INET, AF_INET6)
  - const char *src – character string containing address
  - void *dst – socket address structure pointer

```
int main(int argc, char **argv)
{
    struct sockaddr_in sa;
    char str[INET_ADDRSTRLEN];

    inet_pton(AF_INET, argv[1], &(sa.sin_addr));

    inet_ntop(AF_INET, &(sa.sin_addr), str, INET_ADDRSTRLEN);

    printf("%s\n", str);
    return 0;
}
```

Socket Address Structure(2)

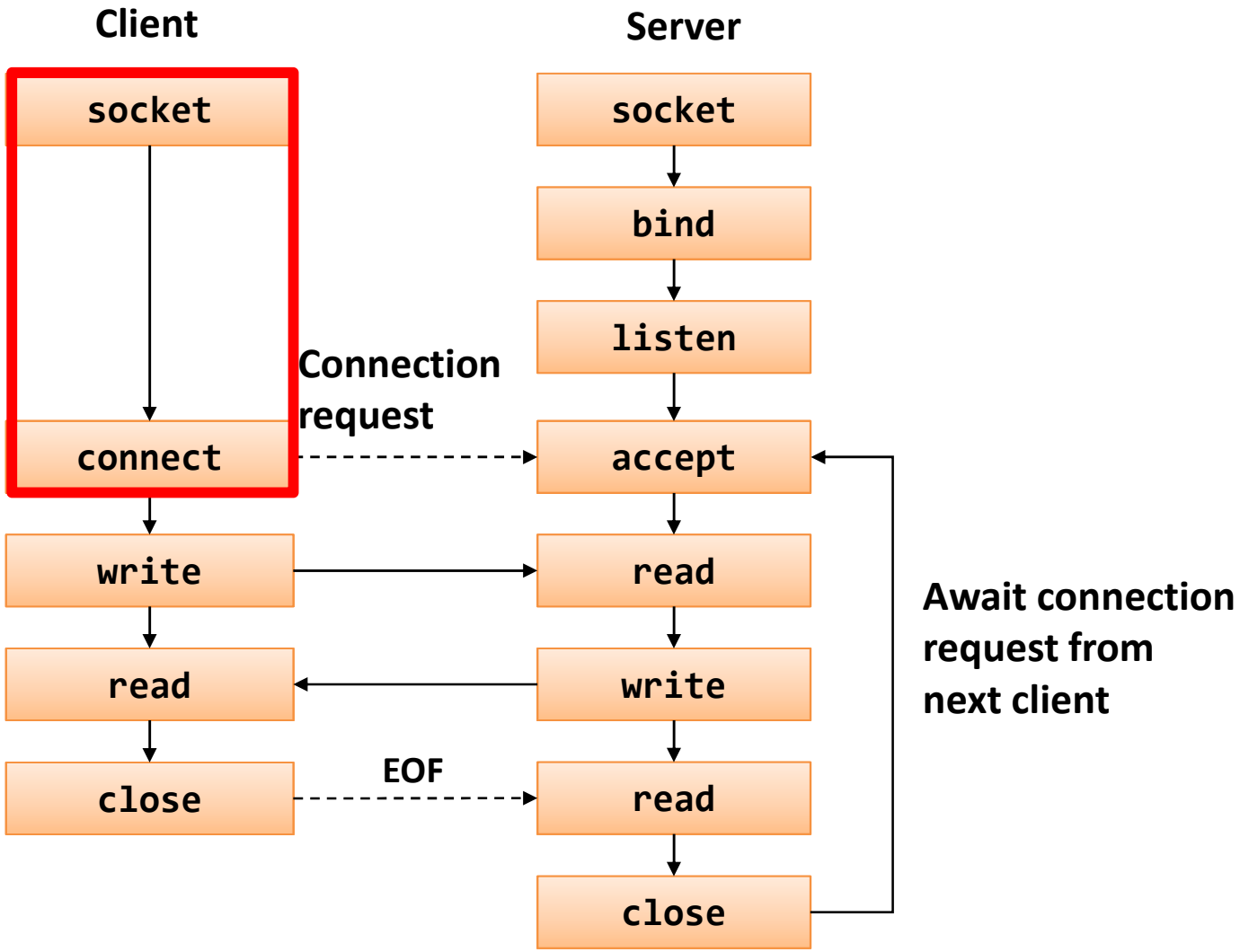
- **How to extract a socket address structure?**

```
const char *inet_ntop(int af, const char *src,
        char *dst, socklen_t size);
```

