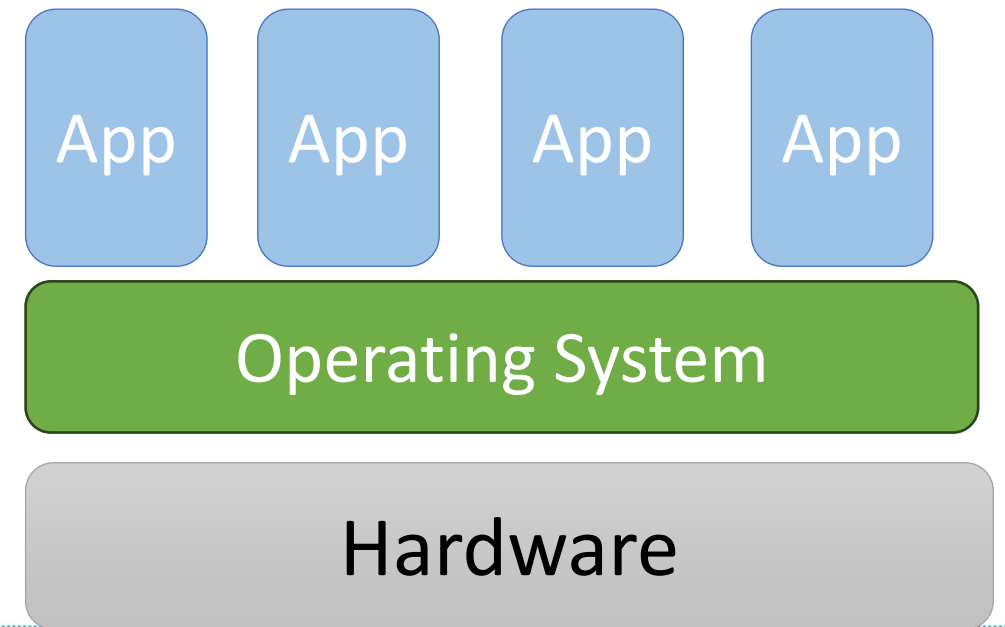


Quick refresher on Modern Operating Systems & CPUs

Introduction

What is an Operating System ?

- An OS is a program
- It sits between hardware and other programs
- It provides services to other programs
- It hides the hardware complexity
- It manages access to resources

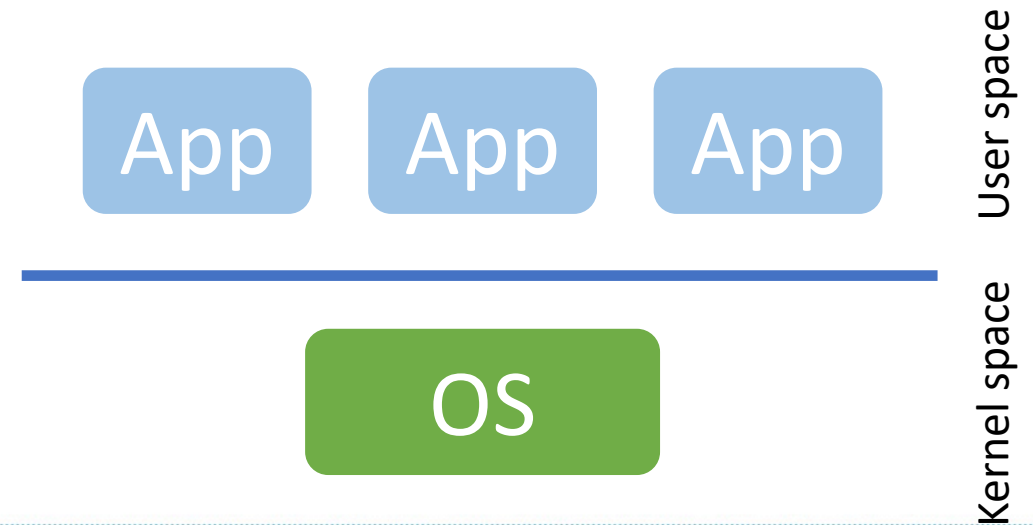


A special program

- The OS is the first program to be executed
 - It has a privileged access to the processor
- The OS can execute special instructions
 - If a standard app tries to call them, the CPU will generate errors
- The OS is not monolithic
 - Kernel
 - Drivers/Modules

