CSCI 460 Operating Systems

Processes (Part III)

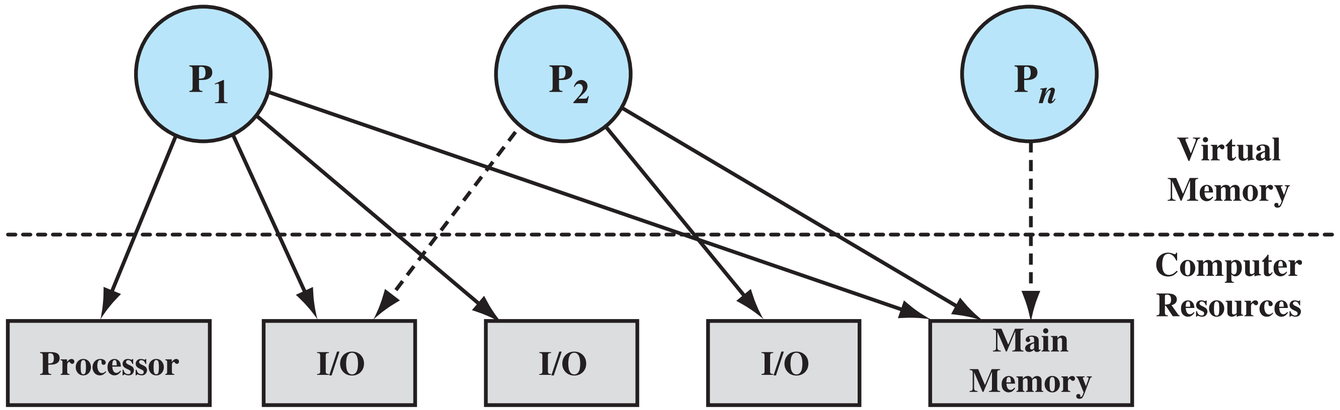
Professor Travis Peters
Fall 2019

Some slides & figures adapted from Stallings instructor resources.

Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

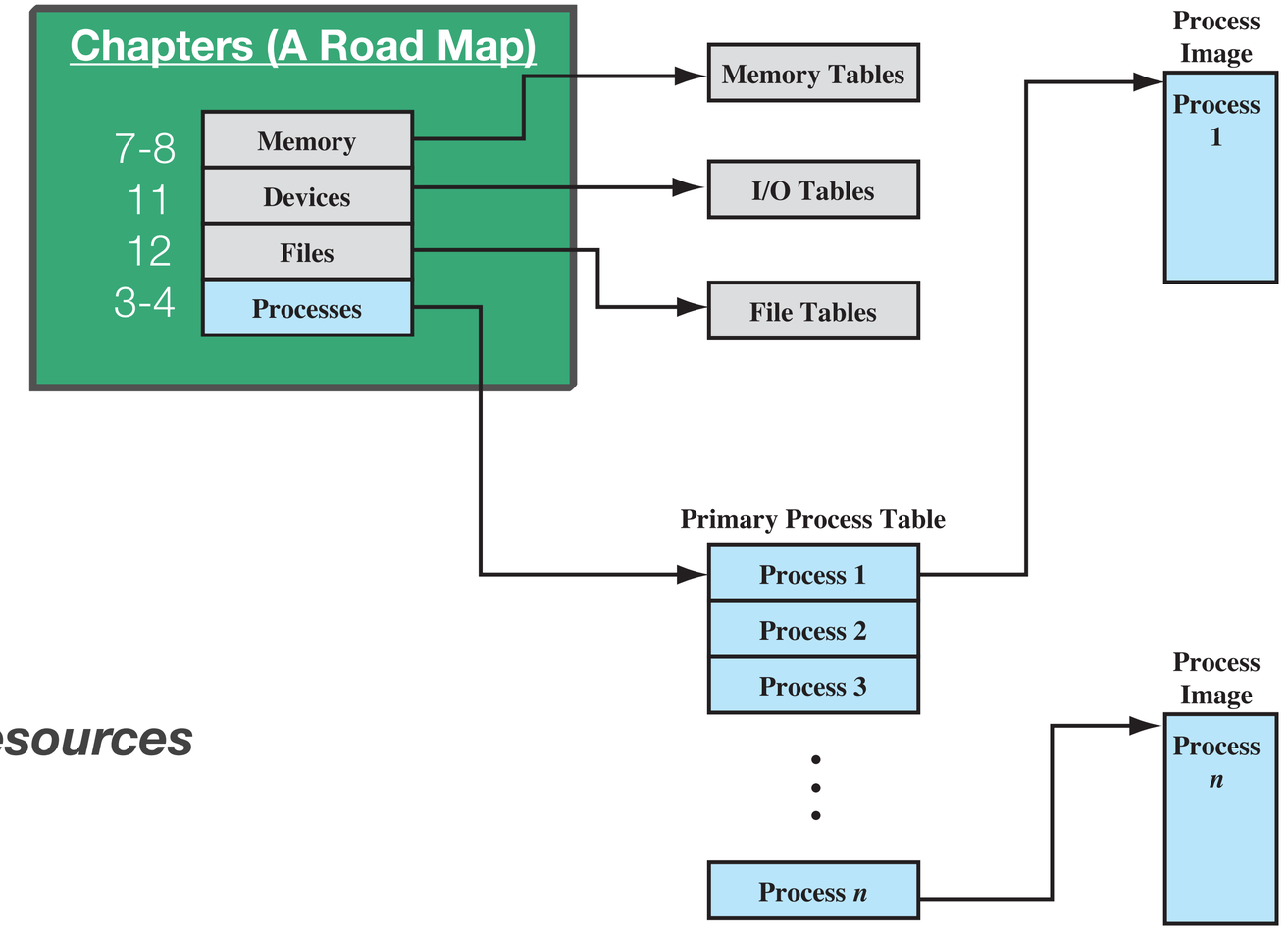
The OS as the Manager of Processes & Resources

