Inter-Process Communication (IPC) Mechanisms

Operating Systems Lecture

Learning Focus

LO2: IPC mechanisms and theoretical foundations

Instructor: Tanaya Bowade & Dr Shabih Fatima

Course: Operating Systems

Introduction to IPC

Definition

Inter-Process Communication (IPC) is a fundamental mechanism within an operating system that enables COOPERATING processes to exchange data and coordinate their activities.

Why Processes Need Communication

Information Sharing
Multiple processes accessing shared data

Computation Speedup
Breaking tasks into concurrent subtasks

Modularity
Enabling modular components to interact

Convenience
Structured way for processes to interact

Real-world Examples

Web Browsers

Separate processes for tabs/extensions communicating to share data

Shell Pipelines

Commands chained together using pipes (e.g., ls | grep .txt)

Client-Server Apps

Client sends requests to server, which processes and responds

IPC Models Overview

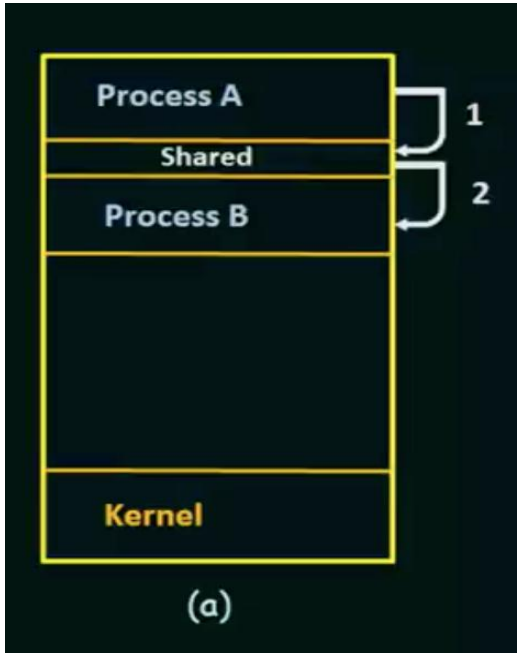
Inter-Process Communication mechanisms rely on two primary models: **shared memory** and **message passing**. The choice between them depends on application requirements.

Shared Memory Model

Speed
High speed - direct memory access

Complexity
Higher - requires synchronization

Synchronization
Explicit needed for data consistency

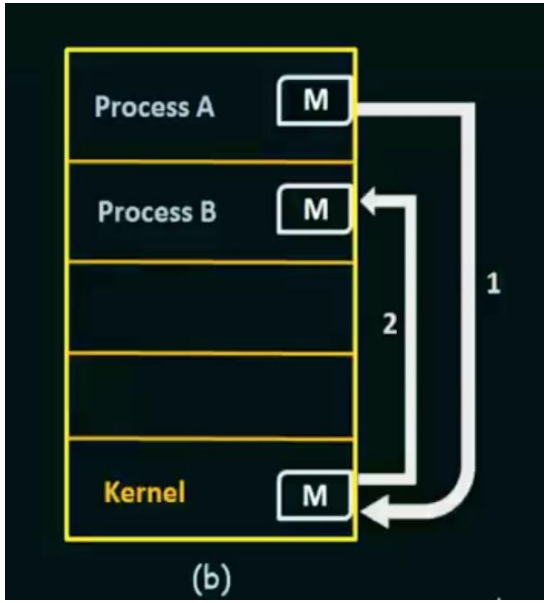


Message Passing Model

Speed
Lower - kernel involvement

Complexity
Lower - simpler to manage

Synchronization
Built-in to message semantics

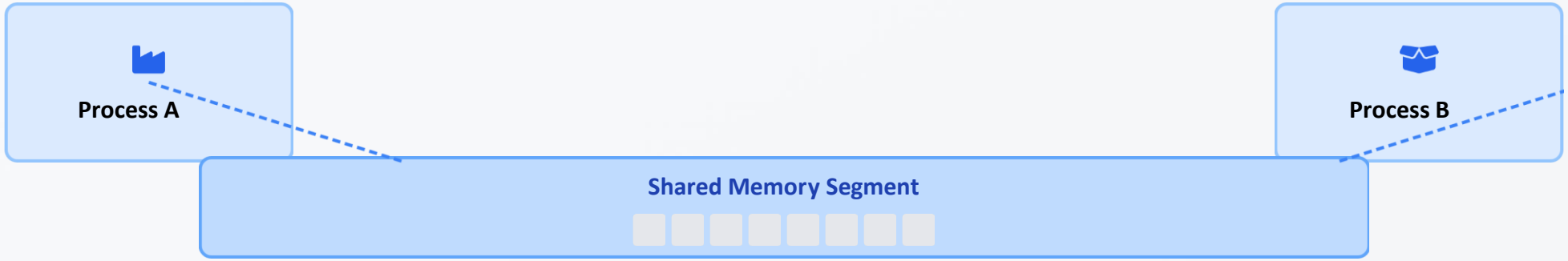


Key Decision Factor: Choose shared memory for performance, message passing for simplicity.

