BMCS3003 Distributed Systems and Parallel Computing

L02 - Inter-process Communication

Presented by

Assoc Prof Ts Dr Tew Yiqi
May 2023

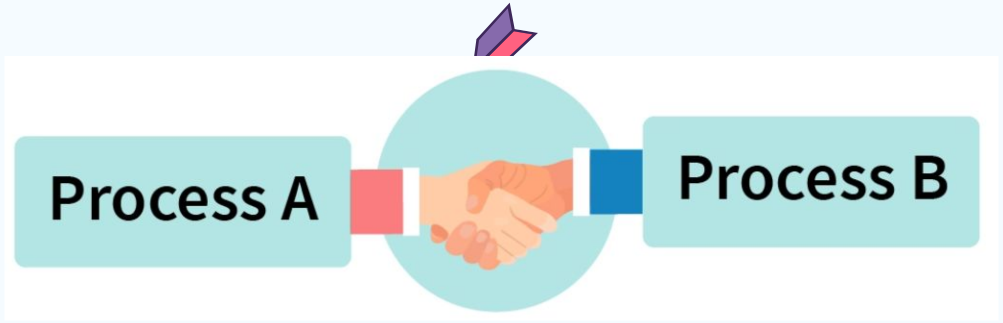
Table of contents

01

Client-server, Peer-to-peer

A relationship in which one program requests a service or resource from another program.

Inter-process communication in an operating system



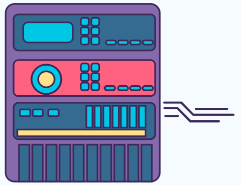
Processes executing concurrently in the operating system may be either **independent processes** or **cooperating processes**

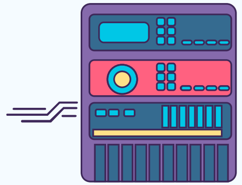
Independent processes:
They cannot affect or be affected by the other processes executing in the system

Cooperating processes:
They can affect or be affected by the other processes executing in the system

Independent Processes

Process 1





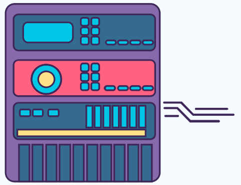
Process 2

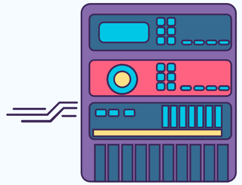
Run concurrently

Does not share resources

Cooperating Processes

Process 1



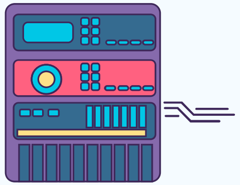
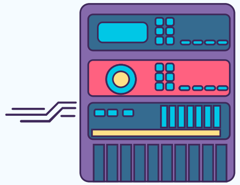


Process 2

Advantages of process cooperation

Information sharing - several users may access to the same file.
Computation speedup - a task to be divided into several concurrent subtasks.
Modularity - system functions to be divided into separate processes or threads.
Convenience - a user may perform several tasks at the same time eg editing, compiling or printing.

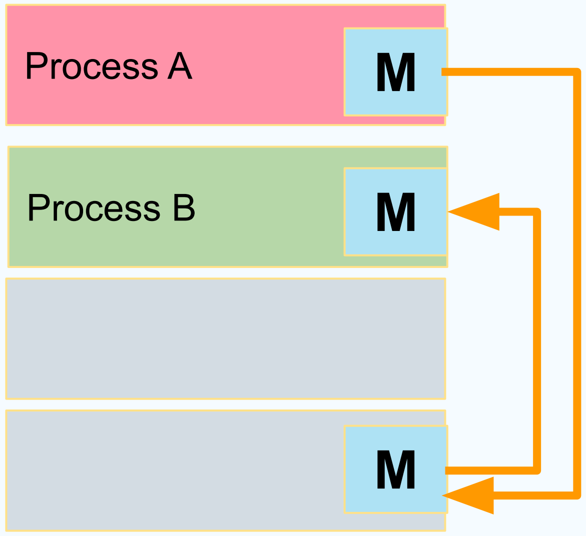
Inter-process communication (IPC)

Process 1   Process 2

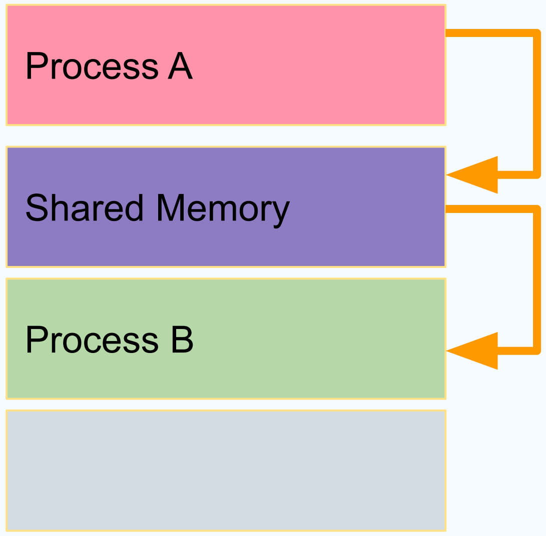
Cooperating processes require an inter process communication(IPC) mechanism that will allow them to exchange data and information.

There are 2 fundamental models:
a. Message passing communication model
b. Shared memory communication model

Communication Models



Message Passing



Shared Memory

Inter-process communication (IPC)

Message passing

- Implemented by using system calls, time consuming task of kernel intervention

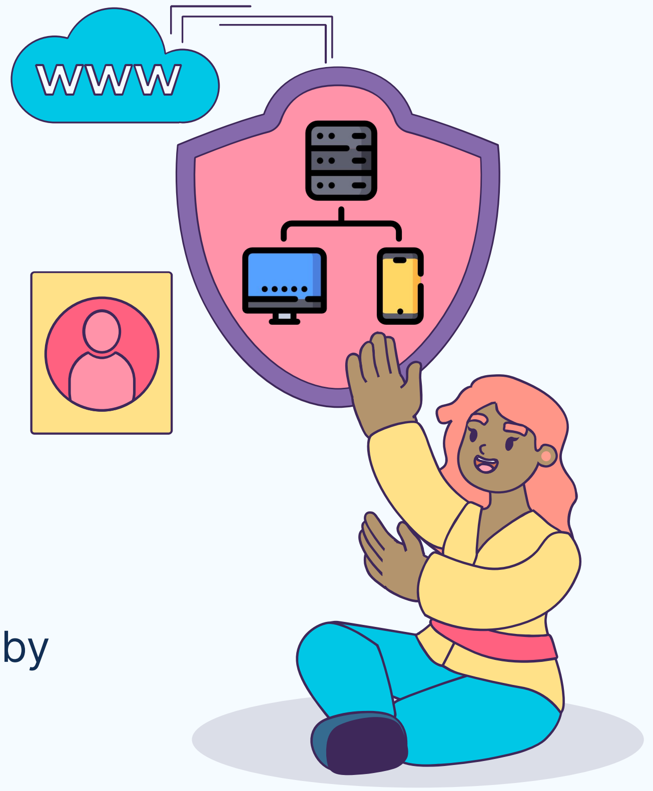
Shared memory

- Faster, kernel is not required
- Message exchange by reading & writing data to the shared region
- Must ensure that they are not writing to the same location simultaneously

01a

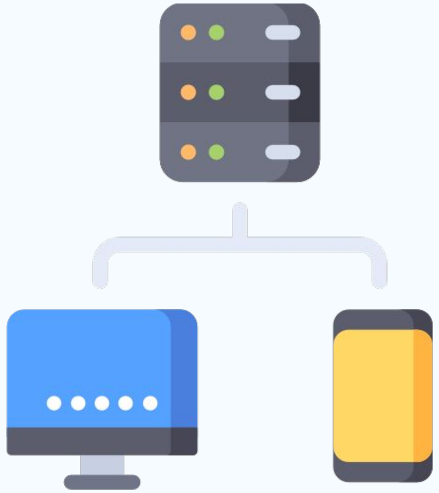
Client-server

Used to distinguish computing by Computers from the monolithic, centralised computing model used by mainframes

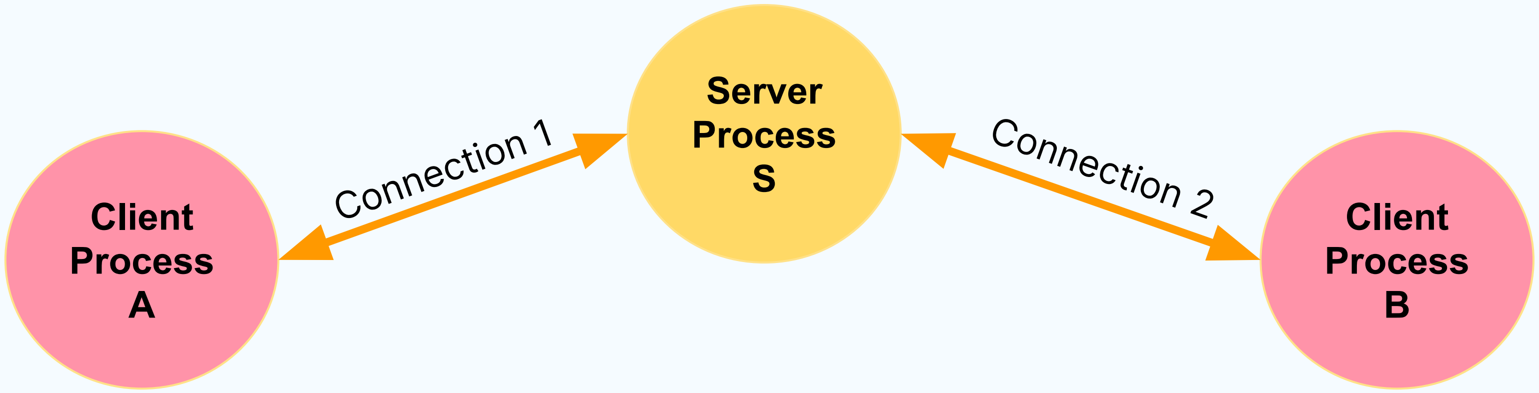


Client - Server

- ➢ Perhaps the most well known model of distributed computing
- ➢ The application logic is split into two parts:
  - ○ The ‘Server’ is a **process** that provides some sort of computation service.
  - ○ The ‘Client’ is a **process** that makes service requests to the Server.
- ➢ The Client is usually associated with human users (it acts as their agent).
- ➢ The Server is usually hosted on a dedicated computer that is designed specially
- ➢ (e.g. it might have larger memory, faster CPU etc to ensure it can handle requests at a high speed and with low delay).



