Operating Systems – Week 5 Lecture Notes

Instructor: SDB
Theme: Execution Management
Topic: *Threads vs Processes*

Lecture Script

Welcome to Week 5!

So far, we've treated the process as the basic unit of execution. But that's just the beginning — today, we introduce **threads**: lightweight execution units within a process.

Think of a process as a house, and threads as people living in it. They share resources (kitchen, bathroom), but they can all be doing different tasks at the same time.

Let's understand the trade-offs, models, and practical uses of multithreading.

Core Concepts and Definitions

❖ *Thread*

Definition: A **thread** is the smallest unit of execution.
It runs within a process and shares:

- Code section
- Data section (heap)
- Open files
- Address space

Each thread maintains its own:

- Stack
- Program counter (PC)
- Registers

❖ *Process vs Thread*

Feature	Process	Thread
Address Space	Independent	Shared
Overhead	High (full context switch)	Low (faster switch)
Creation Time	Relatively heavy	Light
Crash Impact	Isolated	Can bring down siblings
IPC	Via pipes/sockets	Direct (shared memory)

❖ *Types of Threads*

- **User-Level Threads (ULT)**:
  - Managed by user-space libraries
  - Fast creation/switching
  - Kernel unaware of them
- **Kernel-Level Threads (KLT)**:
  - Managed by the OS
  - True concurrency on multi-core systems
  - Higher overhead per thread

*Modern OSes support **POSIX threads** (pthreads) as KLT.*

Analogy: Text Editor + Spellcheck

A modern word processor:

- UI thread: handles user input
- Background thread: performs spellcheck
- Sync thread: saves document to cloud

All run as threads under the same process → share memory, but operate independently.

In-Class Live Code Demo (Optional)

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

void* print_msg(void* msg) {
   printf("%s\n", (char*)msg);
   return NULL;
}

int main() {
   pthread_t t1, t2;
   pthread_create(&t1, NULL, print_msg, "Hello");
   pthread_create(&t2, NULL, print_msg, "World");
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   return 0;
}
```

Use this to demonstrate:

- Thread creation
- Parallel execution
- Join behavior

Glossary of Terms

Term	Meaning
Thread	Lightweight unit of CPU scheduling within a process

Term	Meaning
Pthread	POSIX standard API for thread operations
User-Level Thread	Thread managed in user space
Kernel Thread	Thread managed directly by the OS
Join	Wait for a thread to finish execution
Race Condition	Bug from unsynchronized thread access to shared data

---

Topics for Exploration

- Thread-local storage (TLS)
- Thread safety and mutexes
- Multi-threading in Python (threading, multiprocessing)
- How modern browsers use threads (rendering, I/O)
- Green threads vs OS threads (e.g., Go vs C++)

✅Summary

- Threads are faster and lighter than processes
- Threads **share memory**, but **isolate stack and execution**
- Use threads for tasks that benefit from concurrency without full process overhead
- Modern OSes use kernel threads; applications can also abstract user threads
- Always manage shared data access (e.g., mutexes)

✍ Review & Exercises

1. Compare threads vs processes with respect to:
    - Memory use
    - IPC
    - Crash containment
2. Write a pthread-based program that spawns 5 threads to print a counter
3. Discuss why multithreaded apps need synchronization
4. Use `top` → press `H` to view threads
5. Observe thread behavior of Firefox or Chrome (`ps -Lf`)
6. Research: How does Java manage thread lifecycle?
7. Advanced: Create thread race condition using shared counter (without mutex), then fix it