523454

Computer Network Programming

Lab 1: Introduction to Sockets and Client/Server

Dr. Parin Sornlertlamvanich,

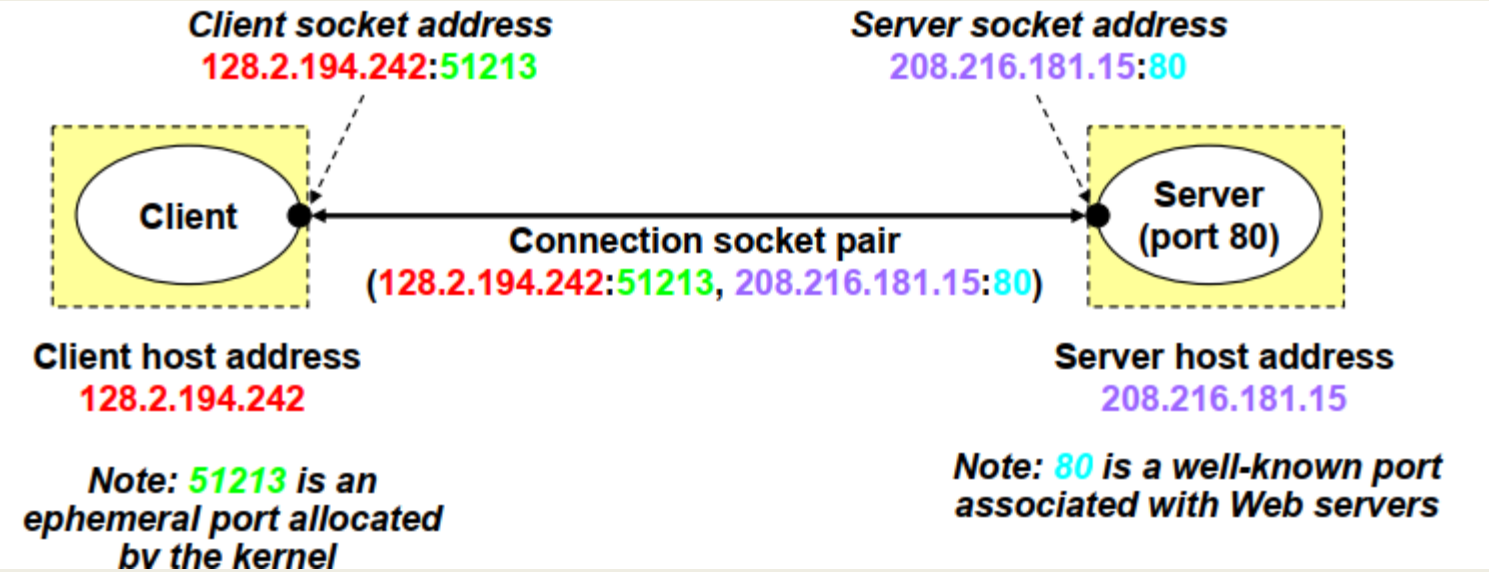
parin.s@sut.ac.th

Internet Connections

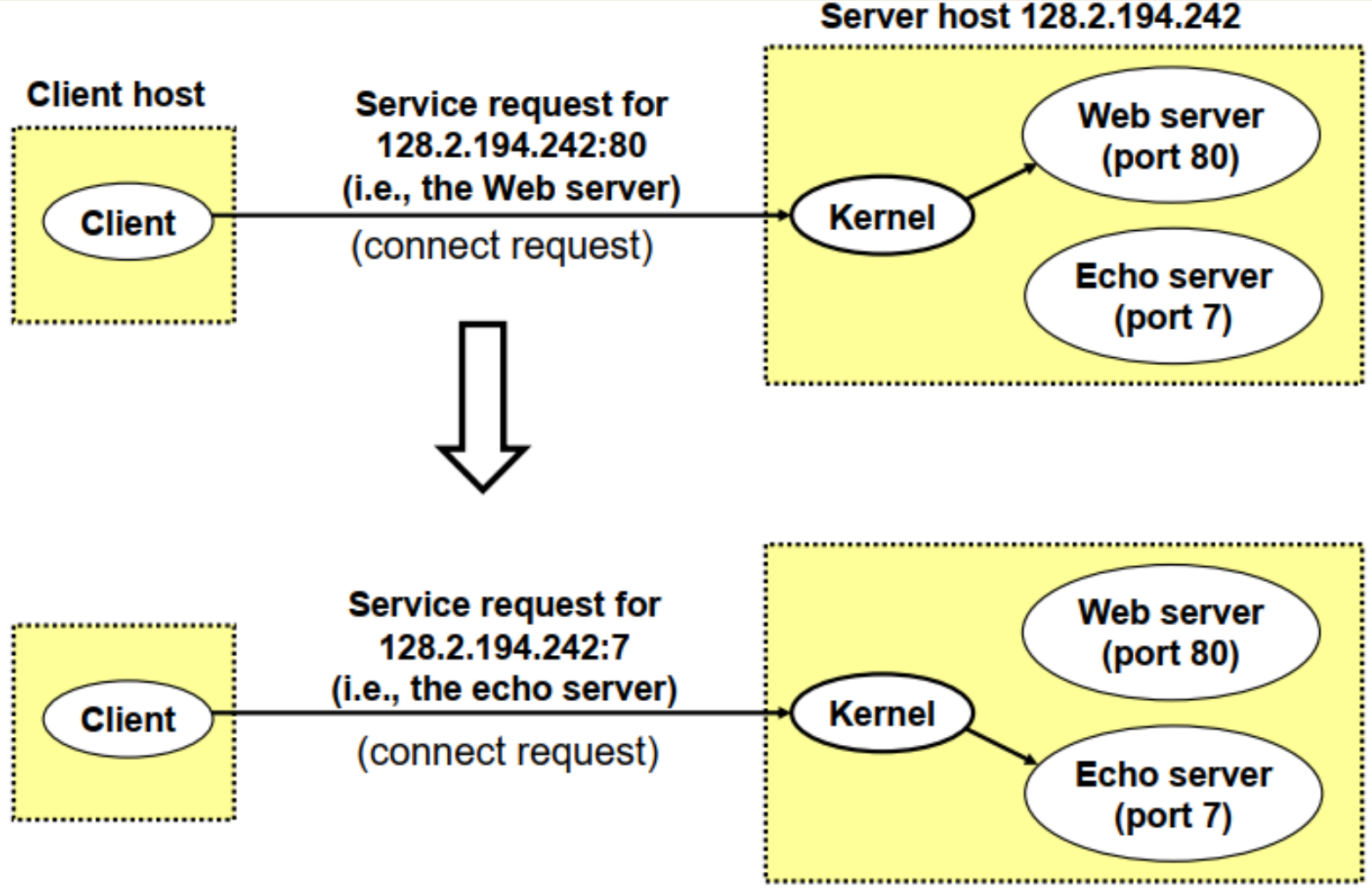
- Most clients and servers communicate by sending streams of bytes over connections
  - TCP
- A socket is an endpoint of a connection between two processes

Sockets

- A host might have many open connections, possibly held by different processes
- A port is a unique communication endpoint on a host, named by a 16-bit integer, and associated with a process