- **Basically:** The OS manages the use of system resources by processes.
- **Q:** But ***what information*** does the OS need to control processes and manage system resources for them?



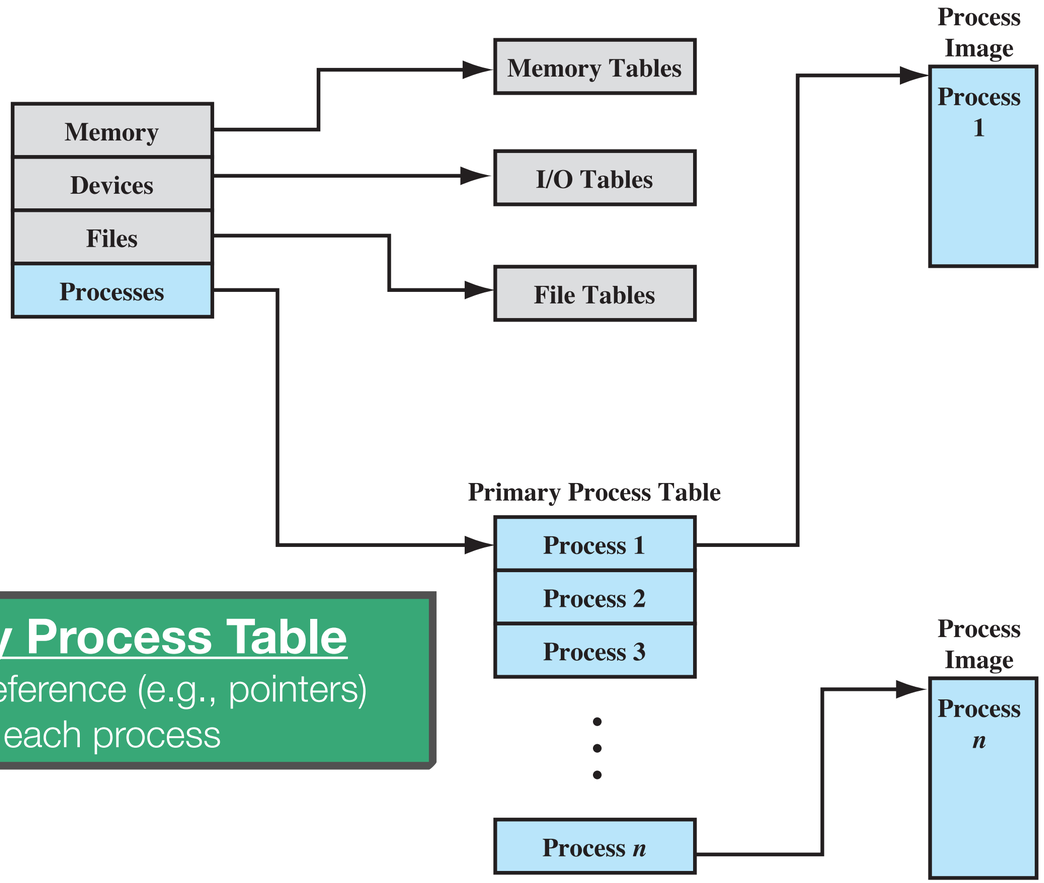
The OS as the Manager of Processes & Resources

- OS Control Structures *(tables…)*
- Specifics vary, but generally we have four types of tables ➡
- **NOTE: Tables are Interconnected**
  - The tables all cross-reference each other
    e.g., P1 has access to files in the file table
- **NOTE: Need Initial Configuration**
  - The OS must have access to relevant ***information about the system and its resources*** in order to create these tables.
    - *How much main memory exists?*
    - *What I/O devices exist? How to access them?*
    - *etc.*



