Computer Network I

Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

(riccardo.caccavale@unina.it)

UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

DIETI

UNINA

Application Layer

Creating Network Applications

- So far, we have looked at several network applications and protocols, we will now see **how network applications can be created**.
- Most network applications include a **client-side** program and a **server-side** program that communicate through the network.
- When these two programs are executed, a client process and a server process are created, and these processes communicate with each other by reading from, and writing to, **sockets**.
- When creating a network application, the developer's main task is therefore to write the code for both the client and server programs.

Application Layer

Creating Network Applications

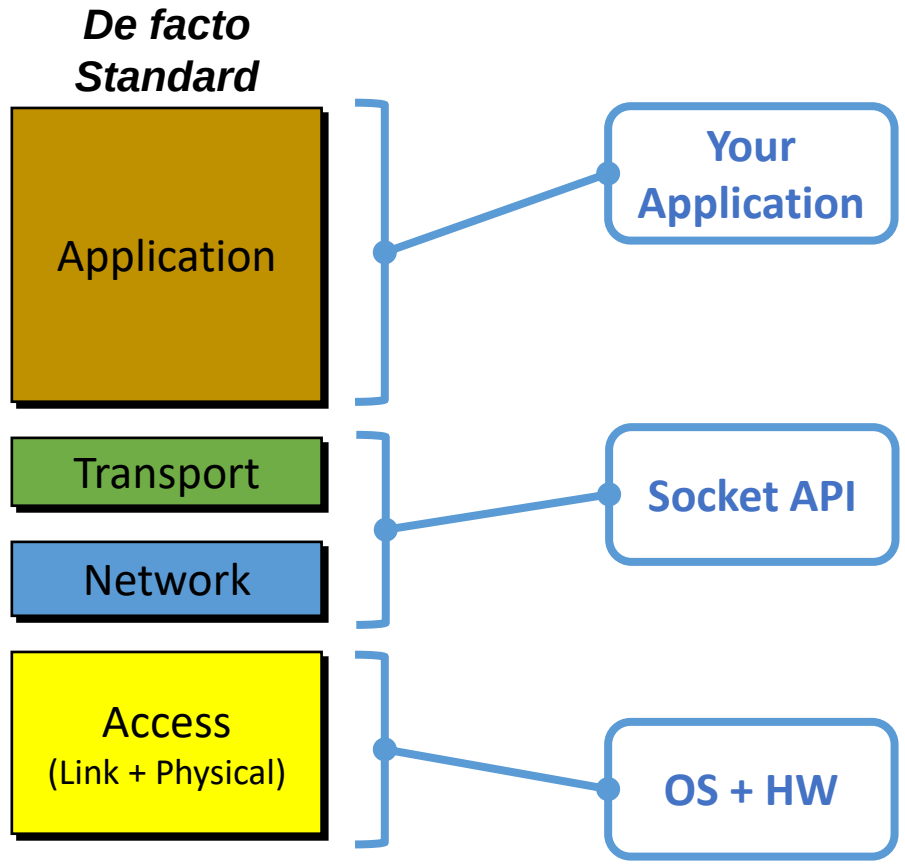
- There are two types of network applications:
  - Applications relying on a **standard protocol**. There are several “open” protocols whose rules are known. In this case, the client and server programs must conform to the rules to be compatible with the protocol.
  - Applications relying on a **proprietary protocol**. In this case the client and server programs employ a custom protocol (not been openly published in an RFC or elsewhere). In this case, a single developer (or development team) creates both the client and server programs.

- There are two transport protocols that can be used:
  - **TCP** (Transmission Control Protocol), which is **connection oriented** and provides a reliable byte-stream channel through which data flows between two end systems.
  - **UDP** (User Datagram Protocol), which is **connectionless** and sends independent packets of data from one end system to the other, without any guarantees about delivery.