  - Returns pointer to dst, NULL on error
  - int af – address family (AF_INET, AF_INET6)
  - const char *src – address structure src
  - char *dst – buffer to copy the address
  - socklen_t size – buffer size

```c
int main(int argc, char **argv)
{
    struct sockaddr_in sa;
    char str[INET_ADDRSTRLEN];

    inet_pton(AF_INET, argv[1], &(sa.sin_addr));

    inet_ntop(AF_INET, &(sa.sin_addr), str, INET_ADDRSTRLEN);

    printf("%s\n", str);
    return 0;
}
```

TCP Sockets


Client
Server
socket
socket
bind
listen
Connection request
connect
accept
write
read
read
write
close
EOF
read
close
Await connection request from next client

socket()

- **int socket (int family, int type, int protocol)**
  - Specifying the type of communication
  - **socket()** creates a socket descriptor. -1 on error.
  - **family** specifies the protocol family.
    - **AF_UNIX**: Local Unix domain protocols
    - **AF_INET**: IPv4 Internet protocols
  - **type** specifies the communication semantics.
    - **SOCK_STREAM**: provides sequenced, reliable, two-way, connection-based byte streams
    - **SOCK_DGRAM**: supports datagrams (connectionless, unreliable messages of a fixed maximum length)
  - **protocol** specifies a particular protocol to be used with the socket.

connect()

- **int connect (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen)**
  - Used by a TCP client to establish a connection with a TCP server.
  - **servaddr** contains <IP address, port number> of the server.
  - The client does not have to call **bind()** before calling **connect().**
    - The kernel will choose both an ephemeral port and the source IP address if necessary.
  - Client process suspends (blocks) until the connection is created.

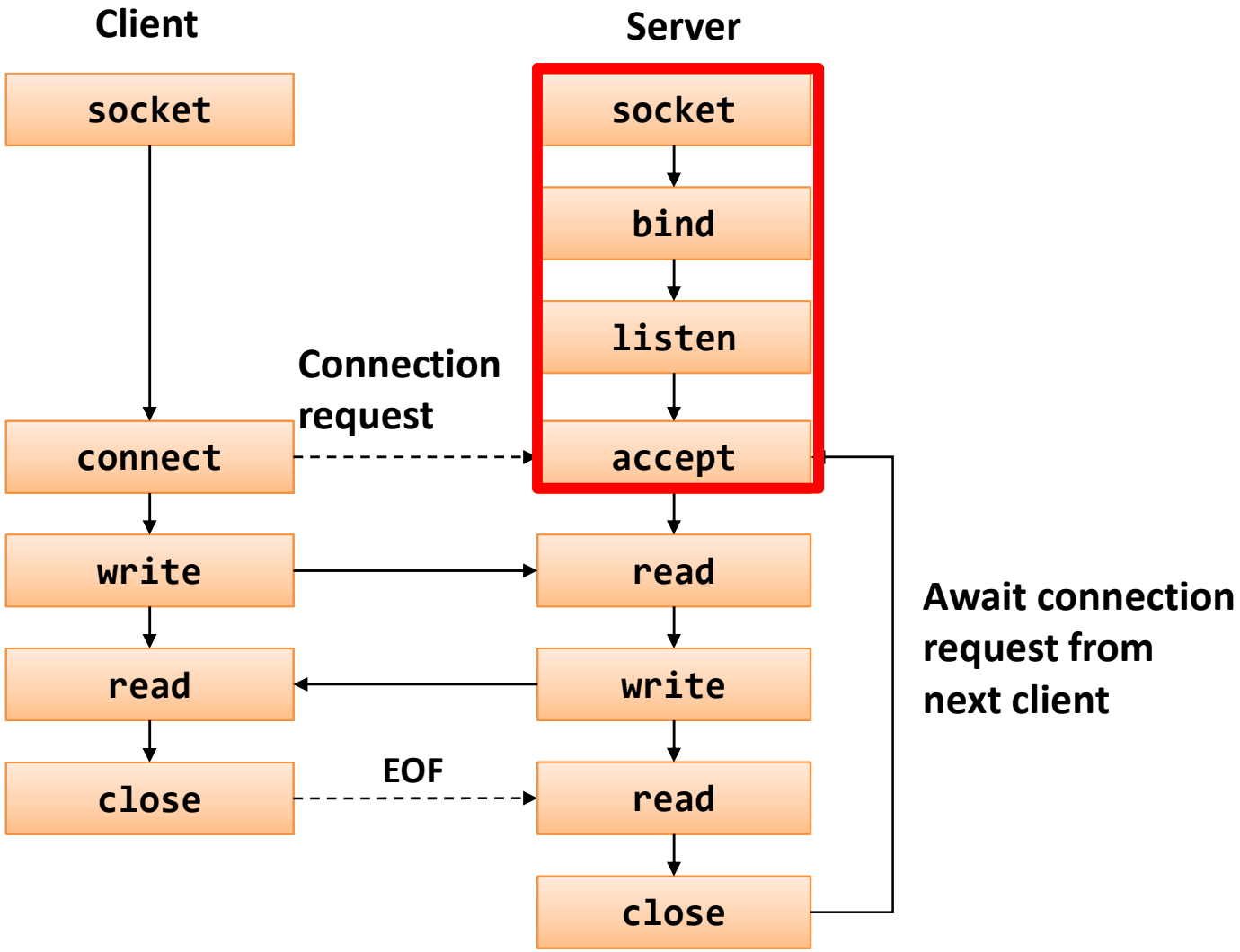
Echo Client (1)

