Socket Programming

Programmazione di Reti

Ing. Chiara Contoli, PhD
chiara.contoli@unibo.it

Corso di Laurea Triennale in
Ingegneria e Scienze Informatiche

Outline

Socket Programming:

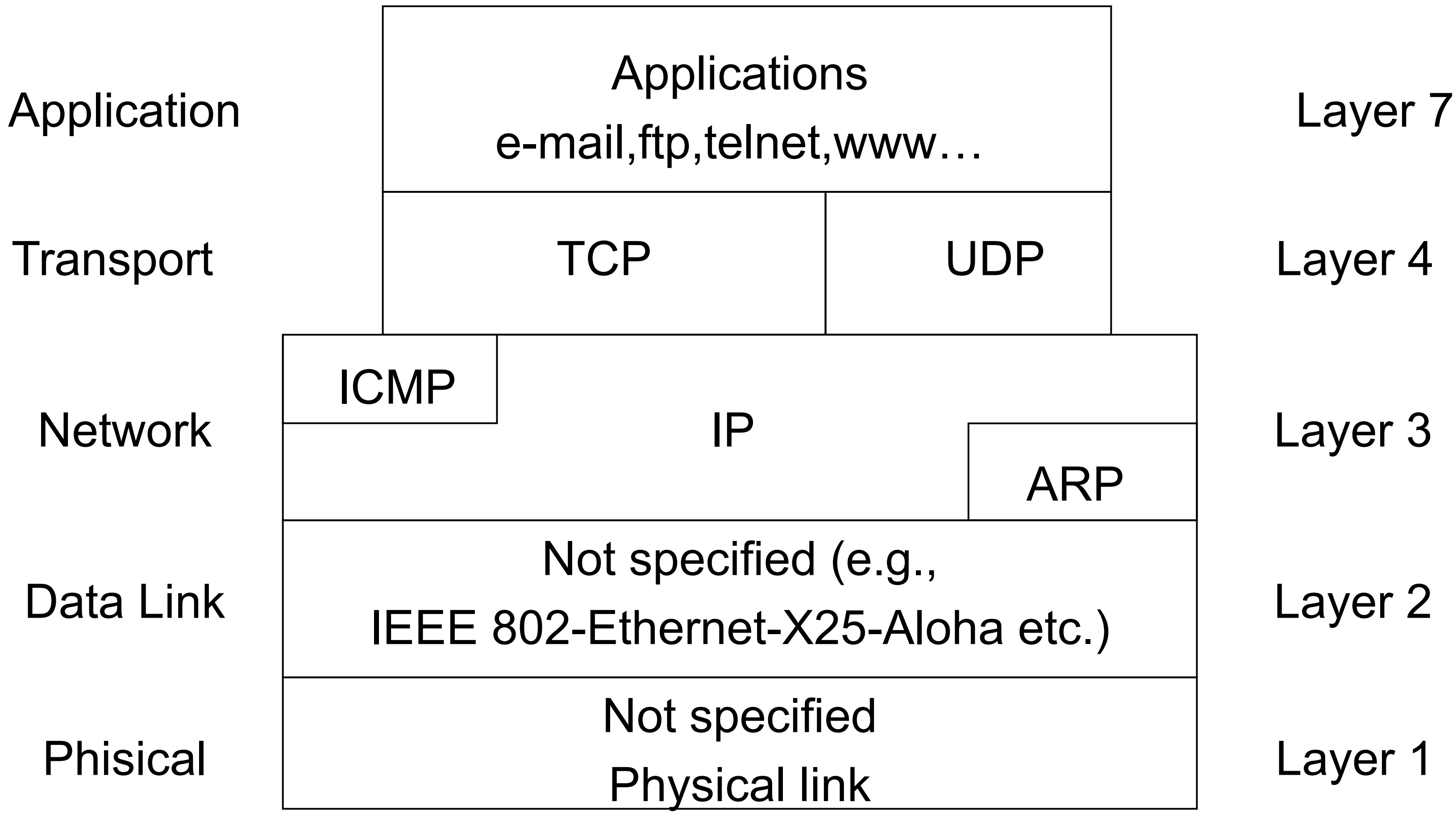
- Basic concepts
- UDP socket
- Client/Server interaction