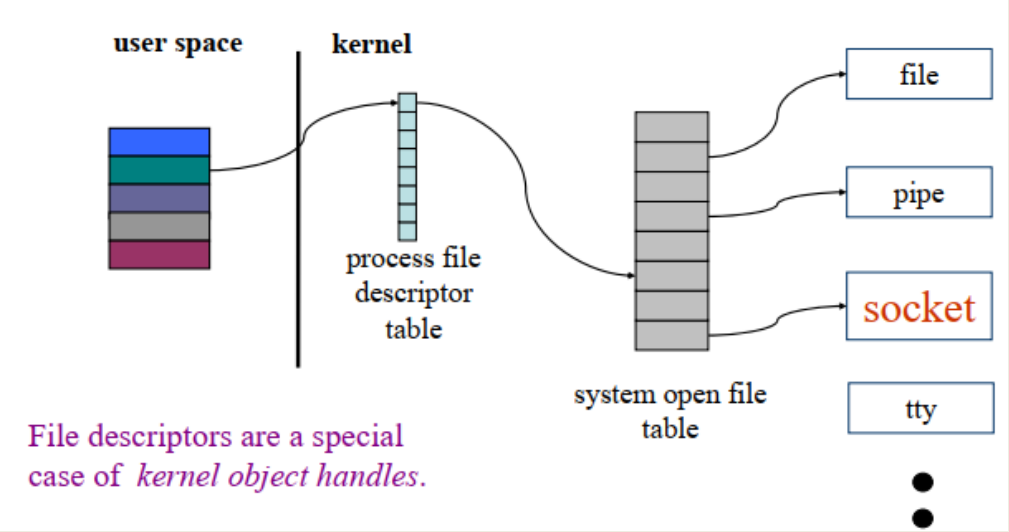
Using Ports to Identify Services



Datagrams and Streams

- **Communication over the Internet uses a selected transport-layer protocol (layer 4) built above the common IP packet protocol**
- **UDP = User Datagram Protocol (AF_INET/SOCK_DGRAM)**
  - Unreliable: messages may be lost or reordered
  - Connectionless: no notion or cost of ‘establishing a connection’
- **TCP = Transmission Control Protocol (AF_INET/SOCK_STREAM)**
  - Send/receive byte streams of arbitrary length (like a pipe)
  - All bytes delivered are correct and delivered in order
  - Connection setup/maintenance: other end is notified if one end closes or resets the connection, or if the connection breaks