Client - Server model 1

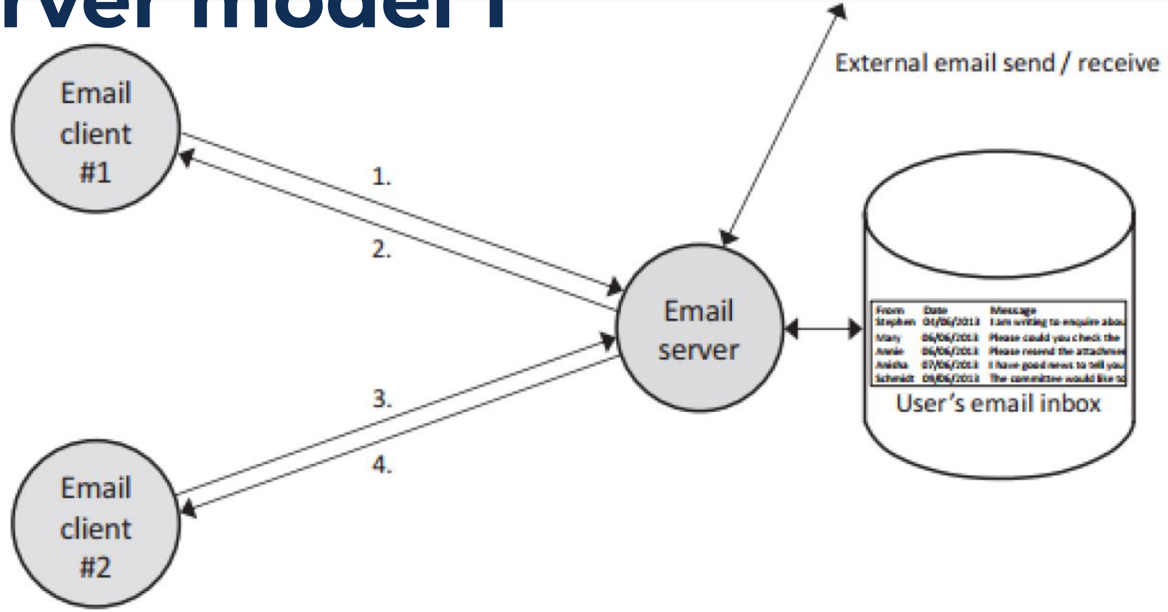


Connections are private between one client and one server.
One server may allow many clients to be connected at one time.
Clients usually initiate communication (as and when service is needed).
All communication is via the server (clients do not communicate directly).

Client - Server model 1

The user uses a desktop computer to check for any new email messages. This computer has a certain email client installed

(a)


Email client #1
1.
2.
External email send / receive
Email server
From Date Message
Stephen 04/06/2013 I am writing to enquire abou
Mary 06/06/2013 Please could you check the
Annie 06/06/2013 Please resend the attachmen
Anisha 07/06/2013 I have good news to tell you
Schmidt 09/06/2013 The committee would like to
User's email inbox
3.
4.
Email client #2

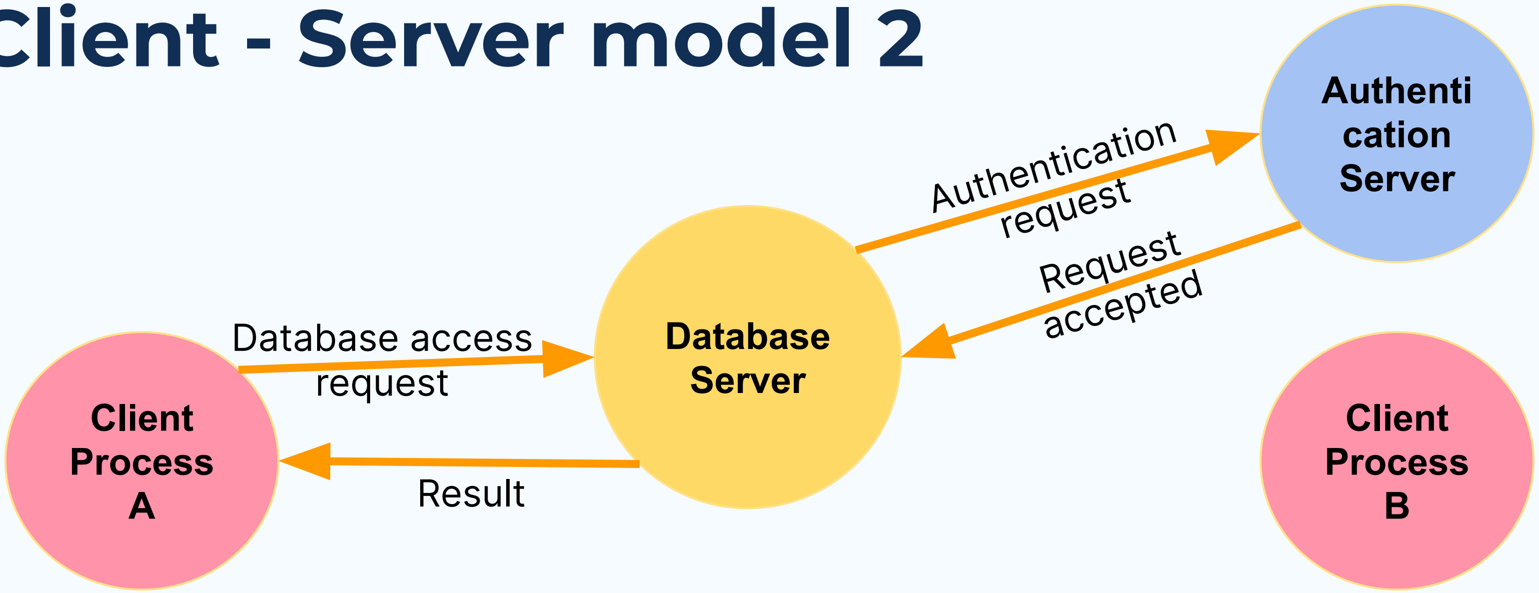
B. Later, the same user uses a mobile device such as a tablet to compose and send an email message. The tablet has a different email client installed

(b)

Key

1. User views email (client requests inbox contents from server)
2. Server sends reply containing email messages from inbox
3. User creates email and 'sends' it (client passes the email message to the server, which actually does the sending)
4. Server sends confirmation that email was sent (some clients may display it and others may not)

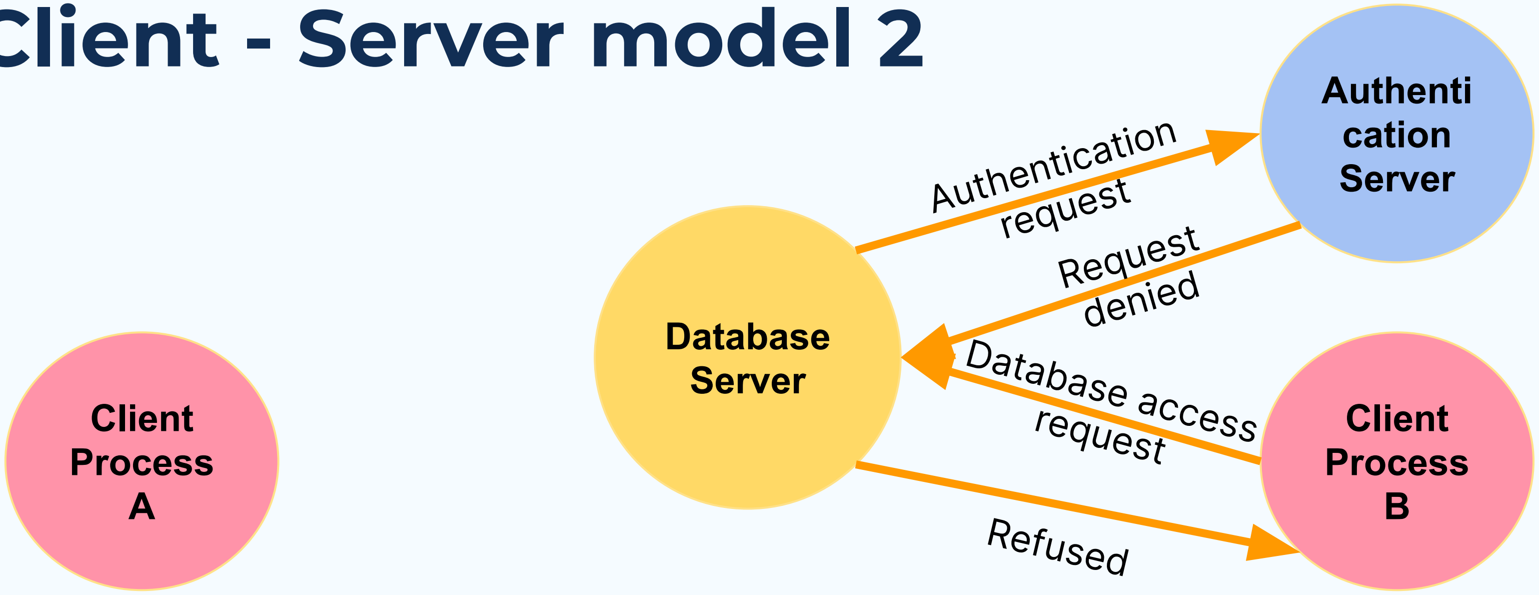
Client - Server model 2



Client - Server is quite a flexible model, can operate at several levels.
Consider a system comprising:

- Database server (holds the database itself and the access / update logic),
- Database clients (user local interfaces to access the database),
- Authentication service (holds information to validate / authenticate users).