C language Socket Programming

- Simple UDP Client application
- Simple UDP Server Application

ISO-OSI

Distributed systems: communications between different processes via messages exchange



IP Network

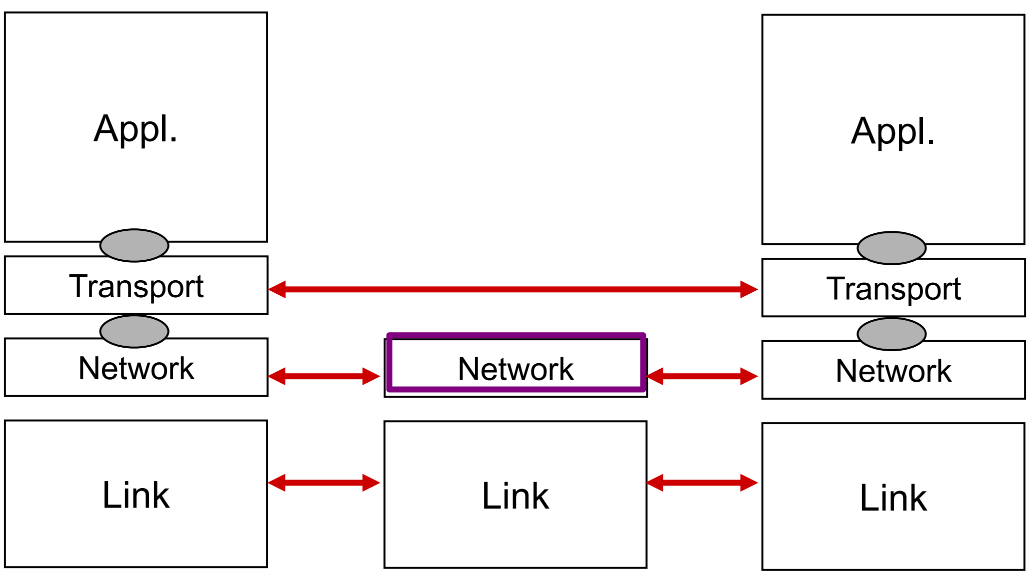
End-to-end communication: IP network allows (distributed) hosts located at the network edges to communicate between each others

Processes use layer 7 protocol in order to exchange information

Communication occurs by exchanging packets between remote hosts
- Layer 4 protocol adoption

Network routes packets via layer 3

Packets are directly delivered to layer 2



Socket programming

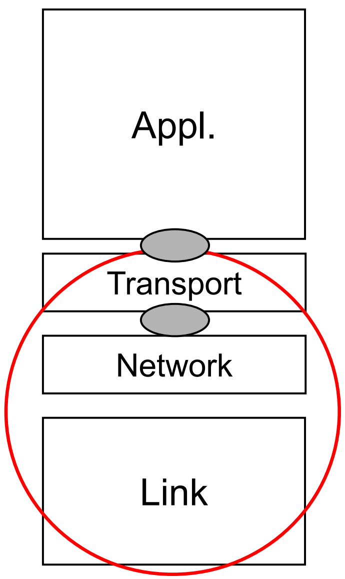
Goal: learn how to build client/server applications that communicate via socket

Socket API

- Introduced in 1981 in UNIX BSD 4.1
  - We will mainly take as reference the Linux kernel networking stack
- **Client** / **server** paradigm
- Sockets are explicitly created and utilized from the application
- Abstract pretty similar to file access
- Two types of transport via socket API:
  - **UDP** (unreliable)
  - **TCP** (reliable, byte stream)

Socket Programming: introduction

Today is pretty likely that systems use some form of networking.