Kernel space vs User space

- When the kernel executes, it's in a privileged mode
 - Called kernel space
- An application executes in a standard mode
 - User space



Access to resources

- A program cannot access resources directly
 - It must ask the OS to do it on its behalf (system call)
- Examples of resources access
 - Memory allocation
 - I/O on devices
 - Display
- For each request, the OS checks
 - The program is allowed (security)
 - The resource is free or can be shared
 - Access is fair

CPU-OS interaction

Process

- A process is a program being executed
 - It's a service offered by the OS
- A process has
 - Its instructions
 - Its data (variables, open files...)
 - A process-specific CPU state
- A process always need at least 2 resources
 - CPU and Memory
- Multiple processes have to be isolated
 - Memory protection

CPU Sharing

- Multitasking when multiple processes
 - Each process needs the CPU to be executed
 - But so does the OS?!
- Consider the CPU as a resource
 - Give each process some time on the CPU
 - Alternate processes rapidly, feeling of continuous execution
- Giving the CPU to another process is called **Context Switch**
 - It's a costly operation

Preemptive multitasking

- General idea
 - Allow a process to execute for at most x milliseconds on CPU
 - If process doesn't need CPU anymore before x , it gives it back
 - If not, process is kicked from the CPU and replaced by OS
- But how can this be done?
- It can't 😞
 - Unless we get help from the CPU

Interrupt

- An interrupt is a signal requiring an immediate action
- Software vs Hardware interrupt
- An interrupt is first processed by the CPU
 - It routes it to the correct interrupt handler (function)
- Who registers interrupt handlers to the CPU ?
 - The OS !
- Workflow
 1. An interrupt is raised
 2. CPU finds the matching interrupt handler
 3. Interrupt handler (OS) is given the CPU (executed)

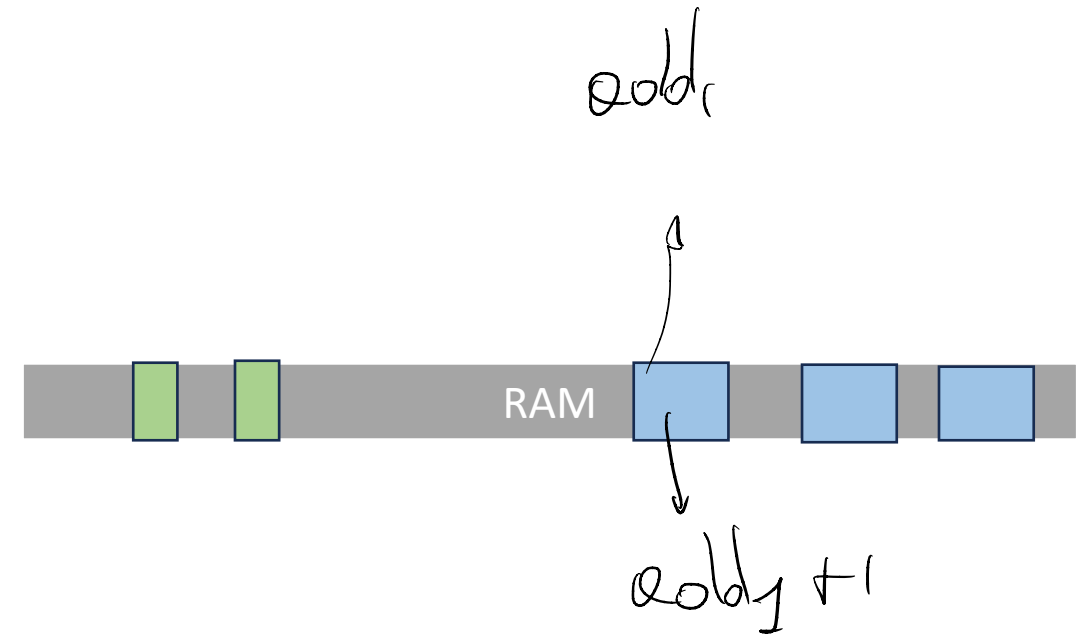
Preemptive multitasking – 2

- The OS registers an interrupt handler for clock interruptions
 - Every 15-50ms (depending on OS), a clock raises an interrupt
 - So even if a process keeps the CPU, the OS will get it back after 15-50ms
- How can a process not need the CPU ?
 - After all it needs it to execute....
 - Many reasons actually
 - Wait for an I/O
 - Wait for another process...