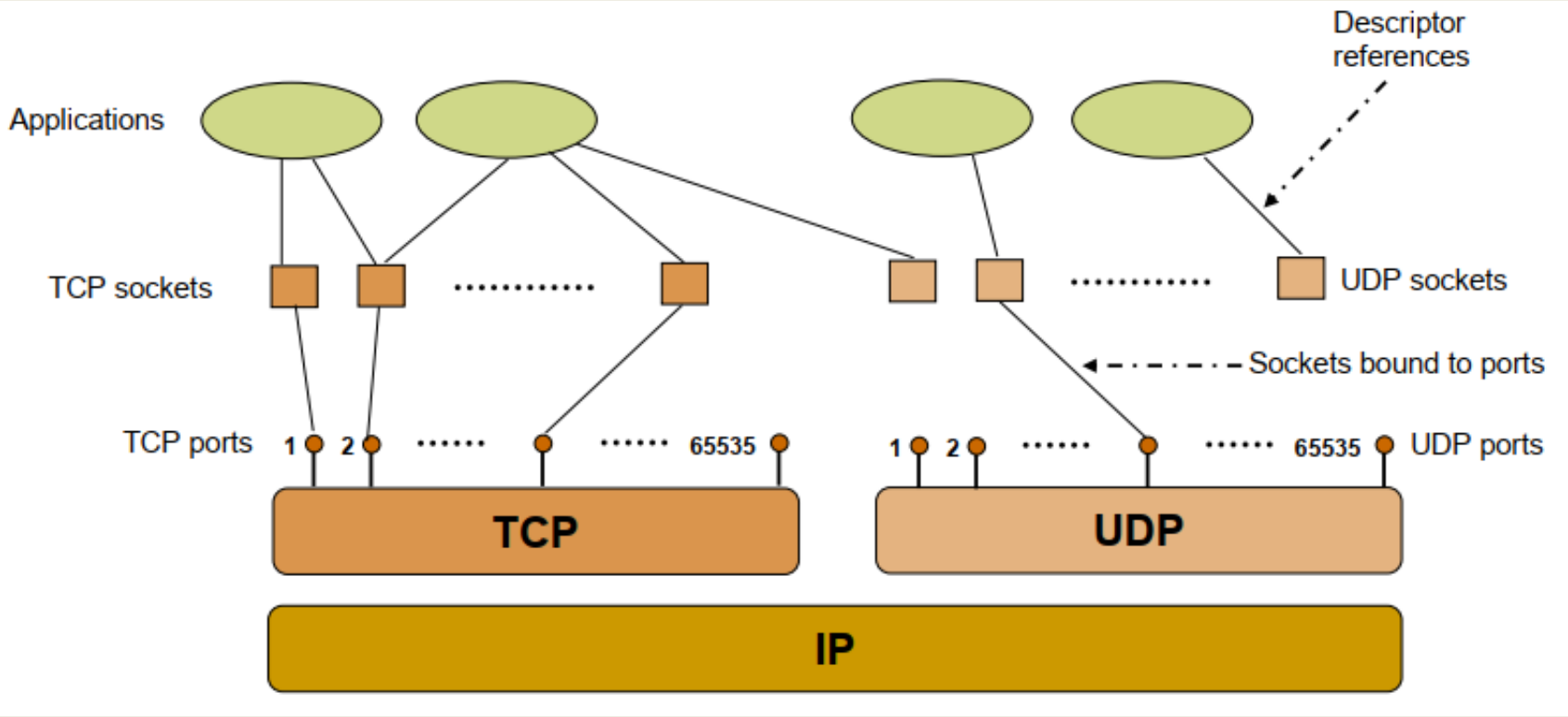
Creating a socket



- **int socket(int domain, int type, int protocol)**
  - domain = AF_INET, AF_UNIX
  - type = SOCK_STREAM, SOCK_DGRAM
- **Socket: An interface between an application process and transport layer**
  - The application process can send/receive messages to/from another application process (local or remote) via a socket
  - A socket is a file descriptor – an integer associated with an open file

Sockets

- file descriptors for network communication



Client-Server communication

- **Server**
  - passively waits for and responds to clients
  - Passive socket
- **Client**
  - initiates the communication
  - must know the address and the port of the server
  - Active socket