Layer 2, 3 and 4 are all handled by the Linux *kernel*. It handles both *incoming* and *outgoing* packets. Incoming packets might be delivered *locally* or *forwarded*.

Application layer is not handled by the kernel, it is handled by the *userspace*.

This implies that (at least) two interfaces are required: one towards the userspace (i.e., towards applications) and one towards the kernelspace (i.e., towards the network).

Such interface is provided by the so called *Socket API*.

Socket Programming: basic concepts

Socket: a *local host* “interface”, *created by applications (controlled by operating system, OS)*, through which the process of an application can send and receive messages to / from the process of another application

The **client** must contact the **server**

Both client and server need to have their own sockets through which they can send and receive datagrams

To each socket is associated a **local port number**

Client **needs to know** the server **IP address** and the **port number** on which the application (and related socket) are listening to

Socket Programming: basic concepts

Several types of sockets exist:

- Stream sockets: byte-stream oriented and reliable communication
- Datagram sockets: message oriented, unreliable communication
- Raw sockets: allow direct access to IP layer (packets are sent/received directly to / from IP layer)
- …

Sockets are defined in a *communication domain*, which identifies:

- Address identification (i.e., socket address format)
- Communication range (i.e., same host or distributed hosts)

Several domains exists:

- Unix domain (i.e., communication occurs between application running on the same host)
- Internet domain (i.e., communication occurs among applications running on different hosts) → TCP/IP protocol stack

Socket Programming: basic concepts

Socket API provides several methods, among which:

- *Socket()*: creates a new socket
- *Bind():* associates a socket with a local IP address
- *Listen()*: allow to receive connection from other socket (not used in datagram sockets)
- *Accept()*: accepts a connection from a peer socket (used in connection oriented socket)
- *Connect():* allow to establish a connection to a peer socket (used in connection oriented socket)

Specific system call are also provided for socket I/O:

- *Send()*: send byte-streams
- *Recv()*: receive byte-streams
- *Sendto()*: send messages
- *Recvfrom()*: receive messages

C Socket

Sockets in C language are implemented through a standard service interface to be invoked on the Operating System

- Socket: communication endpoint
- Socket descriptor: equivalent to a file descriptor in UNIX

Communication domain defines communication semantics

- Each communication domain may support different protocols
  - *UNIX (AF_UNIX)*: local communication via pipe
  - *Internet (AF_INET)*: remote communication via internet protocols (TCP/IP)
  - *XROX (AF_NS)*: Xerox protocol
  - *AF_CCITT (AF_CCITT)*: protocollo X.25

Socket can have different communication domains

Socket Libraries

Several libraries are needed: <*sys/socket.h*>, <*arpa/inet.h*>, <*netinet/in.h*>

From <*sys/socket.h*>

- int **socket**(int *domain*, int *type*, int *protocol*);
  - Returns (socket) file descriptor on success, -1 on error
  - Parameters: i) *domain*: communication domain, ii) *type*: socket type, iii) *protocol*: protocol to be used

- int **bind**(int *sockfd*, const struct sockaddr **addr*, socklen_t *addrlen*);
  - Returns 0 on success, -1 on error
  - Parameters: i) *sockfd*: socket file descriptor, ii) *addr*: address to be bound to the socket; *addrlen*: size of the ii)

Internet domain socket address

Two types: IPv4 and IPv6. Our focus is on IPv4; address is stored in a data structure: *socckaddr_in* (struct *sockaddr_in*, defined in <*netinet/in.h*>)

```
struct sockaddr_in {  // IPv4 socket address
  sa_family_t sin_family;  // Address family
  in_port_t sin_port;  // Port
  struct in_addr sin_addr;  // IPv4 address
  ….
}

struct in_addr {  // IPv4 address (4-byte)
  in_addr_t s_addr;
}
```

UDP Socket

Programmazione di Reti

Ing. Chiara Contoli, PhD
chiara.contoli@unibo.it

Corso di Laurea Triennale in
Ingegneria e Scienze Informatiche

Why UDP?