Application Layer

Creating Network Applications

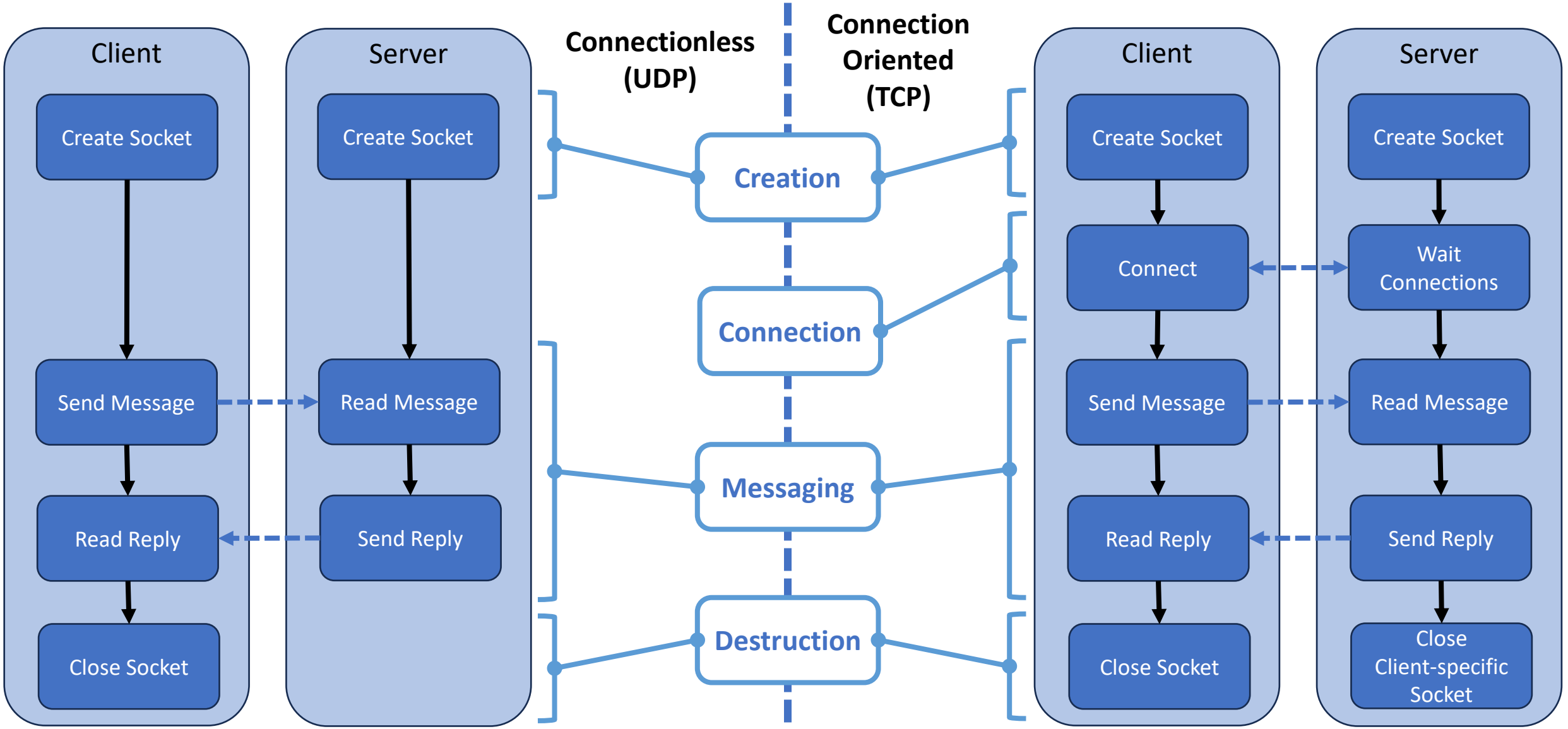
- **Sockets** are a central element for the creation of network applications.
- They typically manages (black-box) the transport (UDP/TCP) and the network (IP) layers functionalities.
- Note that there are several applications (middleware) that may **wrap sockets** simplifying their creation and providing more complex functionalities.
- However, **basic API are still widely used**.



Where layers are implemented

Application Layer

Connection-oriented vs. Connectionless



Application Layer
UDP and TCP

- UDP and TCP sockets implement the previous procedure to allow connectionless and connection-oriented communication, respectively.

