* **Named Pipes** (Windows-only, secure)
* **Shared Memory** (Fastest but needs synchronization)
* **Message Queues** (Asynchronous communication)
* **Sockets** (Network-friendly, supports remote IPC)

**1. Named Pipes (Server-Client Communication)**

A **named pipe** allows bidirectional communication between processes.

**Server (Named Pipe)**

#include <windows.h>

#include <iostream>

#define PIPE\_NAME L"\\\\.\\pipe\\MyPipe"

int main() {

HANDLE hPipe = CreateNamedPipe(PIPE\_NAME, PIPE\_ACCESS\_DUPLEX,

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

1, 1024, 1024, 0, NULL);

if (hPipe == INVALID\_HANDLE\_VALUE) {

std::cerr << "Failed to create named pipe\n";

return 1;

}

std::cout << "Waiting for client connection...\n";

if (ConnectNamedPipe(hPipe, NULL)) {

char buffer[128];

DWORD bytesRead;

ReadFile(hPipe, buffer, sizeof(buffer), &bytesRead, NULL);

std::cout << "Received: " << buffer << "\n";

std::string response = "Message received!";

DWORD bytesWritten;

WriteFile(hPipe, response.c\_str(), response.length() + 1, &bytesWritten, NULL);

}

CloseHandle(hPipe);

return 0;

}

**Client (Named Pipe)**

#include <windows.h>

#include <iostream>

int main() {

HANDLE hPipe = CreateFile(L"\\\\.\\pipe\\MyPipe", GENERIC\_WRITE | GENERIC\_READ,

0, NULL, OPEN\_EXISTING, 0, NULL);

if (hPipe == INVALID\_HANDLE\_VALUE) {

std::cerr << "Failed to connect to pipe\n";

return 1;

}

std::string message = "Hello, Server!";

DWORD bytesWritten;

WriteFile(hPipe, message.c\_str(), message.length() + 1, &bytesWritten, NULL);

char buffer[128];

DWORD bytesRead;

ReadFile(hPipe, buffer, sizeof(buffer), &bytesRead, NULL);

std::cout << "Server response: " << buffer << "\n";

CloseHandle(hPipe);

return 0;

}

**2. Shared Memory (Fast Data Exchange Between Processes)**

Shared memory allows multiple processes to access the same memory region.

**Server (Shared Memory)**

#include <windows.h>

#include <iostream>

int main() {

HANDLE hMapFile = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, 256, L"MySharedMemory");

if (!hMapFile) {

std::cerr << "Could not create shared memory.\n";

return 1;

}

LPVOID pBuf = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, 256);

if (!pBuf) {

CloseHandle(hMapFile);

return 1;

}

std::string message = "Hello from Server!";

memcpy(pBuf, message.c\_str(), message.size() + 1);

std::cout << "Message written to shared memory.\n";

std::cin.get(); // Wait for client to read

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

return 0;

}

**Client (Shared Memory)**

#include <windows.h>

#include <iostream>

int main() {

HANDLE hMapFile = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, L"MySharedMemory");

if (!hMapFile) {

std::cerr << "Could not open shared memory.\n";

return 1;

}

LPVOID pBuf = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, 256);

if (!pBuf) {

CloseHandle(hMapFile);

return 1;

}

std::cout << "Message from shared memory: " << (char\*)pBuf << "\n";

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

return 0;

}

**3. Message Queue (Process-to-Process Messaging)**

Message queues are used for sending messages between processes asynchronously.

**Server (Message Queue)**

#include <windows.h>

#include <iostream>

int main() {

HANDLE hQueue = CreateMailslot(L"\\\\.\\mailslot\\MyMailslot", 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hQueue == INVALID\_HANDLE\_VALUE) {

std::cerr << "Failed to create message queue.\n";

return 1;

}

char buffer[128];

DWORD bytesRead;

while (true) {

if (ReadFile(hQueue, buffer, sizeof(buffer), &bytesRead, NULL)) {

std::cout << "Received: " << buffer << "\n";

}

}

CloseHandle(hQueue);

return 0;

}

**Client (Message Queue)**

#include <windows.h>

#include <iostream>

int main() {

HANDLE hQueue = CreateFile(L"\\\\.\\mailslot\\MyMailslot", GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hQueue == INVALID\_HANDLE\_VALUE) {

std::cerr << "Failed to connect to message queue.\n";

return 1;

}

std::string message = "Hello from Client!";

DWORD bytesWritten;

WriteFile(hQueue, message.c\_str(), message.length() + 1, &bytesWritten, NULL);

CloseHandle(hQueue);

return 0;

}

**4. Socket Communication (Network-Based IPC)**

Sockets allow communication over networks or between local processes.

**Server (Socket)**

#include <winsock2.h>

#include <iostream>

#pragma comment(lib, "ws2\_32.lib")

int main() {

WSADATA wsa;

WSAStartup(MAKEWORD(2, 2), &wsa);

SOCKET server = socket(AF\_INET, SOCK\_STREAM, 0);

sockaddr\_in serverAddr = {AF\_INET, htons(8080), INADDR\_ANY};

bind(server, (sockaddr\*)&serverAddr, sizeof(serverAddr));

listen(server, 1);

SOCKET client = accept(server, NULL, NULL);

char buffer[128];

recv(client, buffer, sizeof(buffer), 0);

std::cout << "Received: " << buffer << "\n";

send(client, "Message received!", 17, 0);

closesocket(server);

WSACleanup();

return 0;

}

**Client (Socket)**

#include <winsock2.h>

#include <iostream>

#pragma comment(lib, "ws2\_32.lib")

int main() {

WSADATA wsa;

WSAStartup(MAKEWORD(2, 2), &wsa);

SOCKET client = socket(AF\_INET, SOCK\_STREAM, 0);

sockaddr\_in serverAddr = {AF\_INET, htons(8080), inet\_addr("127.0.0.1")};

connect(client, (sockaddr\*)&serverAddr, sizeof(serverAddr));

std::string message = "Hello, Server!";

send(client, message.c\_str(), message.length() + 1, 0);

char buffer[128];

recv(client, buffer, sizeof(buffer), 0);

std::cout << "Server response: " << buffer << "\n";

closesocket(client);

WSACleanup();

return 0;

}