No connection required/established (connection introduces latency)

Simple: no state of the connection needs to be memorized neither in the source nor in the destination

Datagram header are **short**: overhead is reduced

No congestion control: UDP can send data without any control

UDP: User Datagram Protocol (RFC 768)

Transport protocol without “frill”

Provides transport service in a “**best effort**” fashion, UDP datagram can be:

- **Lost**
- Delivered **out of sequence** to the application

Connectionless:

- **No hand-shake** between UDP source and UDP destination
- Each UDP datagram is handled in a **totally independent** manner from other UDP datagrams

UDP Socket Programming

UDP: there is no connection between **client** and **server**

There is no handshake

Each application level message needs to be **contained in a single UDP datagram**

To send a response to the client, server needs to extract from each received datagram the source host **IP address** and the **port number** (of the sender source application)

UDP: data transmitted might **not arrive** at the destination or might arrive **out of order** compared to the sending sequence

Application point of view: UDP provides an **unreliable** byte transfer service of "groups of bytes" (known as datagrams) between client and server

UDP Socket Programming

UDP: there is no connection between **client** and **server**

There is no handshake

… to be **contained in a single UDP datagram**

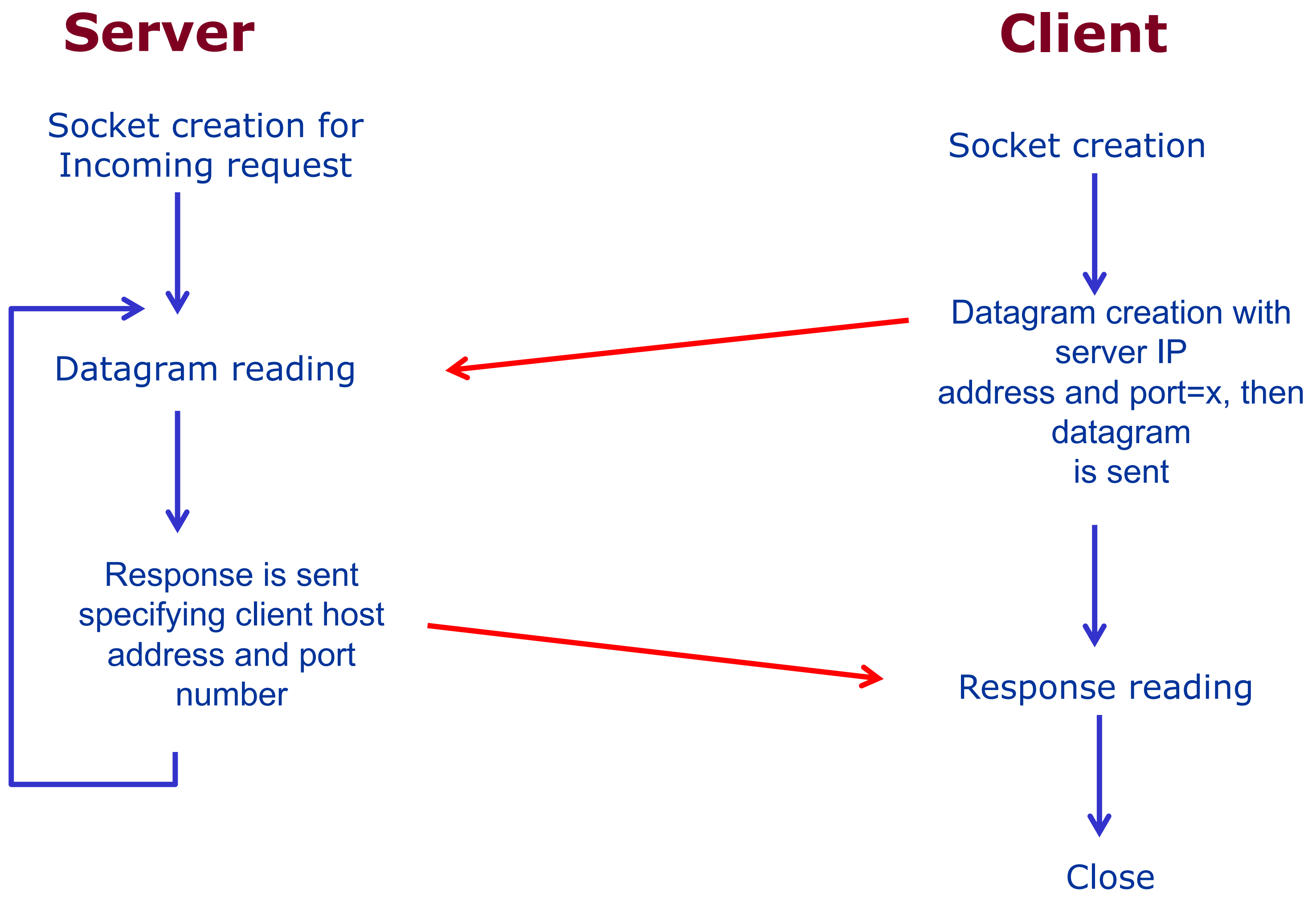
To send a response to the client, server needs to extract from each received datagram the … **address** and the **port number** (of the sender source application)