The OS as the Manager of Processes & Resources

To manage and control a process the OS must know:



Primary Process Table
Maintains reference (e.g., pointers) to each process

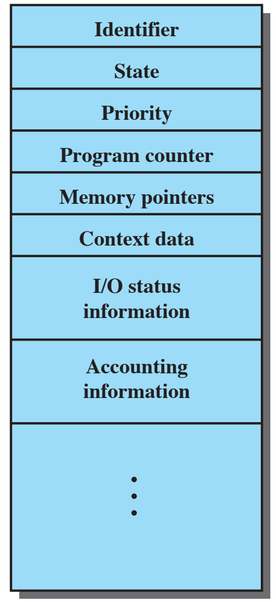
Process Image
Collection of code, data, stack, and attributes (PCB)

Process Attributes

1. Proc. Idenfication Info
PID, PPID, UID, etc.

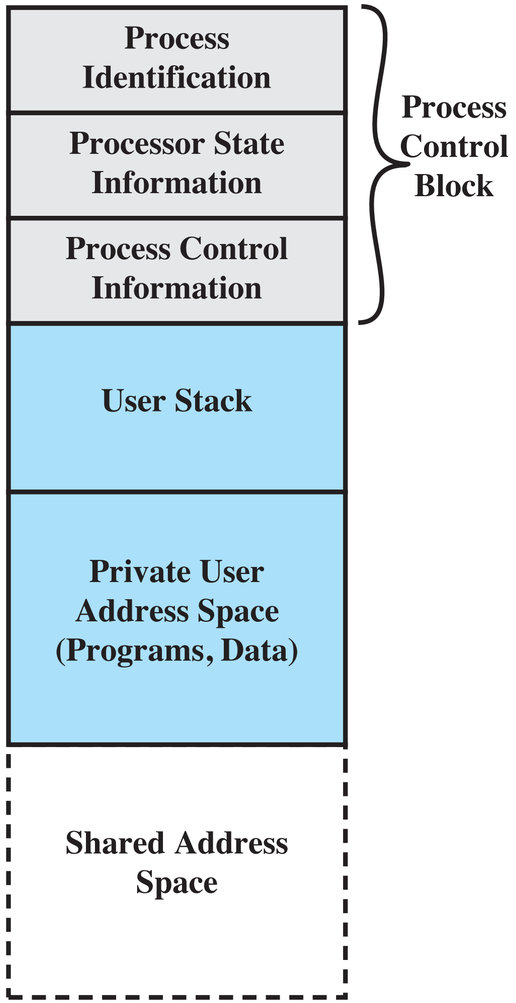
2. Processor State Info
Registers (User-visible, Status, Control), Stack Pointers, etc.
(maintains info after process is interrupted!)

3. Proc. Control Info
Scheduling & State Info, IPC, Privileges, Files, Page Tables (Memory), etc.



—W. Stallings (2018) Operating Systems: Internals and Design Principles (9th Edition).

