Network Named Pipes – Network Communication

Network Named Pipes are a feature in Windows that allows inter-process communication (IPC) between processes running on different machines across a network, using named pipes. Named pipes are traditionally used for local communication on the same machine, but with network support, they can be extended to work over a network.

How It Works:

1. A **Named Pipe Server** listens on a specific named pipe, similar to how it works locally, but with a **UNC path** (Universal Naming Convention) that specifies the machine name in the network.
2. A **Named Pipe Client** connects to the server using the **UNC path**, which looks like \\RemoteMachineName\pipe\PipeName, where RemoteMachineName is the host or IP address of the machine, and PipeName is the name of the pipe being used for communication.

Advantages:

1. Named pipes provide a **high-performance communication mechanism** between processes on different machines in a network.
2. They are ideal for communication between **Windows-based applications** within a **local network**.

Requirements:

1. Both the client and server must have access to the same network.
2. The server must have a **named pipe** listening on a specific name.
3. Proper **permissions and firewall settings** need to be configured to allow access to named pipes over the network.

Example Path:

To access a named pipe on a remote server:

\\RemoteServer\pipe\MyPipe

Use Cases:

1. Legacy systems where named pipes are preferred for communication.
2. Secure, high-performance communication between client and server applications in a trusted network.

Limitations:

1. Named pipes are **Windows-specific** and may not be suitable for cross-platform communication.
2. They rely on the **SMB (Server Message Block)** protocol, which may require specific firewall and security configurations.

In summary, **Network Named Pipes** are a powerful IPC mechanism for communication between processes on different Windows machines over a network, but they are mainly used in environments where security and network configuration are tightly controlled.

Watch following video:

https://www.youtube.com/watch?v=kGT4PcTEPP8

Read following links

https://learn.microsoft.com/en-us/dotnet/api/system.io.pipes.namedpipeclientstream?view=net-8.0

https://learn.microsoft.com/en-us/dotnet/api/system.io.pipes.namedpipeserverstream?view=net-8.0

Develop the following tasks for **Named Pipe Server and Client**

Basic Named Pipe Server and Client

Step 1: Create a Named Pipe server that listens for a client connection. Define a pipe name, such as "testpipe".

Step 2: Once the server is listening, wait for a client to connect.

Step 3: When the client connects, send a simple message like "Hello, Client!".

Step 4: Build a client that connects to the Named Pipe server and reads the message from the server.

Step 5: Display the message on the client side.

Bidirectional Communication

Step 1: Start with the basic Named Pipe server and client from Exercise 1.

Step 2: Modify the server so that after sending "Hello, Client!", it waits to receive a message back from the client.

Step 3: On the client side, add code to send "Hello, Server!" in response to the server's initial message.

Step 4: Ensure both sides can send and receive messages, creating a bidirectional communication flow.

Step 5: Test by exchanging several messages back and forth.

Message Exchange with User Input

Step 1: Set up a Named Pipe server similar to previous exercises.

Step 2: Modify the client so it takes user input (e.g., from the console) and sends each message to the server over the pipe.

Step 3: Update the server to receive messages from the client and print them to its console.

Step 4: After receiving a message, the server should send back a simple acknowledgment (e.g., "Message received").

Step 5: Run both applications and verify the server displays each client message and responds correctly.

File Transfer over Named Pipes

Step 1: Modify the client to read a text file from disk, line by line.

Step 2: Send each line of the text file to the server using Named Pipes.

Step 3: On the server side, receive each line sent from the client and write it to a new file.

Step 4: Ensure the client waits for an acknowledgment after each line is sent, then continues with the next line.

Step 5: Test by transferring a sample file and verifying that the server's file output matches the client's input file.

Multiple Clients Communication

Step 1: Create a Named Pipe server that can handle multiple client connections by creating new threads or tasks for each client.

Step 2: Modify the server to accept a new client, handle its messages, and respond independently of other clients.

Step 3: Write the client code to connect and send messages as usual, but make it possible for several client instances to run concurrently.

Step 4: Test by launching multiple client instances and verifying that each client can send and receive messages without interference.

Step 5: Add error handling on both server and client sides to manage simultaneous connections and handle client disconnections smoothly.