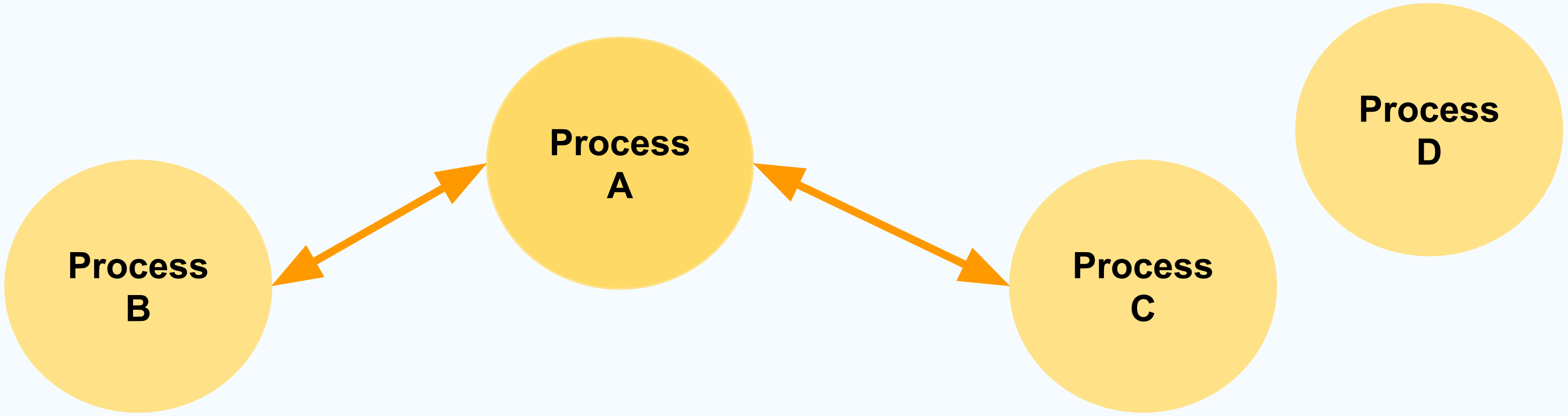
Client - Server model 2



Client - Server is quite a flexible model, can operate at several levels.
Consider a system comprising:

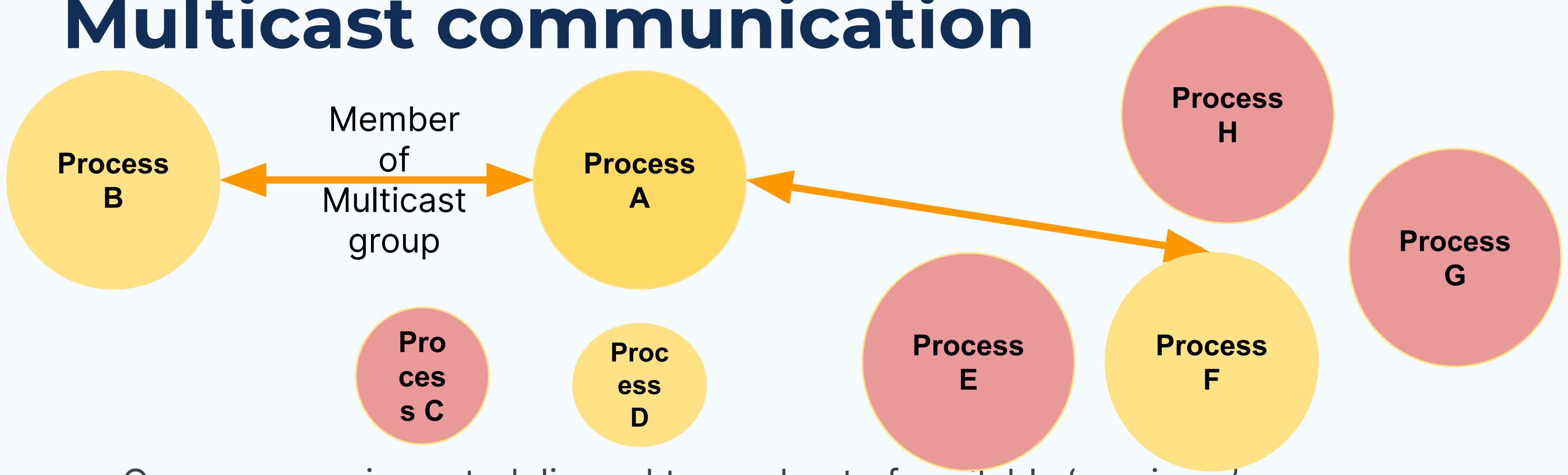
- Database server (holds the database itself and the access / update logic),
- Database clients (user local interfaces to access the database),
- Authentication service (holds information to validate / authenticate users).

Broadcast communication



- One message is sent, delivered to **all** available 'receivers'.
- Sender does not need to know how many, or identities of, receivers.
- Insecure (any process on the **appropriate port** can hear the message).
- Limited to LAN scope (routers block broadcasts).
- Inefficient (interrupts at all receivers even if not interested in the data).

Multicast communication

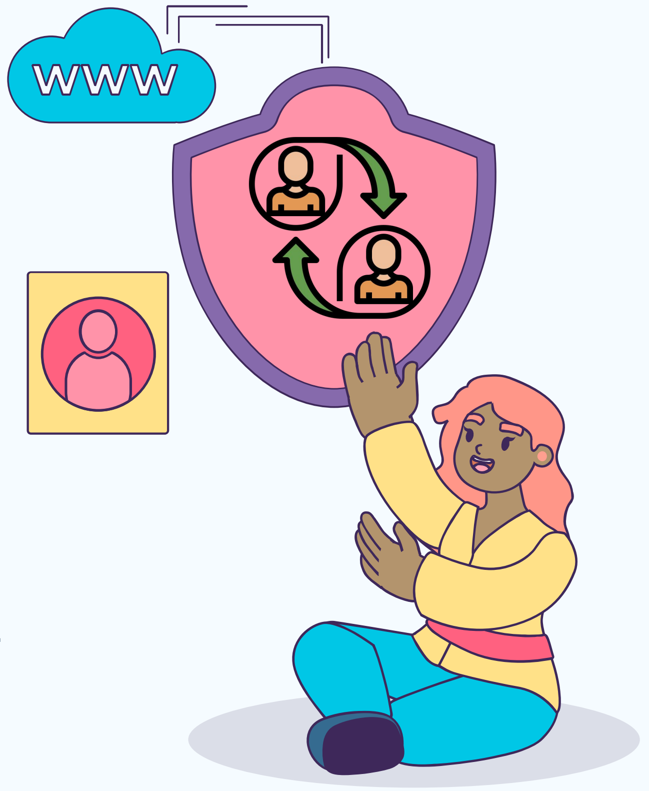


- One message is sent, delivered to a subset of available 'receivers'.
- Sender may not need to know how many, or identities of, receivers (depends on implementation).
- Insecure (any process on the appropriate port can hear the message).
- Usually limited to LAN scope (routers block some application multicasts, but routers use multicast when sharing routing information amongst themselves).
- Inefficient (interrupts at all receivers even if not interested in the data).

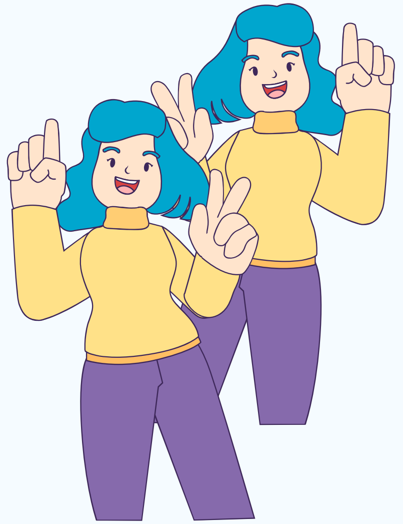
01b

Peer-to-peer

When two or more PCs are connected and share resources without going through a separate server computer.