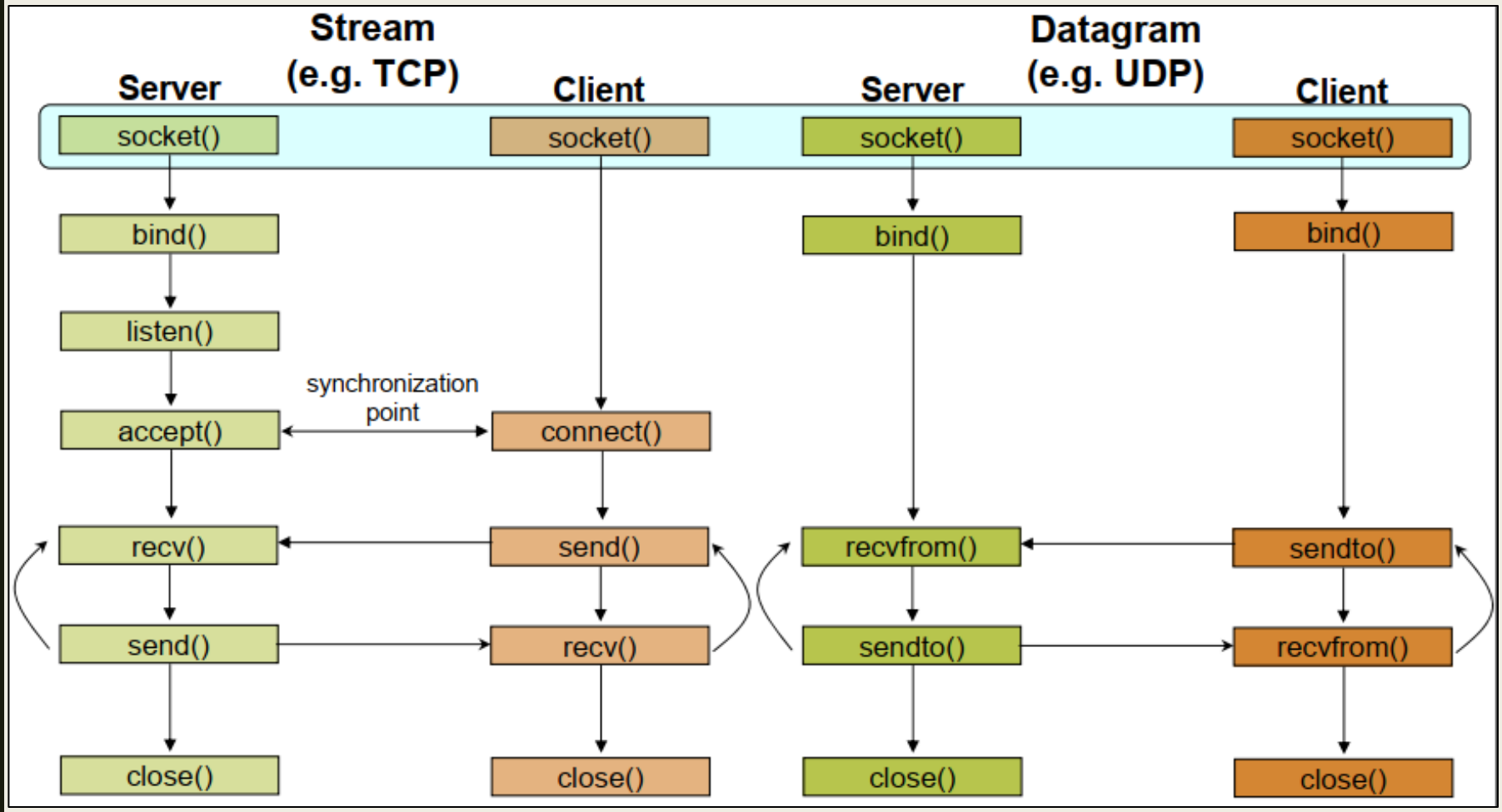
Server-Side Sockets

- **Bind socket to IP address/port**
  - int bind(int socket, struct sockaddr *addr, int addr_len)
- **Mark the socket as accepting connections**
  - int listen(int socket, int backlog)
- **“Passive open” accepts connection**
  - int accept(int socket, struct sockaddr *addr, int addr_len)
  - returns a new socket file descriptor to use for this accepted connection and -1 on error