Memory management

Memory as a resource

- Memory is a resource
 - All processes and OS need some
- Memory is sensitive
 - A process should not be able to
 - read memory from another process
 - Read memory from the OS
- Memory is scarce
 - Limited amount so has to be shared fairly
- General idea
 - Allocate parts of memory to processes/OS
- 3 point of view
 - Processes, OS and CPU



Memory management

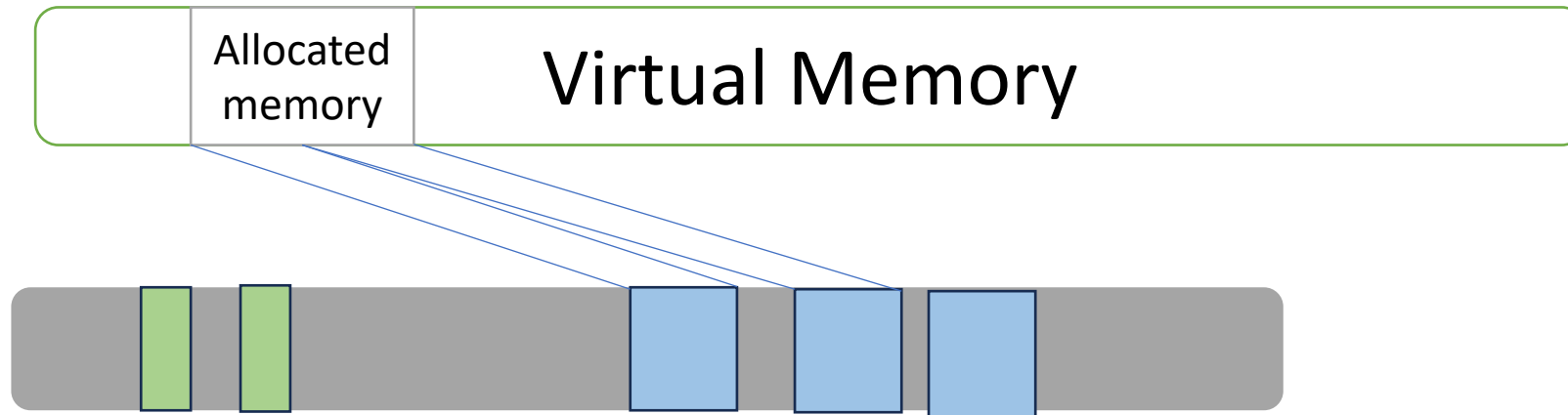
From the process's point of view

Process's point of view

- A process needs memory to execute
 - Needs may vary during its lifetime
- New memory needed
 - At creation
 - When executing (malloc())
 - When accessing files, communicating with other processes...
- Problem
 - Physical memory might be fragmented
 - Non contiguous memory addresses are hard for programmer !
- Solution
 - Use of virtual memory

Virtual Memory

- Each process has the illusion that it can address the whole memory
 - Logical address space
 - 4GB on 32bits machines, a lot on 64 bits machines



Memory management

From the CPU point of view

Managing physical memory

- The CPU manages access to physical memory
 - Different strategies depending on CPU family
- Memory Management Unit
 - Dedicated unit on the CPU to help with memory management
- ~~Pagination~~ *Paging*
 - Main strategy
 - Physical memory is divided in chunk called pages
 - 4KB or 4MB on x86, can be mixed!

