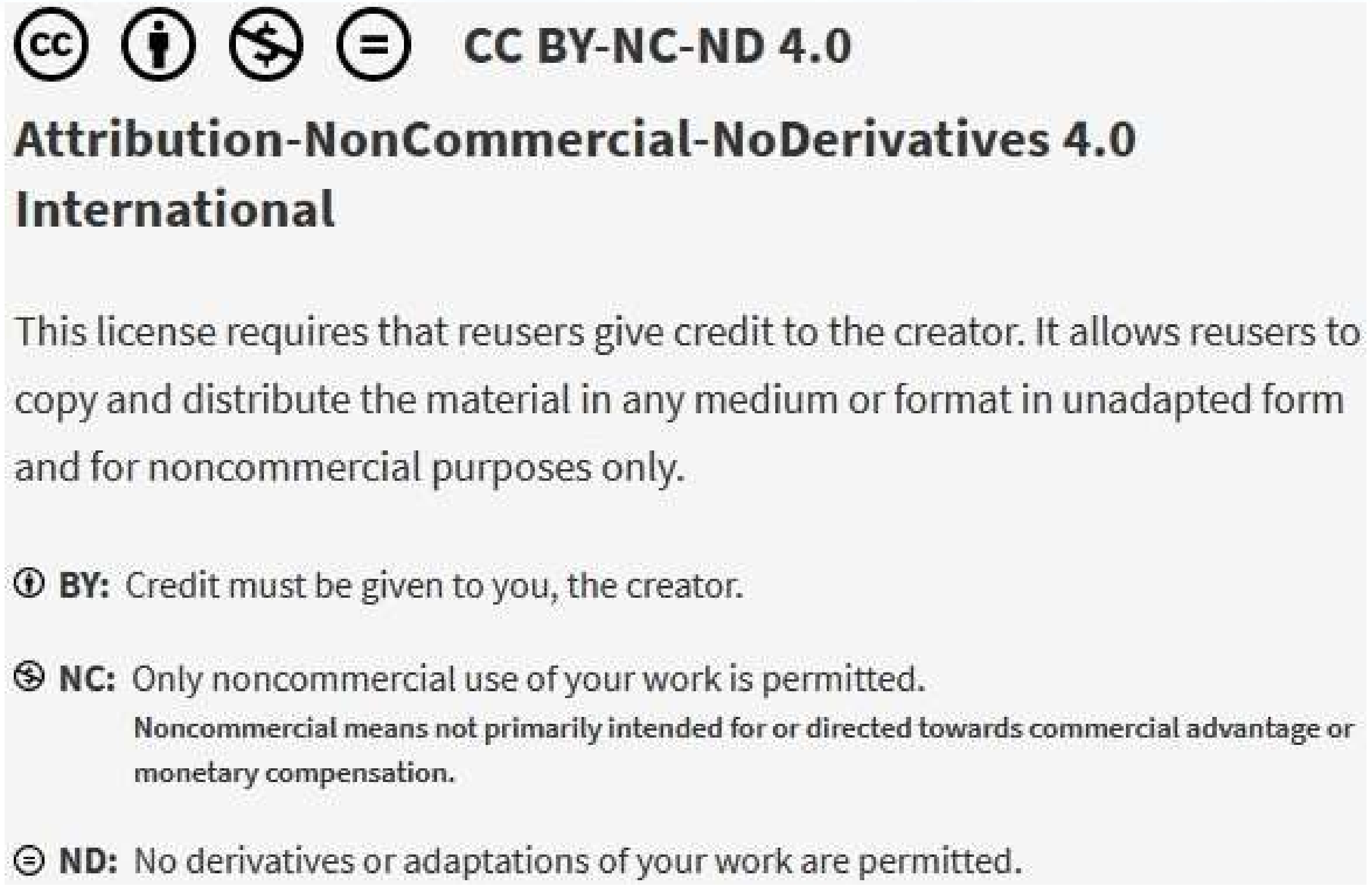
Inter-Process Communication

Network Sockets

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

License Information

This work is licensed under the license



CC BY-NC-ND 4.0

Attribution-NonCommercial-NoDerivatives 4.0 International

This license requires that reusers give credit to the creator. It allows reusers to copy and distribute the material in any medium or format in unadapted form and for noncommercial purposes only.