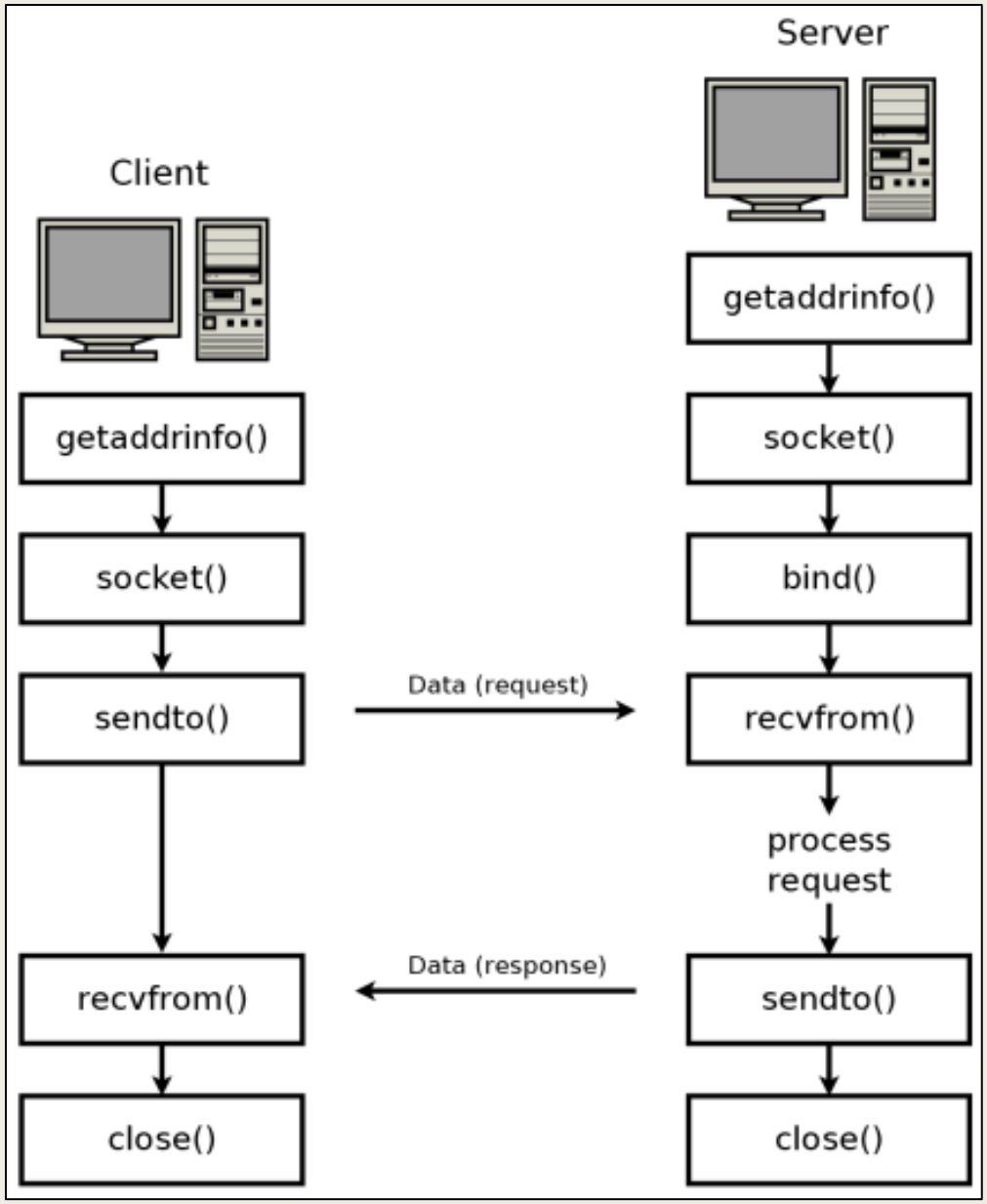
Client Socket

- **Active Open (on client)**
  - int connect(int socket, struct sockaddr *addr, int addr_len)

Client - Server Communication



UDP Socket


Server
Client
getaddrinfo()
socket()
bind()
recvfrom()
process request
sendto()
close()
getaddrinfo()
socket()
sendto()
recvfrom()
close()
Data (request)
Data (response)

socket()

- `int sockid = socket(family, type, protocol);`
  - **sockid**: socket descriptor, an integer (like a file-handle)
  - **family**: integer, communication domain, e.g.,
    - PF_INET, IPv4 protocols, Internet addresses (typically used)
    - PF_UNIX, Local communication, File addresses
  - **type**: communication type
    - SOCK_STREAM - reliable, 2-way, connection-based service
    - SOCK_DGRAM - unreliable, connectionless, messages of maximum length
  - **protocol**: specifies protocol
    - IPPROTO_TCP IPPROTO_UDP
    - usually set to 0 (i.e., use default protocol)
  - upon failure returns -1

☞ NOTE: socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!

close()

- When finished using a socket, the socket should be closed
- `status = close(sockid);`
  - **sockid**: the file descriptor (socket being closed)
  - **status**: 0 if successful, -1 if error
- Closing a socket
  - closes a connection (for stream socket)
  - frees up the port used by the socket

bind()

- associates and reserves a port for use by the socket

- `int status = bind(sockid, &addrport, size);`
  - **sockid**: integer, socket descriptor
  - **addrport**: struct sockaddr, the (IP) address and port of the machine
    - for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses any incoming interface
  - **size**: the size (in bytes) of the addrport structure
  - **status**: upon failure -1 is returned

```
int sockid;
struct sockaddr_in addrport;
sockid = socket(PF_INET, SOCK_STREAM, 0);

addrport.sin_family = AF_INET;
addrport.sin_port = htons(5100);
addrport.sin_addr.s_addr = htonl(INADDR_ANY);
if(bind(sockid, (struct sockaddr *) &addrport, sizeof(addrport))!= -1) {
    …}
```

Skipping the bind()

- **bind can be skipped for both types of sockets**
- **Datagram socket:**
  - if only sending, no need to bind. The OS finds a port each time the socket sends a packet
  - if receiving, need to bind
- **Stream socket:**
  - destination determined during connection setup
  - don't need to know port sending from (during connection setup, receiving end is informed of port)

Exchanging data with datagram socket

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

ssize_t recvfrom(int sockfd, void *buffer, size_t length, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

