Distributed Programming

Lecture 03 - Distributed Systems Architectures and Inter-process Communication

Edirlei Soares de Lima
<edirlei.lima@universidadeeuropeia.pt>

What is the Architecture of a System?

- The architecture of a system is its structure in terms of separately specified components and their interrelationships.
  - In distributed system, the architecture also identifies the location of the components in the network.
  - The architecture impacts on the performance, reliability and security of the system.

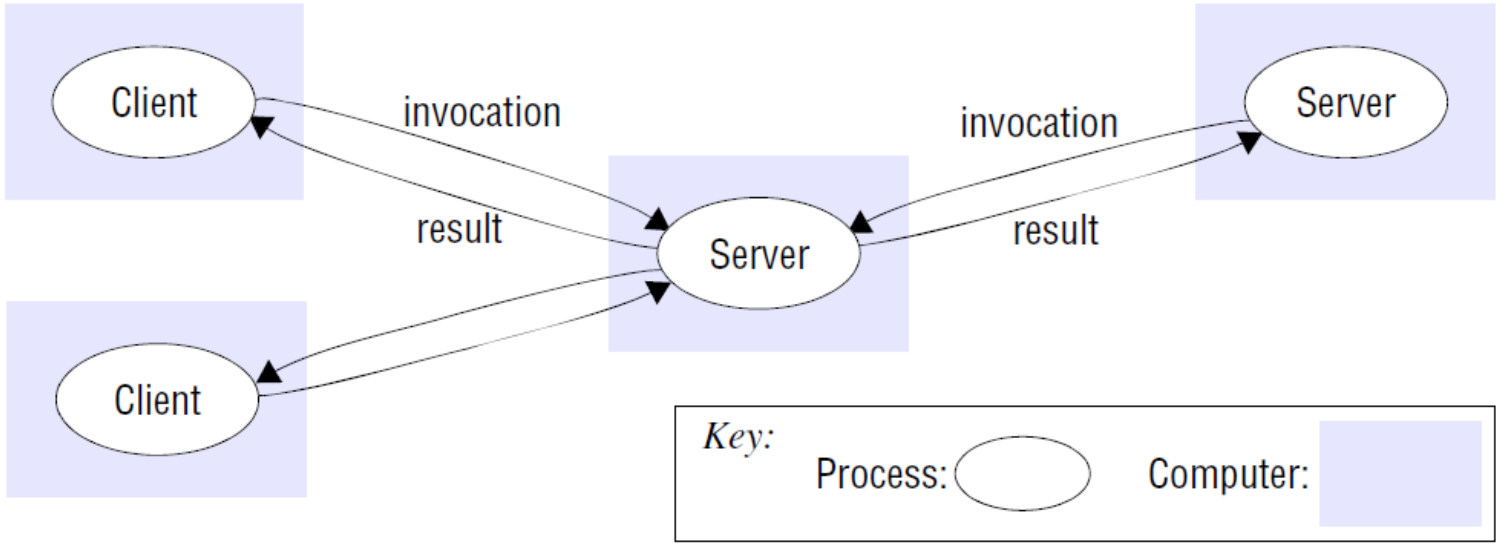
- **Key design questions**:
  - What are the entities of the system?
  - How do they communicate and what communication paradigm is used?
  - What roles and responsibilities do they have in the overall architecture?

Key Design Questions

- **What are the entities of the system?**
  - Modules, components, objects, web services, …
- **How do they communicate and what communication paradigm is used?**
  - Inter-process communication: sockets programming.
  - Remote invocation: request-reply protocols, remote procedure calls (RPC), remote method invocation (RMI).
  - Indirect communication: group communication, message queues, distributed shared memory.
- **What roles and responsibilities do they have in the overall architecture?**
  - Client-server, peer-to-peer, …

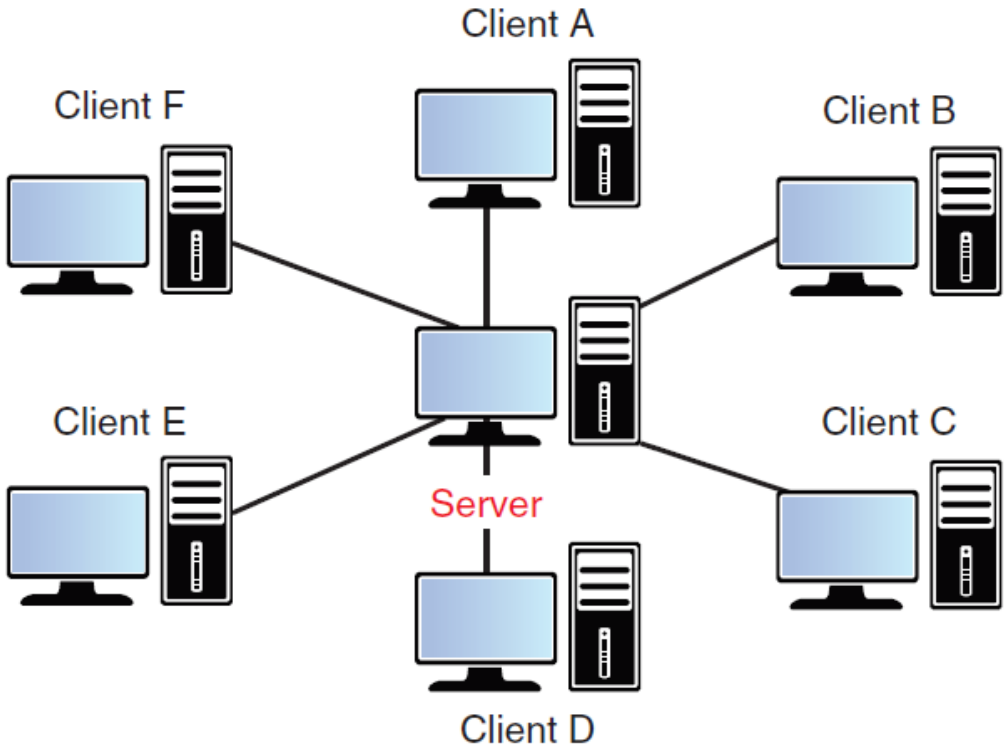
Client-Server Model

- The Client-Server Model comprises two types of nodes:
  - Client: program that makes requests to a process running on a server.
  - Server: program that runs the operations requested by the clients, sending back the result.



