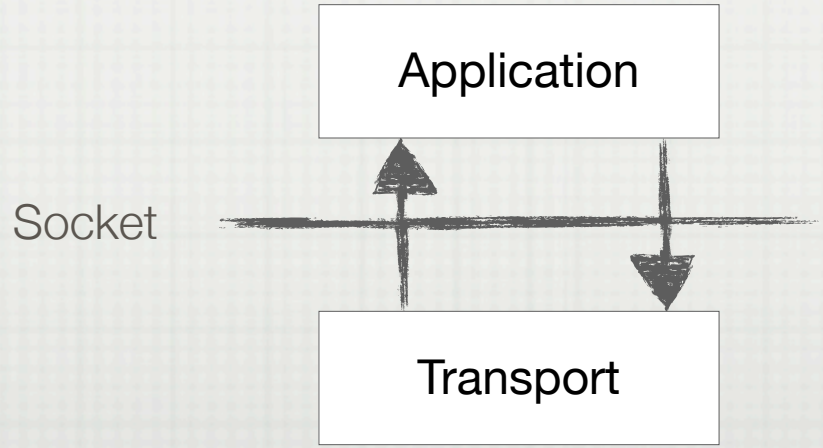
# SOCKET PROGRAMMING

WEI-TE WONG

# WHAT IS SOCKET?

# INTRODUCTION

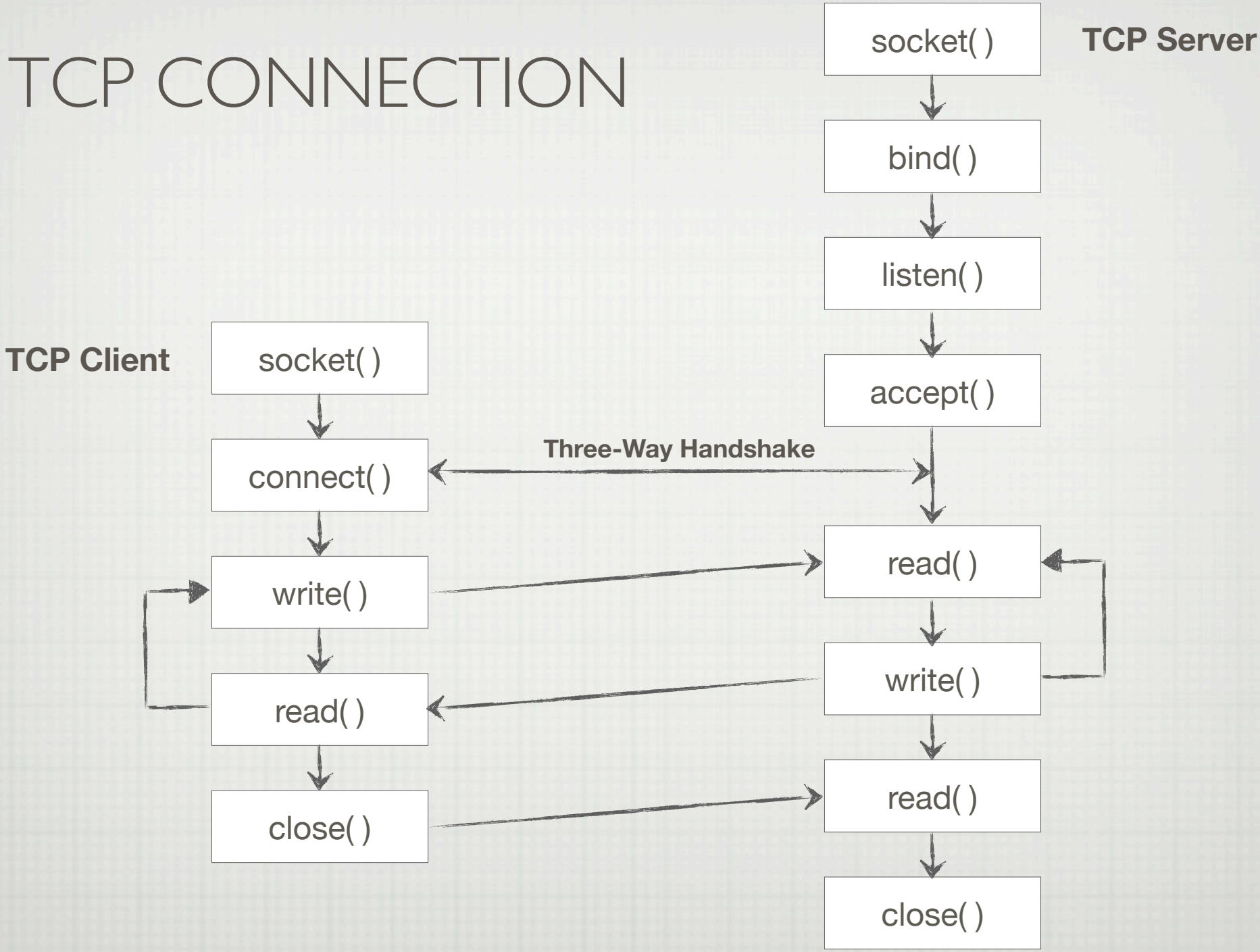
- Socket is the API for the TCP/IP protocol stack
- Provides communication between the Application Layer and the Transport Layer



