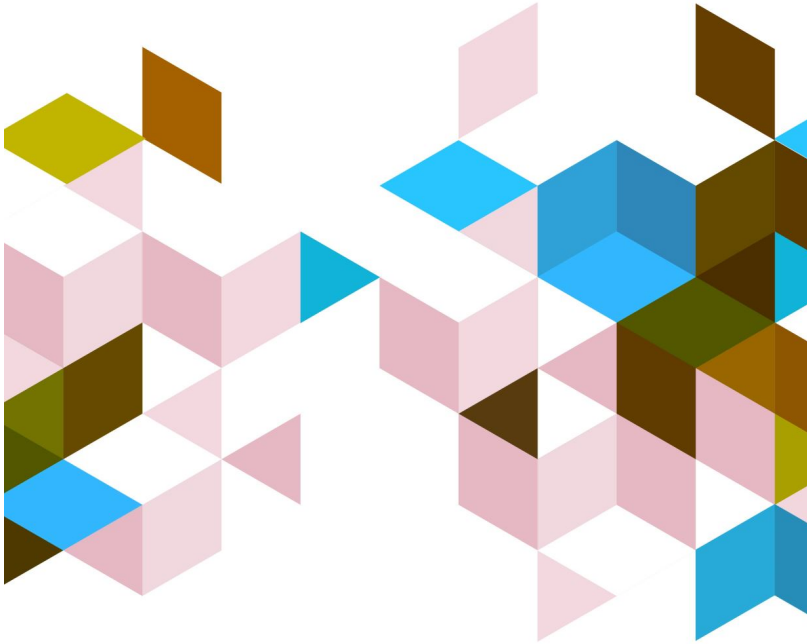
Understanding Process Components and Management



What is a process?

A process is a program in execution.

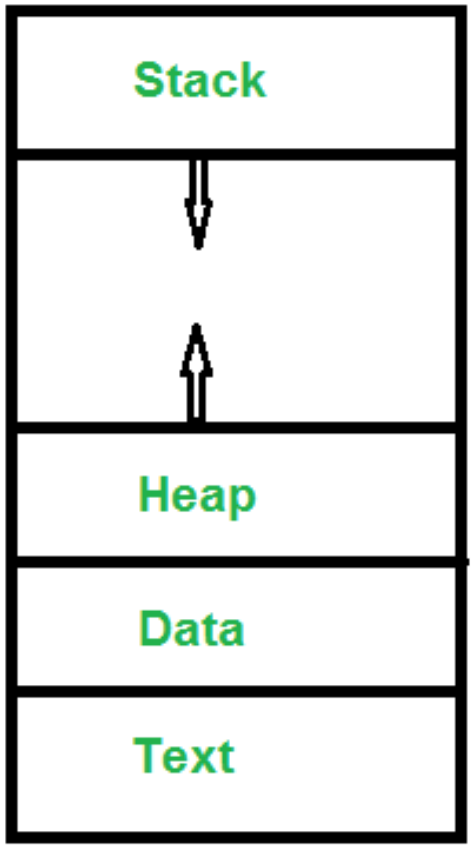
Ex. When we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we run the binary code, it becomes a process.

What is Process Management?

- **Definition**: Techniques and strategies used to design, monitor, and control business processes for efficient and effective goal achievement.
- **Importance in Operating Systems**:
  - Critical for multi-user environments.
  - Tracks and schedules processes.
  - Dispatches processes sequentially.
  - Ensures users feel they have full control of the CPU.