Processes in Memory

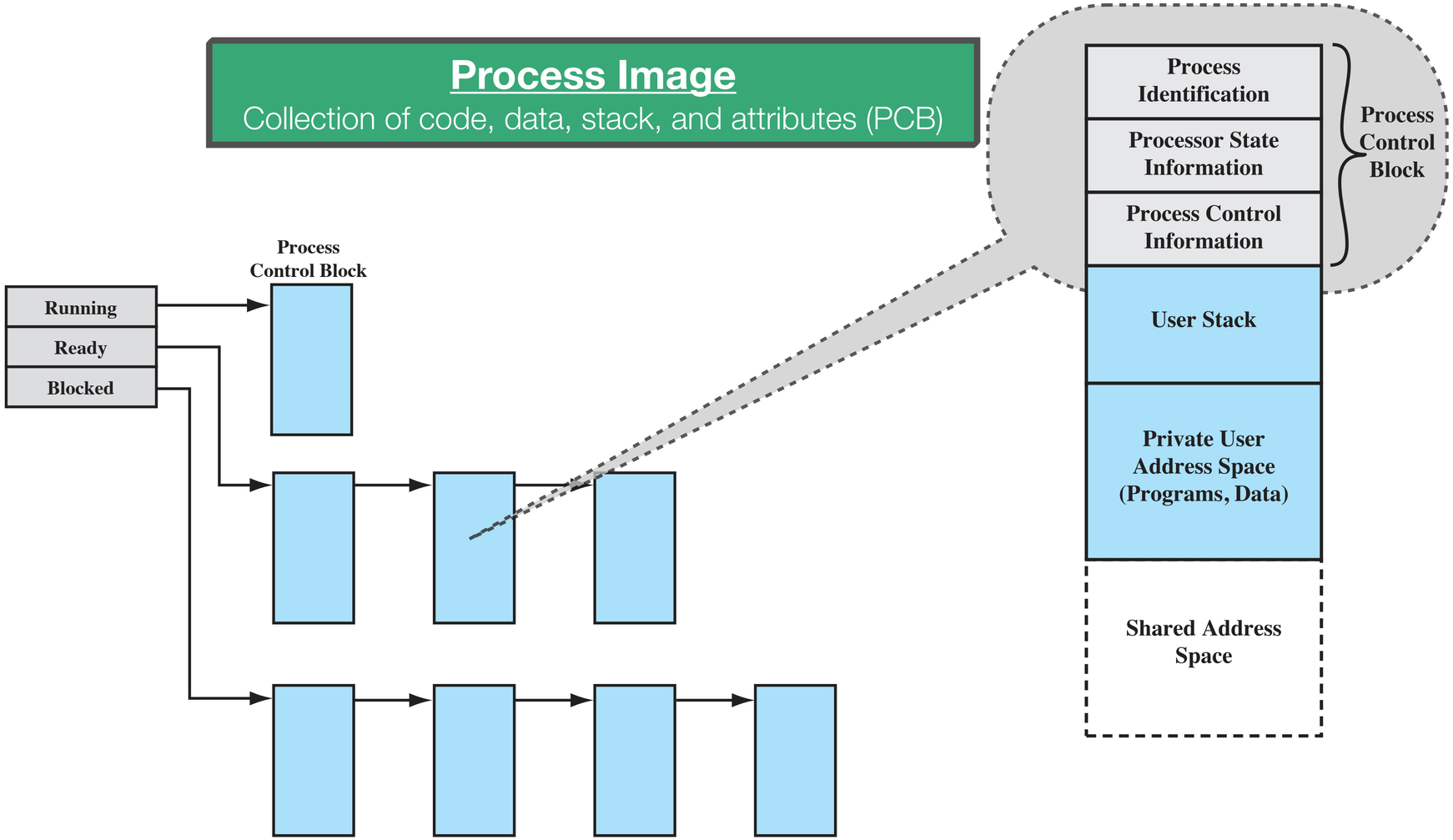


Process Image
Collection of code, data, stack, and attributes (PCB)

More specifically…

- memory
- open streams/files
- devices, including abstract ones like windows
- links to condition handlers (signals)
- processor registers (single thread)
- process identification
- process state - including waiting information
- priority
- owner
- which processor
- links to other processes (parent, children)
- process group
- resource limits/usage
- access rights
- process result - may be waited for by another process
- …

Processes in Memory & Process List Structures



Summary: The Role of the PCB

- PCB is the most important data structure in a modern OS
  - contains ALL of the information about a process that is needed by the OS
  - Blocks are read and/or modified by virtually *every* module in the OS
  - Defines the state of the OS
- **Q:** What could go wrong?!

  Difficulty is ***not access***, but ***protection***
  - A bug in a single OS routine could damage PCBs, which could destroy the system's ability to manage the affected processes
  - A design change in the structure or semantics of the PCB could affect many modules in the OS
  - Must tighly controll access to the PCB…
    ➡ *all PCB changes happen through a designated handler routine*

Privilege & Modes of Execution

- Processors support at least two modes of execution: ***privileged*** ***(kernel mode / system mode)*** VS. ***non-privileged*** ***(user mode)***
  - Sometimes referred to as ***rings*****.**
  - kernel mode = complete control over/access to processor, registers, memory
- **Q:** Why do we need different modes of execution?
- *Things* running in non-privileged mode should not…
  - be able to **access arbitrary memory** (e.g., other processes, I/O devices)
  - be able to **execute certain instructions** (e.g., manipulating control-flow registers)

Privilege & Modes of Execution

- **Q:** How does the processor know which mode to execute code in?

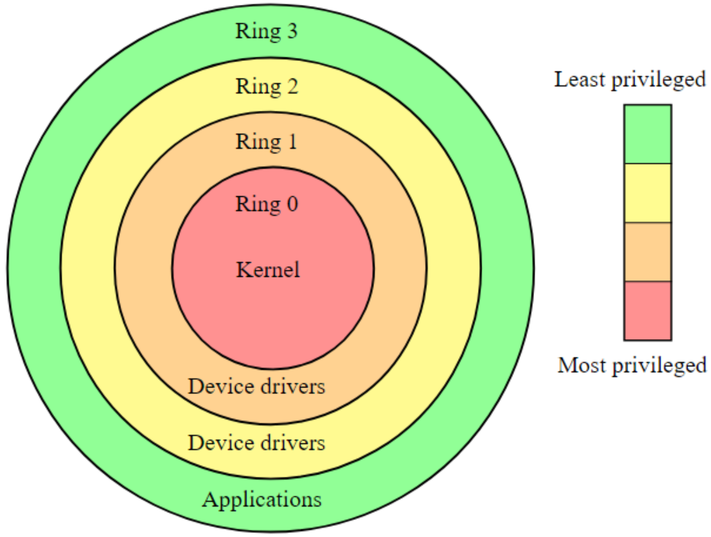
  Usually a single bit can be used → 0/1 == privileged/non-privileged