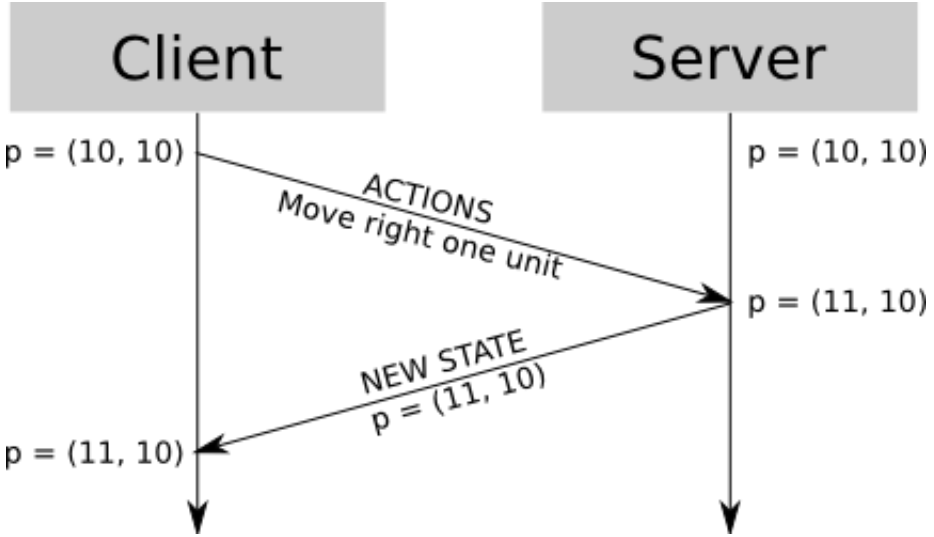
Client-Server Model

- Each client only communicates with the server, while the server is responsible for communicating with all of the clients.



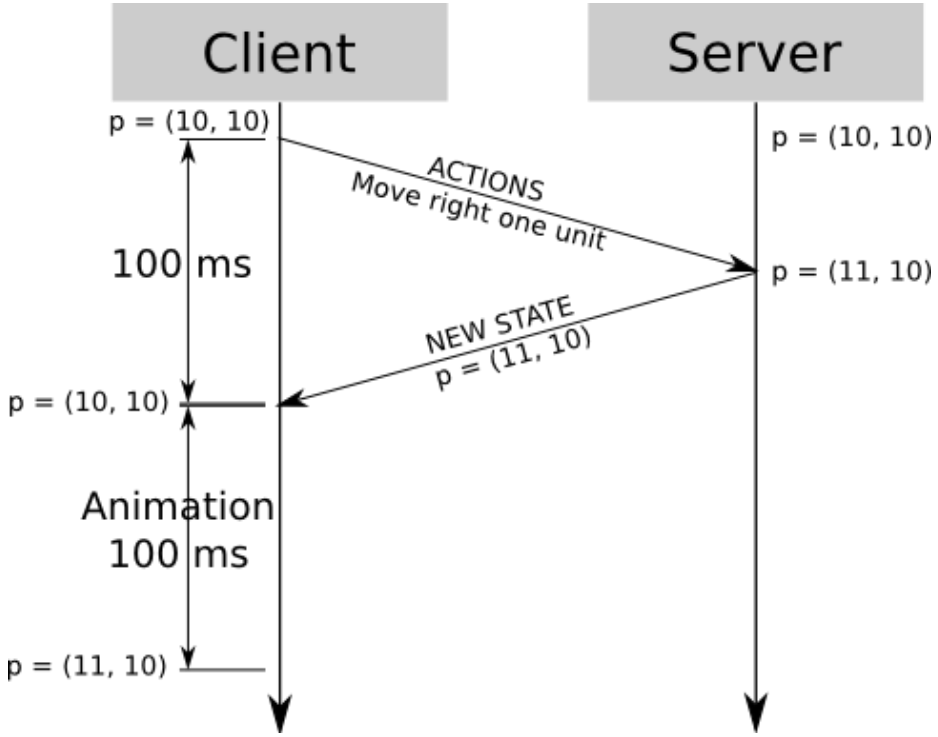
Client-Server Model in Games

- Authoritative servers (to avoid cheating): the server's simulation of the game is considered to be correct. If the client ever finds itself in disagreement with the server, it should update its game state based on what the server says is the game state.



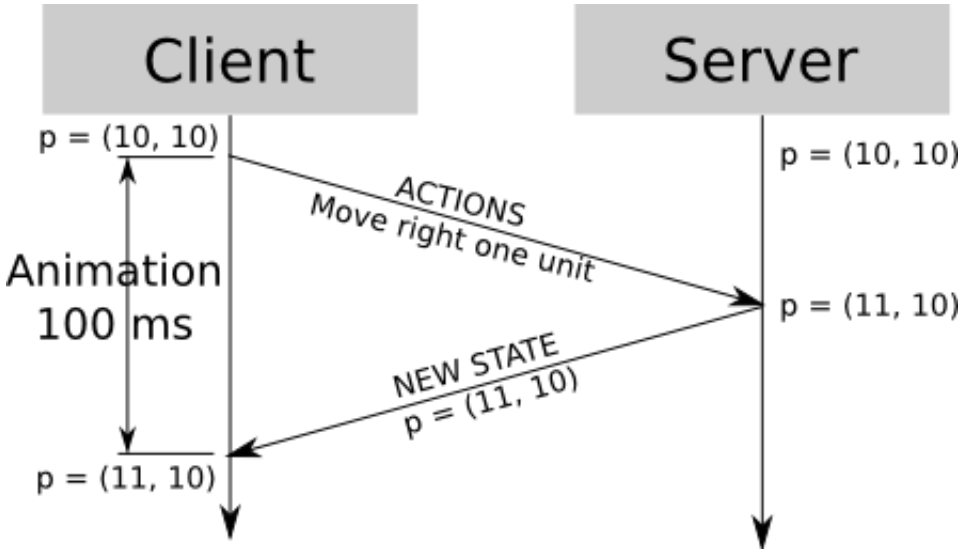
Client-Server Model in Games

- Authoritative servers and network delays problem: simple implementations may make the game quite unresponsive for the player.