- The basic APIs providing UDP and TCP sockets are typically available in almost **all programming languages and operating systems**:
  - C, C++, C#, Java, Perl, Python, Matlab, etc.

- The underlying implementation and the usage can be more or less different depending on the configuration of the machines.

- In Unix domain we use **Berkeley sockets** (aka BSD or POSIX sockets) for both TCP and UDP connections. These are native C APIs.

Application Layer

Sockets: Definitions

- Internet-domain sockets in C rely on some structure to define the addresses and ports of the sending/receiving hosts:

```
#include <netinet/in.h>
#include <arpa/inet.h>              // for inet_addr()
#include <sys/types.h>              // some custom types are defined here


struct sockaddr_in {
   short            sin_family;    // family of the address, typically set to AF_INET (IPv4)
   unsigned short   sin_port;      // port number, e.g. htons(3490)
   struct in_addr   sin_addr;      // see struct in_addr below
   char             sin_zero[8];   // typically zeros, now this structure has same size as sockaddr and can be casted to it
};

struct in_addr {
   unsigned long s_addr;  // IP address, can be loaded with inet_addr() or set to INADDR_ANY for localhost
};
```

Application Layer

Sockets: Creation

- Sockets are created in C by using the socket() function:

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

int sockfd = socket(int domain, int type, int protocol);
```

Where:

- domain: specifies the family as in previous structure (AF_INET).
- type: specifies the type of socket to be created.
  - SOCK_STREAM -> TCP socket
  - SOCK_DGRAM -> UDP socket
- protocol: Specifies a particular protocol to be used within the socket.
  - This is often 0, so no specific protocol is used.
- sockfd: is the file descriptor that identifies the newly created socket.
  - Notice that sockets also follow the Unix philosophy that “everything is a file” where I/O sources are treated as files (and associated to a file descriptor).

Application Layer

Sockets: Binding

- We can associate a socket to local address and port by using the bind() function:

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

int val = bind(int socket, const struct sockaddr *address, socklen_t address_len);
```

Where:

- socket: is the file descriptor of the socket we want to bind.
- address: pointer to a sockaddr structure specifying on which port and address to bind.
  - address.sin_addr often set to INADDR_ANY so connections from all addresses are listened (multiple IPs).
- address_len: specifies the length of the sockaddr structure pointed to by the address argument (we can use sizeof() to get it).
- val: return value which is 0 on success, and -1 otherwise.

- Notice that this function **is used on servers** as we must bind the socket to a specific port but is **optional on clients** (the OS assigns a free port automatically).

Application Layer

Sockets: Send and Receive (UDP)

- Processes of sending or receiving messages in UDP are performed by the sendto() and recvfrom() functions:

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

int ob = sendto(int osock, const void *obuf, size_t olen, int oflags, const struct sockaddr *oaddr, socklen_t oaddr_len);
```

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

int ib = recvfrom(int isock, void *ibuf, size_t ilen, int iflags, struct sockaddr *iaddr, socklen_t *iaddr_len);
```

Where:

- osock/isock: are the file descriptors on which to send/receive a message.
- obuf/ibuf: are pointers to the buffers containing a message to send/receive.
- olen/ilen: are lengths of the messages in bytes.
- oflags/iflags: specifies flags (typical values are 0 or MSG_WAITALL to wait until all olen/ilen bytes are sent/received).
- oaddr/iaddr: pointers to sockaddr structures containing the receiving/sending address.
- oaddr_len/iaddr_len: length in bytes of the sockaddr structure.
- ob/ib: number of bytes that are actually sent/received.

Application Layer

Sockets: Closing

- Sockets can be closed by using the close() function:

```
#include <unistd.h>

int val = close(int socket);
```