Returns number of bytes received, 0 on EOF, or −1 on error

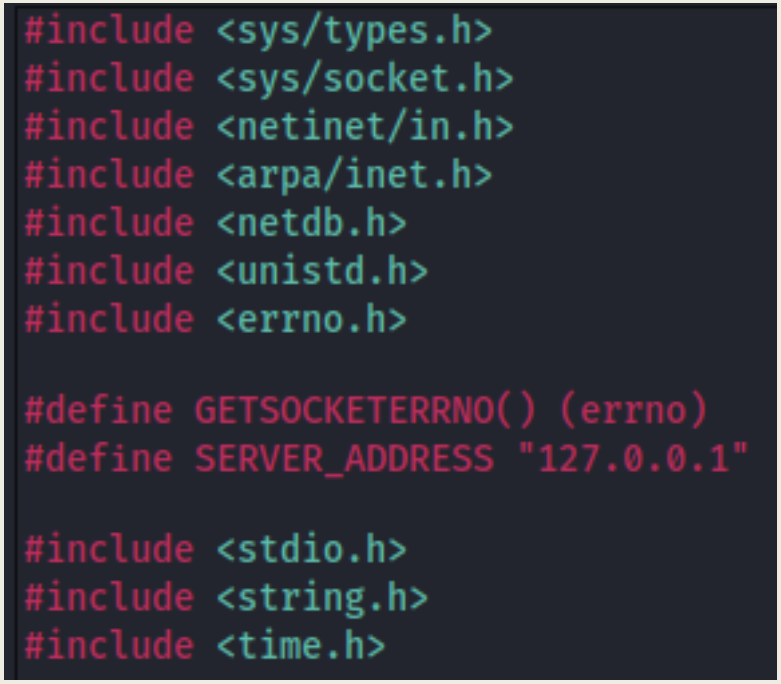
```
ssize_t sendto(int sockfd, const void *buffer, size_t length, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

Returns number of bytes sent, or −1 on error

LAB 1

lab1_server.c

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

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <unistd.h>

#include <errno.h>


#define GETSOCKETERRNO() (errno)

#define SERVER_ADDRESS "127.0.0.1"


#include <stdio.h>

#include <string.h>

#include <time.h>
```



lab1_server.c (cont.)

struct addrinfo hints;

struct addrinfo *bind_address;

struct sockaddr_storage client_address;

int socket_listen;

```c
int main() {

struct addrinfo hints;
struct addrinfo *bind_address;
struct sockaddr_storage client_address;
int socket_listen;
```

getaddrinfo()

```
memset(&hints, 0, sizeof(hints));

hints.ai_family = AF_INET;

hints.ai_socktype = SOCK_DGRAM;

hints.ai_flags = AI_PASSIVE;

getaddrinfo(0, “PORT", &hints, &bind_address);
```

```
printf("Configuring local address...\n");
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE;

getaddrinfo(0, "2020", &hints, &bind_address);
```

- The **getaddrinfo()** function has many uses, but for this, it generates an address that's suitable for bind()
- The advantage to using getaddrinfo() is that it is protocol-independent
  - IPv4 <-> IPv6 (AF_INET to AF_INET6)

getaddrinfo() Function

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

int getaddrinfo(const char *host, const char *service,
                const struct addrinfo *hints, struct addrinfo **result);
```

Returns 0 on success, or nonzero on error

- Returns a list of socket address structures
  - IP address and port number

```
struct addrinfo {
    int     ai_flags;          /* Input flags (AI_* constants) */
    int     ai_family;         /* Address family */
    int     ai_socktype;       /* Type: SOCK_STREAM, SOCK_DGRAM */
    int     ai_protocol;       /* Socket protocol */
    size_t ai_addrlen;         /* Size of structure pointed to by ai_addr */
    char  *ai_canonname;       /* Canonical name of host */
    struct sockaddr *ai_addr;  /* Pointer to socket address structure */
    struct addrinfo *ai_next;  /* Next structure in linked list */
};
```

Creating socket