Shared Memory Systems

Concept

Shared memory allows multiple processes to access the same physical memory region for high-speed data exchange.



Advantages

High Speed
Direct memory access without system calls

Efficient for Large Data
No data copying between address spaces

Challenges

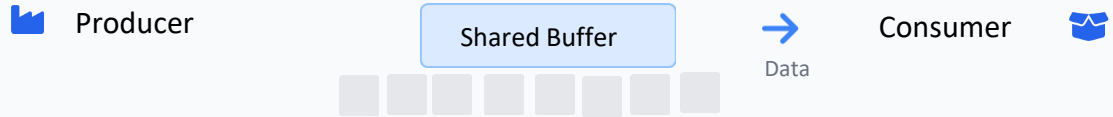
Complex Synchronization
Requires semaphores, mutexes to prevent race conditions

Security Risks
Data vulnerable if not properly managed

Producer-Consumer Example

A classic example of shared memory where:

- Producer generates data into shared buffer
- Consumer retrieves data from same buffer
- Synchronization ensures producer doesn't write to full buffer
- Consumer doesn't read from empty buffer



Message Passing Systems

Concept

Message passing is an IPC mechanism where processes communicate by exchanging messages. Communication occurs through the operating system kernel via system calls such as send() and receive().

Direct Communication

- Processes explicitly name each other
- send(P, message): sends to process P
- receive(Q, message): receives from process Q

Indirect Communication

- Uses mailboxes/ports to facilitate communication
- Messages sent to specific mailboxes
- Mailboxes can be shared by multiple processes

Client-Server Example

A web browser (client) sends a request message to a web server. The server processes the request and sends a response back. This exchange relies on message passing mechanisms.

Pipes and Redirection in Unix

Unix Philosophy

Unix emphasizes building small, simple, and modular programs that each do one thing well, combining them using mechanisms like pipes for complex tasks.

Pipe (|)

Connects stdout of one command to stdin of another:

```
ls -l | wc -l
```

Lists files then counts lines (files/directories)

Redirection

Output (>):

```
echo "Hello" > file.txt
```

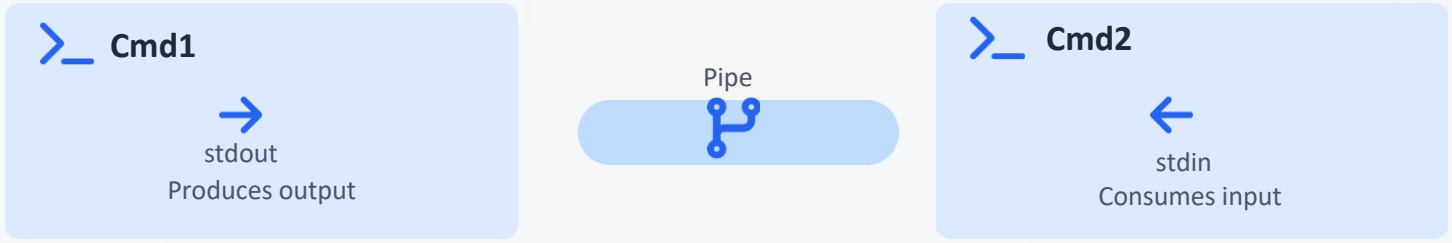
Writes "Hello" to file.txt (overwrites if exists)

Input (<):

```
sort < names.txt
```

Sorts lines in names.txt

How Pipes Work



Pipes, FIFOs, and Sockets

Pipes vs. FIFOs

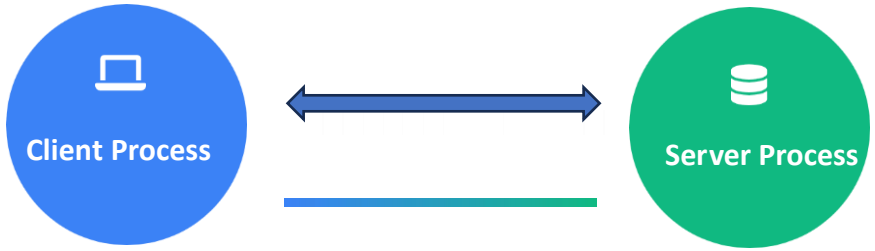
Feature	Pipes (Anonymous)	FIFOs (Named)
Nature	Unidirectional communication	One-way (half-duplex) — but can be opened twice for full-duplex communication
Naming	Unnamed; created in memory	Named; appears as a special file in the file system
Relationship	Used between related processes (e.g., parent-child)	Used between unrelated processes
Persistence	Exist only as long as communicating processes are active	Persist until explicitly deleted from the file system
Creation	Created using the `pipe()` system call	Created using the `mkfifo()` system call

Sockets

Sockets serve as an endpoint for communication, enabling processes to exchange data either on the same machine or across a network.