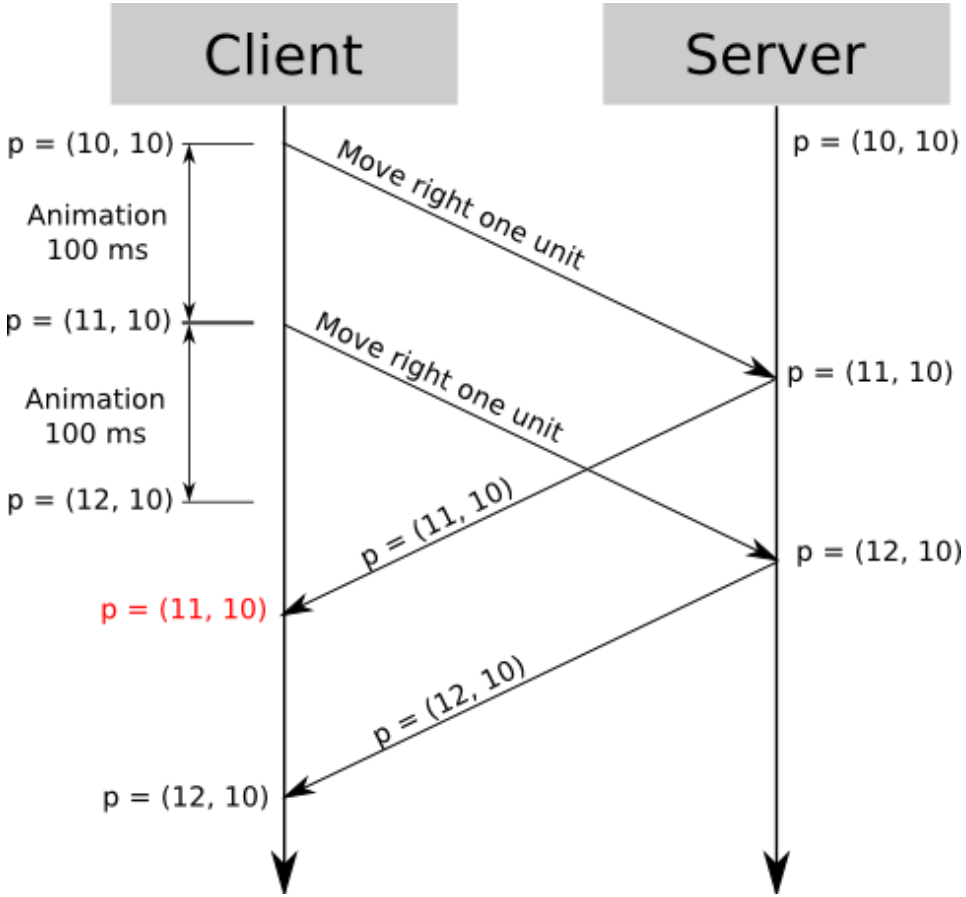
Client-Server Model in Games

- Client-side prediction: the client can assume the inputs send to the server will be executed successfully.



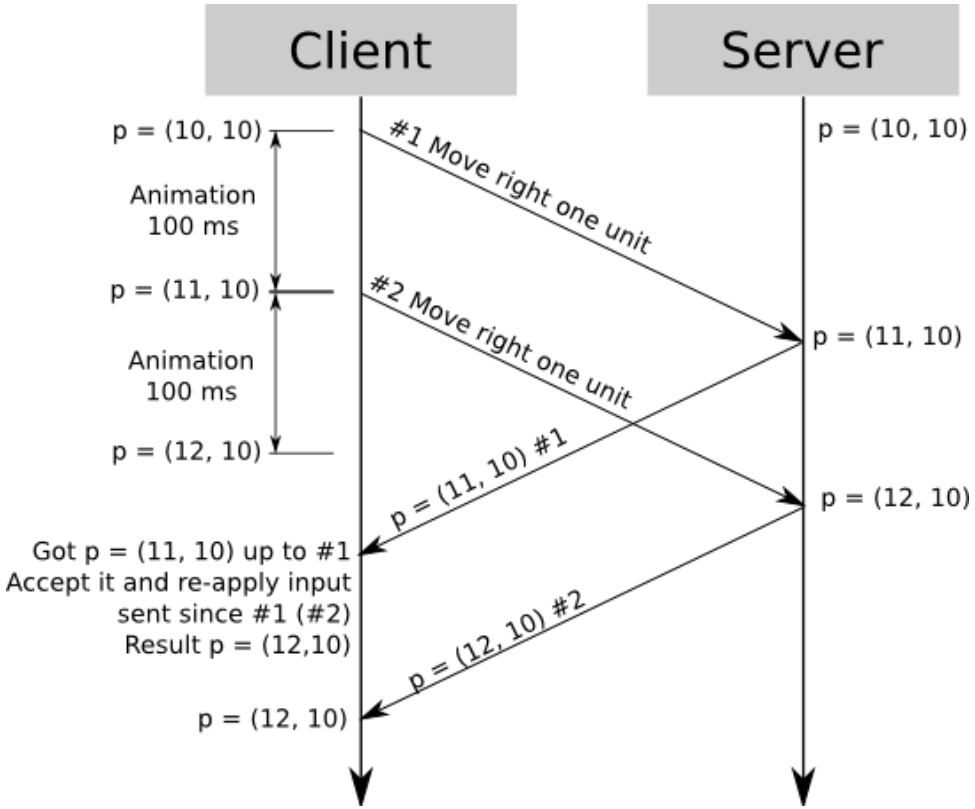
Client-Server Model in Games

- Synchronization issues:



Client-Server Model in Games

- Server reconciliation: the client sees the game world in present time, but because of lag, the updates it gets from the server are actually the state of the game in the past.



Client-Server Model

- The Client-Server Model is the most common architecture used in distributed systems (including games).

- **Advantages:**
  - Simple interaction and communication makes its implementation easy.
  - Security is mainly focused on the server side.

- **Problems:**
  - The server is the single point of failures.
  - Scalability problems (the server can become a bottleneck).

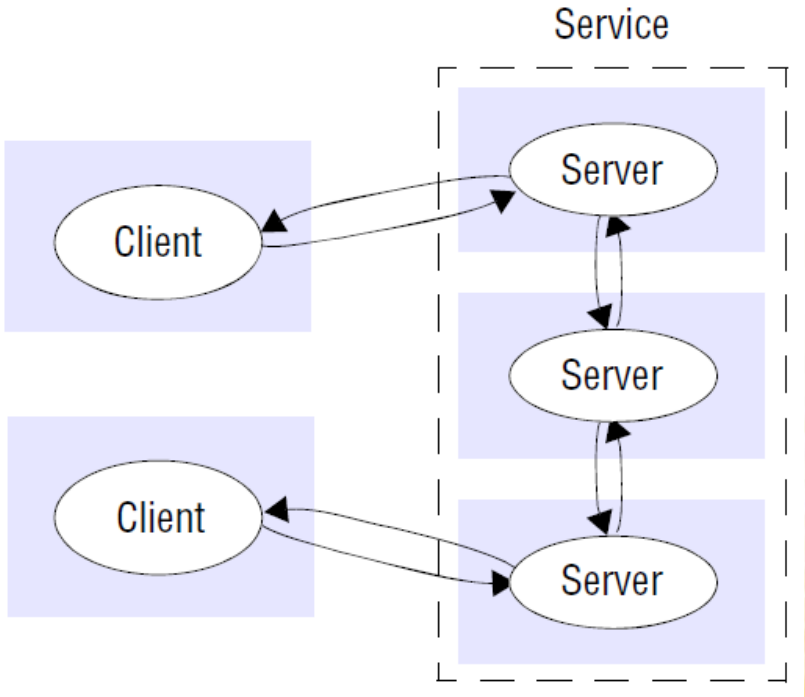
- There are some variants that can reduce the problems.

Client-Server Model Variants

- Multiple server processes: identical servers capable of answering the same requests.