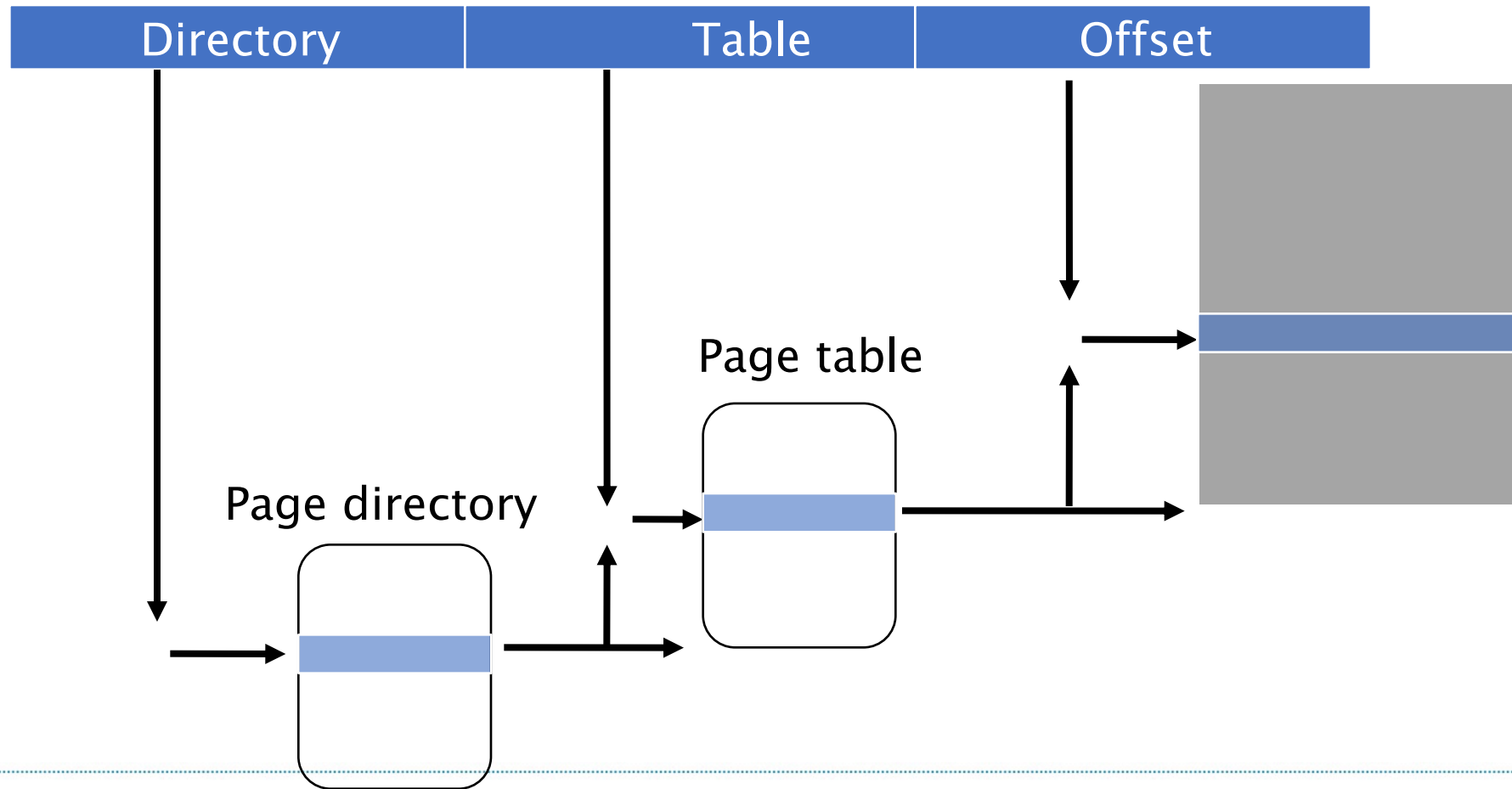
Paging

virtual memory

- General idea
 - Divide a linear address in parts : directory, table, offset
- Maintain 2 data structures (simplified version)
 - Page Directory
 - Page Table
- Conversion from linear to physical in 2 steps
 - Use directory part in Page Directory to get Page Table address
 - Use table part in Page Table to get physical address of page
 - Use offset to get data
- Easy 😊