- More versatile and robust IPC mechanism compared to pipes and FIFOs
- Support various communication protocols (TCP, UDP)
- Unix domain sockets allow IPC between processes on the same machine

Socket Communication Example



Client-server model: Fully bidirectional communication channel

Signals and Process Communication

Definition

Signals are asynchronous notifications in Unix-like systems that indicate events to processes.

Common Signals

SIGINT

Generated by Ctrl+C. Requests graceful termination.

SIGKILL

Forceful termination. Cannot be caught or ignored.

SIGTERM

Request for termination. Can be caught and handled.

How Processes Use Signals

Process Management

- Parent sends signals to children
- Child notifies parent of events

System Calls

kill()

Synchronization Primitives

Introduction

Synchronization primitives are crucial for managing concurrent access to shared resources in multi-process or multi-threaded environments. They ensure data consistency and prevent race conditions.

Semaphores

- Signaling mechanisms for controlling access to resources
- Maintain a count of available resources
- Process can block until resource is available

Mutexes

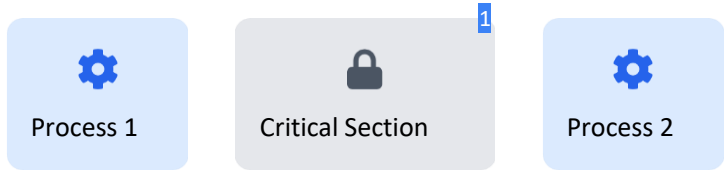
- Mutual exclusion objects for exclusive access
- Ensures only one process/thread accesses critical section
- Process blocks until mutex is available

Monitors

- High-level constructs encapsulating shared data
- Provide mutual exclusion for data access
- Support condition synchronization

Importance in IPC

- Prevent race conditions in shared data
- Ensure data consistency across processes
- Coordinate process execution order



IPC in Practice

Real-world Applications

Databases

Utilize shared memory for efficient cache management, allowing multiple database processes to access and modify shared data structures in memory, reducing disk I/O and improving performance.

Web Servers

Employ multiple processes or threads to handle concurrent client requests. These processes communicate using various IPC mechanisms, such as message queues or sockets, to coordinate tasks and share session information.

OS Components

The kernel and various user-space daemons communicate extensively using IPC. For instance, a system logger might receive messages from different processes via message queues or named pipes.

Microservices

Distributed systems often rely on IPC, particularly network sockets (TCP/UDP), for communication between independent services running on the same or different machines.

Performance Considerations

Kernel Involvement

Message passing has higher overhead due to context switches and data copying, while shared memory allows direct memory access.

Data Amount

For large data transfers, shared memory is more efficient as it avoids copying data between process address spaces.

Communication Frequency

For frequent communication, the overhead of establishing connections in message passing may be offset by its simplicity.

Process Location

Local communication favors shared memory, while networked communication typically requires message passing mechanisms.

Review: IPC Theoretical Foundations

Key Concepts Summary

Shared Memory
Direct memory access for high-speed data exchange, requiring explicit synchronization

Message Passing
Communication via explicit messages, simpler to manage but with higher overhead due to kernel involvement

Pipes & FIFOs
Unidirectional communication channels for related (anonymous) or unrelated (named) processes

Sockets
Network-enabled IPC for communication between processes on the same or different machines

Signals
Asynchronous notifications for event handling or process control

Synchronization
Essential for managing concurrent access to shared resources and preventing data inconsistencies

Exam Preparation

- **Understand Fundamental Purpose**
  Be clear on the basic operational mechanism of each IPC type
- **Know the Trade-offs**
  Be prepared to articulate advantages and disadvantages between different IPC models
- **Provide Examples**
  Be ready to give relevant examples for each IPC mechanism
- **Compare and Contrast**
  Understand the differences between shared memory and message passing models

Focus on conceptual understanding rather than memorization

Review Questions

Test your understanding of the key concepts covered in this lecture on Inter-Process Communication (IPC) Mechanisms.

1 IPC Models

What are the main differences between shared memory and message passing?

2 Pipes and FIFOs

How do pipes and FIFOs differ?

3 Signals

What role do signals play in IPC?

4 Synchronization

Why is synchronization important in IPC?

Processes — Communication — Resources