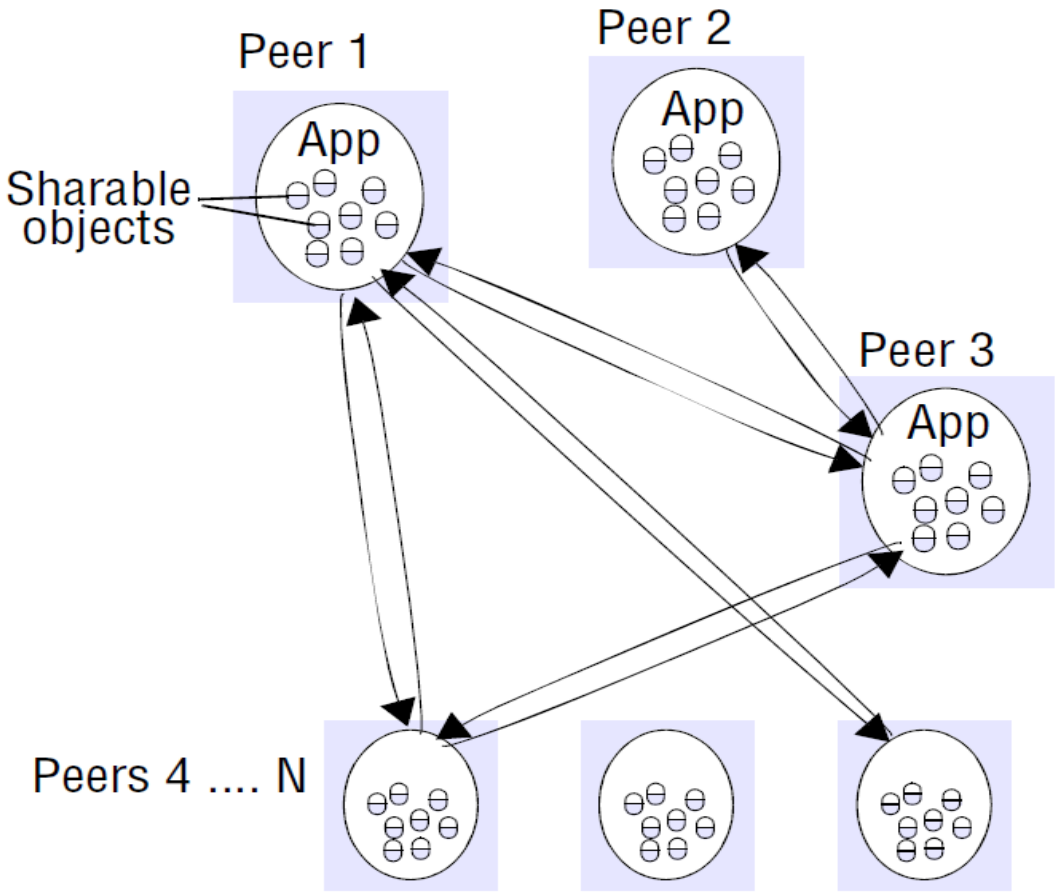
- **Advantages:**
  - Distributes load and improves performance.
  - There is no single point of failure.

- **Problems:**
  - It is necessary to keep the state of the server coherent in all replicas.
  - It is necessary to recover from partial failures of single servers.



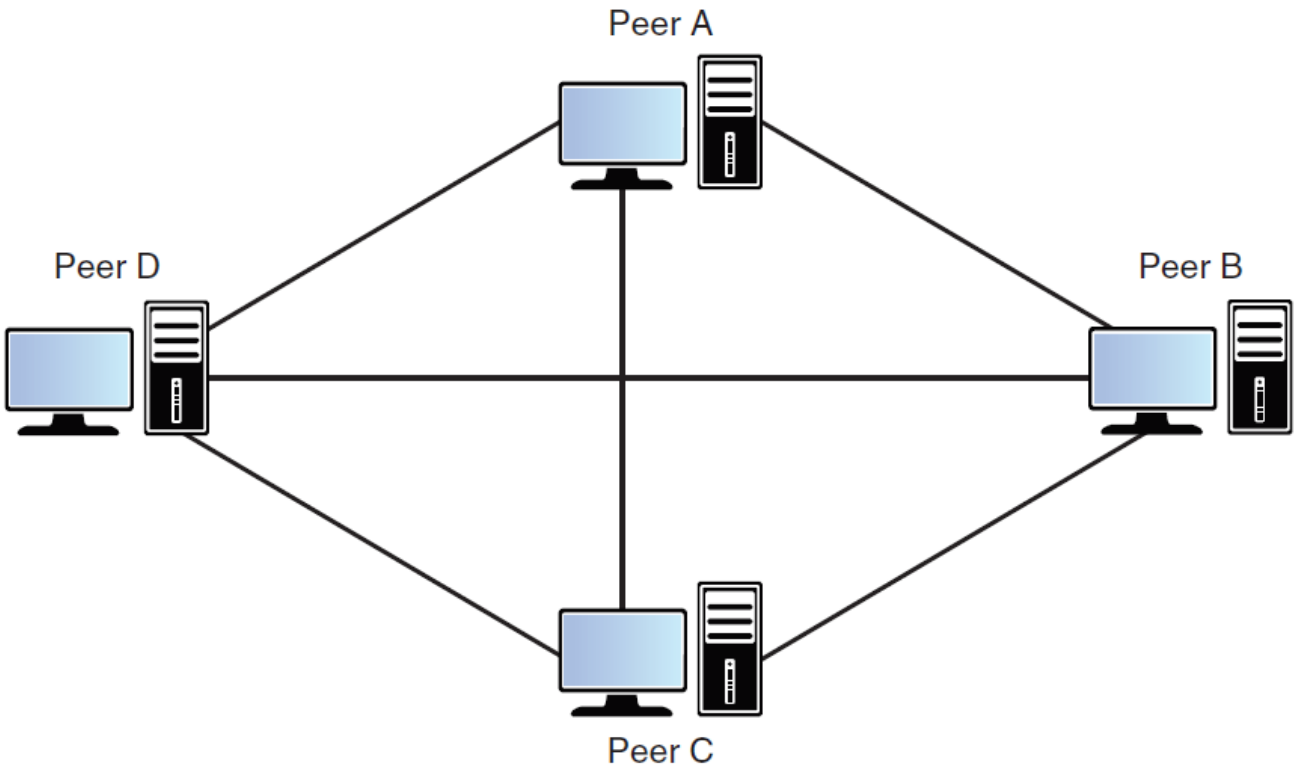
Peer-to-Peer Model (P2P)

- All of the processes play similar roles, interacting cooperatively as peers without any distinction between client and server.



Peer-to-Peer Model (P2P)

- All participating processes run the same program and offer the same set of interfaces to each other.



Peer-to-Peer Model in Games

- The concept of authority is much more nebulous in a peer-to-peer game.
  - Usually, peer-to-peer games share all player actions across every peer, and every peer simulates these actions.

- There is less latency (there is no intermediary server between clients), but the latency that remains leads to the greatest challenge in peer-to-peer games: ensuring that all peers remain synchronized with each other.
  - Furthermore, it is important to ensure that the game state is consistent between all peers.