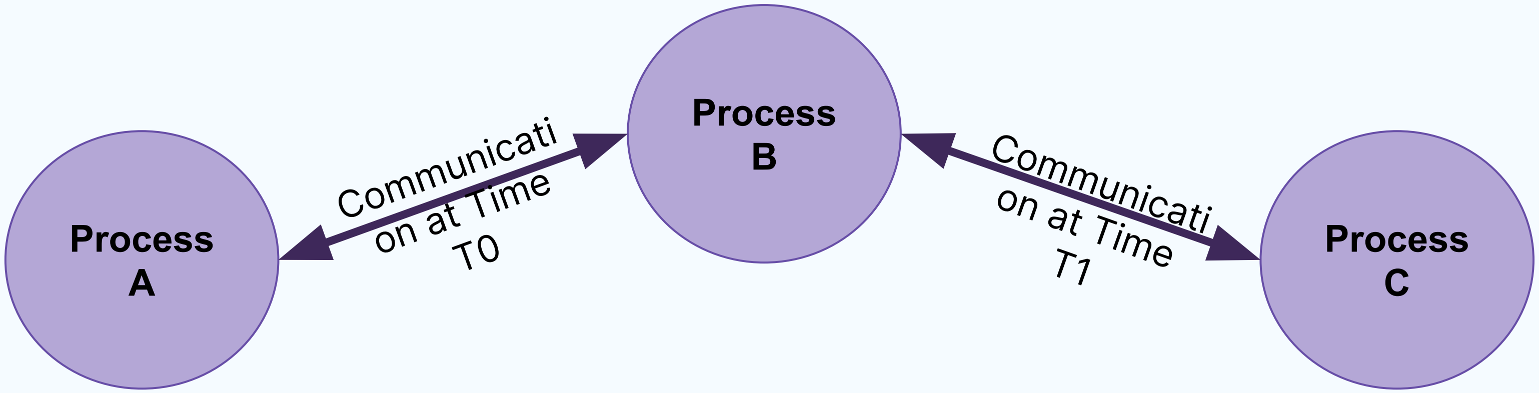


Peer-to-peer



- Peers operate with ‘equal standing’ each part (peer) of the application has the same basic function.
- Peers communicate to achieve their function (e.g. games, messaging, filesharing).
- Tends to be used in limited scope applications (typically single function) but where connection to remote ‘others’ needs to be simple and flexible.
- Either side may offer services to the other side.

Peer-to-peer model

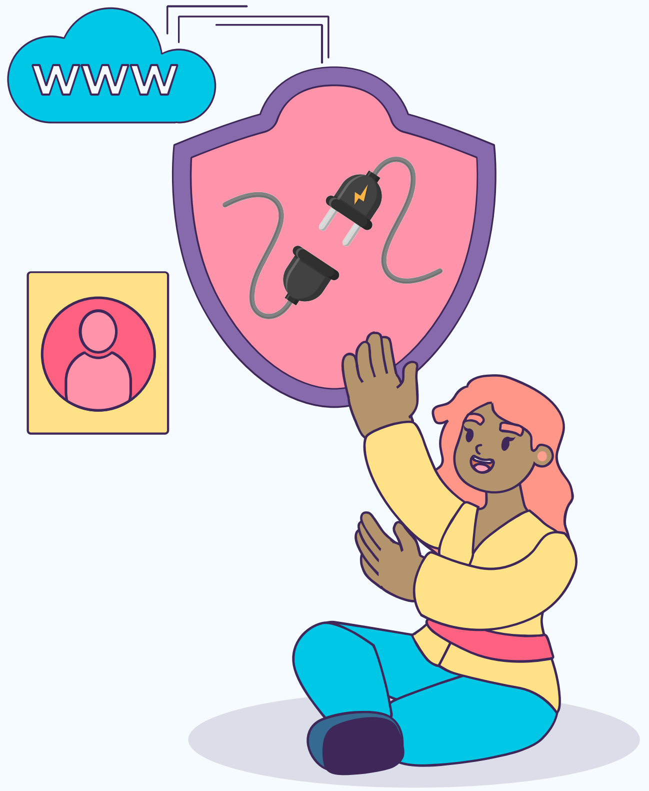


- Connectivity is ad hoc (i.e. it can be spontaneous, unplanned, unstructured).
- Peers can interact with others in any order, at any time.
- Well-suited to mobile applications on mobile devices (some games).
- Some applications (including some ‘sensor network’ applications) rely on ‘promiscuous’ connectivity of peers to pass information across a system.

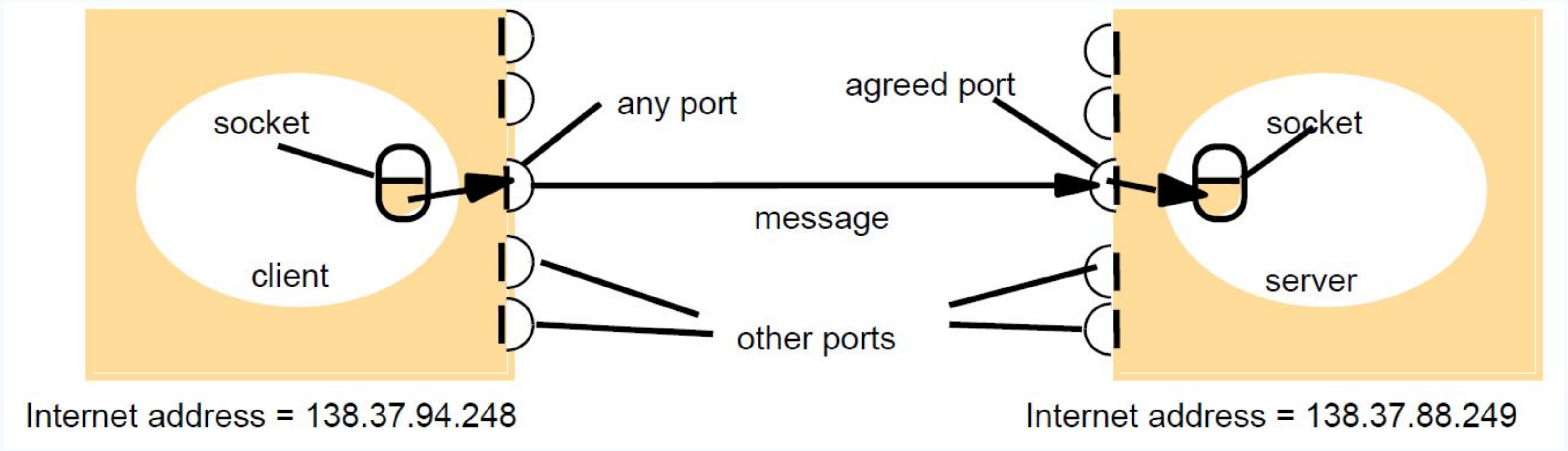
02

Sockets and Pipes

User Datagram Protocol (UDP)
Transmission Control Protocol (TCP)



Sockets and ports


socket
client
any port
agreed port
socket
server
message
other ports
Internet address = 138.37.94.248
Internet address = 138.37.88.249

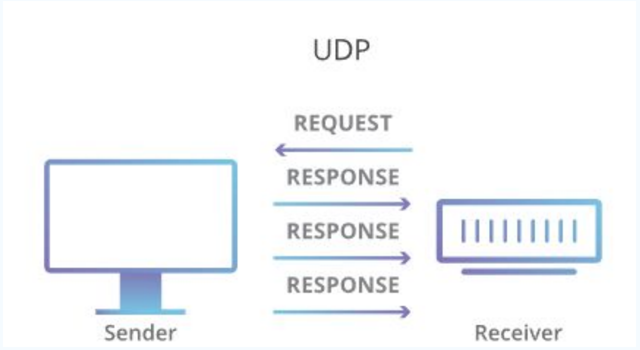
- A library of primitives to support transport layer communication with UDP and TCP.
- The most flexible of all techniques (because it is ‘low level’).
- Requires the application developer to deal with the communication aspects.

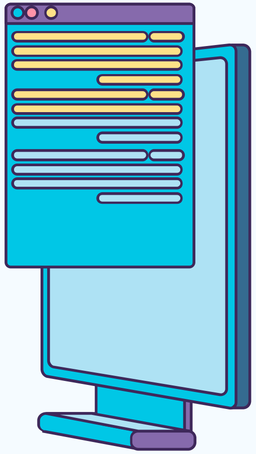
User Diagram Protocol (UDP)

Transmitted from a sending process to a receiving process without acknowledgement or retries. If a failure occurs, the message may not arrive.

Characteristics:

- Connectionless
- Unreliable
- Lightweight
- Point to point (Uni cast)
- Broadcast





Do you know ?

UDP act as an Internet standard,
Under the Request for Comments (RFC) publication,
enforced by Internet Engineering Task Force (IETF),y
Written by Jonathan Bruce Postel in 1980
https://www.rfc-editor.org/info/rfc768

UDP use cases

For service that is liable to occasional omission failures

Example:

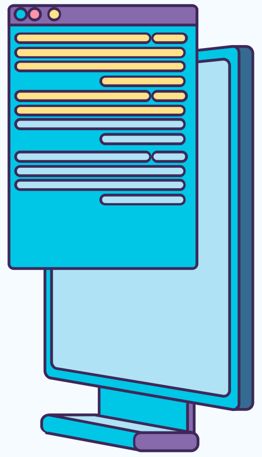
- Domain Name System
- Voice over IP (VOIP)

Activity:

Build the UDP model and send “Hello World” from client to server.

