10. Kernel Design Overview

- Memory Management
- Interrupts and Exceptions
- Low-Level Input/Output
- Low-Level Synchronization
- Devices and Driver Model
- Process Management and Scheduling

10. Kernel Design Overview

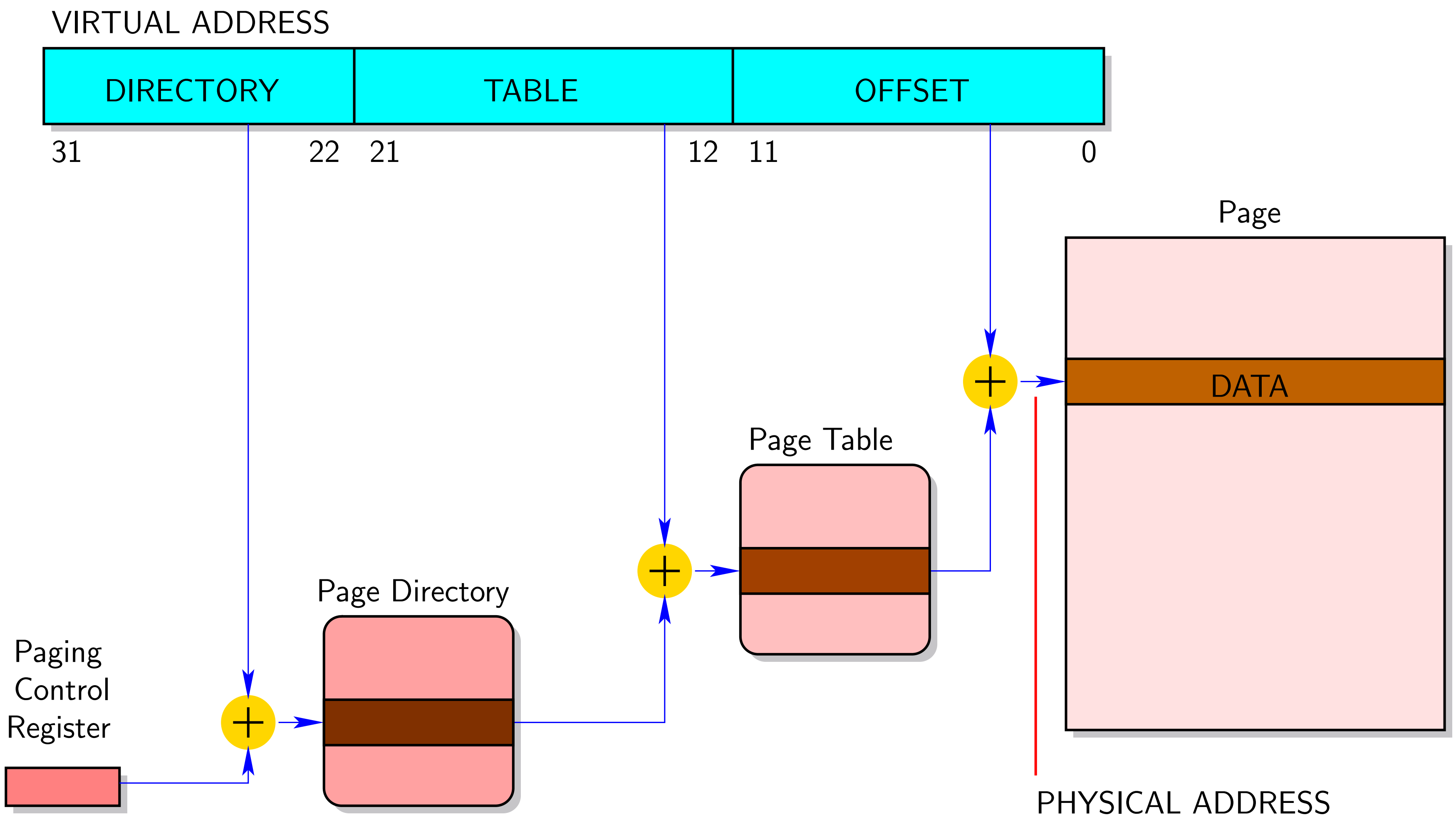
- Memory Management
- Interrupts and Exceptions
- Low-Level Input/Output
- Low-Level Synchronization
- Devices and Driver Model
- Process Management and Scheduling