BY: Credit must be given to you, the creator.

NC: Only noncommercial use of your work is permitted.
Noncommercial means not primarily intended for or directed towards commercial advantage or monetary compensation.

ND: No derivatives or adaptations of your work are permitted.

To view a copy of the license, visit:
https://creativecommons.org/licenses/by-nc-nd/4.0/?ref=chooser-v1

Introduction

- Classical methods of IPC
  - FIFOs, message queues, and shared memory, allow processes running on the **same computer** to communicate with one another
- Network IPC
  - Allows processes running on **different computers** (connected to a common network) to communicate with one another

Main topic of
distributed programming courses

Introduction

- Network sockets can be used by processes to communicate with other processes, regardless of where they are running, i.e., on the same machine or on different machines
  - The same interfaces can be used for both **inter-machine** and **intra-machine** communication
- Sockets can be used to communicate using many different network protocols
  - We restrict our discussion to the TCP/IP protocol, i.e., de facto standard for communicating over the Internet

Logic flow

- Logic flow to use a socket
  - Create a socket with the system call **socket**
  - Create a connection with **connect**
  - Bind an address to a socket with **bind**
  - Accept a connection with **listen**
  - Communicate through a socket with **read** and **write**

Logic flow

- Logic flow to use a socket
  - Create a socket with the system call **socket**
    - Applications use socket descriptors to access sockets
    - Socket descriptors are implemented as file descriptors in the UNIX System
  - Create a connection with **connect**
  - Bind an address to a socket with **bind**
  - Accept a connection with **listen**
  - Communicate through a socket with **read** and **write**

Logic flow

- Logic flow to use a socket
  - Create a socket with the system call **socket**
  - Create a connection with **connect**
    - The connection is between the process requesting the service (the client) and the process providing the service (the server)
  - Bind an address to a socket with **bind**
  - Accept a connection with **listen**
  - Communicate through a socket with **read** and **write**

Logic flow

- Logic flow to use a socket
  - Create a socket with the system call **socket**
  - Create a connection with **connect**
  - Bind an address to a socket with **bind**
  - Accept a connection with **listen**
  - Communicate through a socket with **read** and **write**
    - Read and Write may be not sufficient in general
    - To specify options, receive packets from multiple clients, or send prioritized data, we need to use specific socket functions, such as
      - send, sendto, sendmsg, recv, recvfrom, recvmsg

Operations

Operation	Meaning

int socket (
  int domain,
  int type,
  int protocol
);
	Create a socket. Domain specifies the nature of the communication. Type determines the type of the socket, i.e., the communication characteristics. Protocol is usually zero, to select the default protocol for the given domain and socket type.

int connect(
  int sockfd,
  const struct sockaddr *addr,
  socklen_t len
);
	Create a connection. Addr is the address of the server with which we wish to communicate.

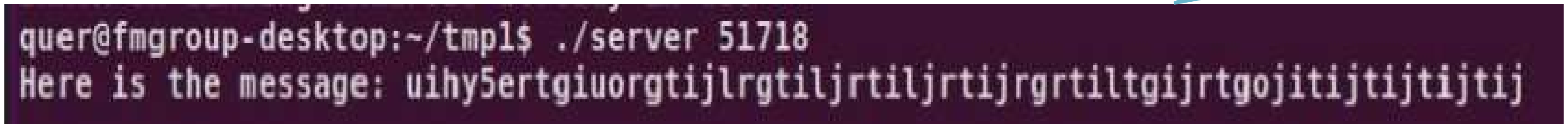
Operations

Operation	Meaning
int bind (int sockfd, const struct sockaddr *addr, socklen_t len);	Associate an address with the server's socket. Sockfd is the cocket descriptor. Addr is the reference of a server address structure that must be properly initialized.
int listen (int sockfd, int backlog);	A server tells that is willing to accept connect request using the function listen. Sockfd is the socket descriptor. Baklog is the number of outstanding connect requests that it should enqueue on behalf pf the process.
int accept (int sockfd, struct sockaddr *addr, socklen_t *restrict len);	Retrieve a connect request and convert it into a connection. The file descriptor returned is the connected socket.

