# Network IPC: Sockets

System Programming

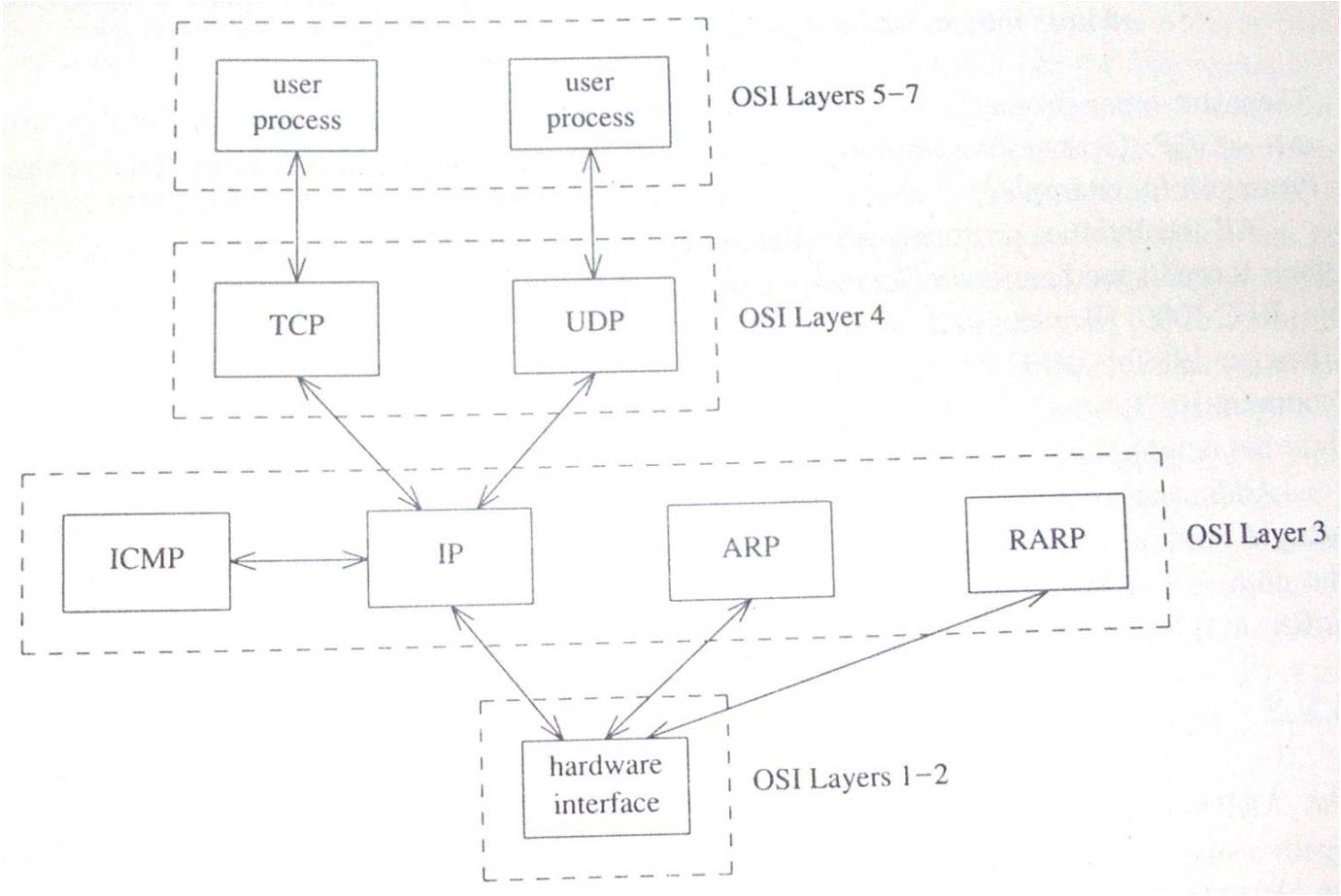
2019 여름 계절학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

# Introduction

- **Socket network IPC interface to support process communication on the same machine or on different machines**
  - Classical IPCs : pipes, FIFOs, message queues, semaphores, and shared memory on the same computer
- **The POSIX.1 socket API based on the 4.4BSD socket interface**
  - Originally introduced in 4.2BSD in the early 1980s.