Peer-to-Peer Model

- Peer-to-peer gaming architectures are promising, but they also come with several of challenges.

- **Advantages:**
  - There is not a single point of failure.
  - Greater potential for scalability.

- **Problems:**
  - Complex interaction (when compared with a client/server system) leads to complex implementations.
  - Security problems.

Inter-process Communication

- Inter process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.

- In distributed systems, this communication is done by sending data over a network interface to another computer/process on the network.
  - The basis for all computer networks is the packet-switching technique developed in the 1960s, which enables data packets addressed to different destinations to share a single communications link.
  - The principles on which computer networks are based include protocol layering, packet switching, routing and data streaming.

Communication Protocols

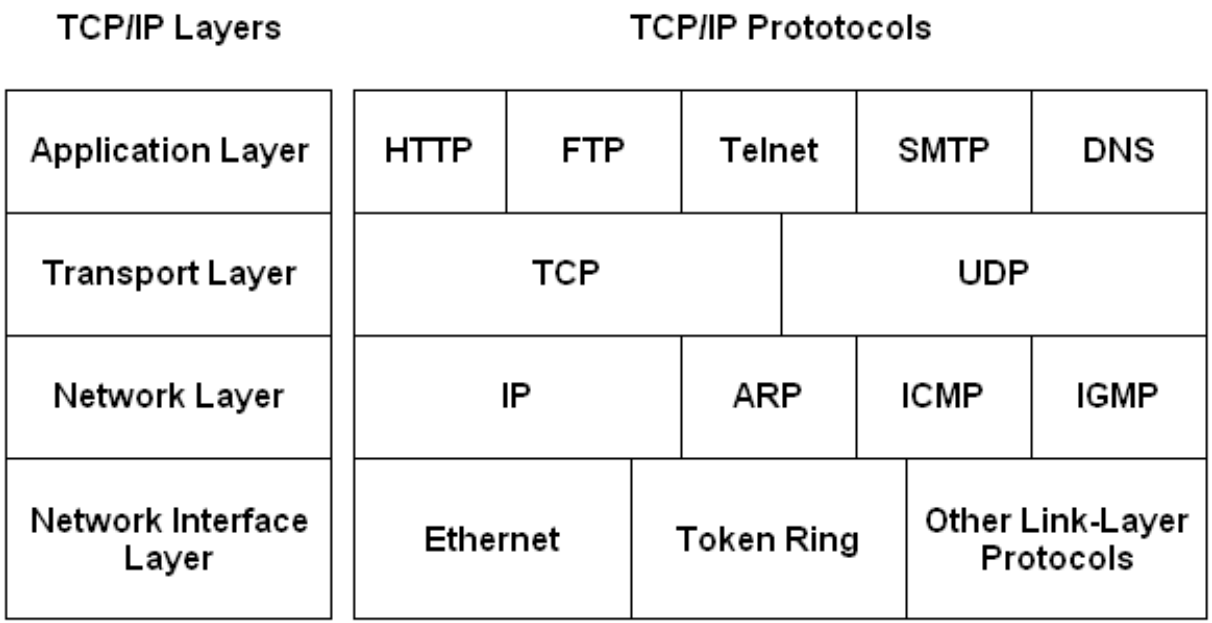
- The term protocol is used to refer to a set of rules and formats to be used for communication between processes in order to perform a given task.

- The definition of a protocol has two important parts:
  1. A specification of the sequence of messages that must be exchanged;
  2. A specification of the format of the data in the messages.

- The Internet protocol suite (known as TCP/IP) is the conceptual model and set of communications protocols used on the Internet and general computer networks.

Communication Protocols

- **The TCP/IP has four main layers:**

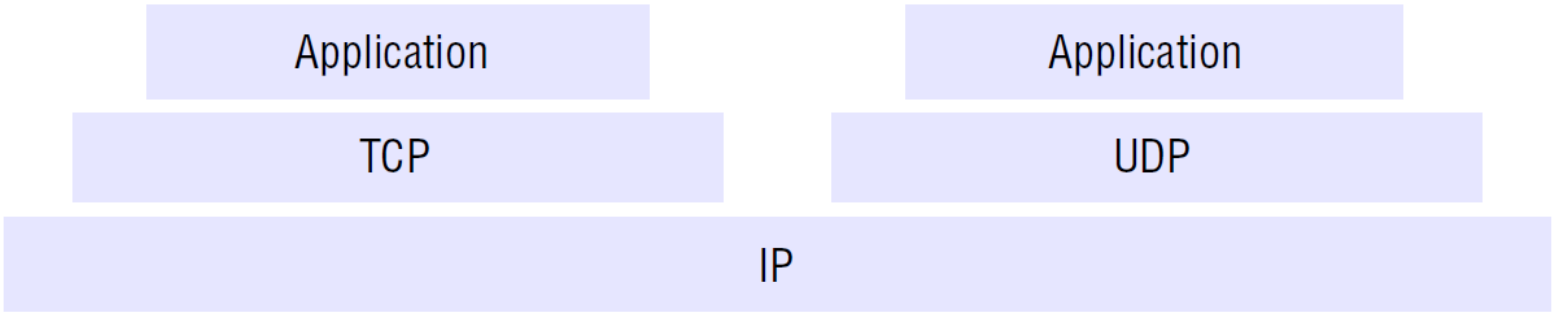


- The transport layer is the lowest level at which messages are handled.
  - Messages are addressed to communication ports attached to processes.

Communication Protocols

- **There are two transport protocols:**
  - TCP (Transport Control Protocol): reliable connection-oriented protocol. It ensures the delivery of packets between hosts.
  - UDP (User Datagram Protocol): datagram protocol that does not guarantee reliable transmission.

- **Programmer's conceptual view of the TCP/IP:**



Windows Sockets API (WSA)

- Defines a standard interface between a Windows TCP/IP client application and the underlying TCP/IP protocol.

- **Main functions:**
  - `WSAStartup`: initiates the Winsock DLL;
  - `socket`: creates a socket that is bound to a transport service provider;
  - `bind`: associates a local address with a socket;
  - `listen`: places a socket in a state of listening for incoming connection;
  - `accept`: permits an incoming connection attempt on a socket;
  - `connect`: establishes a connection to a specified socket;
  - `recv`: receives data from a connected socket;
  - `send`: sends data on a connected socket;
  - `closesocket`: closes an existing socket;
  - `WSACleanup`: terminates use of the Winsock DLL.