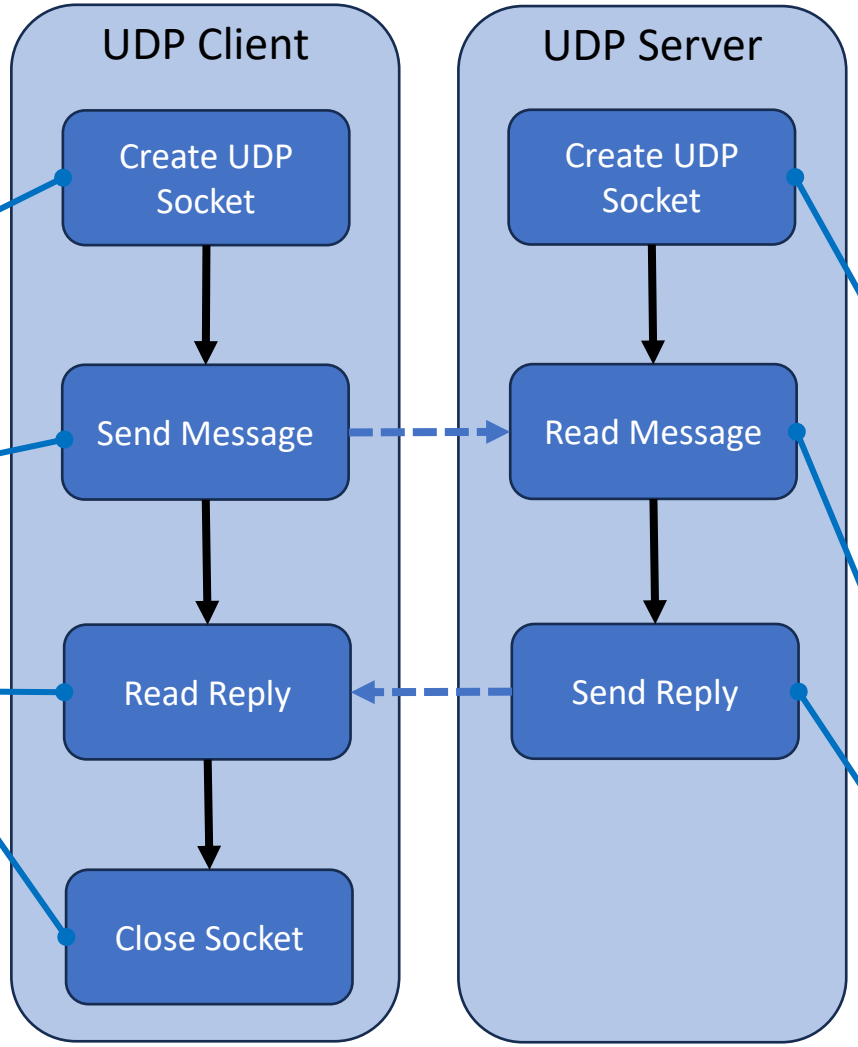
Where:

- socket: is the file descriptor of the socket to close
- val: return value which is 0 on success, and -1 otherwise.

Application Layer

UDP Socket Programming (C/C++)

```
// client-side (includes omitted)

int main() {
   int sockfd;
   char buffer[1024];
   const char *hello = "Hello from client";
   struct sockaddr_in servaddr;

   if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
      perror("socket creation failed");
      exit(EXIT_FAILURE);
   }

   memset(&servaddr, 0, sizeof(servaddr));

   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(8080);
   servaddr.sin_addr.s_addr = INADDR_ANY;

   int n;
   socklen_t len;

   sendto(sockfd, (const char *)hello, strlen(hello),
      MSG_CONFIRM, (const struct sockaddr *) &servaddr, sizeof(servaddr));
   std::cout<<"Hello message sent."<<std::endl;

   n = recvfrom(sockfd, (char *)buffer, 1024,
            MSG_WAITALL, (struct sockaddr *) &servaddr, &len);
   buffer[n] = '\0';
   std::cout<<"Received \""<<buffer<<"\""<<std::endl;

   close(sockfd);

   return 0;
}
```