# The Internet Protocol Suite



# Connection-Oriented vs. Connectionless

## ❑ Connectionless protocol (Datagram interface)

- A datagram is a self-contained message.
- Sending a datagram is analogous to mailing someone a letter.
- No logical connection between peers
  - Subject to out-of-order delivery and loss of packet
- UDP (User Datagram Protocol)

## ❑ Connection-oriented protocol (Stream interface)

- Like making a phone call.
- A point-to-point virtual connection between both ends of the call
- TCP (Transmission Control Protocol)

## Socket Descriptors

- **A socket is an abstraction of a communication endpoint.**
- **Socket descriptors to access sockets**
  - Implemented as file descriptors

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

- **Domain: AF_INET, AF_INET6, AF_UNIX, AF_UNSPEC**
- **Types: SOCK_DGRAM, SOCK_RAW, SOCK_SEQPACKET, SOCK_STREAM**
- **Protocol: zero value to indicate the default protocol for the given domain and socket type**
  - TCP for AF_INET and SOCK_STREAM
  - UDP for AF_INET and SOCK_DGRAM

# Socket Types

| Type | Description |
|---|---|
| SOCK_DGRAM | fixed-length, connectionless, unreliable messages |
| SOCK_RAW | datagram interface to IP (optional in POSIX.1) |
| SOCK_SEQPACKET | fixed-length, sequenced, reliable, connection-oriented messages |
| SOCK_STREAM | sequenced, reliable, bidirectional, connection-oriented byte streams |

# Address Formats

- **An address identifies a socket endpoint. Its format is specific to the particular domain, being cast to a generic `sockaddr` address structure to be passed to socket functions.**

```
struct sockaddr {
  sa_family_t  sa_family;   /* address family */
  char         sa_data[];   /* variable-length address */
  …
}
```