Question to ponder:

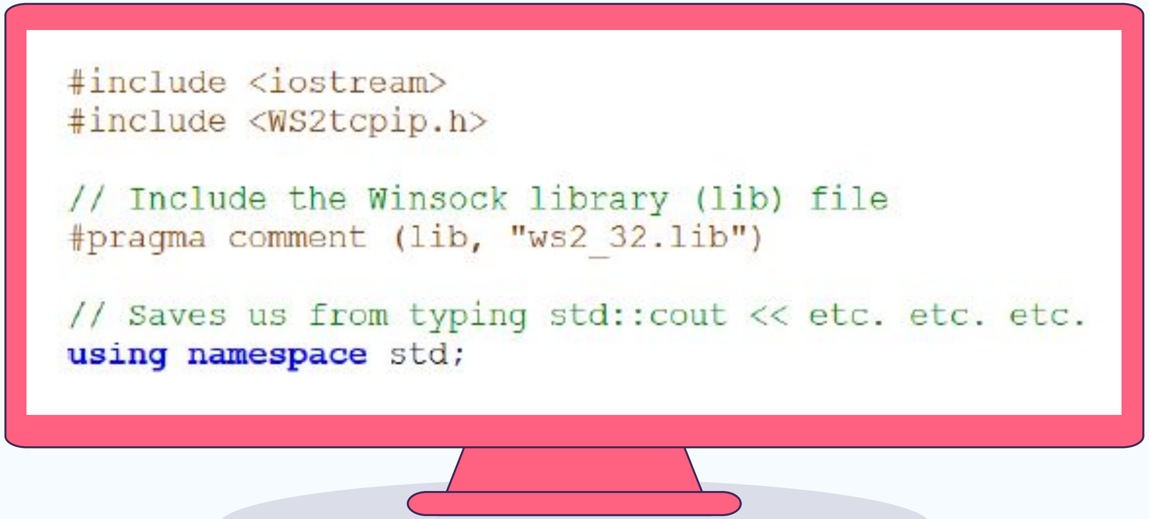
How to generate automatic reply from server to client ?



UDP - server : header

In CPP (under Microsoft Visual Studio, Microsoft Windows OS)

- To include the default Winsock library, for creating a new socket under Windows OS platform.

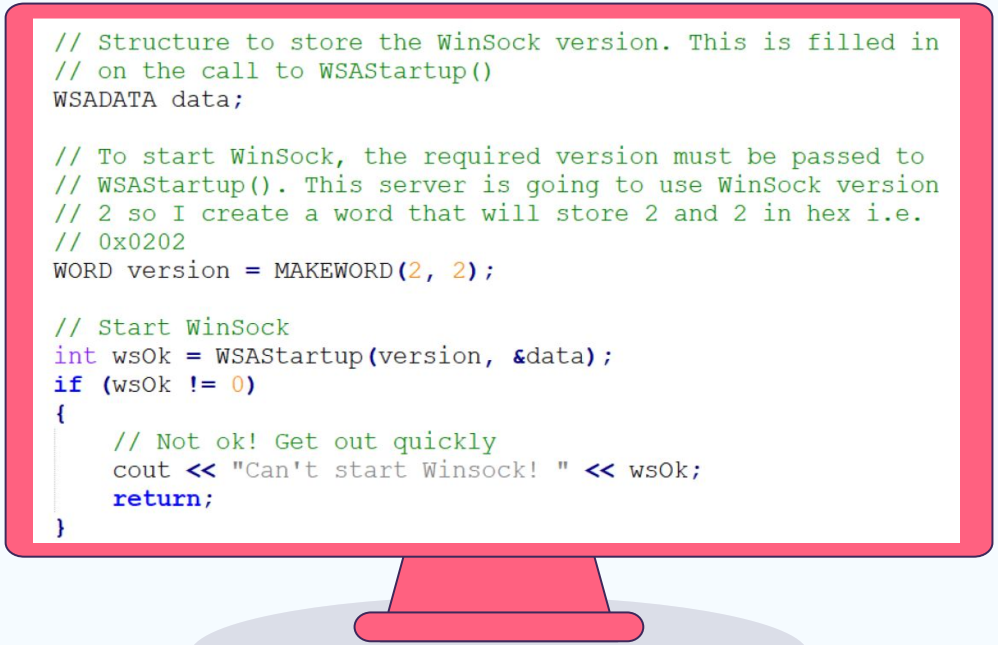


```
#include <iostream>
#include <WS2tcpip.h>

// Include the Winsock library (lib) file
#pragma comment (lib, "ws2_32.lib")

// Saves us from typing std::cout << etc. etc. etc.
using namespace std;
```

UDP - server : Winsock initialisation

In CPP (under Microsoft Visual Studio, Microsoft Windows OS)

- Set up a Winsock with defined WSADATA object, and version of WinSock to be utilised.

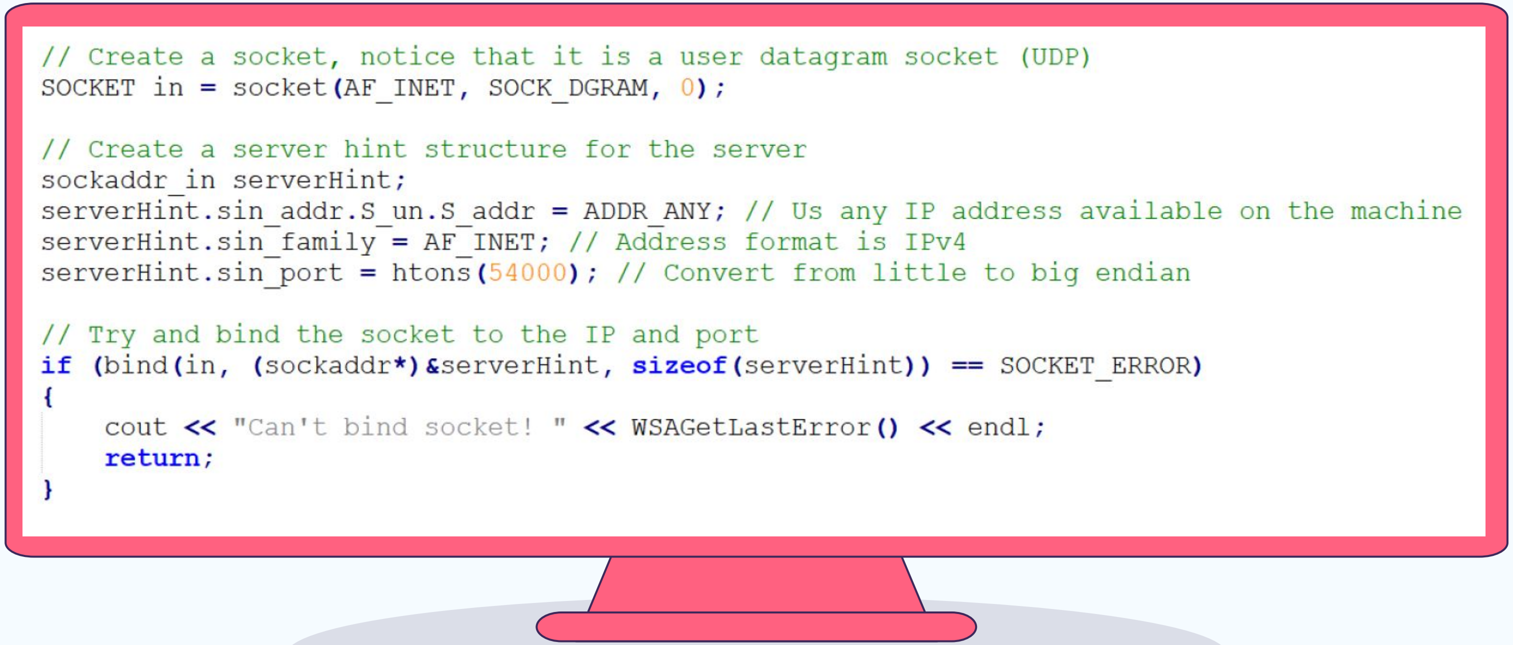


```
// Structure to store the WinSock version. This is filled in
// on the call to WSAStartup()
WSADATA data;

// To start WinSock, the required version must be passed to
// WSAStartup(). This server is going to use WinSock version
// 2 so I create a word that will store 2 and 2 in hex i.e.
// 0x0202
WORD version = MAKEWORD(2, 2);

// Start WinSock
int wsOk = WSAStartup(version, &data);
if (wsOk != 0)
{
    // Not ok! Get out quickly
    cout << "Can't start Winsock! " << wsOk;
    return;
}
```

UDP - server : Socket Creation & Binding

In CPP (under Microsoft Visual Studio, Microsoft Windows OS)

- A socket requests server ip address and port.



```cpp
// Create a socket, notice that it is a user datagram socket (UDP)
SOCKET in = socket(AF_INET, SOCK_DGRAM, 0);

// Create a server hint structure for the server
sockaddr_in serverHint;
serverHint.sin_addr.S_un.S_addr = ADDR_ANY; // Us any IP address available on the machine
serverHint.sin_family = AF_INET; // Address format is IPv4
serverHint.sin_port = htons(54000); // Convert from little to big endian

// Try and bind the socket to the IP and port
if (bind(in, (sockaddr*)&serverHint, sizeof(serverHint)) == SOCKET_ERROR)
{
    cout << "Can't bind socket! " << WSAGetLastError() << endl;
    return;
}
```

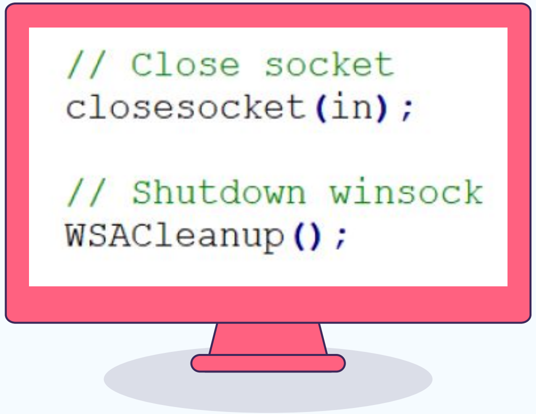
UDP - server : Main Loop Setup & Entry

- Initialize buffer type and size for sending and receiving data.
- Refer to the UDPserver.cpp file to see how to use Send and Recv function.