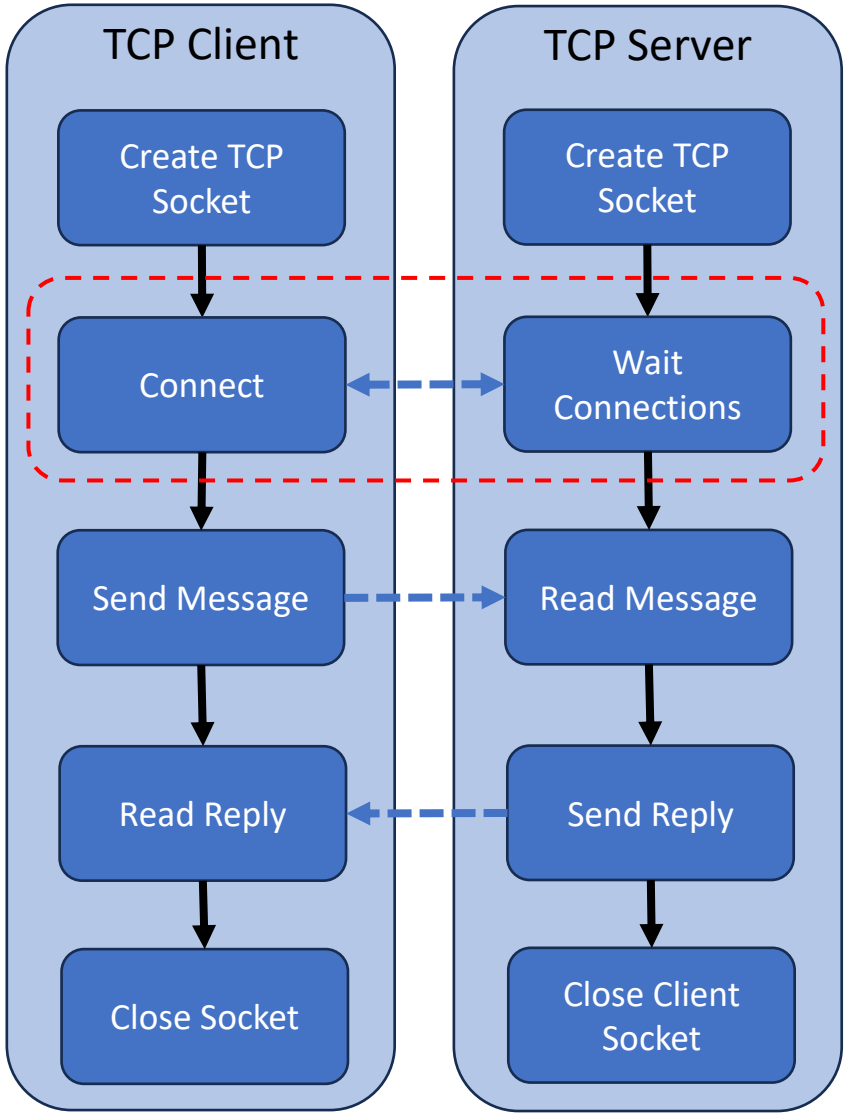

```
//server-side (includes omitted)

int main() {
   int sockfd;
   char buffer[1024];
   const char *hello = "Hello from server";
   struct sockaddr_in servaddr, cliaddr;

   if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
      perror("socket creation failed");
      exit(EXIT_FAILURE);
   }

   memset(&servaddr, 0, sizeof(servaddr));
   memset(&cliaddr, 0, sizeof(cliaddr));

   servaddr.sin_family    = AF_INET;
   servaddr.sin_addr.s_addr = INADDR_ANY;
   servaddr.sin_port = htons(8080);

   if ( bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0 ) {
      perror("bind failed");
      exit(EXIT_FAILURE);
   }

   socklen_t len = sizeof(cliaddr);
   int n;

   n = recvfrom(sockfd, (char *)buffer, 1024,
            MSG_WAITALL, ( struct sockaddr *) &cliaddr, &len);
   buffer[n] = '\0';
   std::cout<<"Received \""<<buffer<<"\""<<std::endl;

   sendto(sockfd, (const char *)hello, strlen(hello),
      MSG_CONFIRM, (const struct sockaddr *) &cliaddr, len);
   std::cout<<"Hello message sent."<<std::endl;

   close(sockfd);
   return 0;
}
```

Note: to talk to another computer use inet_addr() instead of INADDR_ANY (e.g., inet_addr("192.168.1.10"))

Application Layer

From UDP to TCP

- Unlike UDP, in TCP client and server need to **handshake** before messages can be transmitted.
- We use **two sockets on the server-side**:
  - A **welcoming socket**, which is always on, and allows clients to perform handshakes with the server.
  - A **client-specific socket**, which is created after the handshake, and is used by client and server to communicate.
- The three-way handshake, which takes place within the transport layer, is completely **invisible to the client and the server programs**.
- Once the connection is accepted by the server (handshake success) **the communication moves to the newly created socket**.



