

IPC and Threads and Locks

Refresher Concepts:

IPC

Interprocess Communication (IPC): A mechanism allowing processes to exchange data and coordinate their activities.

Each process has its **own private memory**: IPC enables communication **while maintaining isolation**.

Its goal is to enable **cooperation and coordination** among *concurrent or distributed processes*.

Why is IPC needed?

Data sharing: safe, efficient sharing across process boundaries.

Modular design: large programs are split into smaller cooperating processes.

Concurrency/parallelism: synchronize and coordinate processes.

Resource sharing: manage simultaneous access to shared files/resources.

Distributed systems: enable communication between processes on different machines.

Mechanism	Description
Pipes	Unidirectional or bidirectional communication channel; buffered.
Shared Memory	Fastest IPC; multiple processes access a common memory region .
Message Queues	Processes send/receive structured messages asynchronously.
Signals	Lightweight notifications (software interrupts).
Sockets	Allow communication over networks (e.g., between hosts).

Pipes

- **Function**: Transfer data between processes (usually related ones, parent & child).
- **Unidirectional** by default: one writes, the other reads.
- **Buffering**: Temporarily holds data until read.
- **Blocking behavior**:
 - Read blocks if no data.
 - Write blocks if the buffer is full.
 - **Nonblocking mode** can be enabled.
- **File descriptors**: fd[0] = read end, fd[1] = write end.

Named Pipes (FIFOs)

- Special file in the filesystem for IPC.
- **Persistent** (exists beyond process lifetime until removed).
- **Unrelated processes** can communicate through it.
- **Half-duplex**: One-way communication at a time.
- **Blocking**:
 - Read blocks if empty.
 - Write blocks if full.

File Permissions Refresher

- **r (read) = 1, w (write) = 2, x (execute) = 4**
- Permissions are given separately to:
 - **Owner, Group, Others**
- Example:
775 = Owner(7=rwx), Group(7=rwx), Others(5=rx)

Signals

- **Definition**: Software interrupts are sent by the OS, another process, or the process itself.
- **Sources**: OS (e.g., segmentation fault), another process (kill()), or self (raise()).
- **Each signal has**:
 - **Default action**: terminate / ignore / stop / continue
 - **Custom handler**: a function defined to handle it differently

Signal	Default Action	Notes
SIGINT	Terminate	Ctrl + C in the terminal
SIGTERM	Terminate	Graceful termination
SIGKILL	Terminate (uncatchable)	Cannot be handled/ignored
SIGSEGV	Terminate + core dump	Invalid memory access
SIGABRT	Terminate + core dump	From abort()
SIGCHLD	Ignore	Sent to the parent when the child exits
SIGSTOP	Stop (uncatchable)	Suspend process
SIGCONT	Continue	Resume if stopped

SIGUSR1/2	Terminate	User-defined signals
------------------	-----------	----------------------

Signals for IPC

- Useful for **simple event notifications** (e.g., task start/stop).
- **User-defined signals:** SIGUSR1, SIGUSR2.
- **Limitations:** Cannot send large data, only small payloads.

Function	Purpose
signal(signum, handler)	Install a handler for a signal
kill(pid, signum)	Send a signal to the process PID
raise(signum)	Send signal to current process/thread
sigqueue(pid, signum, value)	Send a signal with a small data payload
pause()	Wait until any signal arrives

kill() Misnomer

- Despite the name, it **does not always kill a process**.
- Behavior depends on the signal:
 - kill(pid, SIGKILL) -> uncatchable termination
 - kill(pid, SIGUSR1) -> triggers handler
 - kill(pid, 0) -> checks if process exists (no signal sent)
- Think of it as “**send signal**”, not “kill”.

Threads and Locks

The system creates a new thread of execution for the routine that is being called, and it runs independently of the caller. What runs next is determined by the OS scheduler.

pthread - POSIX threads. POSIX.1 specifies a set of interfaces (functions, header files) for threaded programming commonly known as POSIX threads, or Pthreads.

Read more: <https://man7.org/linux/man-pages/man7/pthreads.7.html>

Concurrency

- So far, each process = single point of execution (one PC).
- Now: Threads -> multiple points of execution within the same process.

- Each thread has its own PC, registers, and stack, but shares:
 - Code
 - Data
 - Heap

Threads share memory space but processes don't.
- A multi-threaded process = one process with multiple threads of execution.

Thread Details

Shared	Private (per thread)
Code	Program Counter (PC)
Data	Stack
Heap	Registers

- OS maintains **Thread Control Block (TCB)** for each thread (analogous to PCB).
- During a thread context switch, OS saves/restores thread state from TCB.
- No page table switch is required (threads share same address space).