- **Internet address defined in `<netinet/in.h>`**

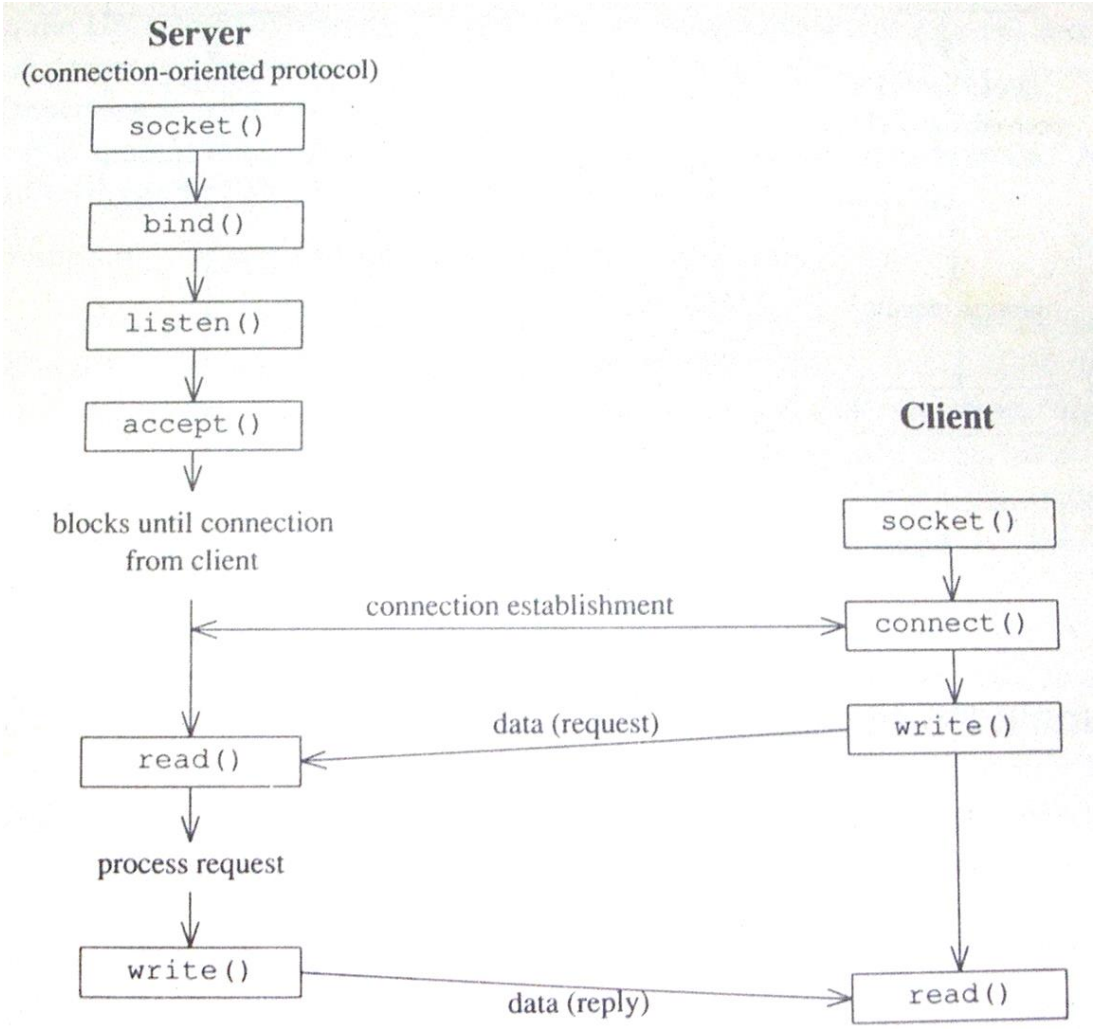
```
struct in_addr {
    in_addr_t   s_addr;       /* IPv4 address */
};

struct sockaddr_in {
    sa_family_t    sin_family;   /* address family */
    in_port_t      sin_port;     /* port number */
    struct in_addr sin_addr;     /* IPv4 address */
};
```

## Address Formats

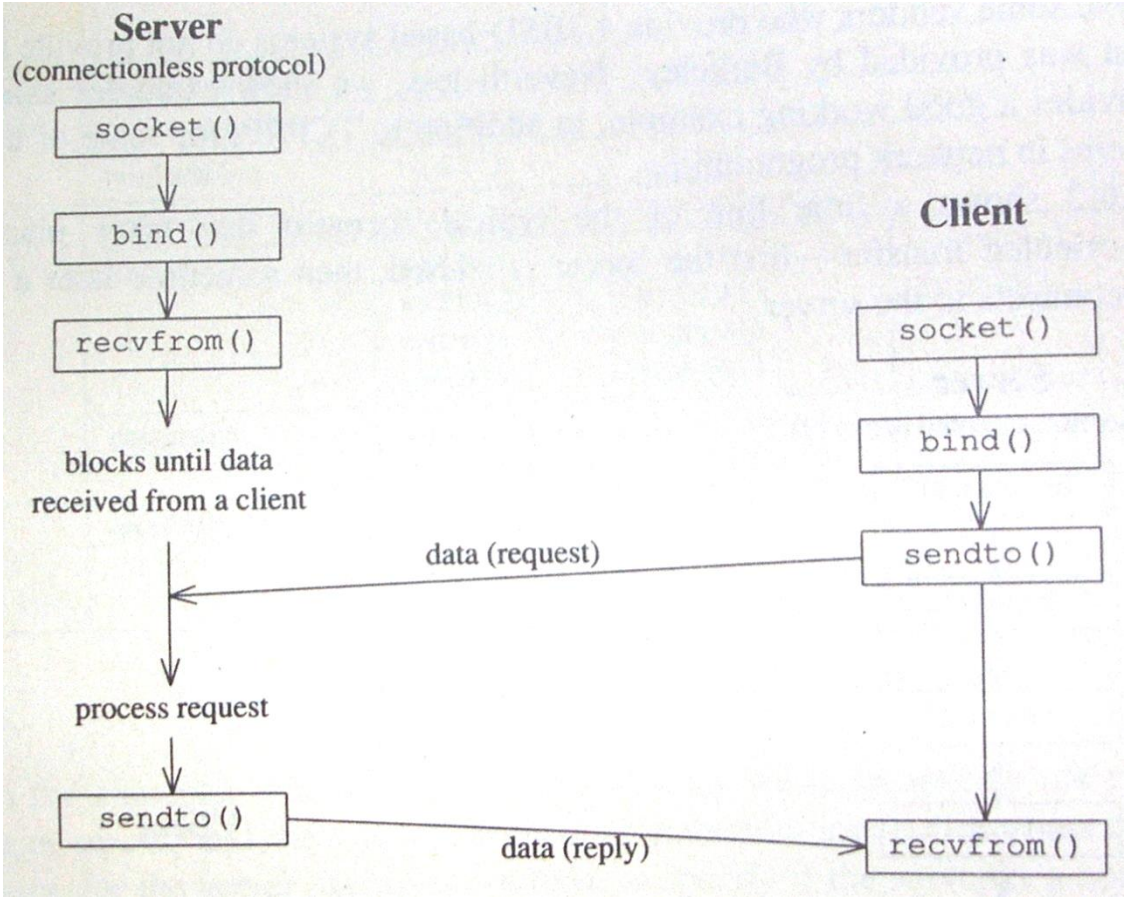
```
#include <arpa/inet.h>

const char *inet_ntop(int domain, const void *addr,
   char *str, socklen_t size);

int inet_pton(int domain, const char *str, void *addr);
```