UDP: data transmitted might **not arrive** at the destination or the might arrive **out of order** compared to the sending sequence

Application point of view: UDP provides an **unreliable** byte transfer service of "groups of bytes" (known as datagrams) between client and server

This term is ambiguos, because usually is adopted for packets in IP networks, but in the Internet terminology it is adopted pretty often

UDP: client – server interaction



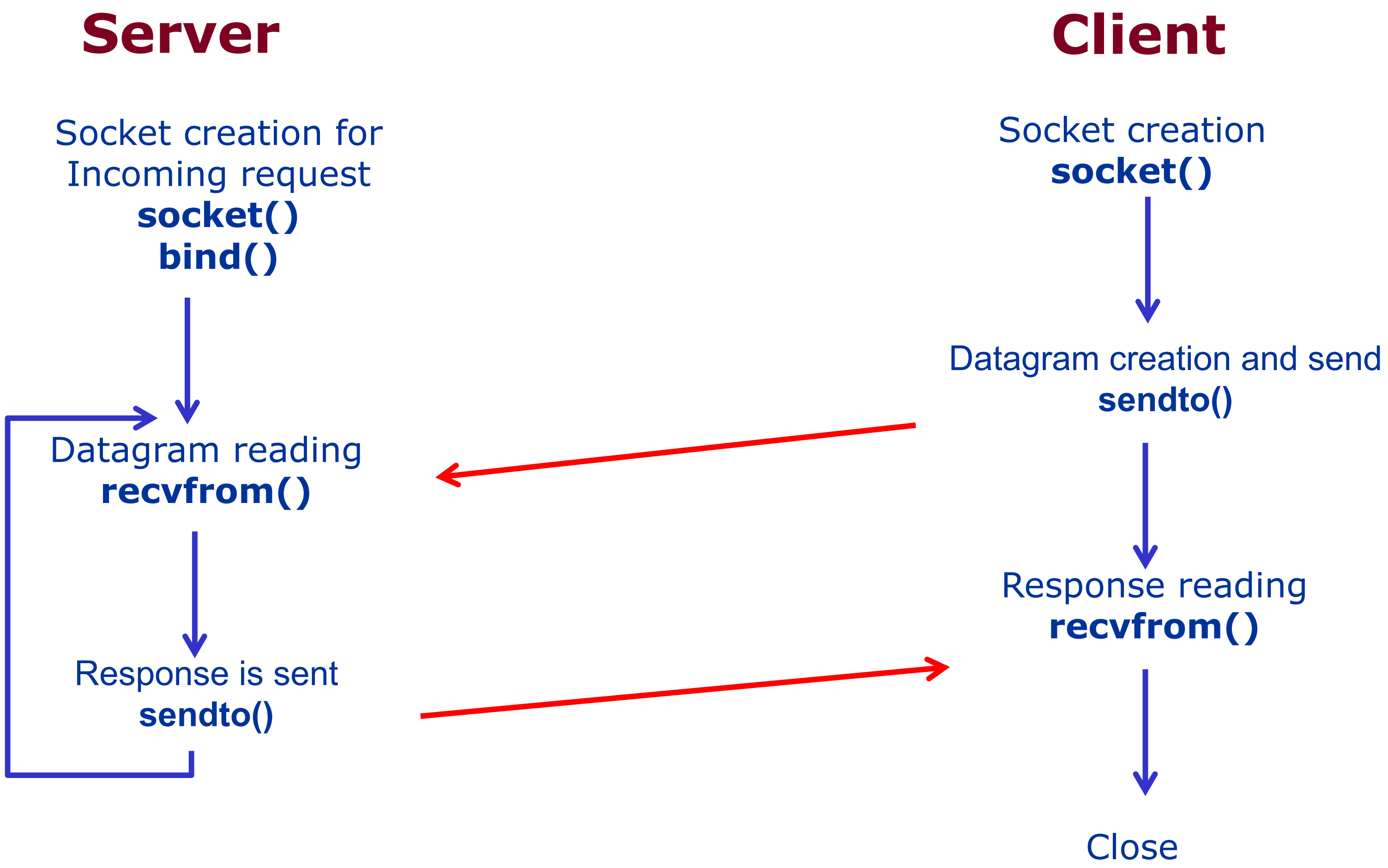
Datagram Socket I/O

From <*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, -1 on error
  - Parameters: i) *sockfd*: socket file descriptor, ii) *buffer*: store received data, iii) *length*: maximum number of bytes to be read, v) and vi) *src_addr* and *addrlen*: store information about message sender

- ssize_t **sendto**(int *sockfd*, const void **buffer*, size_t *length*, int *flags*, struct sockaddr **dst_addr*, socklen_t *addrlen*);
  - Returns number of bytes sent, 0 on EOF, -1 on error
  - Parameters: i) *sockfd*: socket file descriptor, ii) *buffer*: store sent data, iii) *length*: number of bytes to be sent, v) and vi) *src_addr* and *addrlen*: store information about message receiver

UDP: client – server interaction



Example: a simple client / server application

Client:

- User leverage keyboard (standard input) to write a text line
- Client send this text to server

Server:

- Server receive the text line sent by the client