# PROGRAMMER'S VIEW

- Socket is a file descriptor
- Socket allows application to ...
  - Send data to the network
  - Receive data transmitted from other hosts

# TCP CONNECTION



# USEFUL FUNCTIONS

CONNECTION ESTABLISHMENT

# SOCKET()

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

***Return: file descriptor on OK; -1, otherwise***

- *domain* indicates whether IPv4 or IPv6 is used
  - IPv4: AF_INET
- *type* indicates communication type
  - TCP: SOCK_STREAM
  - UDP: SOCK_DGRAM
- *protocol* is defined in /etc/protocols, usually set to 0

# BIND()

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

int bind(int sockfd, struct sockaddr *addr, socklen_t len)
```

***Return: 0 on OK; -1, otherwise***

- *sockfd* specifies the socket file descriptor
- *addr* specifies the address to be associated with *sockfd*
- *len* specifies the size of *addr*

# LISTEN()

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

***Return: 0 on OK; -1, otherwise***

- *sockfd* specifies the socket file descriptor
- *backlog* specifies the number of users allowed in queue
  - Linux typically adds 3 to the number specified
  - Other OS have different implementations

# ACCEPT()

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

***Return: file descriptor on OK; -1, otherwise***

- Blocking until a user *connect( )* call is received
- *sockfd* specifies the socket file descriptor
- *addr* specifies the peer address
- *len* is a value-result argument
  - Must be initialized to the size of *addr*
  - On return, set to the real size of the peer address

# CONNECT()

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

***Return: 0 on OK; -1, otherwise***

- *sockfd* specifies the socket file descriptor
- *addr* specifies the address to be associated with sockfd
- *len* specifies the size of *addr*

# CLOSE()

```
#include <unistd.h>

int close(int fd)
```

***Return: 0 on OK; -1, otherwise***

- *fd* specifies the socket file descriptor to be closed

# USEFUL FUNCTIONS

COMMUNICATION I/O

# READ()

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```

***Return: number of bytes read on OK; -1, otherwise***

- *fd* specifies the socket file descriptor to read data from
- *buf* specifies the buffer to contain the received data
- *count* specifies the size of *buf*

# WRITE()

```
#include <unistd.h>

ssize_t write(int fd, void *buf, size_t count)
```

***Return: number of bytes sent on OK; -1, otherwise***

- *fd* specifies the socket file descriptor to send data to
- *buf* specifies the buffer to contain the data to transmit
- *count* specifies the size of *buf*

# USEFUL FUNCTIONS

CONVERSION

# BYTE ORDERING

- Addresses and port numbers are stored as integers
  - Different machines implements different endian
  - They may communicate with each other on the network
- IP addresses are usually hard to remember
  - We need to translate IP address to hostname

# CONVERSION (1/2)

- Converting IP addresses and port numbers
  - *htonl( )*: for IP addresses (host -> network)
  - *ntohl( )*: for IP addresses (network -> host)
  - *htons( )*: for port number (host -> network)
  - *ntohs*( ): for port number (network -> host)

# CONVERSION (2/2)

- Converting IP addresses between network and human-readable format
  - *inet_ntop( )*: network -> presentation
  - *inet_pton( )*: presentation -> network

# GETHOSTBYNAME()

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name)
```

***Return: host environment on OK; NULL, otherwise***

- Translate a hostname to IP address
- *name* specifies the hostname

# EXAMPLE

HELLO WORLD SERVER/CLIENT

# HELLO WORLD SERVER (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/types.h>

int main(int argc, char *argv[])
{
   char buffer[50];
   int listenfd, connfd;
   socklen_t length;
   struct sockaddr_in serverAddress, clientAddress;
```

# HELLO WORLD SERVER (2/2)

```
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&serverAddress, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(5000);
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(listenfd, (struct sockaddr *) &serverAddress, sizeof(serverAddress));
    listen(listenfd, 1);
    while(1) {
        length = sizeof(clientAddress);
        connfd = accept(listenfd, (struct sockaddr *) &clientAddress, &length);
        write(connfd, “Hello World!\n”, 13);
        close(connfd);
    }
}
```

# HELLO WORLD CLIENT (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

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

int main(int argc, char *argv[])
{
   char buffer[50];
   int sockfd;
   struct sockaddr_in serverAddress;
```

# HELLO WORLD CLIENT (2/2)

```
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&serverAddress, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(5000);
    inet_pton(AF_INET, argv[1], &serverAddress.sin_addr);

    connect(sockfd, (struct sockaddr *) &serverAddress, sizeof(serverAddress));

    bzero(buffer, sizeof(buffer));
    read(sockfd, buffer, sizeof(buffer));
    printf("%s", buffer);
    close(sockfd);
}
```