Introduction to Memory Management

Paging Basics

- Processes access memory through *virtual* addresses
  - Simulates a large *interval* of memory addresses
  - Simplifies memory management
  - Automatic *translation* to *physical* addresses by the CPU
  - Circuits involved: Memory Management Unit (MMU) with Translation Lookaside Buffer (TLB)
- *Paging* mechanism
  - Provide a protection mechanism for memory regions, called *pages*
  - Fixed $\mathbf{2^n}$ page size(s), e.g., 4kB and 2MB on x86
  - The kernel implements a *mapping* of physical pages to virtual ones
    - *Different for every process*
- Key mechanism to ensure *logical separation* of processes
  - Hides kernel and other processes' memory
  - Expressive and efficient address-space protection and separation

Address Translation for Paged Memory



Hardware Support for Memory Management

Hardware Segmentation (Old-Fashioned)

- Hardware to separate types of memory (code, data, static, etc.)
- Supported by x86 but totally unused by Linux/UNIX, except in virtual execution environments

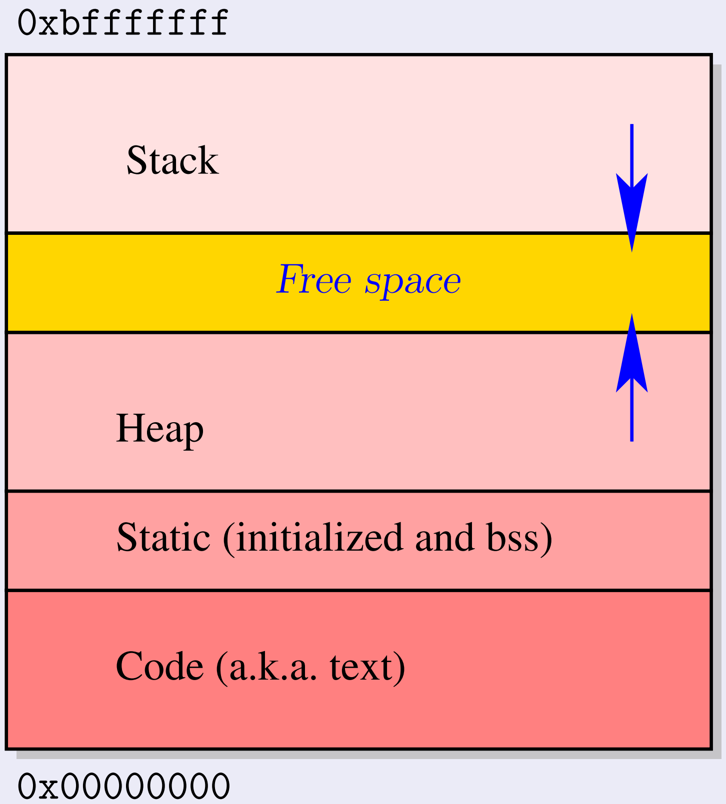
Paging

- Hardware memory protection and address translation (MMU)
- *At each context switch*, the kernel reconfigures the page table
  - Implementation: assignment to a control register at each context switch
  - Note: this flushes the TLB (cache for address translation), resulting in a severe performance hit in case of scattered physical memory pages
  - Kernel structures: *page table* in `/proc/<PID>/pagemap`
- Thread *affinity* and page *pinning* policy for NUMA cache-coherent architectures