TCP Server – Example

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>
#include <thread>

using namespace std;
#define MAXPENDING 5
#define RCVBUFSIZE 1024

void HandleClientThread(SOCKET client, char* clientstr){
  char buffer[RCVBUFSIZE];
  cout << "Client " << clientstr << " connected!" << endl;
  while (recv(client, buffer, sizeof(buffer), 0) > 0){
    cout << "Client " << clientstr << " send: " << buffer << endl;
    memset(buffer, 0, sizeof(buffer));
  }
  if (closesocket(client) == SOCKET_ERROR){
    cout << "closesocket() failed" << endl;
  }
  cout << "Client " << clientstr << " disconnected." << endl;
}
```

The lib **ws2_32.lib** must be added to the linker input.

TCP Server – Example

```
int main()
{
  SOCKET server;
  SOCKADDR_IN server_addr, client_addr;
  WSADATA wsaData;

  if (WSAStartup(MAKEWORD(2, 0), &wsaData) != NO_ERROR)
  {
    cout << "WSAStartup() failed" << endl;
    exit(EXIT_FAILURE);
  }
  if ((server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
      INVALID_SOCKET){
    cout << "socket() failed" << endl;
    exit(EXIT_FAILURE);
  }

  memset(&server_addr, 0, sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  server_addr.sin_port = htons(8856);
```

TCP Server – Example

```
if (::bind(server, (struct sockaddr *) &server_addr,
    sizeof(server_addr)) == SOCKET_ERROR){
  cout << "bind() failed" << endl;
  exit(EXIT_FAILURE);
}

if (listen(server, MAXPENDING) == SOCKET_ERROR)
{
  cout << "listen() failed" << endl;
  exit(EXIT_FAILURE);
}

cout << "Server Started!" << endl;
```

TCP Server – Example

```
  while (true)
  {
    SOCKET client;
    int clientlen = sizeof(client_addr);

    if ((client = accept(server, (struct sockaddr *) &client_addr,
         &clientlen)) == INVALID_SOCKET){
      cout << "accept() failed" << endl;
    }

    char addrstr[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &(client_addr.sin_addr), addrstr,
              INET_ADDRSTRLEN);
    cout << "Connection from " << addrstr << endl;

    thread *clientthread = new std::thread(HandleClientThread,
                                           client, addrstr);
  }
}
```

TCP Client – Example

```
#include <iostream>
#include <string>
#include <winsock2.h>
#include <ws2tcpip.h>

using namespace std;

int main()
{
  SOCKET server;
  WSADATA wsa_data;
  SOCKADDR_IN addr;
  if (WSAStartup(MAKEWORD(2, 0), &wsa_data) != NO_ERROR){
    cout << "WSAStartup() failed" << endl;
    exit(EXIT_FAILURE);
  }
  if ((server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
       INVALID_SOCKET){
    cout << "socket() failed" << endl;
    exit(EXIT_FAILURE);
  }
```

The lib **ws2_32.lib** must be added to the linker input.

TCP Client – Example

```
InetPton(AF_INET, L"127.0.0.1", &addr.sin_addr.s_addr);
addr.sin_family = AF_INET;
addr.sin_port = htons(8856);

if (connect(server, reinterpret_cast<SOCKADDR *>(&addr),
    sizeof(addr)) == SOCKET_ERROR) {
  cout << "connect() failed" << endl;
  exit(EXIT_FAILURE);
}

cout << "Connected to server!" << endl;
```

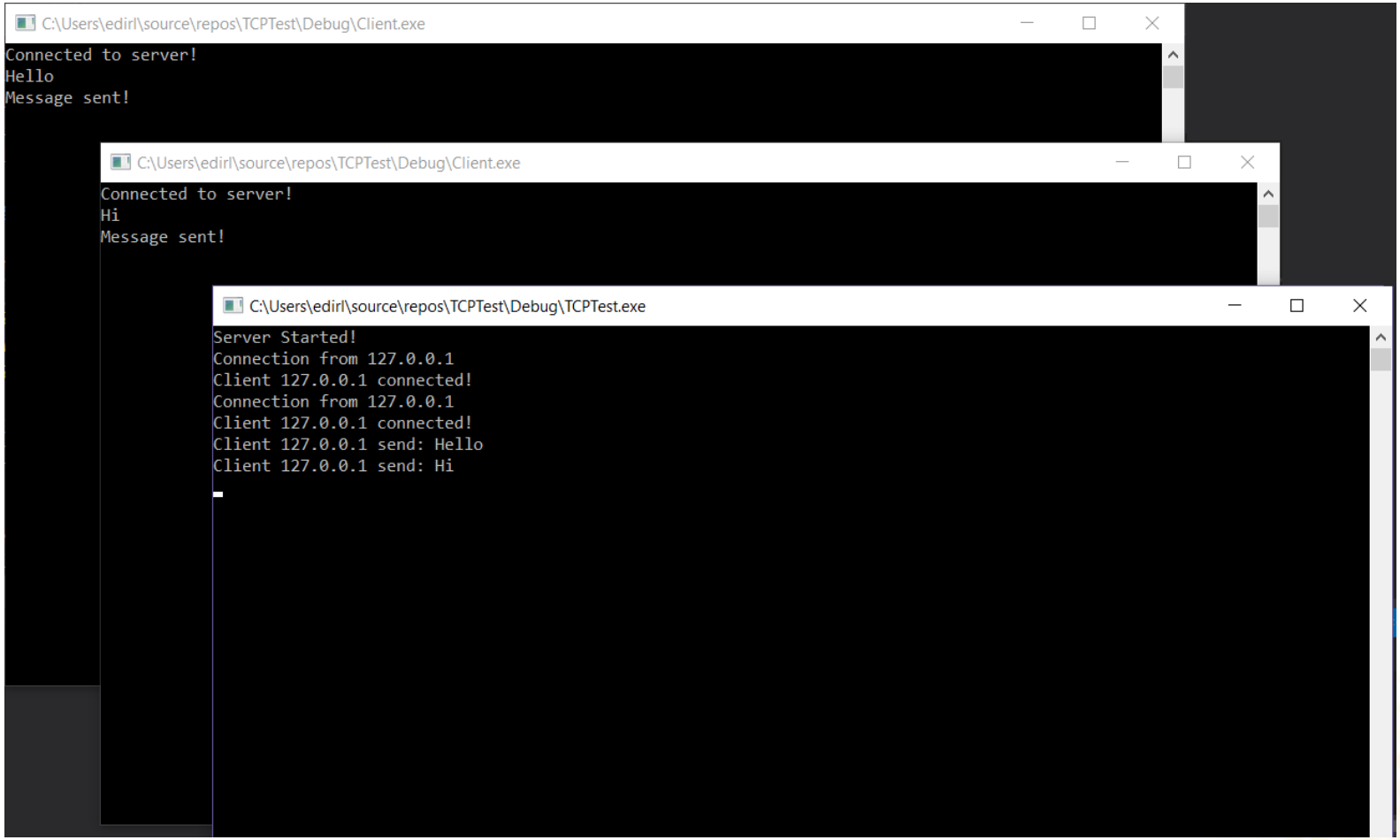
TCP Client – Example