Paging example

virtual



Paging - 2

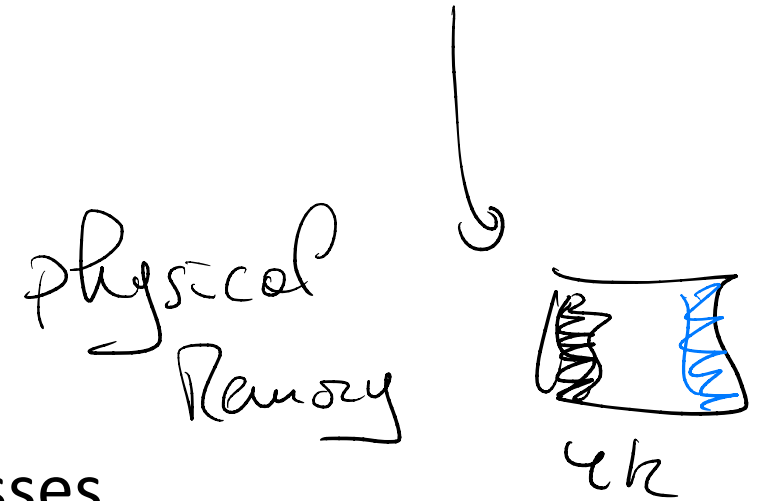
- Each memory access (process or kernel) has to go through these steps
 - Need to be implemented in hardware
 - But not fast enough
 - Use of a caching mechanism called Translation Lookaside Buffer (TLB)
- But this cannot work !!
 - What about 2 processes using the same virtual address (allowed)?
 - Surely they will end up on the same physical address, unless...
- Unless the Page Directory is process dependent !

Memory management

From the OS point of view

OS and memory

- The OS needs memory for itself
 - High priority, not really notion of fairness
- The OS allocates manages memory for processes
 - Keeps track of mapping virtual memory with physical memory with help of CPU
 - Decide whether page should be in memory or on disk (swap)
- The OS handles memory errors
 - CPU error if a process tries to access an unallocated memory page
 - OS will take action depending on situation
 - Kill the process (SegFault)
 - Allocate memory...



ISA

Instruction Set Architecture (ISA)

- An abstract model of a computer
 - Describes how the CPU is controlled by the software
- The ISA defines
 - Supported data types and Registers
 - How the hardware manages main memory (e.g. virtual memory)
 - Which instructions a microprocessor can execute
 - The input/output model
- ISA essentially describes what the computer can do and how it does it