  Example:

  - *Intel uses 2-bit flag (CPL = current privilege level)*
    - *00 = most privileged = kernel*
    - *01 = ?*
    - *10 = ?*
    - *11 = least privileged = user mode*

- **Q:** How is the mode changed?

  Short Answer: Certain instructions clear/set bits



Process Switching

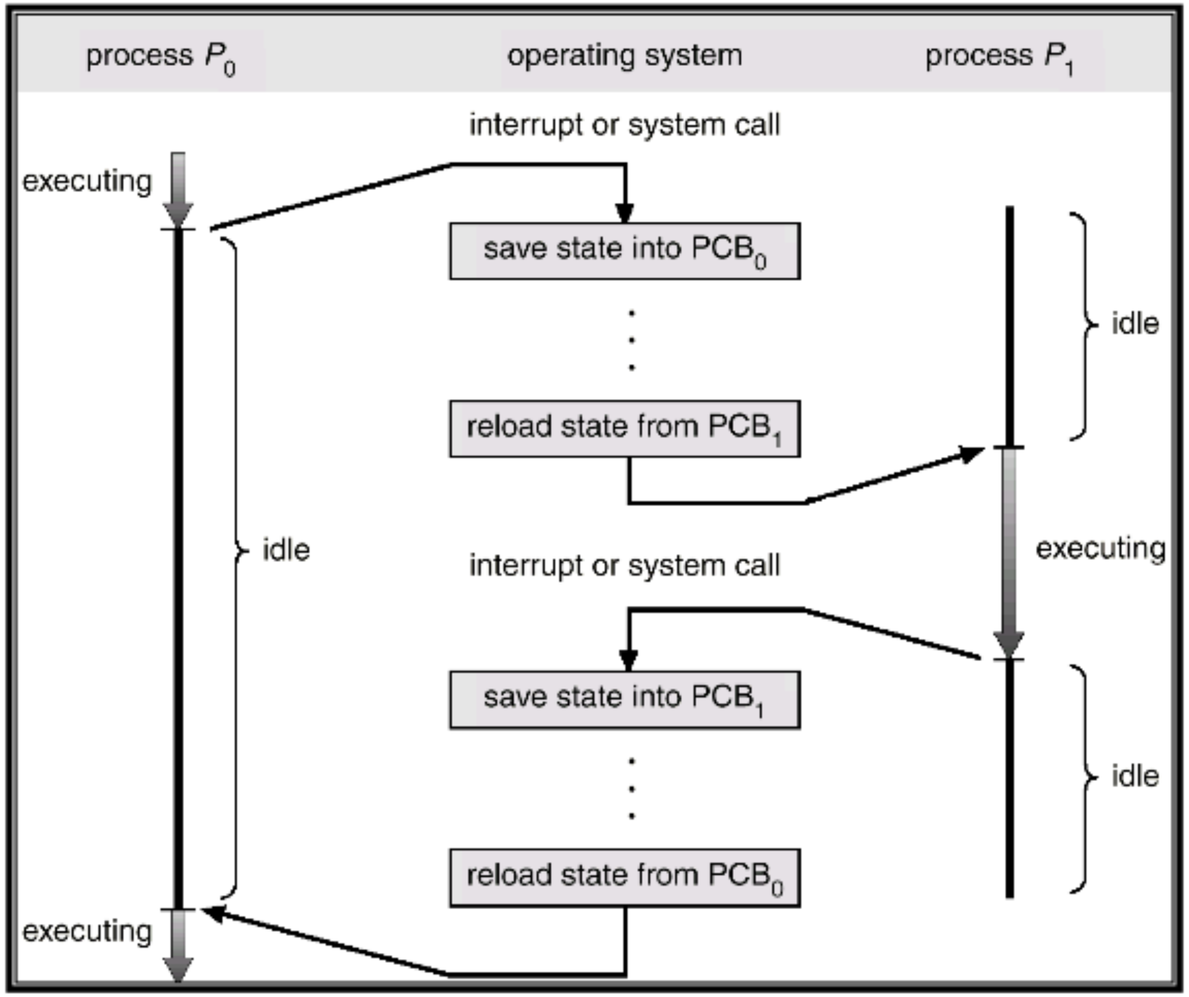
- What events **trigger a process switch**?

