




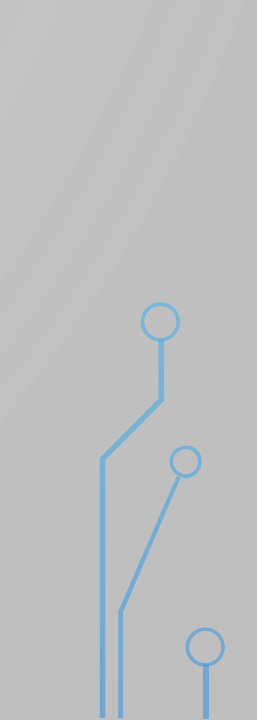
INTERPROCESS COMMUNICATION

CPE / CSC 159: OPERATING SYSTEM PRAGMATICS

GREG CRIST (CRIST@CSUS.EDU)



INTERPROCESS COMMUNICATION

- Overview
 - Why is it needed?
 - Methods and types
 - IPC Challenges
 - How is IPC Implemented
- 
- 

OVERVIEW

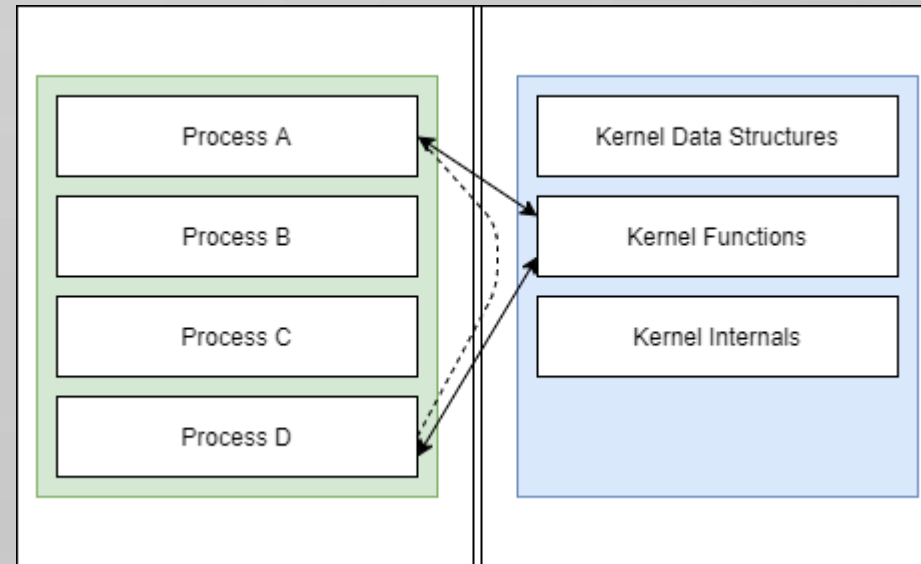
- Up until now, we have instantiated processes that act in isolation
 - While we can run multiple processes, they are not aware of other processes
 - Data that is manipulated is done so within the context of a single process
- Interprocess Communication (IPC) is a mechanism provided by the operating system that enables processes (user or kernel) to communicate with each other
- What does it mean to “communicate with each other”?
 - Coordinate access to shared data / resources
 - Exchange / transfer data across process/memory spaces

WHY IS IT NEEDED?

- Necessity:
 - Maintain the barrier between user and kernel contexts
 - Maintain barriers between user processes
 - Allow for exchange of data / interact with data across process spaces
- Complexity
 - Programs/processes that act alone are inherently limited in their functionality
 - Providing communication methods allows for more complex programs to be developed
- Desires
 - Modularity
 - Performance

WHY IS IT NEEDED?