Application Layer

Sockets: Connection (TCP only)

- Client-side TCP connection is performed by using the connect() function:

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

int val = connect(int socket, const struct sockaddr *address, socklen_t address_len);
```

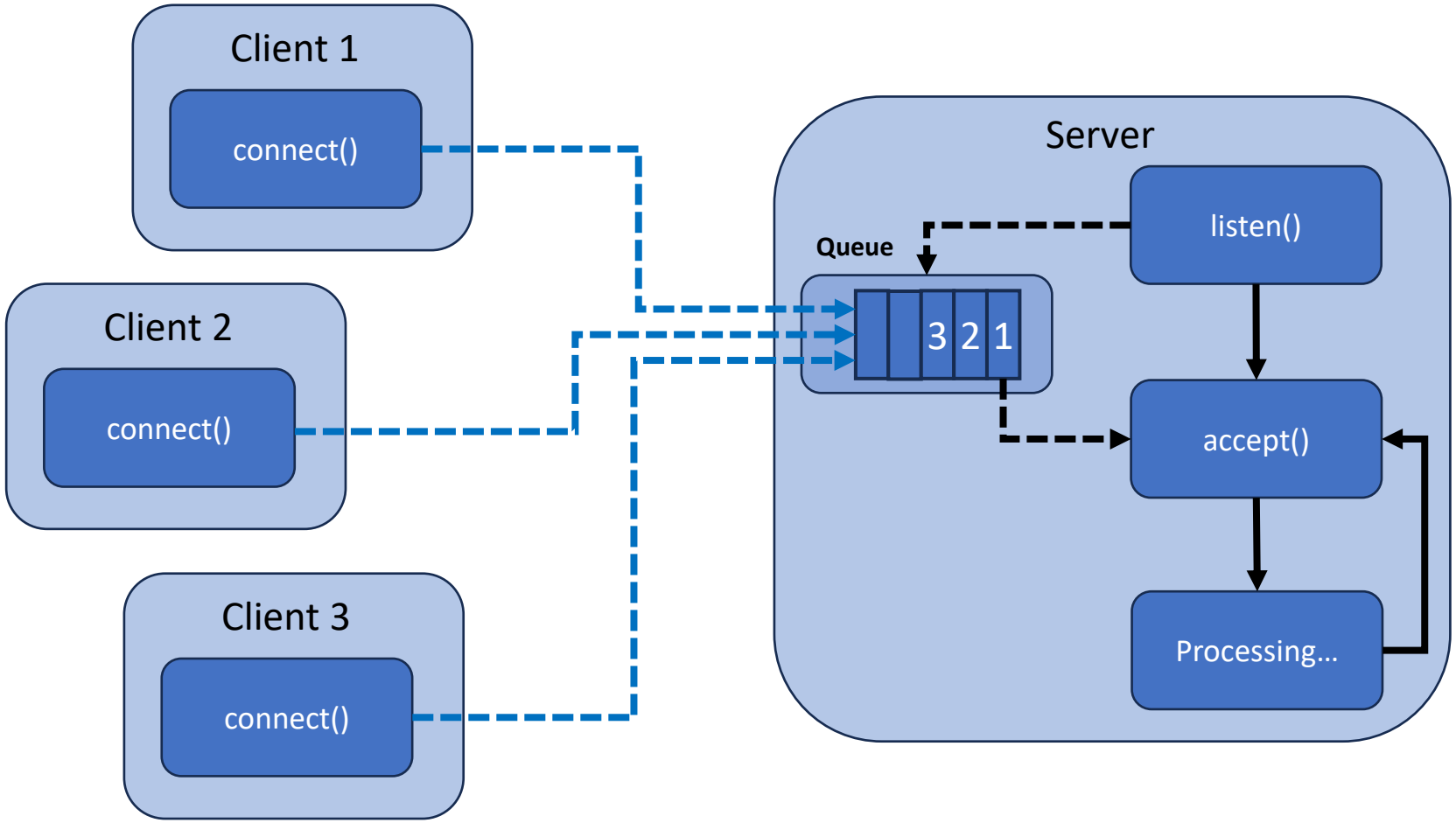
Where:

- socket: is the file descriptor of the socket we use to connect.
- address: pointer to a sockaddr structure containing the server address.
- address_len: specifies the length of the sockaddr structure pointed to by the address argument (we can use sizeof() to get it).
- val: return value which is 0 on success, and -1 otherwise.

Application Layer

Sockets: Wait-for-connection (TCP only)

- Server-side TCP wait-for-connection is performed through two functions: listen() and accept(), which works in combination with the clients connect().
- The listen() opens a queue where incoming connections are stored.
- The accept() opens the client-specific socket.



Application Layer

Sockets: Wait-for-connection (TCP only)

- Server-side TCP wait-for-connection is performed by the listen() and accept() functions:

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

int val = listen(int socket, int backlog);
int new_sockfd = accept(int socket, struct sockaddr *address, socklen_t *address_len);
```

