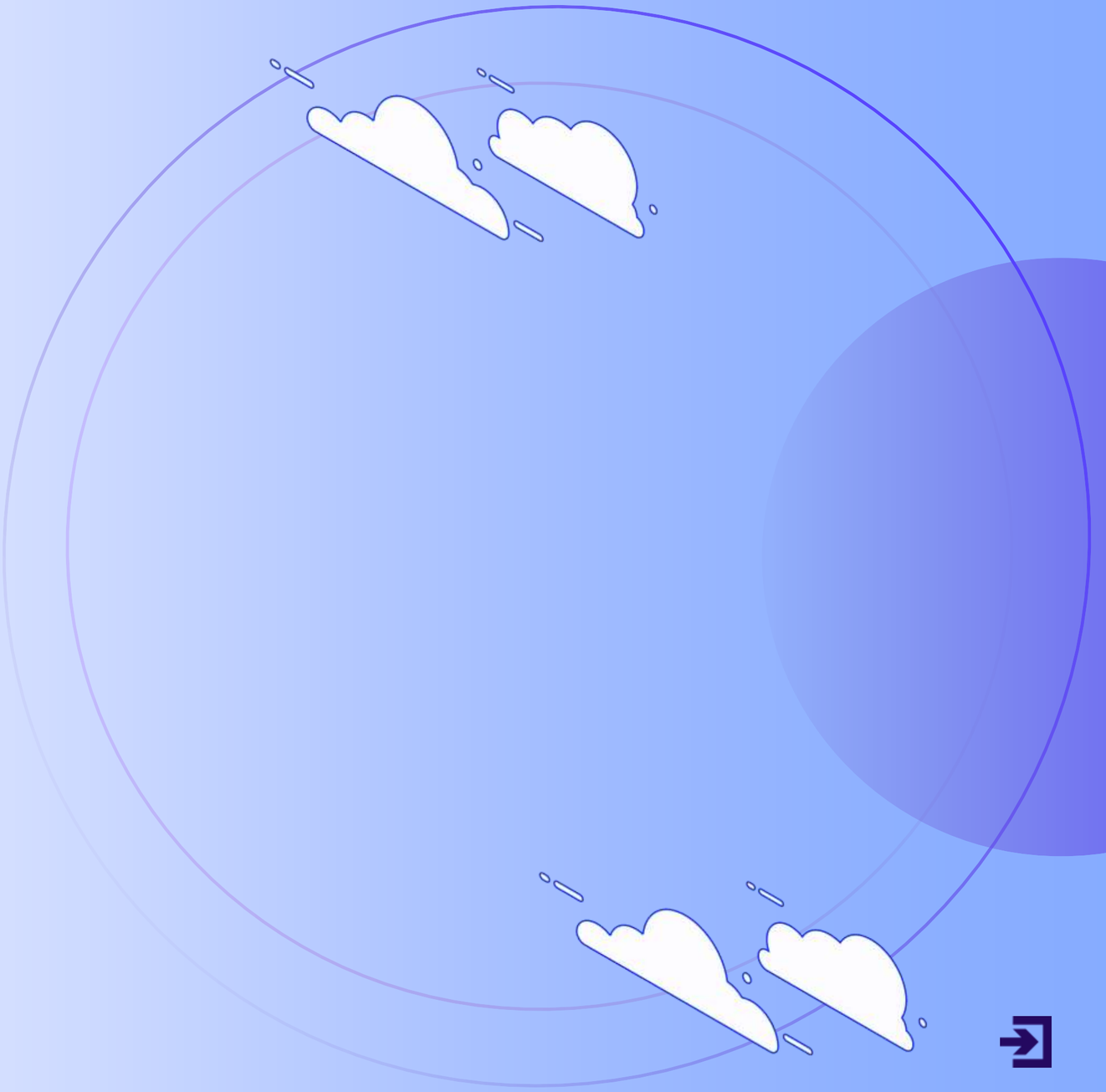
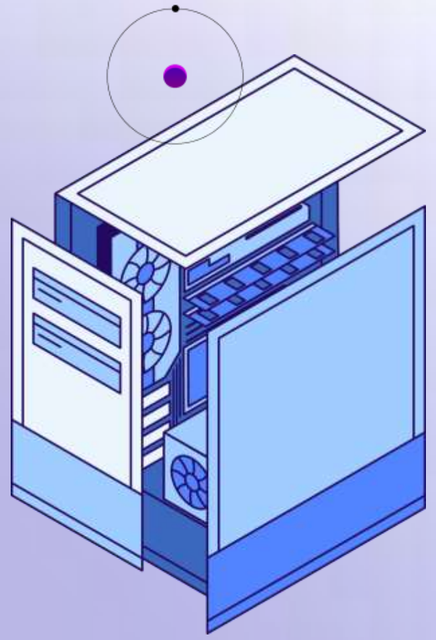
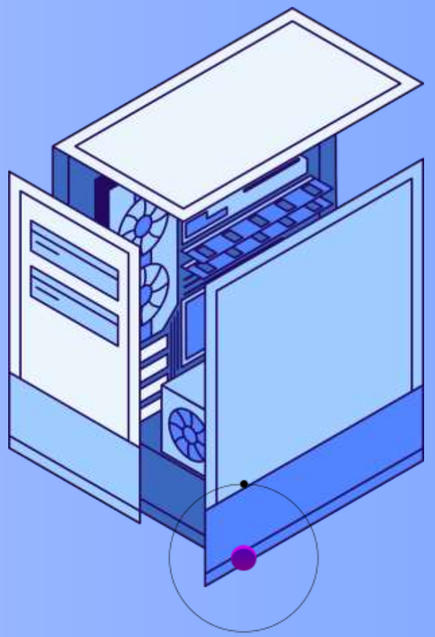
OPERATING SYSTEMS