- Conversion between the binary address format and the dotted-decimal notation
- Only AF_INET and AF_INET6 domains supported

# Socket System Calls for Connection-Oriented Protocol



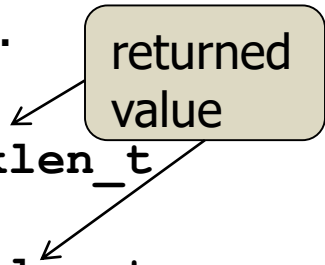
# Socket System Calls for Connectionless Protocol



# Associating Addresses with Sockets

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t
    len);
```

- To associate an address with a socket, to discover the address bound to a socket, and to find out the peer's address
- For a client's socket, we can let the system choose a default address for us, whereas a well-known address should be associated with the server's socket.
- The system will choose an address and bind it to our socket, if we call `connect` or `listen` without first binding an address to the socket.




```
int getsockname(int sockfd, struct sockaddr *addr, socklen_t
    *alenp);
int getpeername(int sockfd, struct sockaddr *addr, socklen_t
    *alenp);
```

## Connection Establishment

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

int connect(int sockfd, const struct sockaddr *addr,
    socklen_t len);
```

- It creates a connection to the specified server, *addr*. If *sockfd* is not bound to an address, a default address will be bound to the caller.
- Figure 16.9

## Connection Establishment

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

- The *backlog* argument provides a hint to the system of the number of outstanding connect requests that it should enqueue on behalf of the process.

## Connection Establishment

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t
   *len);
```

- The file descriptor returned is a socket descriptor that is connected to the client that called `connect`. This new socket descriptor has the same socket type and address family as *sockfd*.
- The original socket passed to `accept` is not associated with the connection, but instead remains available to receive additional connect requests.
- On return, `accept` will fill in the client's address in *addr*.

- Figure 16.10

- `read/write` can be used to communicate with a socket.

```
#include <sys/socket.h>
ssize_t send(int sockfd, const void *buf, size_t nbytes, int
   flags);
```

```
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buf, size_t nbytes, int
   flags, const struct sockaddr *destaddr, socklen_t destlen);
```

- With a connection-oriented socket, the destination address is ignored, as the destination is implied by the connection. (With a connectionless socket, we can't use `send` unless the destination address is first set by calling `connect`.)

# Data Transfer

```
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t nbytes, int
   flags);
```

```
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *buf, size_t len, int
   flags, struct sockaddr *addr, socklen_t *addrlen);
```


Q and A