Processes and Threads

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Understand Linux Processes

Understand Linux Threads

Introduce Interprocess Communication (IPC)

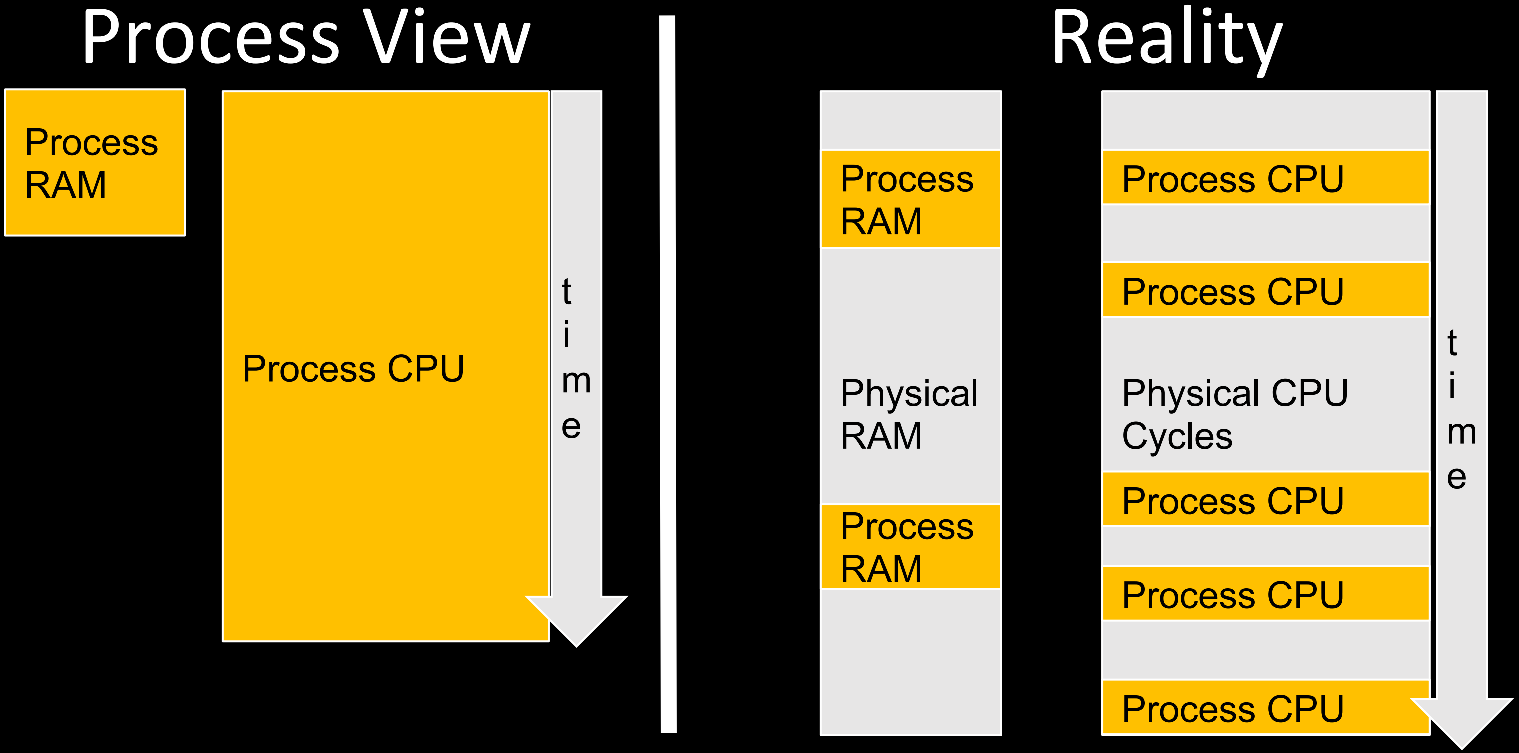
Linux Processes

- Executable object code, running on hardware.
  - ELF (Executable and Linkable Format).
- Necessary resources to run are allocated and managed by the kernel through system calls.
  - Timers
  - Files
  - Hardware access

Linux Processes

- Referenced by Process ID (pid).
- Run on virtualized processor and virtualized view of memory.
    - Appears to software as its own dedicated RAM/CPU.

Virtualized Processor/Memory



Linux Threads

- Unit of activity within a process
- A process may be single-threaded or multithreaded
- Each thread has
  - Stack - stores local variables
  - Processor state/current location
- Memory address space is shared between threads

Linux Threads/Processes and Memory

- Each Process has its own virtual memory.
- Each Thread shares process virtual memory.
- Sharing memory access between threads?
  - Access directly (use synchronization).
- Sharing memory access between processes?
  - Use Inter-Process Communication (IPC).

Linux Signals

- One-way asynchronous notifications sent from:
  - Kernel to process.
  - One process to another process (IPC).
  - A process to itself
- Processes setup signal handlers to control how to respond to a signal
  - Example Ctrl->C or SIGINT to stop a process.

Linux Signals

- Signal handlers must use signal safe functions which are safe to call asynchronously.
  - Use of global variables can introduce unsafe scenarios.
  - Process code can be interrupted at any time by signals.

Linux Interprocess Communication (IPC)

Allows process to exchange information without using a common global memory space.

- Pipes
- Semaphores
- Message Queues
- Shared Memory