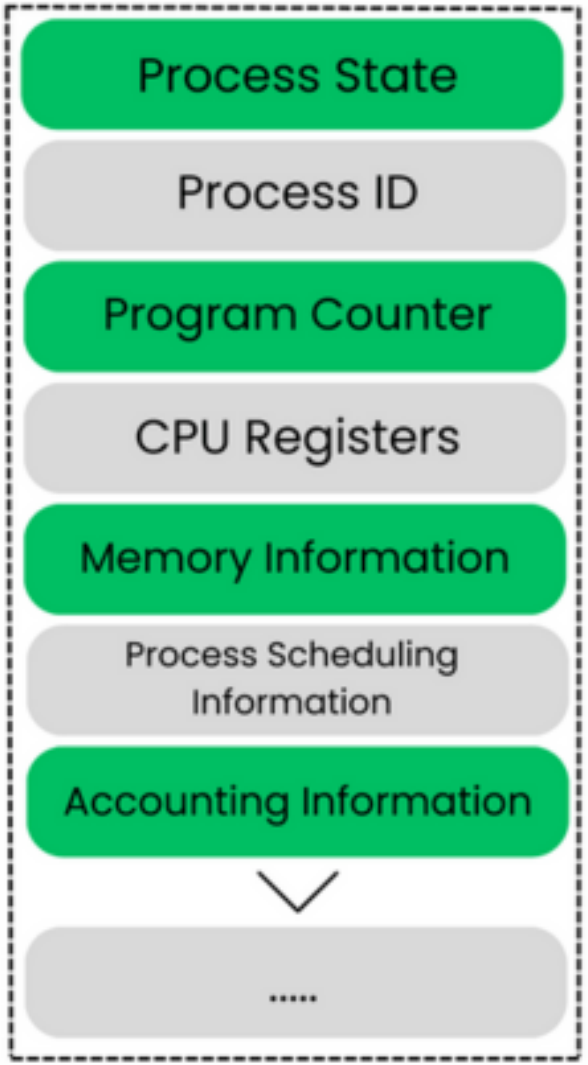
Benefits

- **Operational Efficiency**: Streamlines task completion steps and resource usage.
- **Cost Reduction**: Identifies and eliminates inefficiencies.
- **Customer Satisfaction**: Improves service delivery and responsiveness.
- **Compliance**: Ensures adherence to regulatory standards.



Process Components

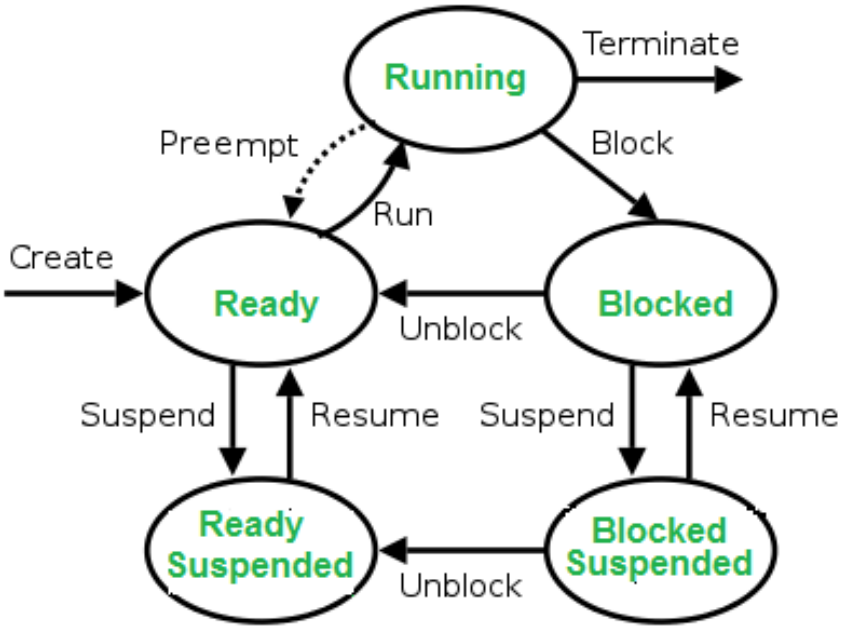
- **Text Section:** Contains the code and current activity (Program Counter).
- **Stack:** Holds temporary data like function parameters and local variables.
- **Data Section:** Stores global variables.
- **Heap Section:** Manages dynamic memory allocation.



Characteristics of a Process

- **Process Id:** A unique identifier assigned by the operating system.
- **Process State:** Can be ready, running, etc.
- **CPU Registers:** Like the Program Counter (CPU registers must be saved and restored when a process is swapped in and out of the CPU).
- **Accounts Information:** Amount of CPU used for process execution, time limits, execution ID, etc.
- **I/O Status Information:** For example, devices allocated to the process, open files, etc.
- **CPU Scheduling Information:** For example, Priority (Different processes may have different priorities, for example, a shorter process assigned high priority in the shortest job first scheduling).