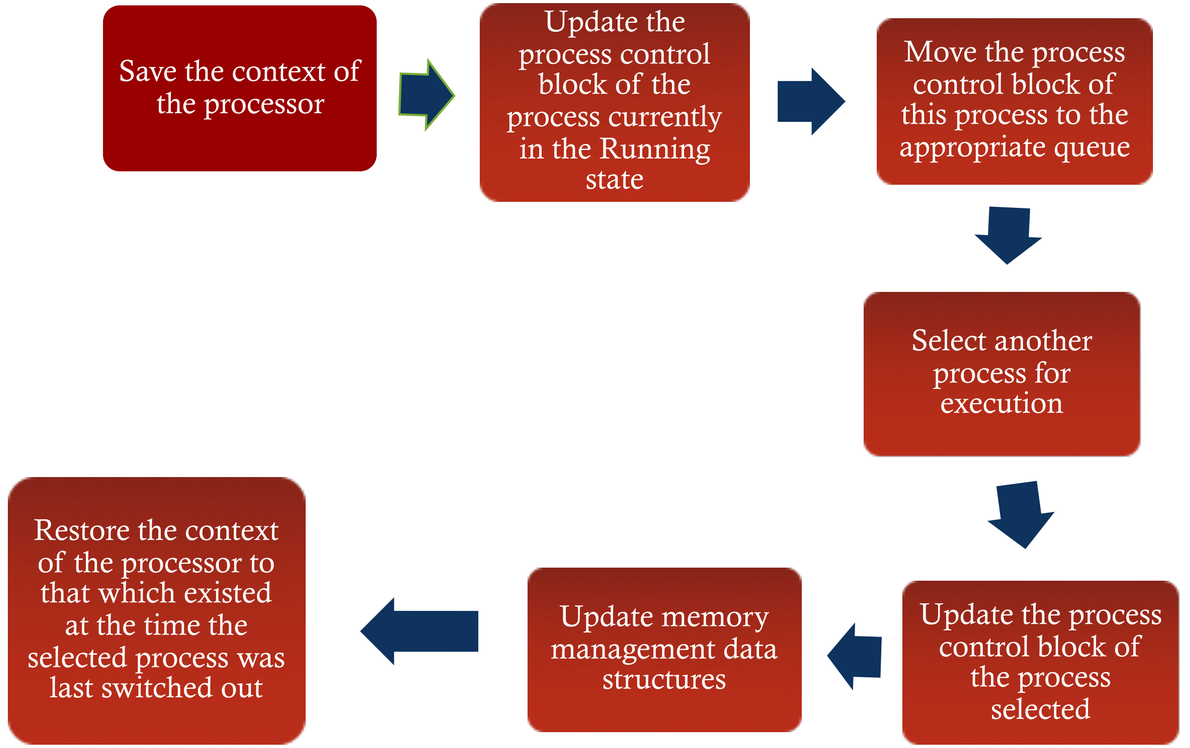
Mechanism	Cause	Use
Interrupt	**External to** the execution of the **current instruction**	Reaction to an asynchronous external event
Trap	**Associated with** the execution of the **current instruction**	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

- What is the difference between **mode switching** and **process switching**?
  - mode switch = change of privilege; process isn't necessarily changed
  - process switch = swap out all process-related info (e.g., registers, page tables)
  - *(Recall the fetch, execute, check interrupt cycles)*
  - *(Also, see next slide)*
- What is **the role of the OS**? (i.e., ***HOW*** is a process switched?)
  - saving context, updating PCB(s), updating resource tables, restoring context of next process, etc.

Process Switching

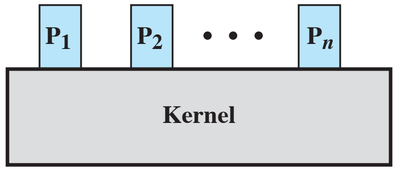
A high-level "sketch" of a full process switch a.k.a. ***"Context Switch"***



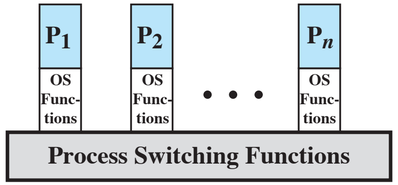


OS Execution

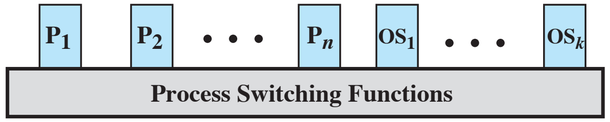
So… is the kernel executed as a process?