Where:

- socket: is the file descriptor of the socket on which to wait for connections (must be bound to address/port).
- backlog: maximum length of the queue of pending connections.
- address: can be a null pointer, or a pointer to a sockaddr structure where the address of the connecting socket shall be returned.
- address_len: length of the address.
- new_sockfd: new socket descriptor on which the client and the server will communicate.
- val: return value which is 0 on success, and -1 otherwise.

Application Layer

Sockets: Send and Receive (TCP)

- Processes of sending or receiving messages in TCP are performed by the send() and read() functions (simpler than UDP ones):

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

int ob = send(int osock, const void *obuf, size_t olen, int flags);
```

```
#include <unistd.h>

int ib = read(int isock, void *ibuf, size_t ilen);
```

Where:

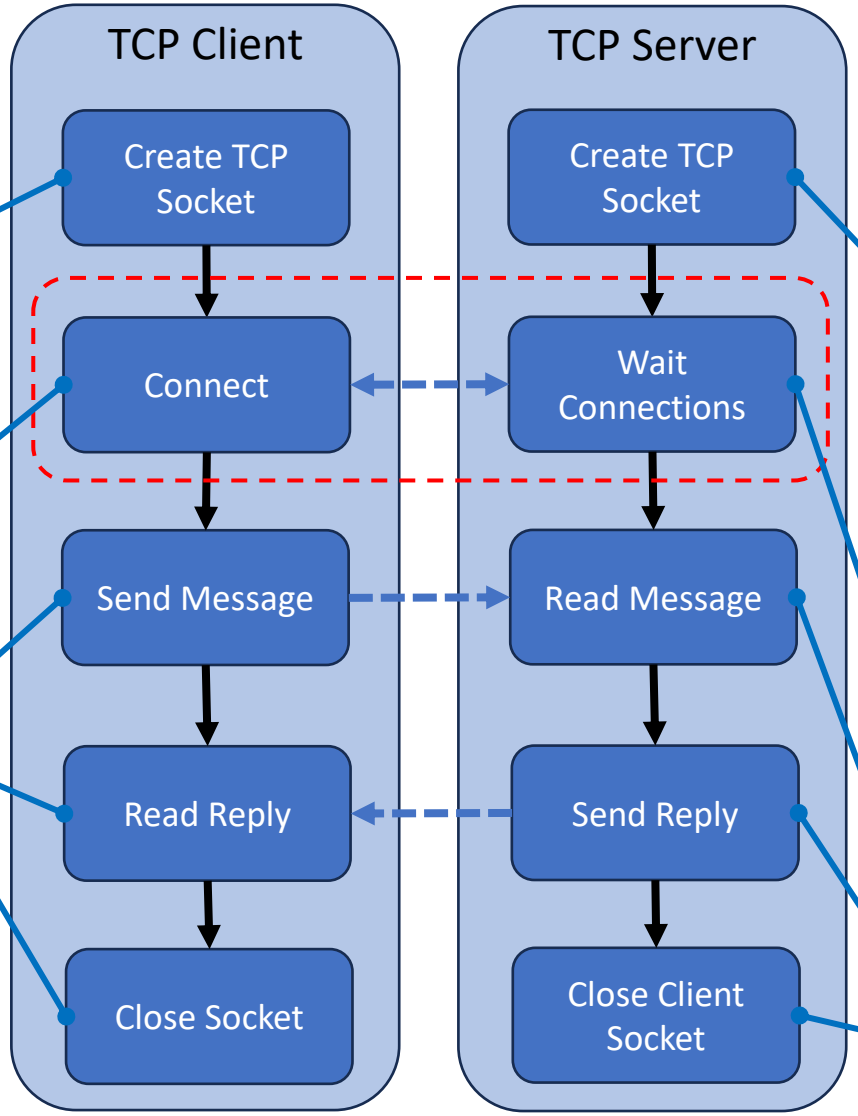
- osock/isock: are the file descriptors on which to send/receive a message.
- obuf/ibuf: are pointers to the buffers containing a message to send/receive.
- olen/ilen: are lengths of the messages in bytes.
- flags: specifies the type of message transmission (depends on protocol, typically 0).
- ob/ib: number of bytes that are actually sent/received.

- Notice that we have already specified client and server addresses during the connection phase, we no need to do it here as in sendto/recvfrom.

Application Layer

TCP Socket Programming (C/C++)