```
  string buffer = "";
  while (buffer != "exit")
  {
    getline(cin, buffer);

    if (send(server, buffer.c_str(), sizeof(buffer), 0) ==
        SOCKET_ERROR)          {
      cout << "send() failed" << endl;
      exit(EXIT_FAILURE);
    }
    cout << "Message sent!" << endl;
  }

  if (closesocket(server) == SOCKET_ERROR)
  {
    cout << "closesocket() failed" << endl;
    exit(EXIT_FAILURE);
  }
  WSACleanup();
  cout << "Socket closed." << endl;
}
```

TCP Client-Server – Example



Exercise 1

1) Modify the code of the TCP Client-Server example to create a chat application. When a client send a message to the server, the server must send the message to all connected clients (except the one that originally send the message).

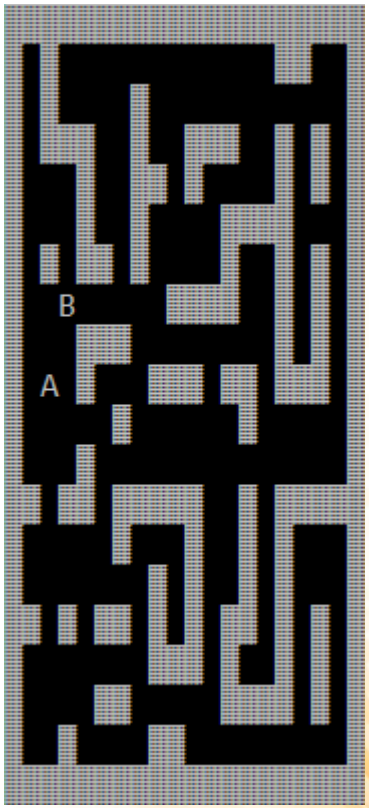
Tips:

- Create a list of connected clients: every time a client connects, the server adds the socket and the address of the client to the list.
  List structure in C++: http://www.cplusplus.com/reference/list/list/
- When the server receives a message, it iterates through the list and send the message to all other clients.
- The client needs a new thread to receive incoming messages from the server.

Exercise 2

2) Create a client-server multiplayer game that allow players to walk together through a maze.

- The server must be authoritative: the player's movements must be validated and performed in the server. The clients must receive only the updated positions.
- A protocol of messages that are exchanged by client and server is indispensable.
  - **Example 1:** when a player wants to move right he send a message to the server: "r|ID|", where r represents the right direction and ID must be a number that identifies the player (every player must know his ID, which is defined by the server – i.e.: the server must send the ID to the player).
  - **Example 2:** when a player moves, the server sends messages to all clients informing the positions of all players: "U|ID|AVATAR|POSX|POSY|", where U indicates that this is an update, ID is the identification of the player, AVATAR is a unique letter that represents the player, and POSX and POSY are the positions of the player.
- The client must show the map and the movements of all players in real-time (as illustrated in the image).



A and B are players

Peer-to-Peer Model

- A peer-to-peer system is designed around the notion of equal peer nodes simultaneously functioning as both "clients" and "servers".

- **If there is no server, how peers find other peers to connect?**
  - Peer discovery is the process of locating peers for data communication in a peer-to-peer network.

  - There are a number of peer discovery methods: tracker servers, message broadcasting, DNS seeding, IP scanning, …

UDP Broadcast

- UDP broadcast is a technique that allows sending UDP datagram from a single source to all computers in a LAN.

```
sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)
...
setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &opcast,sizeof(broadcast));
...
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(BROADCASTPORT);
server_addr.sin_addr.s_addr = INADDR_ANY;
client_addr.sin_family = AF_INET;
client_addr.sin_port = htons(BROADCASTPORT);
client_addr.sin_addr.s_addr = INADDR_BROADCAST;
...
bind(sock, (sockaddr*)&server_addr, sizeof(server_addr));
...
sendto(sock, msg, strlen(msg) + 1, 0, (sockaddr *)&client_addr,
       sizeof(client_addr));
...
recvfrom(sock, buffer, RCVBUFSIZE, 0, (sockaddr *)&client_addr, &len);
```

P2P Client – Example

```
#include <winsock2.h>
#include <iostream>
#include <ws2tcpip.h>
#include <thread>
#include <list>
#include <string>

using namespace std;

#define BROADCASTPORT 9009
#define RCVBUFSIZE 1024
#define MAXPENDING 5
#define GUIDSIZE 64

typedef struct peerinfo {
  SOCKET peer_socket;
  char* peer_ip;
} PeerInfo;

list<PeerInfo> peers;
char *clientGUID = new char[GUIDSIZE + 1];
```

P2P Client – Example

```
bool ClientExist(char* ip){
  bool exist = false;
  for (list<PeerInfo>::iterator it = peers.begin();
       it != peers.end(); it++){
    if (strcmp((*it).peer_ip, ip) == 0)
      exist = true;
  }
  return exist;
}

bool null_client(const PeerInfo& value){
  return (value.peer_socket == -1);
}

void RemoveClient(char* ip){
  for (list<PeerInfo>::iterator it = peers.begin();
       it != peers.end(); it++){
    if (strcmp((*it).peer_ip, ip) == 0)
      (*it).peer_socket = -1;
  }
  peers.remove_if(null_client);
}
```

P2P Client – Example

```
void AddClient(SOCKET sock, char* ip){
  PeerInfo pinfo;
  pinfo.peer_socket = sock;
  pinfo.peer_ip = ip;
  peers.push_back(pinfo);
}

void HandlePeerThread(SOCKET sock, char *ip){
  char buffer[RCVBUFSIZE];
  memset(buffer, 0, sizeof(buffer));
  cout << "Peer " << ip << " connected!" << endl;
  while (recv(sock, buffer, RCVBUFSIZE, 0) > 0){
    cout << ip << " says " << buffer << endl;
    memset(buffer, 0, sizeof(buffer));
  }
  if (closesocket(sock) == SOCKET_ERROR){
    cout << "closesocket() failed" << endl;
  }
  RemoveClient(ip);
  cout << "Client " << ip << " disconnected." << endl;
}
```

P2P Client – Example

```
void SendDataToAllPeers(string data){
  for (list<PeerInfo>::iterator it = peers.begin();
       it != peers.end(); it++){
    if (send((*it).peer_socket, data.c_str(), data.length(), 0) ==
        SOCKET_ERROR){
      cout << "send() failed" << endl;
    }
  }
}


void SendBroadcastThread(SOCKET sock, SOCKADDR_IN client_addr){
  while (true){
    sendto(sock, clientGUID, strlen(clientGUID) + 1, 0,
           (sockaddr *)&client_addr, sizeof(client_addr));
    this_thread::sleep_for(chrono::seconds(5));
  }
}
```

P2P Client – Example