```
printf("Creating socket ... \n");
socket_listen = socket(bind_address→ai_family,
        bind_address→ai_socktype, bind_address→ai_protocol);
if (socket_listen == -1) {
    fprintf(stderr, "socket() failed. (%d)\n", GETSOCKETERRNO());
    return 1;
}
```

socket_listen = socket(bind_address->ai_family,

bind_address->ai_socktype, bind_address->ai_protocol);

- After we have the local address information
  - To create the socket
  - Using our address information from getaddrinfo()

Binding the socket to local address

```
printf("Binding socket to local address ... \n");
if (bind(socket_listen,
            bind_address->ai_addr, bind_address->ai_addrlen) == -1) {
    fprintf(stderr, "bind() failed. (%d)\n", GETSOCKETERRNO());
    return 1;
}
freeaddrinfo(bind_address);
```

- bind(socket_listen, bind_address->ai_addr, bind_address->ai_addrlen)
- After we have bound to bind_address, we can call the freeaddrinfo() function to release the address memory

recvfrom()

```
socklen_t client_len = sizeof(client_address);
char read[1024];
int bytes_received = recvfrom(socket_listen,
        read, 1024,
        0,
        (struct sockaddr*) &client_address, &client_len);

printf("Received (%d bytes): %.*s\n",
        bytes_received, bytes_received, read);
```

int bytes_received = recvfrom(socket_listen, read, 1024, 0,

(struct sockaddr) &client_address, &client_len);*

- Once the local address is bound, it can simply start to receive data
- Recvfrom() returns the sender's address, as well as the received data
- close(socket_listen);

lab1_client.c

int main() {

struct addrinfo hints;

struct addrinfo *server_addr;

int socket_peer;

const char *message = “YOUR NAME";

```c
int main() {
struct addrinfo hints;
struct addrinfo *server_addr;
int socket_peer;
const char *message = "Parin Sornlertlamvanich";
```

getaddrinfo()

```c
printf("Configuring remote address ... \n");
memset(&hints, 0, sizeof(hints));
hints.ai_socktype = SOCK_DGRAM;

if (getaddrinfo(SERVER_ADDRESS, "2020", &hints, &server_addr)) {
    fprintf(stderr, "getaddrinfo() failed. (%d)\n", GETSOCKETERRNO());
    return 1;
}
```

Creating socket and Sending data

```
printf("Creating socket...\n");
socket_peer = socket(server_addr->ai_family,
        server_addr->ai_socktype, server_addr->ai_protocol);
if (socket_peer == -1) {
    fprintf(stderr, "socket() failed. (%d)\n", GETSOCKETERRNO());
    return 1;
}


printf("Sending: %s\n", message);
int bytes_sent = sendto(socket_peer,
        message, strlen(message),
        0,
        server_addr->ai_addr, server_addr->ai_addrlen);
printf("Sent %d bytes.\n", bytes_sent);

freeaddrinfo(server_addr);
close(socket_peer);
```

Checkpoint

- **Server**
  - เปิด Port ด้วยเลข4ตัวสุดท้ายรหัสนักศึกษา
  - เมื่อได้รับชื่อจาก client ให้ print ชื่อออกมา
  - และให้ส่ง “Hello from server” กลับไป
  - ปิดการเชื่อมต่อพร้อมโชว์ Finish
- **Client**
  - ส่งชื่อและนามสกุลตัวเองไปที่ Port ที่เปิด
  - รอรับ “Hello from server”
  - เมื่อได้รับทำการปิดการเชื่อมต่อพร้อมโชว์ Finish

Checkpoint

```
┌──(kali㉿kali)-[~/lab_netPro/my_lab/lab1]
└─$ ./client_checkpoint
Configuring remote address ...
Creating socket ...
Sending: Parin Sornlertlamvanich
Sent 23 bytes.
═══════════════════════════════
Received (17 bytes): Hello from server
Finished.
```

Client

```
┌──(kali㉿kali)-[~/lab_netPro/my_lab/lab1]
└─$ ./server_checkpoint
Configuring local address ...
Creating socket ...
Binding socket to local address ...
Received (23 bytes): Parin Sornlertlamvanich
═══════════════════════════════
Hello message sent.
Finished.
```

Server