UDP - server : Close socket

In CPP (under Microsoft Visual Studio, Microsoft Windows OS)

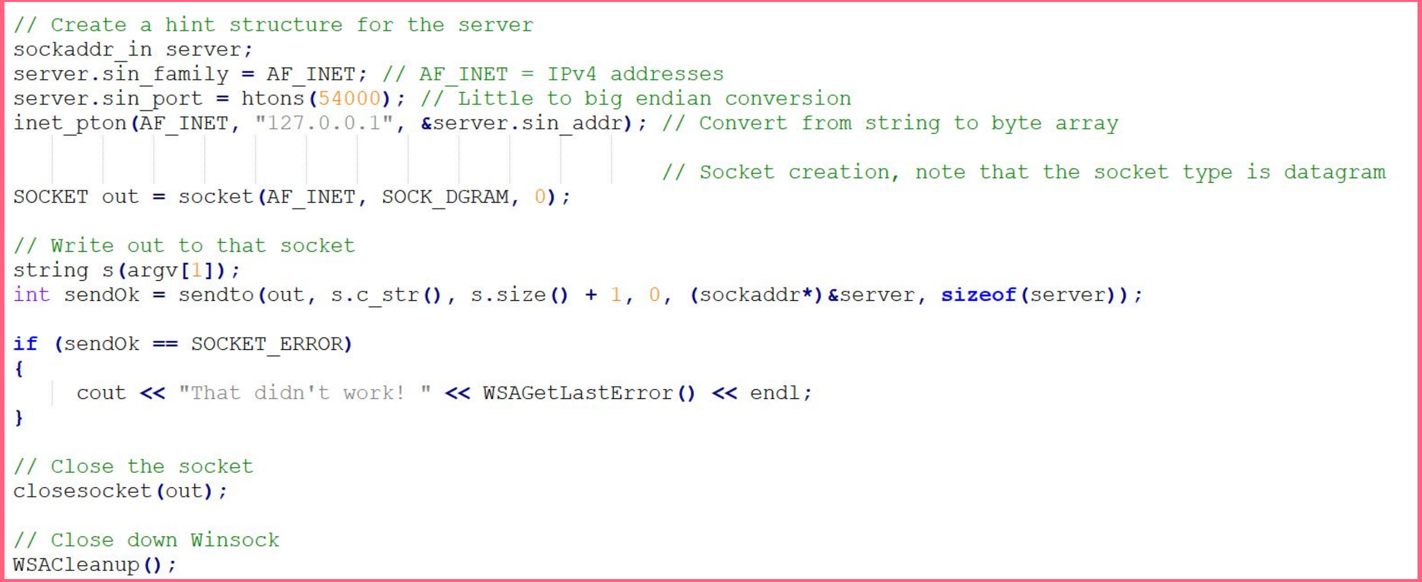
- The defined WSADATA object, will be clean up before ending the programme.



```
// Close socket
closesocket(in);

// Shutdown winsock
WSACleanup();
```

UDP - client : Send message

- Same as server for initialising and ending the socket network configurations.
- Send the message based on the data size.



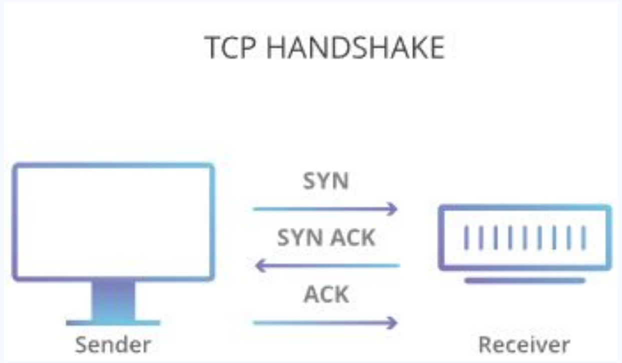
```
// Create a hint structure for the server
sockaddr_in server;
server.sin_family = AF_INET; // AF_INET = IPv4 addresses
server.sin_port = htons(54000); // Little to big endian conversion
inet_pton(AF_INET, "127.0.0.1", &server.sin_addr); // Convert from string to byte array

                                                   // Socket creation, note that the socket type is datagram
SOCKET out = socket(AF_INET, SOCK_DGRAM, 0);

// Write out to that socket
string s(argv[1]);
int sendOk = sendto(out, s.c_str(), s.size() + 1, 0, (sockaddr*)&server, sizeof(server));

if (sendOk == SOCKET_ERROR)
{
    cout << "That didn't work! " << WSAGetLastError() << endl;
}

// Close the socket
closesocket(out);

// Close down Winsock
WSACleanup();
```

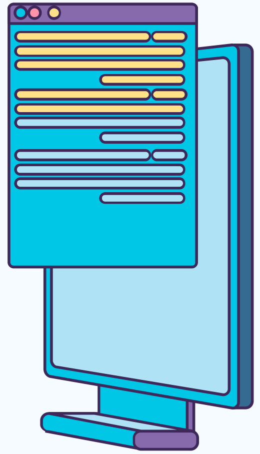
Transmission Control Protocol (TCP)

Stream communication assumes that when a pair of processes are establishing a connection, one of them plays the client role and the other plays the server role , but thereafter they could be peers

Characteristics:

- Connection oriented
- Reliable
- Relatively high overheads (larger header, acks , more processing overhead)
- Point to point





Do you know ?

TCP act as an Internet standard,
Under the Request for Comments (RFC) publication,
enforced by Internet Engineering Task Force (IETF).y
Written by Jonathan Bruce Postel in 1980
https://www.rfc-editor.org/info/rfc761
Latest: https://datatracker.ietf.org/doc/html/rfc9293

TCP use cases

HTTP : The Hypertext Transfer Protocol

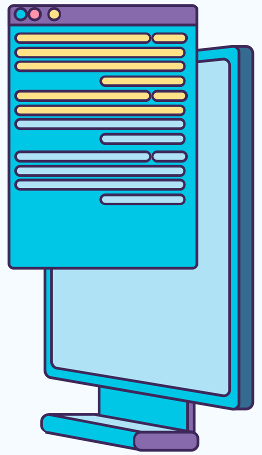
FTP : The File Transfer Protocol

Telnet : Telnet provides access by means of a terminal session to a remote computer

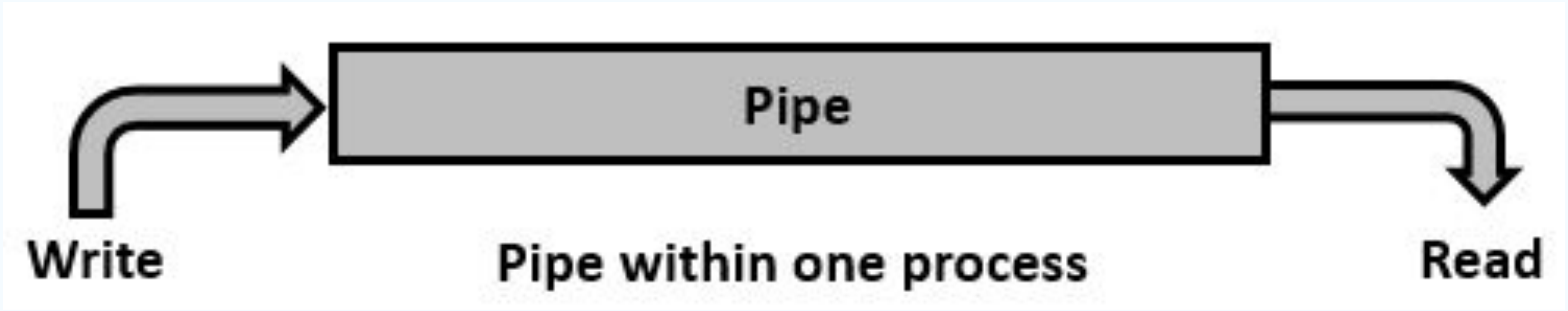
SMTP : The Simple Mail Transfer Protocol

Activity:

Demonstration on TCP



Pipes

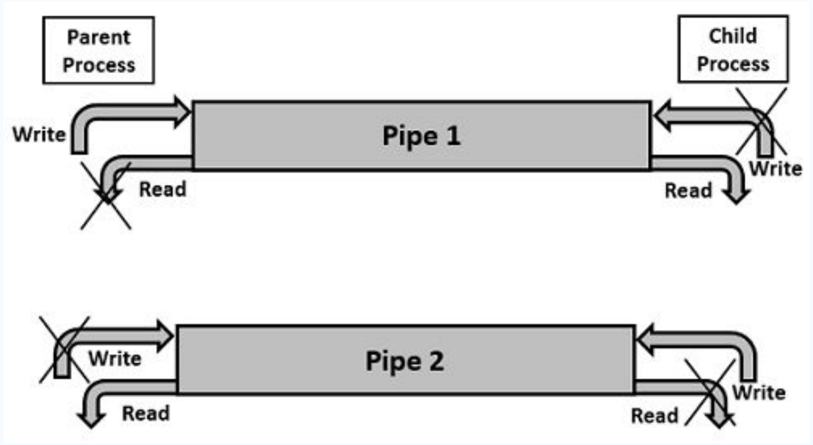


- Unidirectional
- Standard producer consumer mechanism
- Parent-child relationship between communicatingprocesses
- Once communication is over and processes terminated, the ordinary pipe ceases to exist

Pipes properties (with example)

- More powerful with assigned pipe's name
- Do not require parent child relationship
- Once a named pipe is created, multiple non related processes can communicate over it
- Must be explicitly deleted

Example:
Pipe communication with two pipes
tutorialspoint.com



03

Communication Issue

Message Passing, Remote Procedure Call, Remote Method Invocation, Middleware.