```
void ReceiveBroadcastThread(SOCKET sock){
  while (true){
    SOCKADDR_IN client_addr;
    char buffer[RCVBUFSIZE];
    int len = sizeof(SOCKADDR_IN);
    if (recvfrom(sock, buffer, RCVBUFSIZE, 0,
       (sockaddr *)&client_addr, &len) == SOCKET_ERROR){
      cout << "recvfrom() failed" << endl;
    }

    char ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET,&(client_addr.sin_addr),ip,INET_ADDRSTRLEN);

    if (strcmp(clientGUID, buffer) != 0){
      if (!ClientExist(ip)){ //connect to the new discovered peer
        SOCKET peersocket;
        SOCKADDR_IN peer_addr;
        if ((peersocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP))
            == INVALID_SOCKET){
          cout << "socket() failed" << endl;
          exit(EXIT_FAILURE);
        }
```

P2P Client – Example

```
        wchar_t cAddr[INET_ADDRSTRLEN + 1];
        size_t i;
        mbstowcs_s(&i, cAddr, strlen(ip) + 1, ip, strlen(ip) + 1);

        InetPton(AF_INET, cAddr, &peer_addr.sin_addr.s_addr);
        peer_addr.sin_family = AF_INET;
        peer_addr.sin_port = htons(8856);

        if (connect(peersocket, reinterpret_cast<SOCKADDR *>
            (&peer_addr), sizeof(peer_addr)) == SOCKET_ERROR){
          cout << "connect() failed" << endl;
          exit(EXIT_FAILURE);
        }

        AddClient(peersocket, ip);

        thread *peerthread = new thread(HandlePeerThread,
                                        peersocket, ip);
      }
    }
  }
}
```

P2P Client – Example

```
void TCPServerThread()
{
  SOCKET server;
  SOCKADDR_IN server_addr, client_addr;
  if ((server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
       INVALID_SOCKET) {
    cout << "socket() failed" << endl;
    exit(EXIT_FAILURE);
  }

  memset(&server_addr, 0, sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  server_addr.sin_port = htons(8856);

  if (::bind(server, (struct sockaddr *) &server_addr,
      sizeof(server_addr)) == SOCKET_ERROR){
    cout << "bind() failed" << endl;
    exit(EXIT_FAILURE);
  }
```

P2P Client – Example

```
  if (listen(server, MAXPENDING) == SOCKET_ERROR){
    cout << "listen() failed" << endl;
    exit(EXIT_FAILURE);
  }
  while (true){
    SOCKET client;
    int clientlen = sizeof(client_addr);

    if ((client = accept(server, (struct sockaddr *) &client_addr,
         &clientlen)) == INVALID_SOCKET){
      cout << "accept() failed" << endl;
    }
    char ip[INET_ADDRSTRLEN];
    inet_ntop(AF_INET,&(client_addr.sin_addr),ip,INET_ADDRSTRLEN);

    if (!ClientExist(ip)){
      AddClient(client, ip);
      thread *peerthread = new thread(HandlePeerThread,client,ip);
    }
  }
}
```

P2P Client – Example

```
void GenerateGUIDString(char *guidstr)
{
  wchar_t szGUID[GUIDSIZE] = { 0 };
  GUID peerGUID;
  CoCreateGuid(&peerGUID);
  StringFromGUID2(peerGUID, szGUID, GUIDSIZE);
  memset(guidstr, 0, GUIDSIZE+1);
  size_t i;
  wcstombs_s(&i, guidstr, GUIDSIZE, szGUID, GUIDSIZE);
}

int main(){
  WSADATA wsa_data;
  SOCKET sock;
  SOCKADDR_IN server_addr;
  SOCKADDR_IN client_addr;
  GenerateGUIDString(clientGUID);

  if (WSAStartup(MAKEWORD(2, 0), &wsa_data) != NO_ERROR){
    cout << "WSAStartup() failed" << endl;
    exit(EXIT_FAILURE);
  }
```

P2P Client – Example

```
if ((sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) ==
     INVALID_SOCKET){
  cout << "socket() failed" << endl;
  exit(EXIT_FAILURE);
}

char broadcast = '1';
if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &broadcast,
    sizeof(broadcast)) == SOCKET_ERROR){
  cout << "setsockopt() failed" << endl;
  exit(EXIT_FAILURE);
}

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(BROADCASTPORT);
server_addr.sin_addr.s_addr = INADDR_ANY;

client_addr.sin_family = AF_INET;
client_addr.sin_port = htons(BROADCASTPORT);
client_addr.sin_addr.s_addr = INADDR_BROADCAST;
```

P2P Client – Example

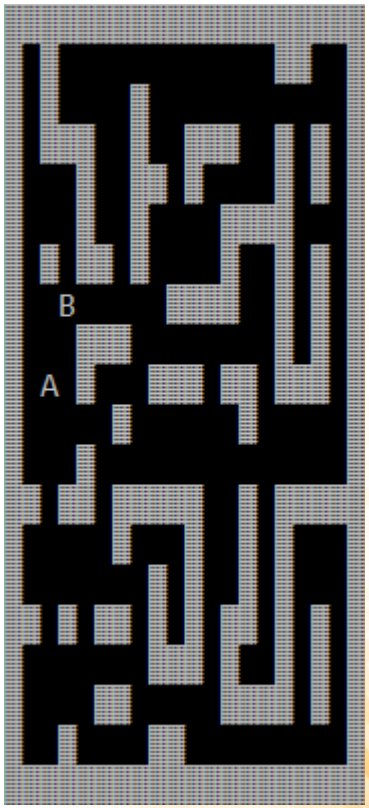
```
  if (::bind(sock, (sockaddr*)&server_addr, sizeof(server_addr)) ==
      SOCKET_ERROR){
    cout << "bind() failed" << endl;
    exit(EXIT_FAILURE);
  }
  thread tcphread(TCPServerThread);
  thread receivethread(ReceiveBroadcastThread, sock);
  thread broadcastthread(SendBroadcastThread, sock, client_addr);

  cout << "Waiting for peers..." << endl << endl;
  string cmd = "";
  while (cmd != "exit"){
    getline(cin, cmd);
    SendDataToAllPeers(cmd);
  }
  if (closesocket(sock) == SOCKET_ERROR){
    cout << "closesocket() failed" << endl;
    exit(EXIT_FAILURE);
  }
  WSACleanup();
  return 0;
}
```

Exercise 3 (Challenge)

3) Create a peer-to-peer multiplayer game that allow players to walk together through a maze.

- The game must be similar to the one created for Exercise 2, but without a server to control the state of the game.
- Use the code of the P2P Client example as base for the project.
- When a player moves, the client must send his movement to all other connected clients.
- The client must be always waiting for movements from other players and for incoming connections from new player.



A and B are players

Further Reading

- Coulouris, G., Dollimore, J., Kindberg, T., Blair, G. (2004). **Distributed Systems: Concepts and Design** (5th edition), Pearson.
  ISBN: 978-0132143011.

  - **Chapter 2: System Models**
  - **Chapter 3: Networking and Internetworking**
  - **Chapter 4: Interprocess Communication**