Abstraction: Virtual Memory

Per-Process Virtual Memory Layout

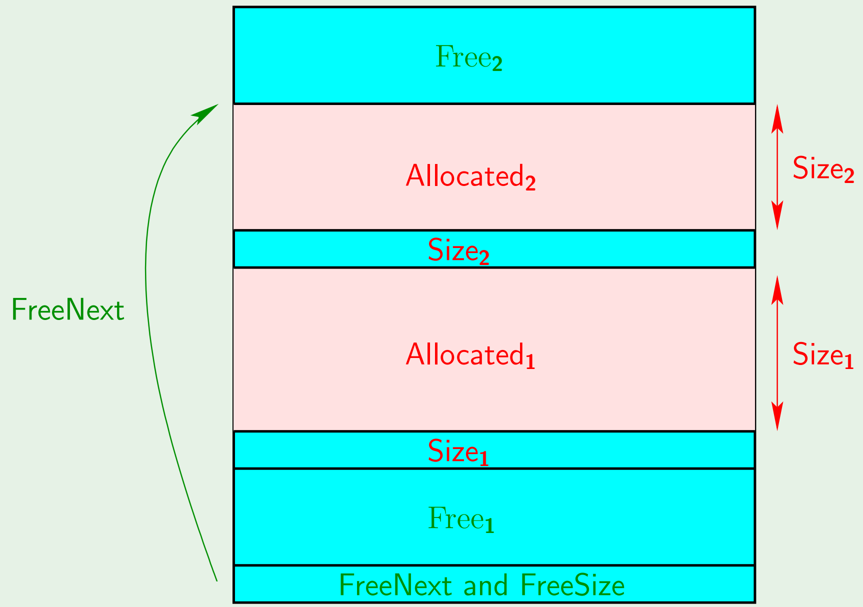
- *Code* (also called *text*) segment
  - Linux: ELF format for object files (`.o` and executable)
- *Static data* segment(s)
  - Global, `static` variables
- *Stack* segment
  - Stack frames for method arguments and local variables
- *Heap* segment
  - Dynamic allocation of objects: `new`
- Kernel structures: *segments* in `/proc/<PID>/map`



Heap Memory Management of User Processes

Memory Allocation: `malloc` and `free`

- Often the most complex part of a kernel
  - Appears in every aspect of the system
  - Major performance impact: highly optimized
- *Free list*: record linked list of free zones in the *free* memory space only
  - Record the address of the *next free zone*
  - Record the size of the allocated zone prior to its effective bottom address



10. Kernel Design Overview

- Memory Management
- Interrupts and Exceptions
- Low-Level Input/Output
- Low-Level Synchronization
- Devices and Driver Model
- Process Management and Scheduling

Hardware Support: Interrupts

- Typical case: electrical signal asserted by external device
  - Filtered or issued by the *chipset*
  - Lowest level hardware synchronization mechanism
- Multiple priority levels: Interrupt ReQuests (IRQ)
- Processor switches to kernel mode and calls specific *interrupt service routine*
- Multiple drivers may share a single IRQ line
  $\rightarrow$ IRQ handler must identify the source of the interrupt to call the proper service routine

Hardware Support: Exceptions

- Typical case: unexpected program behavior
  - Filtered or issued by the *chipset*
  - Lowest level of OS/application interaction
- Processor switches to kernel mode and calls specific *exception service routine*
- Mechanism to implement *system calls*

10. Kernel Design Overview

- Memory Management
- Interrupts and Exceptions
- Low-Level Input/Output
- Low-Level Synchronization
- Devices and Driver Model
- Process Management and Scheduling

Hardware Support: Memory-Mapped I/O

External Remapping of Memory Addresses

- Builds on the chipset rather than on the MMU
  - Address translation + redirection to device memory or registers
- Unified mechanism to
  - Transfer data: just load/store values from/to a memory location
  - Operate the device: reading/writing through specific memory addresses actually sends a command to a device
    E.g., *strobe* registers: writing anything triggers an action
- Supports *Direct Memory Access* (DMA) block transfers
  - Operated by the DMA controller, not the processor
  - Choose between *coherent* (a.k.a. synchronous) or *streaming* (a.k.a. non-coherent or asynchronous) DMA mapping

10. Kernel Design Overview

- Memory Management
- Interrupts and Exceptions
- Low-Level Input/Output
- Low-Level Synchronization
- Devices and Driver Model
- Process Management and Scheduling

Hardware Interface: Kernel Locking Mechanisms

Spin-Lock

- Busy waiting

```
while (true) {
  while (lock == 1) { pause_for_a_few_cycles(); }
  synchronize {
    if (lock == 0) { lock = 1; break; } // Atomic execution
  }
}
// Critical section: atomic sequence of instructions
// accessing a shared resource
lock = 0;
```

Applications

- Wait for short periods, typically less than **1 μs**
  - Busy-waiting for short period, then continue into a *passive*, *interrupt*-based lock if necessary
  - Only way to implement mutual exclusion in interrupts

