Operating Systems – Week 3 Lecture Notes

Instructor: SDB
Theme: Execution Management
Topic: *Process Concepts, Process Control Block (PCB), and Process Lifecycle*

Lecture Script

Welcome back to our deep dive into execution management.

Last week we saw how system calls allow programs to interact with the kernel. Today, we turn our attention to the fundamental entity the OS manages: the **process**.

If the OS is a traffic manager, then processes are the vehicles — each with its own destination, fuel, speed, and identity. And the OS needs to track and switch between thousands of them without crashing.

Let's understand what a process is, how it's represented internally, and how it flows through different states in its lifetime.

Core Concepts and Definitions

❖ *What is a Process?*

Definition: A process is an executing instance of a program, encapsulating:

- Code
- Data
- Execution state
- Allocated resources (memory, open files, etc.)

It is created, scheduled, and managed by the OS.

❖ *Program vs Process*

- A **program** is passive (e.g., a `.c` file or binary).
- A **process** is active, with its own **program counter**, **stack**, and **execution context**.

Multiple processes can run the same program (e.g., 5 tabs in Chrome).

❖ *Process States*

State	Description
New	Process is being created
Ready	Waiting to be scheduled
Running	Actively executing on the CPU
Waiting	Blocked (e.g., waiting for I/O)

State	Description
Terminated	Finished execution

These transitions are managed by the **scheduler** and system events.

❖ Process Control Block (PCB)

Definition: The **PCB** is a kernel data structure that stores everything the OS needs to manage a process.

Contents include:

- Process ID (PID)
- Process state
- Program counter (PC)
- CPU register values
- Scheduling info (priority, time slices)
- Memory info (base, limit, stack pointers)
- Open files and I/O resources
- Accounting info (CPU time used, UID, etc.)

It's like a passport and black box of a process.

Caselet: A Browser Tab

Each Chrome tab runs in its own process:

- Has its own memory, state, open file handles (network sockets)
- If one crashes, others stay safe
- OS tracks each using a **separate PCB**
- The scheduler decides which tab (process) gets CPU next

Process Lifecycle Diagram (Textual)

```
+--------+     admit     +--------+     dispatch     +--------+
|  New   | ------------> | Ready  | ---------------> | Running|
+--------+               +--------+                  +--------+
                          ^  |                      |
                          |  |   interrupt, time-out    |
          I/O complete    |  +--------------------------+
          or child exit   |                            |
                          v                         v
                      +--------+                  +----------+
                      | Waiting| <----------------- | Terminated|
                      +--------+    I/O wait        +----------+
```

Shell Demo (In-Class)

```
ps -ef | grep bash           # Observe multiple bash processes
sleep 120 &                  # Background process creation
ps -p <pid> -o pid,ppid,stat,cmd
cat /proc/<pid>/status       # Inspect PCB fields (indirectly)
```

Optional: Run `top`, identify state transitions in real-time.

Glossary of Terms

Term	Meaning
Process	Running instance of a program
PCB	Process metadata managed by the OS
Context Switch	Saving/restoring process CPU state
Zombie	Terminated, not yet reaped process
Orphan	A process whose parent exited
Fork	Clones a process
Exec	Replaces current process memory with new code

Exploration Topics

- Threads and multithreaded processes
- Fork bomb and process limits (ulimit -u)
- /proc/[pid]/ virtual filesystem
- Process namespaces and containers
- How Windows and Linux differ in PCB layout

✓Summary

- A **process** is the basic unit of execution.
- The **PCB** stores all critical metadata to track, switch, and manage a process.
- A process transitions between states: new → ready → running → waiting → terminated.
- The OS uses **scheduling queues** to manage transitions and CPU allocation.

✍ Review & Exercises

1. Draw and label a complete **process lifecycle diagram**
2. Explain all fields in a **PCB** and their role in context switching
3. Differentiate **program vs process** with examples
4. Use ps, sleep, top, and /proc to observe real-time state transitions
5. Write a C program using fork() and observe two concurrent processes
6. What is a zombie? How is it created and removed?
7. Challenge: Create 10 background processes and monitor their behavior