(a) Separate kernel



(b) OS functions execute within user processes



(c) OS functions execute as separate processes

Figure 3.15 Relationship Between Operating System and User Processes

OS Execution

So… is the kernel executed as a process? Sort of…

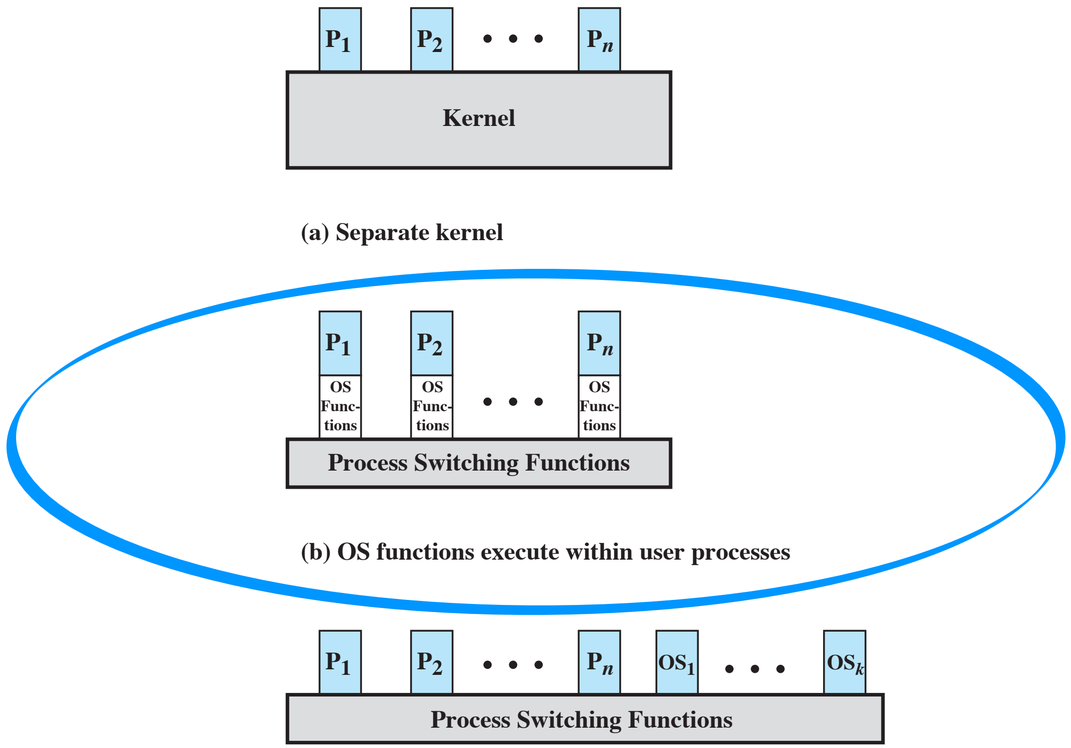


Figure 3.15 Relationship Between Operating System and User Processes

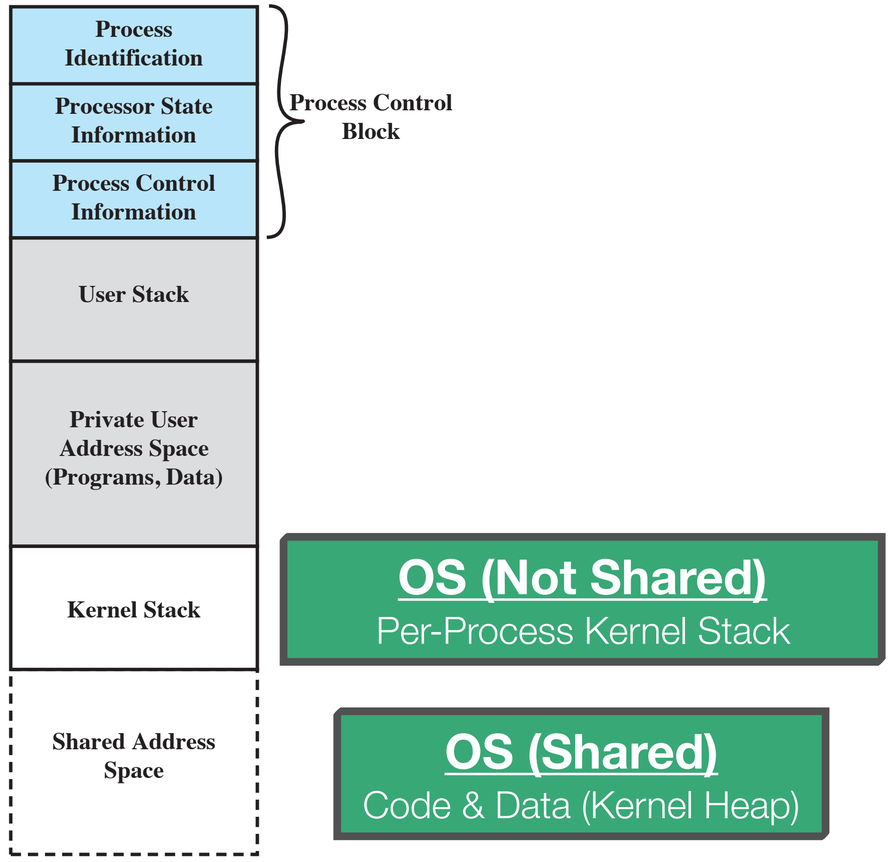
