Module 5: Interprocess Communication and Remote Procedure Call (RPC) in .NET

(Tutorial / Lab)

Subjects: Systems Programming

Minor: For all bachelor programs

Lab Objectives

By completing this lab, students will:

- Understand how independent processes communicate
- Implement basic IPC mechanisms in C#
- Compare local IPC and remote communication
- Design a simple RPC-style client–server system
- Reason about performance, reliability, and design trade-offs

Question 1 – Understanding the Need for IPC (Conceptual + Demo)

Write two separate C# console applications:

- **Process A**: produces data
- **Process B**: consumes data

Tasks:

1. Explain why Process A cannot directly access memory of Process B.
2. Run both programs independently and show that they are isolated.
3. Propose at least **two IPC mechanisms** that could allow them to communicate.

Discussion:

- Why is process isolation important for system stability and security?
- What problems would occur without isolation?

Trả lời: link Question_01

Lý do Process A không thể truy cập trực tiếp memory của Process B là do mỗi process có virtual address space riêng do OS quản lý. Memory của process này không được ánh xạ vào process khác. Nếu một process cố đọc địa chỉ không thuộc vùng của nó thì sẽ bị OS chặn lại (Access Violation).

Process Isolation quan trọng vì Isolation đảm bảo:

- Một process crash không làm sập toàn hệ thống
- Không thể đọc trộm memory process khác
- Tránh corruption dữ liệu giữa các chương trình

Nếu không có isolation:

- Một chương trình lỗi có thể ghi đè bộ nhớ chương trình khác
- Virus dễ dàng đọc password trong memory
- Toàn hệ thống có thể sập chỉ vì 1 ứng dụng lỗi

Question 2 – Local IPC Using Named Pipes

Implement interprocess communication using **Named Pipes**.

Tasks:

1. Create a **Named Pipe Server** that waits for client connections.
2. Create a **Named Pipe Client** that sends a message to the server.
3. The server reads the message and sends a response.
4. Display the communication result on both sides.

Requirements:

- Use `NamedPipeServerStream` and `NamedPipeClientStream`
- Ensure proper resource cleanup

Discussion:

- Why are Named Pipes suitable for local IPC?
- What are the limitations of Named Pipes?

Trả lời: link Question_02

Named Pipes phù hợp cho Local IPC do:

- Tốc độ cao vì không đi qua network stack
- OS quản lý bảo mật và quyền truy cập
- Không cần mở port
- Dễ dùng trong Windows môi trường local

Hạn chế của Named Pipes:

- Chủ yếu dùng local
- Không linh hoạt như TCP cho hệ thống phân tán
- Không phù hợp cho high-scale distributed system
- Phải xử lý blocking nếu nhiều client

Question 3 – IPC Using Sockets (Network-Based Communication)

Implement IPC using **TCP sockets**.

Tasks:

1. Create a TCP server that listens on a port.
2. Create a TCP client that sends a request message.
3. The server processes the request and returns a response.
4. Run both programs on the same machine.

Discussion:

- How does socket-based IPC differ from Named Pipes?
- What changes if the client runs on another machine?

Trả lời: link Question_03

	Named Pipes	**TCP Sockets**
Phạm vi	Chủ yếu local	Cần port
Cách kết nối	Không cần port	Linh hoạt hơn
Hiệu năng	Nhanh hơn trong cùng máy	Linh hoạt hơn
Khả năng mở rộng	Windows-centric	Cross-platform

Nếu client chạy trên máy khác thì chỉ cần:

- Đổi host thành IP server (vd: 192.168.1.10)
- Mở firewall cho port 5000
- Server lắng nghe IPAddress.Any thay vì Loopback

Question 4 – Implementing a Simple RPC Mechanism

Design a **simple RPC-style system**.

Scenario:
A client wants to call a remote function:

```
MoneyExchange(string currency, double amount)
```

Tasks:

1. Define a request message containing:
    - Method name
    - Parameters
2. Serialize the request using **JSON**
3. Send the request to the server
4. The server:
    - Deserializes the request
    - Executes the function
    - Sends back the result
5. The client receives and displays the result

Discussion:

- How does this differ from a normal function call?
- What responsibilities does the RPC framework handle?

Trả lời: link Question_04

So với gọi hàm bình thường:

- Gọi hàm bình thường sẽ chạy trong cùng process, cùng memory.
- RPC:
    1. Phải serialize dữ liệu
    2. Gửi qua network
    3. Deserialize
    4. Có độ trễ mạng
    5. Có thể fail

RPC framework phải xử lý:

- Serialize / Deserialize
- Quản lý kết nối
- Retry khi lỗi mạng
- Timeout
- Bảo mật (TLS)
- Versioning

- Load balancing

Question 5 – JSON-RPC and Design Trade-offs

Extend Question 4 to follow **JSON-RPC style** more closely.

Tasks:

1. Add:
   - Request ID
   - Error handling
2. Support at least **two remote methods**
3. Handle invalid method calls gracefully

Discussion:

- Advantages of JSON-RPC
- Performance overhead of serialization
- Security considerations in RPC systems

Trả lời: link Question_05

Advantages of JSON-RPC:

- Chuẩn hóa request/response
- Dễ debug vì là JSON text
- Cross-language
- Có request id để match response
- Có error structure rõ ràng

Performance overhead of serialization:

- So với function call local:
  1. Tốn CPU serialize/deserialize
  2. Tăng băng thông vì JSON text verbose
  3. Có latency mạng
  4. Allocation nhiều hơn
- Binary protocol (gRPC, Protobuf) nhanh hơn JSON.

Security considerations in RPC systems:

- Authentication (API key, token)
- Authorization
- TLS encryption
- Input validation

- Rate limiting
- Không cho phép gọi method tùy ý (reflection abuse)

Nếu không có dễ bị Remote code execution, Data leak, DoS attack hoặc Injection attack