- Convert the text line to upper case letter
- Send modified text to the client

Client:

- Receive modified text line
- Print text on the screen (standard output)

Example: UDP Server

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "myfunction.h”

#define MAX_BUF_SIZE 1024 // Maximum size of UDP messages
#define SERVER_PORT 9876 // Server port

int main(int argc, char *argv[]){
  struct sockaddr_in server_addr; // struct containing server address
information
  struct sockaddr_in client_addr; // struct containing client address
information
  int sfd; // Server socket filed descriptor
  int br; // Bind result
  int i;
  ssize_t byteRecv; // Number of bytes received
  ssize_t byteSent; // Number of bytes to be sent
```

Example: UDP Server

```
socklen_t cli_size;
char receivedData [MAX_BUF_SIZE]; // Data to be received
char sendData [MAX_BUF_SIZE]; // Data to be sent

sfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

if (sfd < 0){
  perror("socket"); // Print error message
  exit(EXIT_FAILURE);
}

// Initialize server address information
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(SERVER_PORT); // Convert to network byte
order
server_addr.sin_addr.s_addr = INADDR_ANY; // Bind to any address

br = bind(sfd, (struct sockaddr *) &server_addr, sizeof(server_addr));
if (br < 0){
  perror(”bind"); // Print error message
  exit(EXIT_FAILURE);
}
```

Example: UDP Server