Note: Diagram shows a typical structure of a **Process Control Block (PCB)** containing these attributes.



States of Process

- **New:** Newly Created Process (or) being-created process.
- **Ready:** After creation, process moves to Ready state, i.e., ready for execution.
- **Run:** Currently running process in CPU (only one process at a time can be under execution in a single processor).
- **Wait (or Block):** When a process requests I/O access.
- **Complete (or Terminated):** The process completed its execution.
- **Suspended Ready:** When the ready queue becomes full, some processes are moved to a suspended ready state.
- **Suspended Block:** When the waiting queue becomes full.

Real-Life Scenario: Cooking a Meal in a Kitchen

Imagine you're cooking a meal in your kitchen. You have several tasks to complete, like chopping vegetables, boiling water, and frying ingredients. Your kitchen has limited resources—only so many burners on the stove, a single cutting board, and one set of knives.

- **Processes:** Cooking a dish, boiling water, chopping vegetables.
- **Resources:** Stove, pots, knives, ingredients.
- **States:**
  - **Ready:** Ingredients are prepped, but not yet cooking.
  - **Running:** A dish is being cooked on the stove.
  - **Waiting:** Waiting for the water to boil.
  - **Terminated:** The dish is cooked and ready to be served.

Activity

Case Scenario - The Café Analogy

Imagine you are the manager of a busy café. The café has a limited number of tables, baristas, and kitchen space to serve customers. Each customer who enters needs to be seated, place an order, and have their food or drinks prepared by the baristas.

Questions

1. In the café scenario, who or what represents a process? Explain how this analogy helps you understand what a process is in an operating system.
2. In the café scenario, who or what represents a CPU in a computer system? How does this help you understand the "running" state of a process?
3. In the café scenario, who or what represents a system resource? How does this help you understand the "waiting" or "blocked" state of a process?
4. What happens when multiple customers arrive at the café at the same time? How does this scenario relate to processes in an operating system?
5. Consider a scenario where a customer must leave suddenly before finishing their coffee. How would you relate this to the concept of process?
6. How does the café handle peak hours when many customers arrive simultaneously? Relate this to how an operating system manages multiple processes that need CPU time and other resources.

In the café scenario, who or what represents a process? Explain how this analogy helps you understand what a process is in an operating system.

Each customer represents a process. Just like a process in an OS, a customer performs tasks (ordering, eating) and requires resources (table, barista). This helps us see a process as an active entity that moves through different stages.

In the café scenario, who or what represents a CPU in a computer system? How does this help you understand the "running" state of a process?

The barista is like the CPU, executing tasks (making coffee) for customers (processes). When a barista is working on an order, the process is in the "running" state, similar to the CPU executing a process.

In the café scenario, who or what represents a system resource? How does this help you understand the "waiting" or "blocked" state of a process?

Tables represent system resources like memory. If all tables are occupied, customers must wait, similar to how a process waits when needed resources are unavailable.

What happens when multiple customers arrive at the café at the same time? How does this scenario relate to processes in an operating system?

When multiple customers arrive, the baristas must decide who to serve first, similar to how an OS schedules processes. The order of service reflects process scheduling.

Consider a scenario where a customer must leave suddenly before finishing their coffee. How would you relate this to the concept of process?

If a customer leaves suddenly, it's like a process being terminated early, freeing up resources (table, barista time) for others. This is similar to closing or crashing a program.

How does the café handle peak hours when many customers arrive simultaneously? Relate this to how an operating system manages multiple processes that need CPU time and other resources.

During peak hours, the café manages a high load by prioritizing orders and optimizing service. This is like an OS managing multiple processes efficiently under heavy load.

