

**Department of Computer Engineering**

**Course: Operating Systems**

**Course Code: BTECCE21502**

**Mini-Project -Report**

**PART - A**

**Guidance By - Prof. Noshir Tarapore**

**Topic: IPC-Based Process Coordination System**

**By**

|  |  |  |
| --- | --- | --- |
| **Roll No** | **SRN** | **Name of Student** |
| 38 | 202200930 | Shyamal S. Patil |
| 39 | 202200931 | Shaikh M. Asad |
| 33 | 202101389 | Tanmay Gawali |

**Part A: Design of system for coordinating multiple processes or threads using IPC mechanisms**

**What is interprocess communication?**

Official definition:

Interprocess communication (IPC) is used for programs to communicate data to each other and to synchronize their activities. Semaphores, shared memory, and internal message queues are common methods of interprocess communication.

IPC is a method for two or more separate programs or processes to communicate with each other. This avoids using real disk-based files and the associated I/O overhead to pass information. Like a file, you must first create or open the resource, use it and close it. Like real files, the resources have an owner, a group, and permissions. Until you remove the resource it continues to exist. Unlike real disk-based files, semaphores, message queues and shared memory do not persist across reboots.

Designing a system for coordinating multiple processes or threads using IPC (Inter-Process Communication) mechanisms requires careful consideration of the system's architecture, the communication methods, and the coordination of activities.

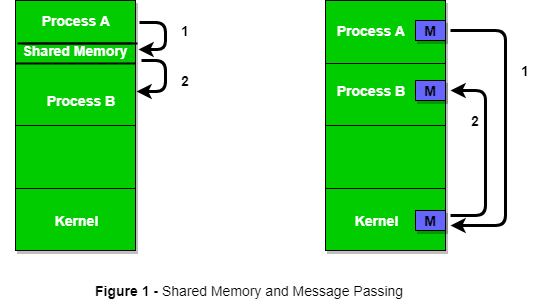
**System Architecture:**

1. **Coordinator:**
   * Central component responsible for managing and coordinating the activities of multiple processes or threads.
   * Maintains a dependency graph to represent the order in which processes or threads should execute.
   * Schedules the execution based on the dependency graph, ensuring that processes with unmet dependencies do not start prematurely.
   * Manages shared resources and synchronization primitives for inter-process synchronization.
2. **Worker Processes/Threads:**
   * Individual entities that perform specific tasks or computations.
   * Can be separate processes or threads that run concurrently.
   * Are aware of their dependencies and wait for the necessary processes or threads to complete before they start.
   * Communicate with the coordinator using IPC mechanisms for coordination and to receive instructions.

**Inter-Process Communication Mechanisms:**

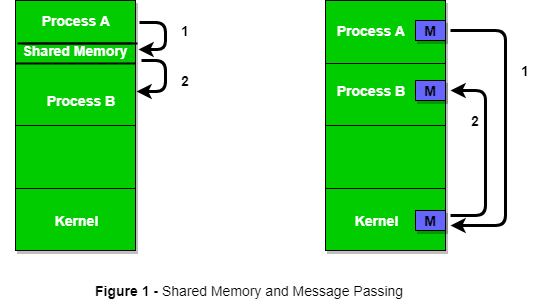
1. **Shared Memory:**

* Use shared memory to allow processes or threads to share data. Shared memory regions can be used for exchanging information.
* Processes or threads can read and write to shared memory segments.



1. **Message Passing:**

* Implement message passing for processes or threads to send and receive messages.
* Message queues, pipes, sockets, or other communication channels can be used for message passing.



1. **Synchronization Primitives:**

Use synchronization primitives such as semaphores, mutexes, and condition variables for managing access to shared resources and synchronizing the activities of processes or threads.

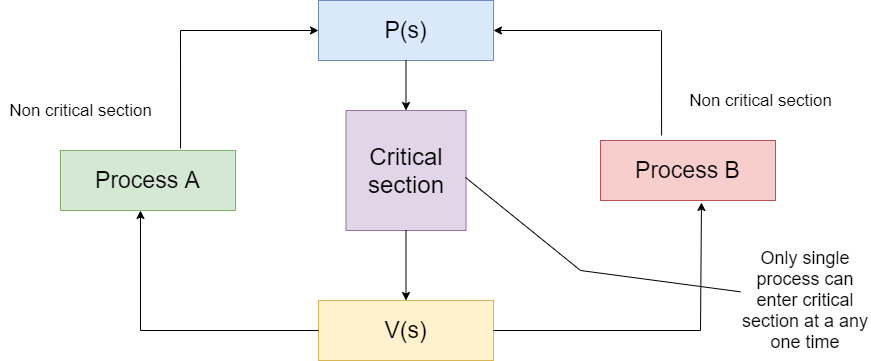
To avoid a series of issues caused by multiple processes (or programs) accessing a shared resource at the same time, we need a method that can be authorised by generating and using a token, so that only one execution thread can access the critical section of the code at any given time. The critical section is a code segment where the shared variables can be accessed and the atomic action is required in this section.

A semaphore is implemented as an integer variable with atomic increment and decrement operations; so long as the value is not negative the thread will continue, it will block otherwise. The increment operation is called **V**, or **signal**; the decrement is called **P**, or **wait**.

* **wait()**: decreases the counter by one; if the counter is negative, then it puts the thread on a queue and blocks.
* **signal()**: increments the counter; wakes up one waiting process.

The most common and simplest kind of semaphore is called a**binary semaphore** because they have two states *locked*or *unlocked*. It guarantees that only one process will be in a critical section at a time.

Diagram showing how the P and V operators act as a gate into critical sections of code.



**Dependency Management:**

1. Dependency Graph:
   * Maintain a dependency graph that represents the relationships between processes or threads.
   * Each node in the graph corresponds to a process or thread, and edges represent dependencies.
2. Scheduling Algorithm:
   * Implement a scheduling algorithm to determine the execution order of processes or threads based on the dependency graph.
   * Ensure that processes or threads with unmet dependencies wait until their dependencies are satisfied.

**Error Handling:**

1. Error Detection:
   * Implement error detection mechanisms to handle unexpected issues such as deadlocks, process failures, or communication errors.
2. Corrective Actions:
   * Define actions to be taken in the event of errors or failures, such as restarting processes or releasing resources.

**Scalability and Performance:**

1. Consider the scalability of the system, including the ability to handle a growing number of processes or threads.
2. Optimize the system's performance, including minimizing communication overhead and ensuring efficient resource usage.

**Security:**

1. Implement security measures to protect the system from unauthorized access or tampering.
2. Ensure data integrity and confidentiality, especially if sensitive information is being exchanged.

**Logging and Monitoring:**

1. Implement logging and monitoring mechanisms to track the activities and performance of the system.
2. Log critical events, errors, and resource utilization to facilitate debugging and analysis.