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

#define MAXLINE 80

int main (int argc, char *argv[]) {
   int n, cfd;
   struct hostent *h;
   struct sockaddr_in saddr;
   char buf[MAXLINE];
   char *host = argv[1];
   int port = atoi(argv[2]);

   if ((cfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      printf("socket() failed.\n");
      exit(1);
   }
```

Echo Client (2)

```
    if ((h = gethostbyname(host)) == NULL) {
        printf("invalid hostname %s\n", host);
        exit(2);
    }
    bzero((char *)&saddr, sizeof(saddr));
    saddr.sin_family = AF_INET;
    bcopy((char *)h->h_addr, (char *)&saddr.sin_addr.s_addr, h->h_length);
    saddr.sin_port = htons(port);

    if (connect(cfd,(struct sockaddr *)&saddr,sizeof(saddr)) < 0) {
        printf("connect() failed.\n");
        exit(3);
    }
    while ((n = read(0, buf, MAXLINE)) > 0) {
        write(cfd, buf, n);
        n = read(cfd, buf, MAXLINE);
        write(1, buf, n);
    }
    close(cfd);
}
```

TCP Sockets


Client
Server
socket
socket
bind
listen
Connection request
connect
accept
write
read
read
write
EOF
close
read
close
Await connection request from next client

bind()

- **int bind (int sockfd, struct sockaddr *myaddr, socklen_t addrlen)**
  - **bind()** gives the socket **sockfd** the local address **myaddr**.
  - **myaddr** is **addrlen** bytes long.
  - Servers bind their well-known port when they start.
  - If a TCP server binds a specific IP address to its socket, this restricts the socket to receive incoming client connections destined only to that IP address.
  - Normally, a TCP client let the kernel choose an ephemeral port and a client IP address.

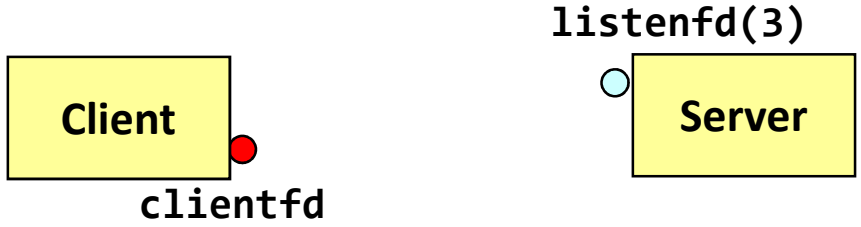
listen()

- **int listen (int sockfd, int backlog)**
  - **listen()** converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests.
    - When a socket is created, it is assumed to be an active socket, that is, a client socket that will issue a **connect().**
  - **backlog** specifies the maximum number of connections that the kernel should queue for this socket.
  - Historically, a backlog of 5 was used, as that was the maximum value supported by 4.2BSD.
    - Busy HTTP servers must specify a much larger backlog, and newer kernels must support larger values.

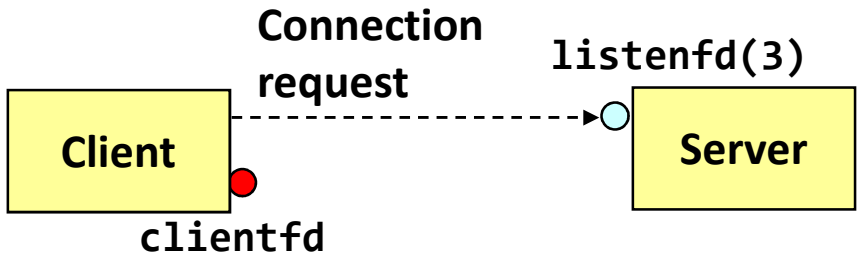
accept() (1)

- **int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen)**
  - **accept()** blocks waiting for a connection request.
  - **accept()** returns a connected descriptor with the same properties as the listening descriptor.
    - The kernel creates one connected socket for each client connection that is accepted.