Example

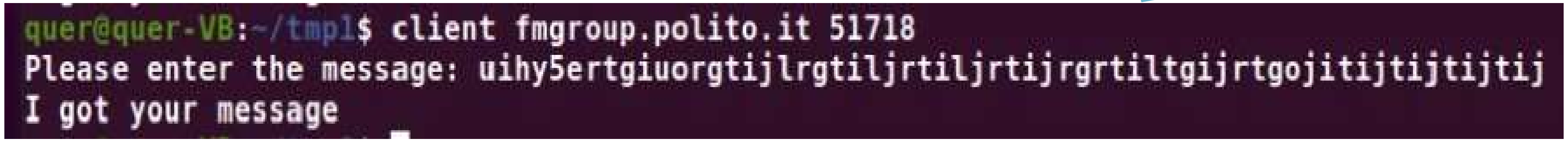
- Write two processes on two different servers
  - One process acts as a server
  - The other one acts as a client
  - Create a socket
  - Establish a connection
  - Send and receive information

fmgroup.polito.it



```
quer@fmgroup-desktop:~/tmp1$ ./server 51718
Here is the message: uihy5ertgiuorgtijlrgtiljrtiljrtijrgrtiltgijrtgojitijtijtijtij
```

fmastq.polito.it



```
quer@quer-VB:~/tmp1$ client fmgroup.polito.it 51718
Please enter the message: uihy5ertgiuorgtijlrgtiljrtiljrtijrgrtiltgijrtgojitijtijtijtij
I got your message
```

Example

- Write two processes on two different servers
  - One process acts as a server
  - The other one acts as a client
  - Create a socket
  - Establish a connection
  - Send and receive information
- Observations
  - The **client** needs to know of the existence of, and the address of, the server and connects to it
  - The **server** does not need to know the address of (or even the existence of) the client and makes a request for information

Solution: Server

- The server must perform the following steps to use a socket
  - Create a socket with the **socket** system call
  - Bind the socket to an address using the **bind** system call
    - For a server socket on the Internet, an address consists of a port number on the host machine
  - Listen for connections with the **listen** system call
  - Accept a connection with the **accept** system call
    - This call typically blocks until a client connects with the server
  - Send and receive data with **read** and **write**

Solution: Server

Server

Server ⇔ Client

The port number is passed as an argument

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

int main(int argc, char *argv[]) {
  int sockfd, newsockfd, portno;
  socklen_t clilen;
  char buffer[256];
  struct sockaddr_in serv_addr, cli_addr;
  int n;
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) error ...
  bzero((char *) &serv_addr, sizeof(serv_addr));
  portno = atoi(argv[1]);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(portno);
  if (bind(sockfd, (struct sockaddr *) &serv_addr,
    sizeof(serv_addr)) < 0) error ...
```

Create a socket

Bind the socket to an address

Solution: Server

Server

Server ⟺ Client

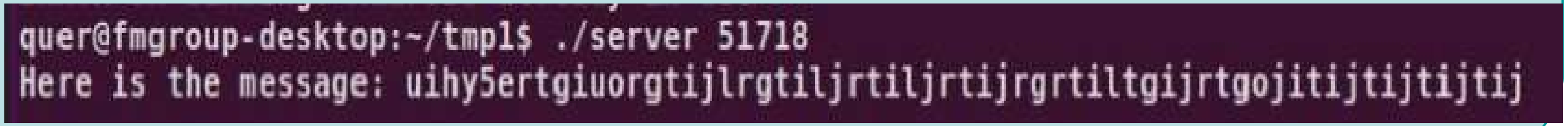
Listen the connection

Accept the connection

Read a message from the client

Write a message to the client