Message Passing

- This is a simple unstructured form of communication in which messages are passed from a sender to a receiver.
- Conceptually this is very similar to datagram communication provided by UDP.

Message Passing Interface (MPI)

- ➢ A specific implementation of message passing.
- ➢ Very popular in parallel processing applications.
- ➢ Has a large number of enhancements (beyond for example UDP datagrams), added specifically to support the synchronisation aspects of communication in parallel applications.

Message Passing Model (Abstraction)

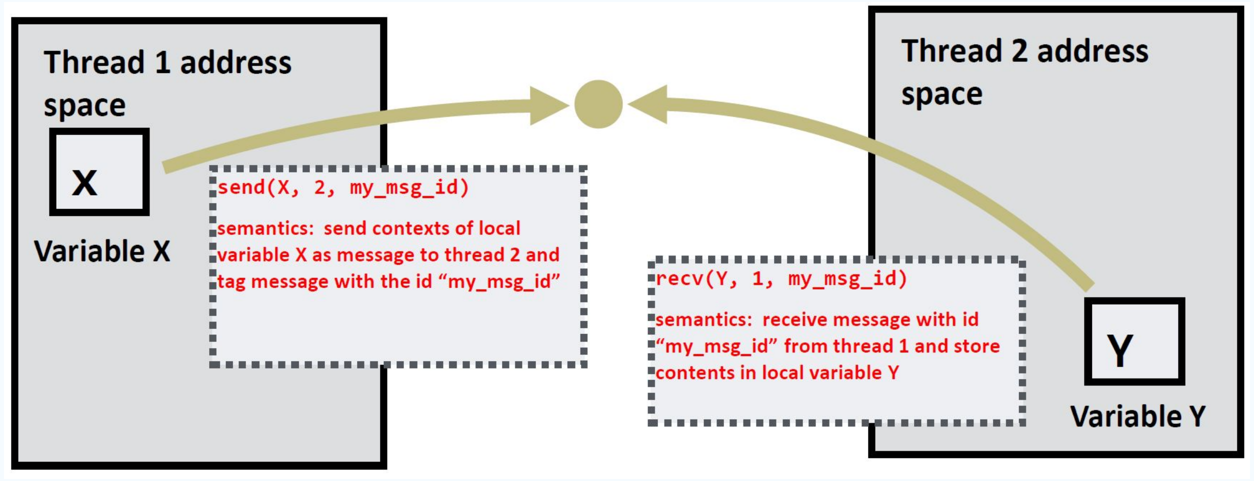
Threads operate within their own private address spaces

Threads communicate by sending/receiving messages

send : specifies recipient, buffer to be transmitted, and optional message identifier (“tag”)

receive : sender, specifies buffer to store data, and optional message identifier

Sending messages is the only way to exchange data between threads 1 and 2



```
send(X, 2, my_msg_id)
```

semantics: send contexts of local variable X as message to thread 2 and tag message with the id “my_msg_id”

```
recv(Y, 1, my_msg_id)
```

semantics: receive message with id “my_msg_id” from thread 1 and store contents in local variable Y

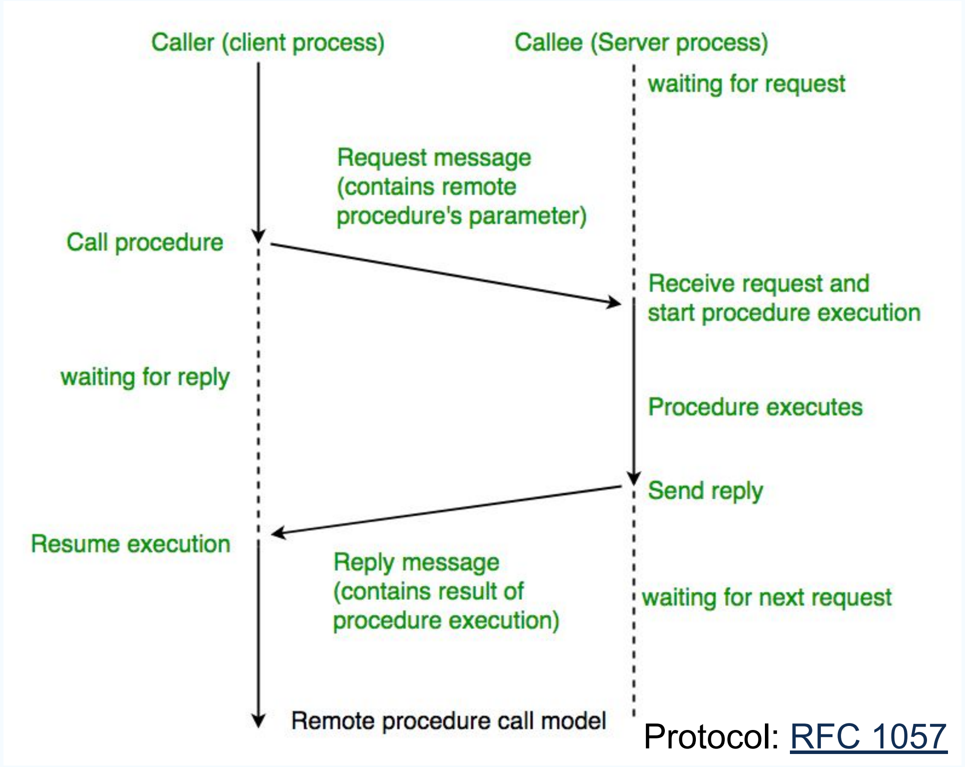
Message Passing Systems

Popular software library: **MPI (message passing interface)**
Hardware need not implement system wide loads and stores to execute message passing programs (need only be able to communicate messages)
◦ Can connect commodity systems together to form large parallel machine (message passing is a programming model for clusters).

Source:
Technicians working on a cluster consisting of many computers working together by sending messages over a network.

MEGWARE Computer GmbH - http://www.megware.com
Chemnitzer Linux Cluster (CLIC) an der Technischen Universität Chemnitz

Remote Procedure Call (RPC)



Caller (client process)
Callee (Server process)
waiting for request
Request message (contains remote procedure's parameter)
Call procedure
Receive request and start procedure execution
waiting for reply
Procedure executes
Send reply
Resume execution
Reply message (contains result of procedure execution)
waiting for next request
Remote procedure call model

Protocol: [RFC 1057]

For procedural code such as C, Fortran, Pascal (i.e. not object oriented).

One side makes a procedure call to a procedure within the other process.

Proxies are provided so that the caller actually makes a local call.

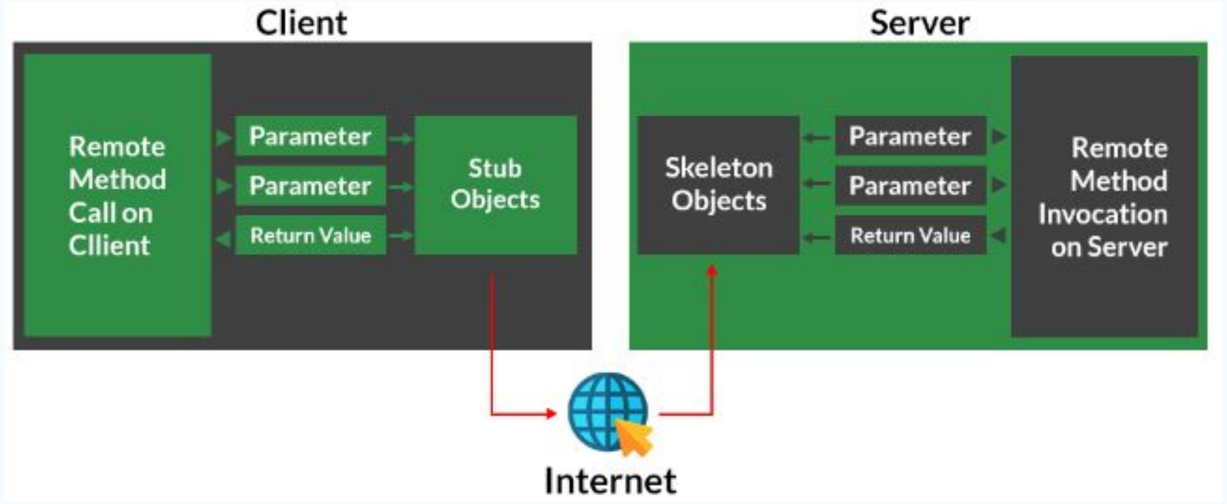
The callee , likewise is given the impression that the caller is a local

This greatly simplifies the application developer's work - they are not concerned with the communication aspects in great detail.