Hardware Interface: Kernel Locking Mechanisms

Low-Level Mutual Exclusion Variants

- Very short atomic sequences of instructions
  - Spin-lock: active loop polling a memory location
- Other fine-grain locks and asynchronous notification mechanisms
- Brute-force methods
  - Disable preemption and interrupts: can be used for very short periods
  - The "big kernel lock"
    - Non scalable on parallel architectures
    - Only for very short periods of time
    - Now mostly in legacy drivers and in the virtual file system

10. Kernel Design Overview

- Memory Management
- Interrupts and Exceptions
- Low-Level Input/Output
- Low-Level Synchronization
- Devices and Driver Model
- Process Management and Scheduling

Hardware Interface: Driver Model in Linux

Device Special Files

- *Block*-oriented device
  Disks, file systems: `/dev/hda` `/dev/sdb2` `/dev/md1`
- *Character*-oriented device
  Serial ports, console terminals, audio: `/dev/tty0` `/dev/pts/0` `/dev/usb/hiddev0` `/dev/mixer` `/dev/null`
- *Major* and *minor* numbers to (logically) project device drivers to device special files

Hardware Interface: Driver Model in Linux

Low-Level Device Driver

- Automatic configuration: "plug'n'play"
  - Memory mapping regions, access rights, device capabilities
  - Configuration of interrupts (IRQ)
- Automatic configuration of device mappings
  - *Device numbers: kernel anchor for driver interaction*
  - Automatic assignment of *major* and *minor* numbers
    - At *discovery-time*: when a driver recognizes the signature of a device (e.g., PCI number)
    - At boot-time or plug-time
  - Hot pluggable devices

Hardware Interface: Device Drivers

Overview

- Abstracted by system calls or kernel processes
  - Generic I/O system calls (`open()`, `read()`, `write()`, etc.)
  - Device-specific `ioctl()`: `$ man ioctl_list`
- Manage buffering between device and local buffer
- Control devices through memory-mapped I/O
- Devices trigger interrupts (end of request, buffer full, etc.)
- Many concurrency challenges (precise synchronization required)
- Multiple layers for portability and reactivity

Hardware Interface: Driver Model in Linux

Low-Level Statistics and Management

- Generic device abstraction: `proc` and `sysfs` pseudo file systems
  - Class (`/sys/class`)
  - Module (parameters, symbols, etc.)
  - Resource management (memory mapping, interrupts, etc.)
  - Bus interface (PCI: $ `lspci`)
  - Power management (sleep modes, battery status, etc.)

Example: Block Device

```
$ cat /sys/class/scsi_device/0:0:0:0/device/block:sda/dev
8:0

$ cat /sys/class/scsi_device/0:0:0:0/device/block:sda/sda3/dev
8:3
```

Hardware Interface: Driver Model in Linux

High-Level Device Driver Interface

- Automatic device node creation (udev)
  - *Device name: application anchor to interact with the driver*
  - User level
  - Reconfigurable rules
  - Hot pluggable devices

Example: Block Device

```
$  cat /sys/class/scsi_device/0:0:0:0/device/uevent
DEVTYPE=scsi_device
DRIVER=sd
PHYSDEVBUS=scsi
PHYSDEVDRIVER=sd
MODALIAS=scsi:t-0x00
```

Hardware Interface: Driver Model in Linux

High-Level Device Driver Interface

- Automatic device node creation (`udev`)
  - *Device name: application anchor to interact with the driver*
  - User level
  - Reconfigurable rules
  - Hot pluggable devices

Example: Network Interface

```
$  cat /sys/class/net/eth0/uevent
PHYSDEVPATH=/devices/pci0000:00/0000:00:1c.2/0000:09:00.0
PHYSDEVBUS=pci
PHYSDEVDRIVER=tg3
INTERFACE=eth0
IFINDEX=2
```

10. Kernel Design Overview

- Memory Management
- Interrupts and Exceptions
- Low-Level Input/Output
- Low-Level Synchronization
- Devices and Driver Model
- Process Management and Scheduling