Typical ISA : instructions + HW archi

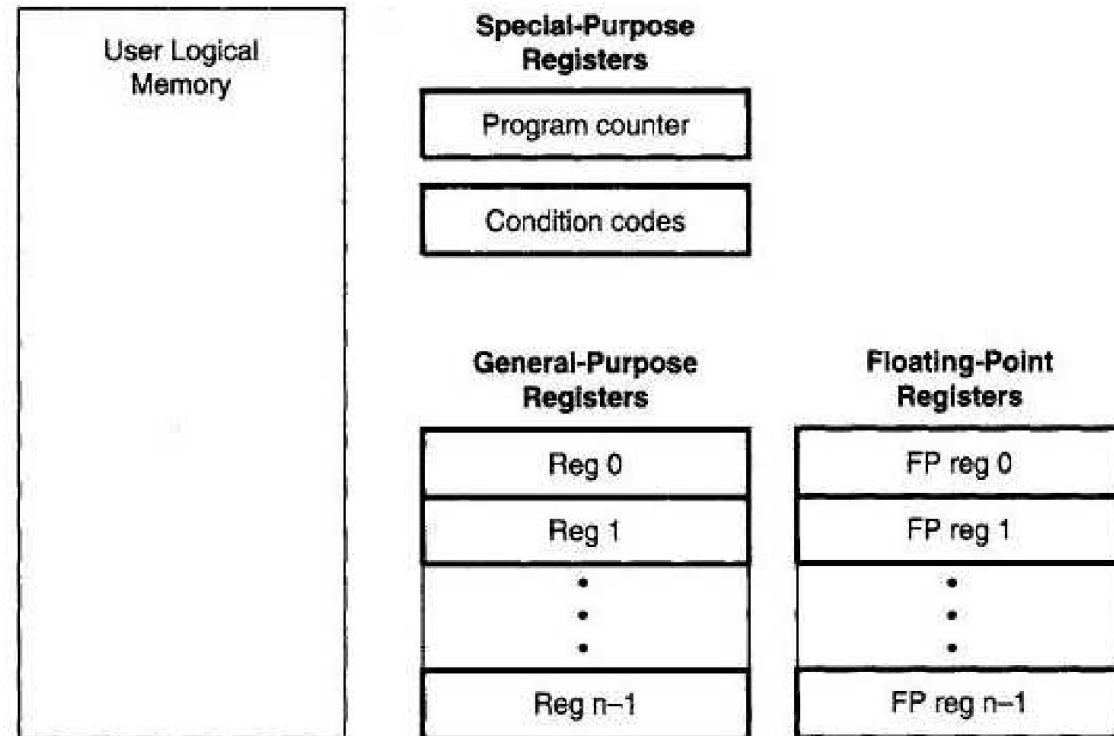


Figure A.6 Key Architected User State of a Typical ISA. This includes a logical (virtual) memory space, a special-purpose register, general-purpose registers, and floating-point registers.

Registers

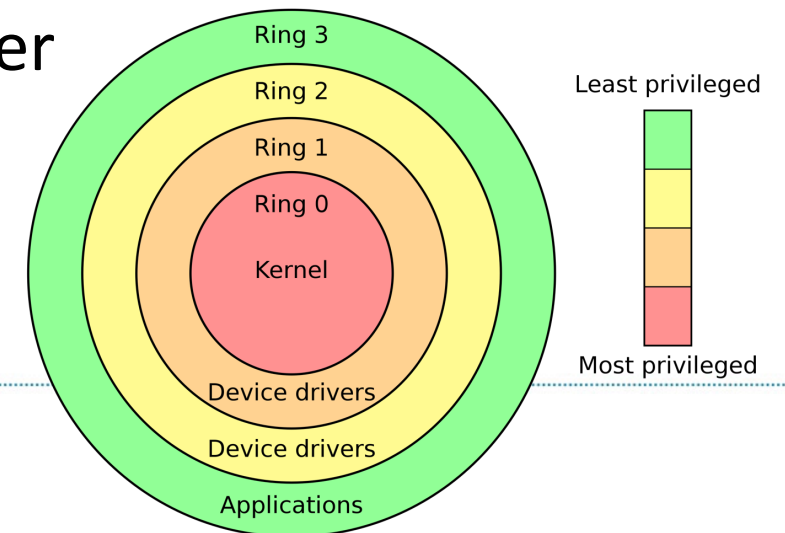
- Generic registers
 - Can hold different types of values
- Typed registers for specific operands
 - ISA dependent
 - Ex : pointer to a memory segment in Intel IA-32 ISA
- Specific registers
 - Program counters
 - Condition

User ISA

- Part of the ISA instructions for user applications
- 4 categories
 - Load/store memory \leftrightarrow registers
 - Integer/logical operations and bit shifting
 - Floating point operations
 - Branching and jumps

System ISA

- Processors features several executions modes
 - Numbered from 0 to 3 in general
 - Intel/AMD has 4, ARM v7 has 3, ARM v8 has 4
- Used to protect data and functionality from faults
 - Hierarchical, lower is higher privilege
- Current execution mode stored in specific register



System registers

- Time register
 - E.g. Time stamp counter on x86
- Traps and interruptions registers
- Traps and interruptions masking registers
- Page directory pointer
 - E.g. CR3 on Intel (Control Register 3 or PDBR for Page Directory Base Register)

Traps and interruptions

- Traps and interruptions
 - Transfer of control to the OS
- Interruption
 - Clock
 - Request from an I/O device
- Trap
 - Error during execution of an instruction (page fault, division by 0)
 - Explicit call from a program