```
  cli_size = sizeof(client_addr);

  for(;;){
    byteRecv = recvfrom(sfd, receivedData, MAX_BUF_SIZE, 0, (struct
sockaddr *) &client_addr, &cli_size);

    if(byteRecv == -1){
      perror("recvfrom");
      exit(EXIT_FAILURE);
    }
    printf("Received data: ");
    printData(receivedData, byteRecv);
    if(strncmp(receivedData, "exit", byteRecv) == 0){
      printf("Command to stop server received\n");
      break;
    }

    convertToUpperCase(receivedData, byteRecv);
    printf("Response to be sent back to client: ");
    printData(receivedData, byteRecv);
```

Example: UDP Server

```
    byteSent = sendto(sfd, receivedData, byteRecv, 0, (struct sockaddr *)
&client_addr, sizeof(client_addr));

    if(byteSent != byteRecv){
      perror("sendto");
      exit(EXIT_FAILURE);
    }
  } // End of for(;;)

  return 0;

}
```

Example: UDP Client

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include "myfunction.h”

#define MAX_BUF_SIZE 1024 // Maximum size of UDP messages
#define SERVER_PORT 9876 // Server port

int main(int argc, char *argv[]){
  struct sockaddr_in server_addr; // struct containing server address
information
  struct sockaddr_in client_addr; // struct containing client address
information
  int sfd; // Server socket filed descriptor
  int br; // Bind result
  int i;
  int stop = 0;
```

Example: UDP Client

```
ssize_t byteRecv; // Number of bytes received
ssize_t byteSent; // Number of bytes to be sent

size_t msgLen;
socklen_t serv_size;
char receivedData [MAX_BUF_SIZE]; // Data to be received
char sendData [MAX_BUF_SIZE]; // Data to be sent

sfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

if (sfd < 0){
  perror("socket"); // Print error message
  exit(EXIT_FAILURE);
}

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(SERVER_PORT);
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

serv_size = sizeof(server_addr);
```

Example: UDP Client

```
  while(!stop){
    printf("Insert message:\n");
    scanf("%s", sendData);
    printf("String going to be sent to server: %s\n", sendData);

    if(strcmp(sendData, "exit") == 0){
      stop = 1;
    }
    msgLen = countStrLen(sendData);
    byteSent = sendto(sfd, sendData, msgLen, 0, (struct sockaddr *)
&server_addr, sizeof(server_addr));
    printf("Bytes sent to server: %zd\n", byteSent);

    if(!stop){
      byteRecv = recvfrom(sfd, receivedData, MAX_BUF_SIZE, 0, (struct
sockaddr *) &server_addr, &serv_size);
      printf("Received from server: ");
      printData(receivedData, byteRecv);
    }
  } // End of while
  return 0;
}
```

“myfunction.h”

```
#include <ctype.h>

size_t countStrLen(char *str){
  size_t c = 0;

  while(*str != ‘\0’){
    c += 1;
    str++;
  }
  return c;
}

void printData(char *str, size_t numBytes){
  for(int i = 0; i < numBytes; i++){
    printf("%c", str[i]);
  }
  printf("\n");
}
```

“myfunction.h”

```
void convertToUpperCase(char *str, size_t numBytes){
  for(int i = 0; i < numBytes; i++){
    str[i] = toupper(str[i]);
  }
}
```

Useful references

Socket API, to cite a few:

- http://man7.org/linux/man-pages/man3/
  - http://man7.org/linux/man-pages/man3/socket.3p.html
  - http://man7.org/linux/man-pages/man3/bind.3p.html
- https://www.gnu.org/software/libc/manual/html_node/Sockets.html
- … and much more on the web
- Kurose, Ross book (Chapter 2, 2.7 and 2.8)

C language reference:

- http://en.cppreference.com/w/c