Process Scheduling

Distribute Computations Among Running Processes

- Infamous optimization problem
- Many heuristics... and objective functions
  - Throughput?
  - Reactivity?
  - Deadline satisfaction?
- General (failure to) answer: *time quantum* and *priority*
  - Complex dynamic adaptation heuristic for those parameters
  - `nice()` system call
  - $ `nice` and $ `renice` commands

Process Scheduling

Scheduling Algorithm

- Process-dependent semantics
  - *Best-effort* processes
  - *Real-time* processes

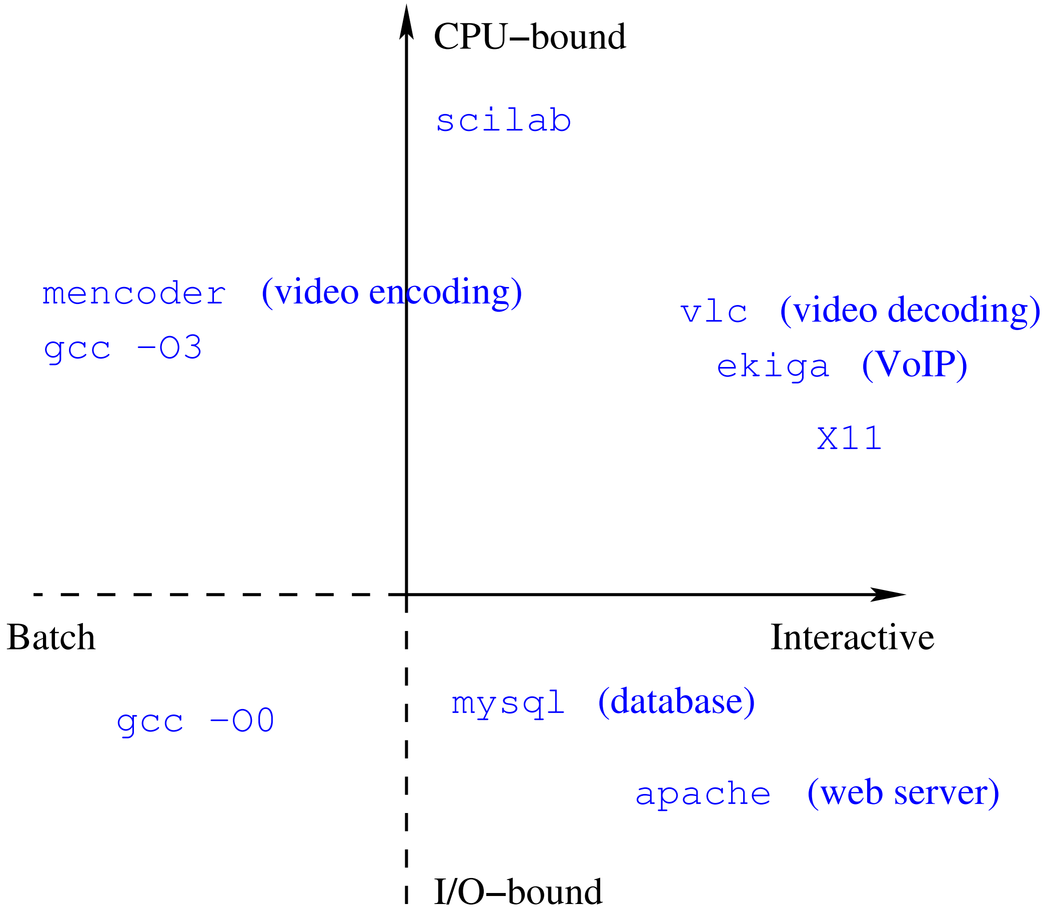
Scheduling Heuristic

- Multiple *scheduling queues*
  - Semantics: split processes according to scheduling algorithm (e.g., preemptive or not)
  - Performance: avoid high-complexity operations on priority queues (minimize context-switch overhead)
- Scheduling *policy*: prediction and adaptation

Scheduling Policy

Classification of Best-Effort Processes

- Two independent features
  - I/O behavior
  - Interactivity



Scheduling Policy

Real-Time Processes

- Challenges
  - Reactivity and low response-time variance
  - Coexistence with normal, time-sharing processes
- `sched_yield()` system call to relinquish the processor voluntarily without entering a suspended state
- Policies: *FIFO* or *Round-Robin (RR)*