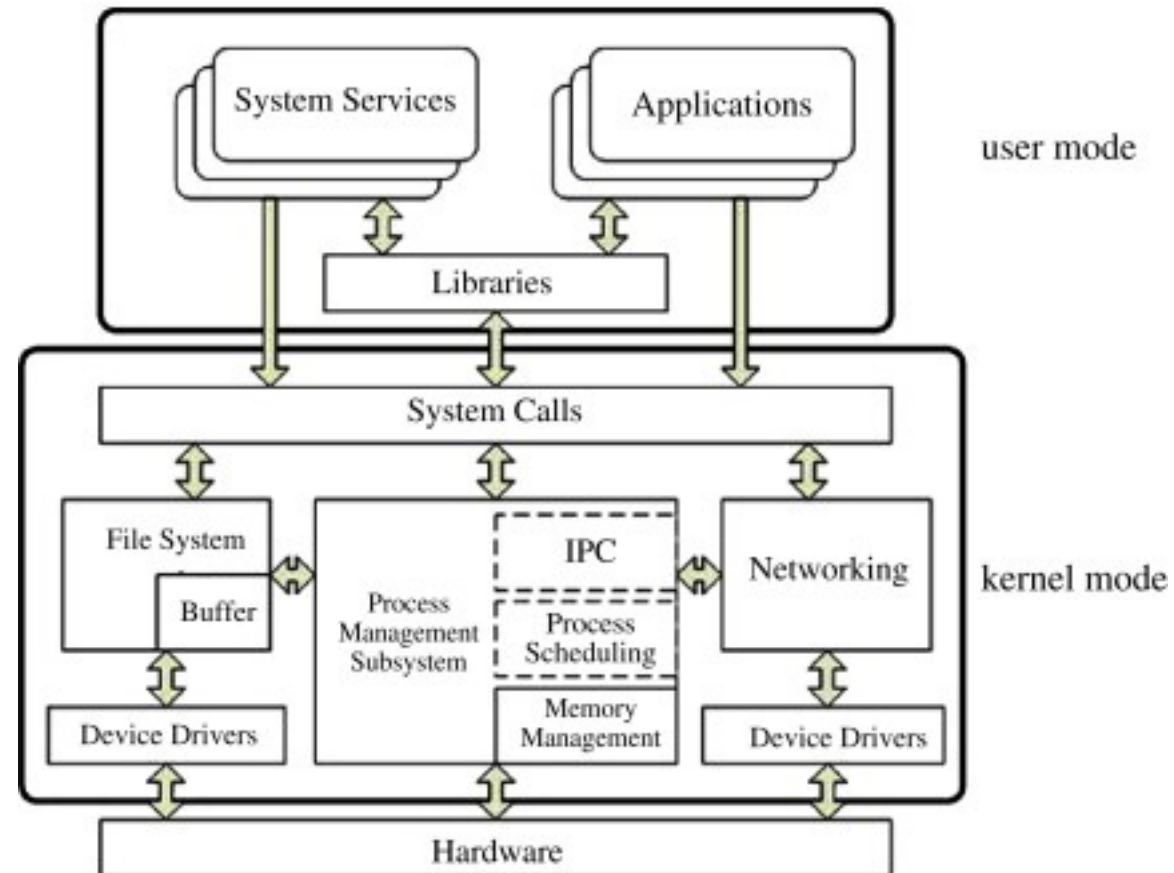
System ISA : processor management

- OS must be able to hand over control of the processor to processes
 - Jump to an instruction of the user program
 - Modification of CPU registers
- And regain control later
 - Use of interrupts

System ISA : I/O Management

- 3 models
- Memory Mapped I/O
 - Peripheral device's memory is mapped to CPU's memory
 - Transfer between memory and device uses memory transfer instructions
- Instruction based I/O
 - Special instruction to access I/O ports
- DMA I/O
 - Direct Memory Access
 - Peripheral can access main memory directly without CPU

Linux kernel schematic view



Conclusion

Conclusion

- The OS
 - Is a program
 - Provides services to other programs (processes)
- Some services require collaborating with the CPU
 - Multitasking
 - Memory management (safety)
- Different CPUs can have similar behaviour
 - Described in ISA
 - ISA specifies how the OS should interact