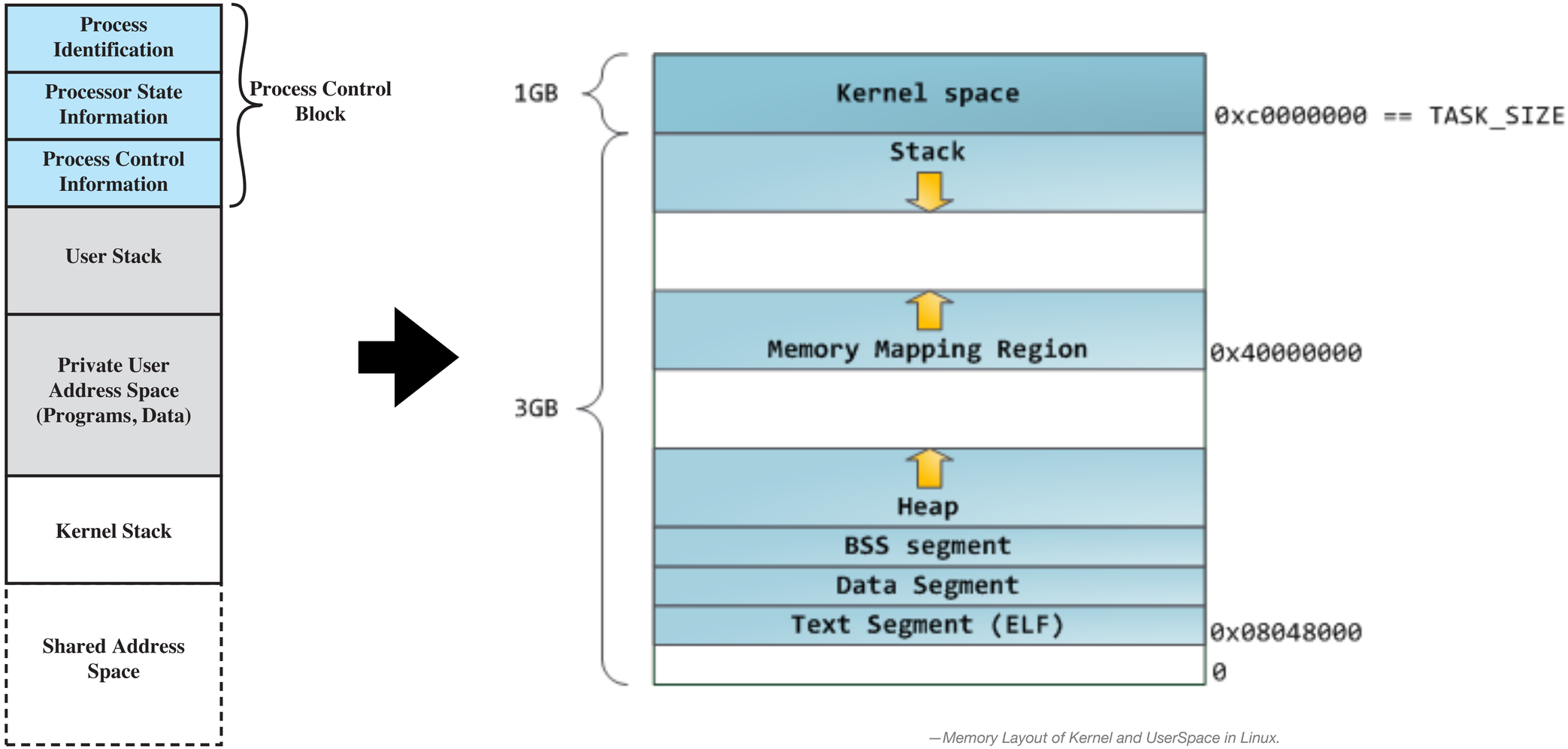


Figure 3.16 Process Image: Operating System Executes Within User Space

OS Execution

From abstract representation to modern OSs (e.g., Linux)



—Memory Layout of Kernel and UserSpace in Linux.
https://learnlinuxconcepts.blogspot.com/2014/03/memory-layout-of-userspace-c-program.html

UNIX Example @ End of Chapter (Process Management in a Real OS)

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.