- All interactions for a process outside of it's own memory space / instructions require coordination of the kernel



METHODS AND TYPES

- Various types of interprocess communication
 - Files, Sockets, Pipes
 - Shared Memory, Memory Mapped Files
 - Message Queues, Message Passing
 - And more!
- Different methods for different purposes

FILES, SOCKETS, PIPES

- Files
 - File, data, or “record” stored on disk
 - Can be accessed by multiple processes
- Sockets
 - Data sent/received between processes through a network protocol
 - Can be local or remote
- Pipes (unnamed or named)
 - Unidirectional data channel (bidirectional uses two pipes)
 - Data can be buffered until it is read from the receiving end

SHARED MEMORY, MEMORY MAPPED FILES



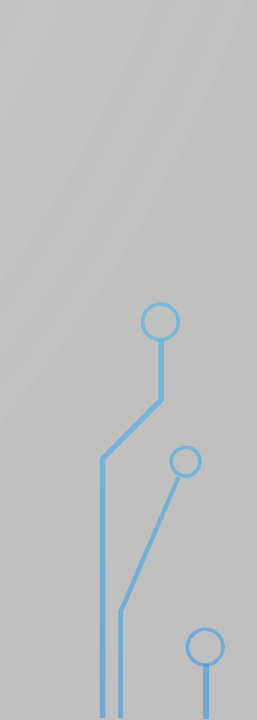
- Shared Memory
 - Multiple processes utilize a common memory location and can read/write
- Memory mapped file
 - Data for a file on disk is mapped into memory so it can be accessed like normal memory, but is also synchronized like a normal file

MESSAGE QUEUES, MESSAGE PASSING

- Message Queues
 - Allows "messages" or data-packets to be sent / received between processes
 - Unidirectional queues
- Message Passing
 - Makes use of message queues to exchange data / communicate
 - Can also be used for signaling



IPC CHALLENGES

- Concurrency
 - Critical Sections
 - Synchronous-vs-Asynchronous Operations
- 
- 
- 

CONCURRENCY

- Consider multiple processes may read/write data at any given time
- As multiple processes access data, how do you ensure data integrity?
 - Multiple readers / multiple writers
 - Partial write operations
- Problem scales as more processes are involved with accessing the same data

CRITICAL SECTIONS

- A critical section is a piece of code / data that needs to be protected from a concurrent operation
- Not exclusive to a specific context
 - Can span user processes in user-contexts
 - Can span the user context / kernel context boundary
- Requires methods to protect these critical sections
 - Synchronization

SYNCHRONIZATION

- Synchronization is used to ensure protection of critical sections that arise out of concurrent operations
- Different types of approaches:
 - Semaphores: simple variable used to control access to a critical section
 - Mutexes: mutual exclusion – allows only one process to enter a critical section
 - Barriers: processes wait for others before proceeding into a critical section

SEMAPHORES


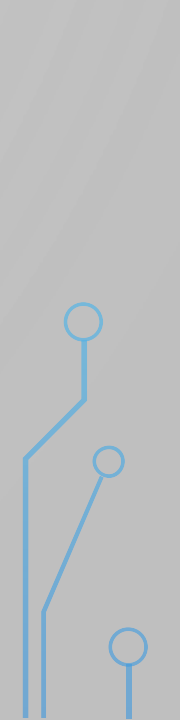
- What is a semaphore?
 - Oxford dictionary: a system of sending messages by holding the arms or two flags or poles in certain positions to an alphabetic code
 - Computer science: A variable used to signal when a resource is available
- Can be a simple variable that is toggled, or incremented/decremented to keep track of accesses to a given resource
 - A resource can be “anything”

MUTEXES

- What is a mutex?
 - Mutex = Mutual Exclusion
- Typically used in terms of a “lock” to ensure that only one process can enter a critical section at a time



SYNCHRONOUS VS ASYNCHRONOUS

- Synchronous
 - Some operations **need to be** synchronous: in a specific order
 - Typically requires some sort of “blocking” mechanism
 - Asynchronous
 - Some operations **may be** asynchronous: can occur in any order
 - Typically allows for “non-blocking” mechanisms
- 
- 

BLOCKING VS NON-BLOCKING

- **Blocking Operations**

- The calling process must wait on some external trigger
- Only continues with execution once complete

- **Non-Blocking Operations**

- The calling process doesn't need to wait and can continue with execution

HOW IS IPC IMPLEMENTED?

- User process perspective:
 - system calls
- Kernel perspective:
 - new data structures!
 - new process lifecycle!