TEAM MEMBERS:





Mohamed Adly Mohamed Ali (2301000)

Yassen Omar Sayed Farrag (2301784)

Mahmoud Osama Salah Taha (2302601)

Ali Osama Ali Meselhy (2302942)

Ahmed Nasser Ahmed Abdel Hamid (2302978)

PROJECT IDEA

1. Shared Memory Chat System:

Server: Uses shared memory, mutex, and event for local inter-process communication (IPC).
Client: Connects to the same shared memory segment, prompts for username, and syncs messages.

2. Socket-Based Chat System (TCP/IP):

Server: Listens on a port, accepts multiple clients, and broadcasts messages using threads and mutex.
Client: Connects via IP/port, sends/receives messages in real time using a receiver thread.

PROJECT

OVERVIEW&OBJECTIVES

Unified Goal
Develop two distinct real-time chat systems on Windows to demonstrate core operating system and networking concepts:

Local communication using OS-native IPC mechanisms.
Networked communication using TCP/IP sockets and multithreading.

Project 1: Shared Memory Chat (Local IPC)

Purpose: Enable multiple processes on the same machine to exchange messages instantly.
Mechanisms Used:
CreateFileMapping / MapViewOfFile → Shared memory buffer.
CreateMutex → Synchronized access.
CreateEvent → Signal new messages (event-driven, no polling).
Features: Circular message buffer (10 messages), username support, native Win32 GUI.

Project 2: Socket-Based Chat (TCP/IP Networking)

Purpose: Enable multiple clients across machines (or same machine) to chat over a network.
Mechanisms Used:

Winsock2 API (socket, bind, listen, accept, recv, send).
One thread per client + std::mutex for thread safety.
Broadcast to all connected clients.
Features: Configurable IP/port, real-time log, scalable client handling.

KEY TECHNICAL HIGHLIGHTS

Shared Memory System

Uses Global* names to ensure visibility across sessions (e.g., admin vs. user).
Sequence number prevents message duplication or loss.
Event object (Auto-reset in client, Manual-reset in server) ensures all clients receive new messages.

Socket System

Server uses detach() threads for simplicity (production systems might use thread pools).
std::remove + erase safely removes disconnected clients.
Broadcast excludes sender to avoid echo.

KEY TECHNICAL HIGHLIGHTS

Common UI Design

Consistent color scheme (RGB(203, 204, 246) — soft blue background).
Edit boxes, buttons, and log displays using raw Win32 (no frameworks).

PROJECT CONCLUSION

These two applications successfully illustrate two fundamental paradigms of inter-process communication:

1. Local IPC (Shared Memory)

Extremely fast, zero-copy, ideal for same-machine coordination (e.g., game engines, system tools).
Requires careful synchronization but avoids network overhead.

2. Networked Communication (Sockets)

Flexible, scalable, and interoperable across devices.
Introduces latency and complexity (connection management, error handling) but enables distributed systems.

THANK YOU!