More Example : IBM

Exercise : Dave Marshall

Remote Method Invocation (RMI)



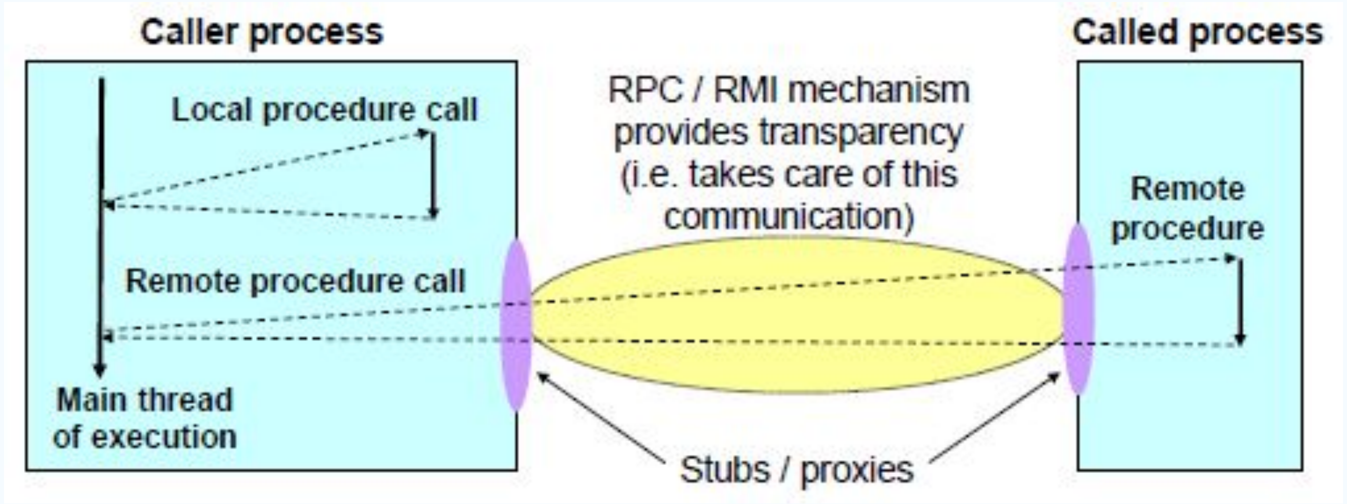
The object oriented version of RPC.

For object oriented languages such as (java, C++).

One side makes a method call to a method within the other process.

More Example : RMI in Java
Exercise : Java Card

RPC / RMI model

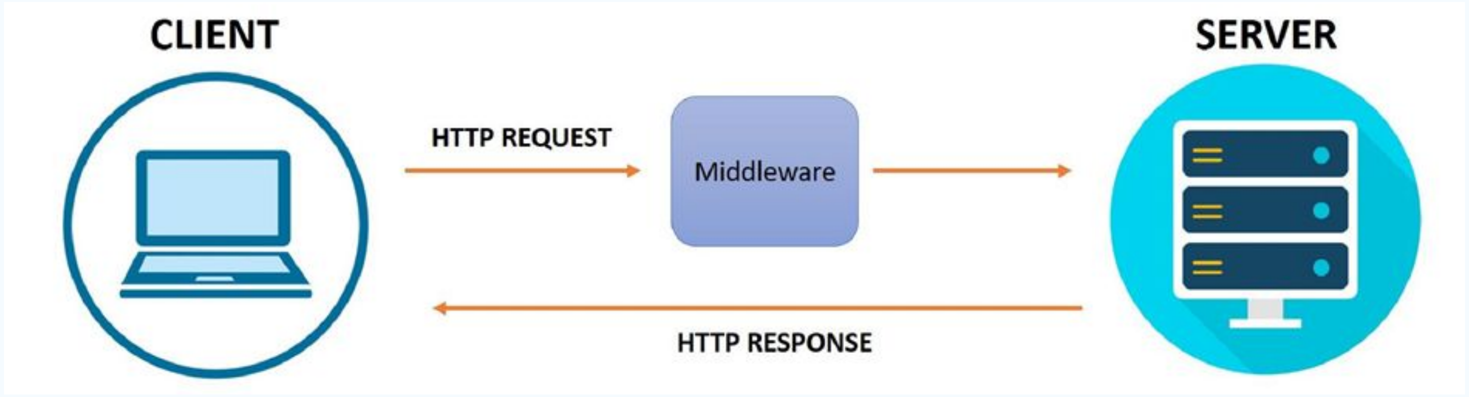


A process can call a procedure (or invoke a method) in another process

To the programmer this is made to look as much like a local procedure call as possible.

The RPC / RMI mechanisms provide local ‘stub’ mechanisms which act as communication proxies, so all communication appears to be ‘process local’.

Middleware



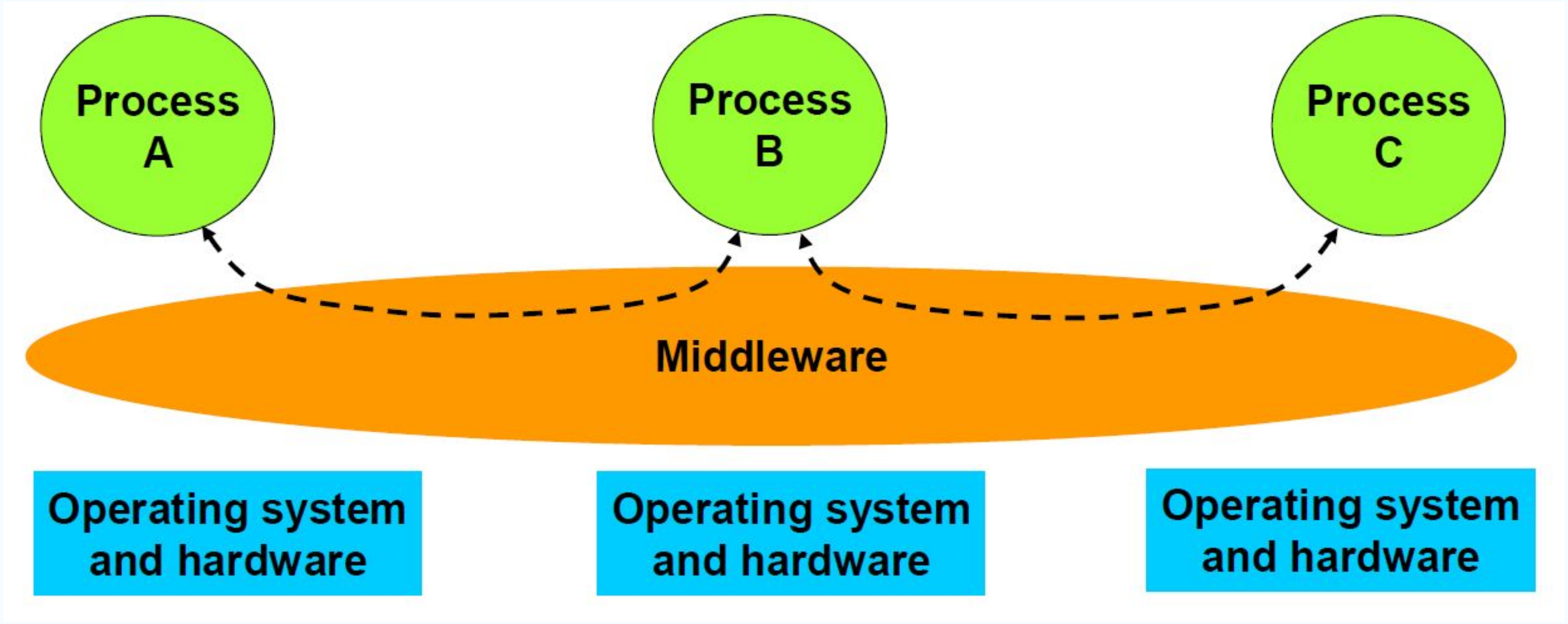
A software ‘layer’ that sits between processes and the network.

Can comprise part of the operating system and some special services.

Services include location and remote invocation services.

The underlying communication is usually sockets.

Middleware Concept


Process A
Process B
Process C
Middleware
Operating system and hardware
Operating system and hardware
Operating system and hardware

More Description: Amit Saha
Exercise : Middleware Web Apps with Go, Js, Python

Middleware

Application level communication resembles RMI / RPC but with greater transparency:

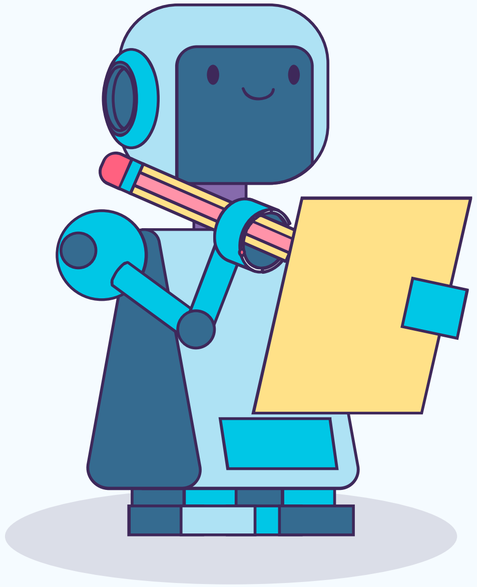
- Application developer does not need to be concerned with the communication mechanism, OR the location of processes.
- Supports relocation of objects transparent to application.

As shown in the conceptual diagram, processes communicate via the middleware layer.

The middleware hides details of location (a process need not know where the other process is).

The middleware itself is implemented with sockets or RMI

The middleware consists of processes and services on each computer, but acts like a continuous ‘layer’ or ‘channel’ that is spread across all computers.



Thank you

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**