    - Returns when the connection between client and server is created and ready for I/O transfers.
    - All I/O with the client will be done via the connected socket.
  - The **cliaddr** and **addrlen** arguments are used to return the address of the connected peer process (the client)

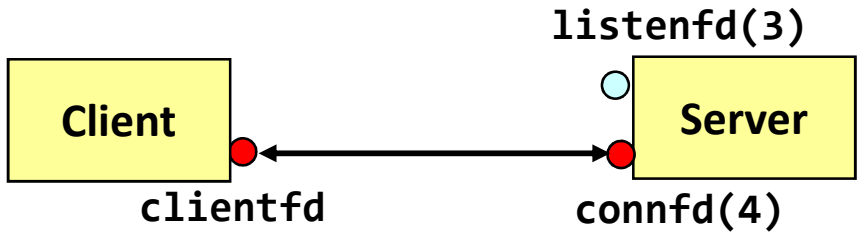
accept() (2)



1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`.



2. Client makes connection request by calling and blocking in `connect`.



3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`.

accept() (3)

- **Listening descriptor**
  - End point for client connection requests
  - Created once and exists for lifetime of the server
- **Connected descriptor**
  - End point of the connection between client and server
  - A new descriptor is created each time the server accepts a connection request from a client.
  - Exists only as long as it takes to service client.
- **Why the distinction?**
  - Allows for concurrent servers that can communicate over many client connections simultaneously.

Echo Server (1)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <arpa/inet.h>

#define MAXLINE 80

int main (int argc, char *argv[]) {
   int n, listenfd, connfd, caddrlen;
   struct hostent *h;
   struct sockaddr_in saddr, caddr;
   char buf[MAXLINE];
   int port = atoi(argv[1]);

   if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      printf("socket() failed.\n");
      exit(1);
   }
```

Echo Server (2)

```
bzero((char *)&saddr, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);
saddr.sin_port = htons(port);
if (bind(listenfd, (struct sockaddr *)&saddr,
         sizeof(saddr)) < 0) {
   printf("bind() failed.\n");
   exit(2);
}
if (listen(listenfd, 5) < 0) {
   printf("listen() failed.\n");
   exit(3);
}
while (1) {
   caddrlen = sizeof(caddr);
   if ((connfd = accept(listenfd, (struct sockaddr *)&caddr,
                        &caddrlen)) < 0) {
      printf ("accept() failed.\n");
      continue;
   }
```

Echo Server (3)

```
        h = gethostbyaddr((const char *)&caddr.sin_addr.s_addr,
            sizeof(caddr.sin_addr.s_addr), AF_INET);
        printf("server connected to %s (%s)\n",
            h->h_name,
            inet_ntoa(*(struct in_addr *)&caddr.sin_addr));

        // echo
        while ((n = read(connfd, buf, MAXLINE)) > 0) {
           printf ("got %d bytes from client.\n", n);
           write(connfd, buf, n);
        }

        printf("connection terminated.\n");
        close(connfd);
    }
}
```

Exercises

- **Make Connection oriented(TCP) DB!**

- **Client requests followings to server…**
  - GetValue: Request the number
  - PutValue: Put a word to server
  - GetRank: Find the rank for the word.

- **Server should manage the DB**