Why Use Threads?

1. Parallelism
 - Utilize multiple CPUs/cores simultaneously.
 - e.g., adding two large arrays using multiple threads.
2. Avoid Blocking
 - Overlap I/O with computation.
 - While one thread waits for I/O, another can continue processing.

Shared Data & Race Conditions: when multiple threads update shared data without synchronization:

- Execution order depends on uncontrolled scheduling.
- Output becomes indeterminate (different results on different runs).
- This happens because threads access shared variables concurrently.

Critical Section

- The code segment accessing shared variables.
- Must not be executed by more than one thread at the same time.
- Otherwise, race condition occurs (nondeterministic results).

Avoiding Race Conditions

- Ensure Mutual Exclusion (Mutex): Only one thread executes a critical section at any given time. Threads must synchronize before accessing shared data. Goal: deterministic and safe behavior.
- Locks (Synchronization Mechanism): Used to enforce mutual exclusion. Programmers place locks around critical sections.

Evaluating Locks

Criterion	Description
Correctness	Ensures true mutual exclusion: no simultaneous access.
Fairness	All threads get a fair chance to acquire lock eventually.
Performance	Adds minimal overhead and scales well with contention.

Naïve Lock (Simple Flag)

- A simple shared flag (e.g., lock = 0/1) fails because:
 - Race conditions can still occur if both threads check the flag simultaneously.
 - Leads to busy waiting (spin-waiting), thus wasting CPU cycles.
- Hence, this is not a valid mutual exclusion solution. *What then?*

Spin Locks & Test-And-Set Instruction

- Spin Lock: Threads continuously check (spin) until the lock is free.
- Built using atomic hardware instructions such as Test-and-Set (TAS).
- TestAndSet(): Atomically reads and sets the lock value and prevents multiple threads from entering simultaneously.

Questions:

Q1 (from last week's syllabus). Use the clock algorithm that considers reference/use and dirty bits for replacing pages. Find the number of hits, misses and the hit rate for the following pages. The cache/frame size is 4. The reference bit of all the pages is initially set to 1. The dirty bit of pages 1, 2, 3 are set to 1 and is 0 for the rest of the pages. Show all the steps, including the state of the cache at each step, the reference bit and use bit for each page. Clearly state your assumptions, for example, the position of the hand.

Page sequence: 1, 2, 3, 4, 5, 1, 6, 2, 7, 8, 1, 2, 9, 10

Ans. Assumptions: Clock hand always starts with page 1 in the cache.

Step	Page	Cache state after insertion -	Hit/Miss	Comments
------	------	-------------------------------	----------	----------

		(page, use bit, dirty bit)		
--	--	----------------------------	--	--

1	1	(1,1,1)	Miss	
2	2	((1,1,1), (2,1,1))	Miss	
3	3	((1,1,1), (2,1,1), (3,1,1))	Miss	
4	4	((1,1,1), (2,1,1), (3,1,1), (4,1,0))	Miss	
5	5	((1,0,1), (2,0,1), (3,0,1), (5,1,0))	Miss	Replace 4 in the second pass since that is the only page with dirty bit 0.
6	1	((1,1,1), (2,0,1), (3,0,1), (5,1,0))	Hit	
7	6	((1,0,1), (2,0,1), (3,0,1), (6,1,0))	Miss	Replace 5 in the second pass since that is the only page with dirty bit 0.
8	2	((1,0,1), (2,1,1), (3,0,1), (6,1,0))	Hit	
9	7	((1,0,1), (2,0,1), (3,0,1), (7,1,0))	Miss	Replace 6 in the second pass since that is the only page with dirty bit 0.
10	8	((1,0,1), (2,0,1), (3,0,1), (8,1,0))	Miss	Replace 7 in the second pass since that is the only page with dirty bit 0.
11	1	((1,1,1), (2,0,1), (3,0,1), (8,1,0))	Hit	
12	2	((1,1,1), (2,1,1), (3,0,1), (8,1,0))	Hit	
13	9	((1,0,1), (2,0,1), (3,0,1), (9,1,0))	Miss	Replace 8 in the second pass since that is the only page with dirty bit 0.
14	10	((1,0,1), (2,0,1), (3,0,1), (10,1,0))	Miss	Replace 9 in the second pass since that is the only page with dirty bit 0.

- Hits: 4
- Total requests: 14
- Hit Rate = (Number of Hits) / (Total requests) = 0.286

Discuss the codes shared.