```
// client-side (includes omitted)

int main() {
   int sockfd, status;
   char buffer[1024];
   const char *hello = "Hello from client";
   struct sockaddr_in servaddr;

   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
      perror("socket creation failed");
      exit(EXIT_FAILURE);
   }

   memset(&servaddr, 0, sizeof(servaddr));

   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(8080);
   servaddr.sin_addr.s_addr = INADDR_ANY;

   int n;

   if ((status = connect(sockfd, (struct sockaddr*)&servaddr,
            sizeof(servaddr))) < 0) {
      printf("\nConnection Failed \n");
      return -1;
   }

   send(sockfd, hello, strlen(hello), 0);
   std::cout<<"Hello message sent."<<std::endl;

   n = read(sockfd, buffer, 1024);
   std::cout<<"Received \""<<buffer<<"\""<<std::endl;

   close(sockfd);

   return 0;
}
```




```
//server-side (includes omitted)

int main() {
   int sockfd, new_socket;
   int opt = 1;
   char buffer[1024];
   const char *hello = "Hello from server";
   struct sockaddr_in servaddr, cliaddr;

   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
      perror("socket creation failed");
      exit(EXIT_FAILURE);
   }
   if (setsockopt(sockfd, SOL_SOCKET,  SO_REUSEADDR | SO_REUSEPORT, &opt,
            sizeof(opt))) {
      perror("setsockopt");
      exit(EXIT_FAILURE);
   }

   memset(&servaddr, 0, sizeof(servaddr));
   memset(&cliaddr, 0, sizeof(cliaddr));

   servaddr.sin_family    = AF_INET;
   servaddr.sin_addr.s_addr = INADDR_ANY;
   servaddr.sin_port = htons(8080);

   if ( bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0 ) {
      perror("bind failed");
      exit(EXIT_FAILURE);
   }

   if (listen(sockfd, 3) < 0) {
      perror("listen");
      exit(EXIT_FAILURE);
   }
   int addrlen = sizeof(cliaddr);
   if ((new_socket = accept(sockfd, (struct sockaddr*)&cliaddr,
            (socklen_t*)&addrlen)) < 0) {
      perror("accept");
      exit(EXIT_FAILURE);
   }

   int n = read(new_socket, buffer, 1024);
   std::cout<<"Received \""<<buffer<<"\""<<std::endl;

   send(new_socket, hello, strlen(hello), 0);
   std::cout<<"Hello message sent."<<std::endl;

   close(new_socket);
   close(sockfd);
   return 0;
}
```

Note: to talk to another computer use inet_addr() instead of INADDR_ANY (e.g., inet_addr("192.168.1.10"))