Types of Processes

- **I/O Bound**:
  - Spends more time doing I/O than computations
  - Many short CPU bursts
- **CPU Bound**:
  - Spends more time doing computations
  - Few very long CPU bursts

* CPU burst is a period when the CPU is fully engaged with a process's tasks

Ex. I/O-Bound Processes

A text editor that frequently reads from and writes to files is I/O-bound. It spends more time waiting for file I/O operations than it does processing text.

Ex. CPU-Bound Processes

A video encoding application or a scientific simulation is CPU-bound. It spends most of its time performing complex calculations, utilizing the CPU heavily, with minimal I/O operations.

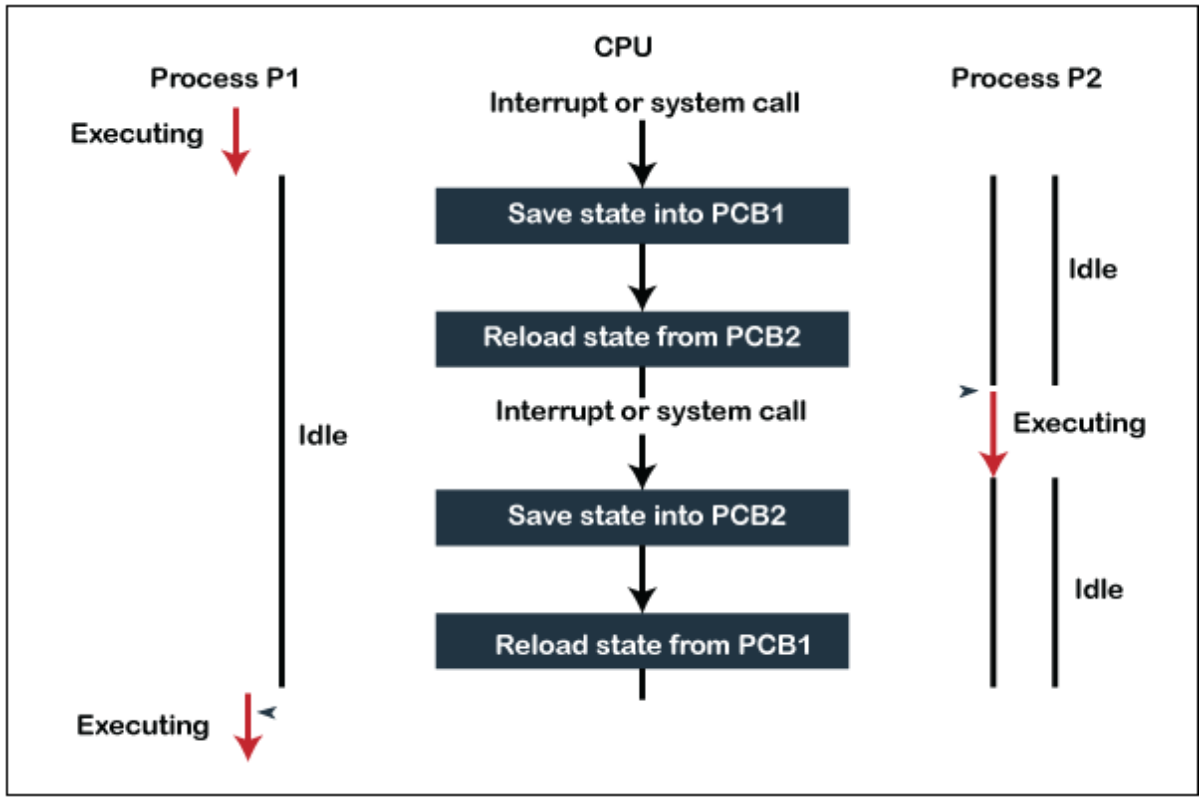
Importance in Scheduling

- Understanding whether a process is I/O-bound or CPU-bound helps the OS's scheduler to allocate CPU time more effectively.

For example, Scheduling algorithms may prioritize I/O-bound processes to keep the system responsive. Since I/O-bound processes often spend time waiting for I/O and quickly free up the CPU, this allows CPU-bound processes to run during those idle periods, making the system more efficient.