```
  listen(sockfd,5);
  clilen = sizeof(cli_addr);
  newsockfd = accept (sockfd,
     (struct sockaddr *) &cli_addr, &clilen);
  if (newsockfd < 0) error ...
  bzero(buffer,256);
  n = read(newsockfd,buffer,255);
  if (n < 0) error ...
  printf("Here is the message: %s\n",buffer);
  n = write(newsockfd,"I got your message",18);
  if (n < 0) error ...
  close(newsockfd);
  close(sockfd);
  return 0;
}
```



```
quer@fmgroup-desktop:~/tmp1$ ./server 51718
Here is the message: uihy5ertgiuorgtijlrgtiljrtiljrtijrgrtiltgijrtgojitijtijtijtij
```

Solution: Client

- The client must perform the following steps to use a socket
  - Create a socket with the **socket** system call
  - Connect the socket to the address of the server using the **connect** system call
  - Send and receive data
    - There are a number of ways to do this, but the simplest is to use the **read** and **write** system calls

Solution: Client

Client

Server ⟺ Client

Name serve and port number are passed as arguments

```
#include ...

int main(int argc, char *argv[]) {
  int sockfd, portno, n;
  struct sockaddr_in serv_addr;
  struct hostent *server;
  char buffer[256];

  portno = atoi(argv[2]);
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) error ...
  server = gethostbyname(argv[1]);
  if (server == NULL) {
    fprintf(stderr,"ERROR, no such host\n");
    exit(0);
  }
  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  bcopy((char *)server->h_addr,
    (char *)&serv_addr.sin_addr.s_addr, server->h_length);
```

Create a socket

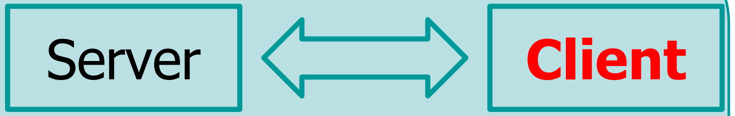
Get host structure from server name

Zero a byte string

Copy byte sequence

Client

Solution: Client

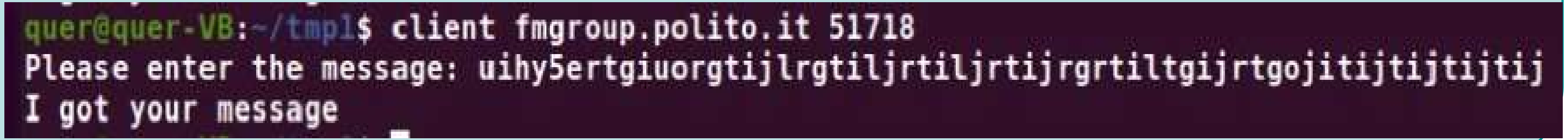


Create the connection

```
  serv_addr.sin_port = htons(portno);
  if (connect(sockfd,(struct sockaddr *)
     &serv_addr,sizeof(serv_addr)) < 0)
     error("ERROR connecting");
  printf("Please enter the message: ");
  bzero(buffer,256);
  fgets(buffer,255,stdin);
  n = write(sockfd,buffer,strlen(buffer));
  if (n < 0) error ...
  bzero(buffer,256);
  n = read(sockfd,buffer,255);
  if (n < 0) error ...
  printf("%s\n",buffer);
  close(sockfd);
  return 0;
}
```

Write a message to the server

Read a message from server



```
quer@quer-VB:~/tmp1$ client fmgroup.polito.it 51718
Please enter the message: uihy5ertgiuorgtijlrgtiljrtiljrtijrgrtiltgijrtgojitijtijtijtij
I got your message
```