Context Switching

- **When CPU switches to another process:**
  - Save the state of the old process (to PCB)
  - Load the saved state (from PCB) for the new process
- **Context Switch Time**: Dependent on hardware

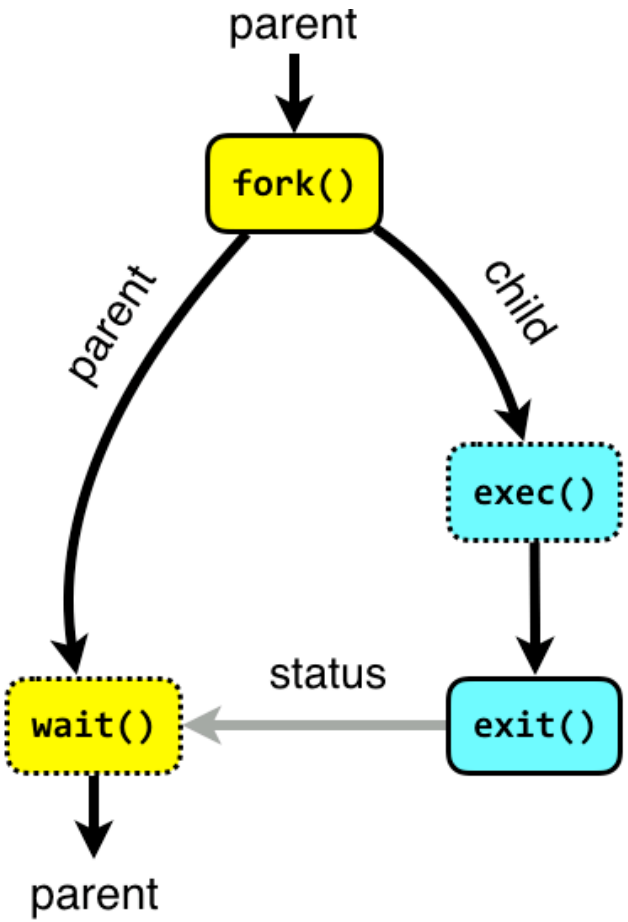


Process Creation

- **Parent Processes Create Children Processes** (form a tree)
- **Process Management**: Using PID
- **Resource Sharing**: All, some, or none
- **Execution**:
  - Concurrently with children
  - Wait until children terminate

System Calls

- **fork()**: Creates a new process
- **exec()**: Replaces the process's memory space with a new program



Understanding Forking with Illustrative Example

- Imagine one participant draws a simple picture or scene on their paper. This drawing represents the **"parent process"** in an operating system.
- Now, another participant is asked to replicate this drawing exactly on their own sheet of paper. This second drawing represents the **"child process"** that is created when a process is forked.
- Both participants work independently. Changes made to one drawing do not affect the other. This illustrates how **each process has its own space and does not automatically share changes with the other.**

Reflection

- **Process Independence:** The drawings show how processes run separately and do not affect each other.
- **Initial Copy, Independent Evolution:** The child process starts as a copy with the same state but can develop differently, similar to the drawings.
- **Separation of Tasks:** Changes in one process do not impact the other, demonstrating effective multitasking.

Modification Impact

- Changes made to shared resources by either the parent or child process will affect both.

Example: If the parent process modifies a file descriptor, the child process will see the change too.

Undesirable Changes

- Unintended modifications can lead to unexpected behavior in either process.

Example: If one process updates a shared variable without coordination, the other process might operate on outdated or incorrect data.

To address Modification Impact and Undesirable Changes, you should:

- Implement synchronization mechanisms.
- Manage file descriptors carefully to avoid unintended modifications.
- Use atomic operations for shared variables.
- Employ IPC mechanisms and locking strategies.
- Design consistent access patterns for shared resources.

Alternatives to fork

- **posix_spawn:** Simplifies process creation with reduced overhead and fewer pitfalls.
- **vfork:** Creates a new process sharing the parent's address space until exec is called.
- **clone:** Provides fine-grained control over shared resources between parent and child.
- **CreateProcess:** Windows API for creating and initializing new processes.
- **Microkernel APIs:** Offers modular process creation and mangement without fork.

Despite its limitations, fork is still commonly used because:

- **Legacy Systems:** Many existing applications depend on fork, and changing them would be complex.
- **Simplicity:** fork offers a straightforward way to create processes, especially in traditional Unix applications.
- **Copy-on-Write:** Modern fork implementations use copy-on-write to efficiently manage memory.
- **Historical Precedent:** fork is a long-standing standard in Unix-like systems, maintaining consistency with past practices.
- **Application Design:** Some applications are built around fork, making it challenging to switch.
- **Limited Alternatives:** Alternatives like posix_spawn and vfork don't fully replace all of fork's use cases or functionality.

Activity

Step 1: Read the Code

Read through the provided code snippet to understand its structure and what it does.

Step 2: Identify Fork Calls

Identify the locations in the code where the fork() function is called. Each fork() call creates a new child process.

Step 3: Count Forks

Count the number of fork() calls in the code. For each fork() call, determine how many additional processes it creates.

Step 4: Calculate Total Forks

Add up the number of additional processes each fork() call creates to find the total number of forks created in the code.

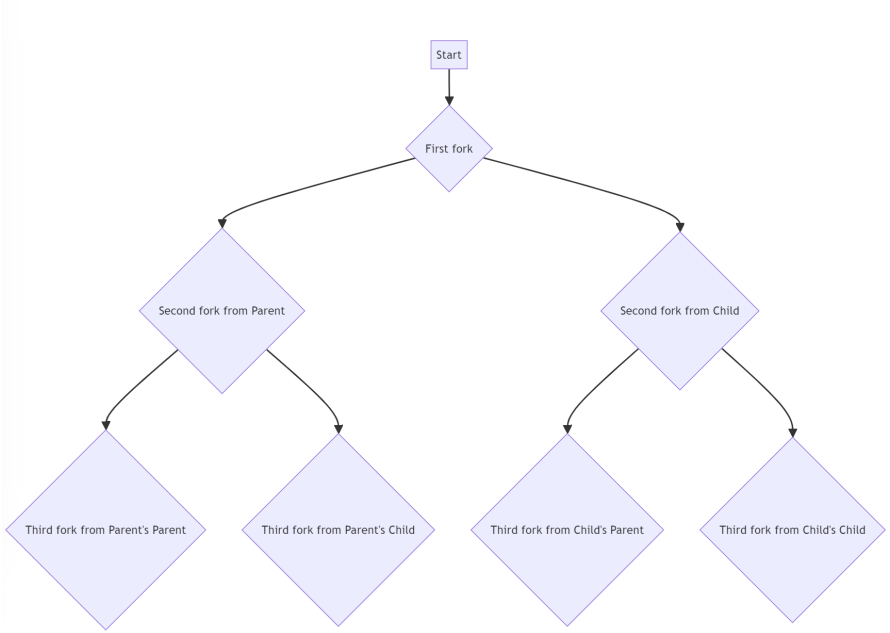
Step 5: Final Answer

Share your findings. How many forks are created in the provided code snippet?

Code Snippet

```
#include <stdio.h>
#include <unistd.h>

int main() {
    fork();
    fork();
    fork();
    return 0;
}
```



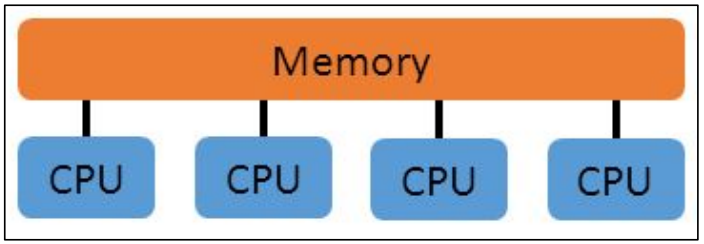
Answer

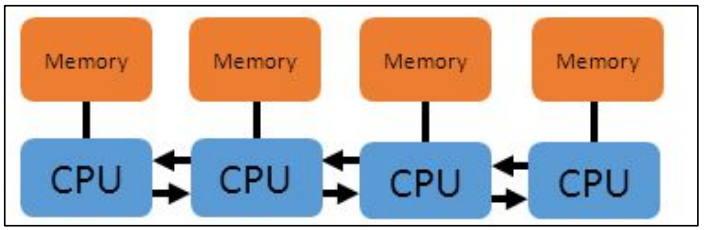
Let's count the number of forks created in the code:

- The first fork() creates one additional process.
- The second fork() is encountered after the first fork, so it creates another two processes (one for each existing process).
- The third fork() is encountered after the second fork, so it creates another four processes (one for each existing process).

So, in total, there are 1 (first fork) + 2 (second fork) + 4 (third fork) = 7 forks created.

Interprocess Communication (IPC)





- Shared memory: A method where multiple processes share a common memory space to communicate and exchange data.
- Message passing: A method where processes communicate by sending and receiving messages, usually through a communication channel or a messaging queue.

Message Passing

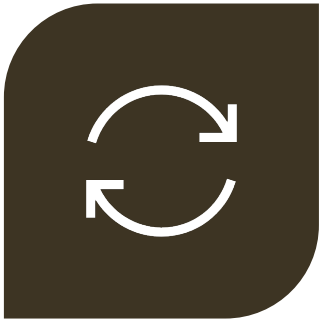
Blocking (Synchronous):

- Blocking send: Sender waits until the message is received.
- Blocking receive: Receiver waits until a message is available.

Non-blocking (Asynchronous):

- Non-blocking send: Sender sends and proceeds without waiting.
- Non-blocking receive: Receiver checks for a message; continues if none is found.

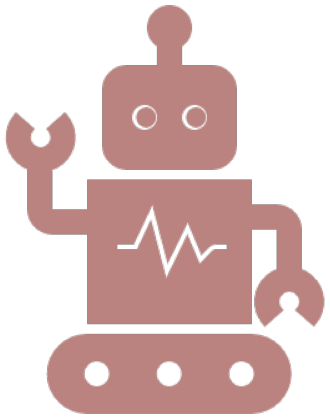
Current Challenges in OS Processes



CONCURRENCY ISSUES: MANAGING MULTIPLE PROCESSES EFFICIENTLY, ESPECIALLY IN MULTI-CORE SYSTEMS.

SECURITY IN PROCESSES: ENSURING THAT PROCESSES DO NOT INTERFERE WITH EACH OTHER AND PROTECTING SYSTEM INTEGRITY.

RESOURCE ALLOCATION: FAIR AND EFFICIENT DISTRIBUTION OF CPU, MEMORY, AND I/O RESOURCES AMONG COMPETING PROCESSES.



With AI and Large Language Models rising, data sovereignty, user control, identity, and the proprietary vs. open-source debate are back. Should we rethink yesterday's OS Processes for tomorrow?

The Need for Advanced OS Processes

- **Dynamic Resource Management:** Adaptive algorithms that adjust resource allocation based on real-time needs.
- **AI-Assisted Process Scheduling:** Utilizing AI to predict and optimize task scheduling for better performance.
- **Enhanced Security Protocols:** Advanced techniques to isolate and protect processes from malicious interference.

Core Processes of a Future OS

- **Modular Process Management:** A system where processes can be independently updated and optimized.
- **Context-Aware Processing:** Processes that adapt based on the current environment, device, and user behavior.
- **Scalable Process Handling:** Efficient process management across diverse devices, from IoT to high-performance computing.

Potential Features in Process Management

- **AI-Driven Resource Allocation:** AI optimizes how resources are distributed to processes based on usage patterns.
- **Seamless Cross-Platform Processes:** Processes that maintain state and functionality across different devices.
- **User-Controlled Process Prioritization:** Allowing users to prioritize which processes should receive more system resources.

How can an operating system balance process efficiency with user experience?

An OS can balance efficiency and user experience by implementing adaptive process management. For instance, the OS could **prioritize foreground applications** (those actively used by the user) while optimizing background tasks to minimize resource use. AI-driven predictions can enhance responsiveness, ensuring **critical tasks receive more CPU and memory without causing delays**. Additionally, providing users with customizable settings allows them to decide how much system performance is devoted to specific tasks, enhancing overall satisfaction while maintaining efficiency.