10. Overview of Key Kernel Components

Outline

- Low-level mechanisms
  - Interrupts and exceptions
  - Memory-mapped I/O
  - Kernel locks, global and fine grain mechanisms
- File systems, I/O and devices
  - Devices and drivers
  - The virtual file system
  - Disk operation
- Memory management
  - Segmentation, paging, address translation
  - Memory allocation
- Processes
  - Hardware context (registers) and support (interrupts)
  - Lightweight processes (generic support for threads and processes)
  - Scheduling

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 a specific *interrupt service routine*
- Multiple drivers may share a single IRQ line
  $\longrightarrow$ IRQ handler must identify the source of the interrupt to call the proper service routine

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 a specific *exception service routine*
- Typical mechanism to implement *system calls*

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
    Typical example: *strobe* registers (writing anything triggers an event)
- Supports Direct Memory Access (DMA) block transfers
  Operated by the DMA controller, not the processor

Old-Fashioned Alternative: I/O Ports

- Old interface for x86 and IBM PC architecture
- Rarely supported by modern processor instruction sets
- Low-performance (ordered memory accesses, no DMA)

Kernel Locking Mechanisms

Low-Level Mutual Exclusion Variants

- Very short critical sections
  - Spin-lock
- Fine grain
  - Read/write lock: traditional read/write semaphore
  - Seqlock: speculative readers
  - Read-copy-update lock: concurrent writers in special cases
- Coarse grain
  - Disable preemption
  - Disable interrupts
  - 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

Kernel Locking Mechanisms

Spin Lock

- Busy waiting

```
Acq:     while (lock == 1) { pause_for_a_few_cycles; }
ATOMIC  if (lock == 0) lock = 1;
         else goto Acq;
         // Critical section
         // ...
         lock = 0;
         // Non-critical section
```

- Wait for short periods, typically less than **1 μs**
  - As a proxy for other locks
  - As a *polling* mechanism
  - Mutual exclusion in interrupts

I/O Implementation in Linux

Abstraction Levels: Low Level

- Automatic configuration: plug'n'play
  - Memory mapping
  - Interrupts (IRQ)
- Generic device abstraction (`sysfs`)
  - Class
  - Power management
  - Resources: memory mapping, interrupts
  - ...
- Automatic configuration of device mappings
  - *Device numbers: kernel anchor for driver interaction*
  - Kernel level
  - Automatic assignment of major and minor numbers
  - Hot pluggable devices

I/O Implementation in Linux

Abstraction Levels: OS Interface

- Automatic device node creation (`udev`)
  - *Device name: application anchor to interact with the driver*
  - User level
  - Reconfigurable rules
  - Hot pluggable devices
- File system mounting and virtual file system (`mount`)
  - Software layer below POSIX I/O system calls
  - Superset API for the features found in UNIX file systems
  - Also supports pseudo file systems (`/proc`, `/sys`, `/dev`, `/dev/shm`...)
  - Also supports foreign and legacy file systems (FAT, NTFS, ISO9660)

I/O Concurrenty Challenges

Typical Kernel Control Path

1. Page fault of user application
2. Exception, switch to kernel mode
3. Lookup for cause of exception, detect access to swapped memory
4. Look for name of swap device (multiple swap devices possible)
5. Call non-blocking kernel I/O operation
6. Retrieve device major and minor numbers (no VFS in this special case)
7. Forward call to the driver
8. Retrieve page (possibly swapping another out)
9. Update the kernel and process's page table
10. Switch back to user mode and proceed

Executing concurrently with...

- Other processes
- Other kernel control paths (interrupts, parallel or preemptive kernel)
- Deferred interrupts (softirq/tasklet mechanism)
- Real-time deadlines: timers, buffer overflows (e.g., CDROM)

Disk Operation

Disk Structure

- Plates, tracks, cylinders, sectors
- Multiple R/W heads
- Quantitative analysis
  - Moderate peak bandwidth in continuous data transfers
    E.g., up to 160MB/s on a modern SATA, 320GB/s on a modern SCSI
    Plus a read (and possibly write) cache in DRAM memory
  - Very high latency when moving to another track/cylinder
    Typically a few milliseconds on average, slightly faster on SCSI

Request Handling Algorithms

- Idea: queue pending requests and select them in a way that minimizes *head movement* and *idle plate rotation*
- Multiple variants of the "elevator" algorithm
- Heuristics dependent on block size, disk type (number of heads)
- Strong influence on process scheduling: avoid *disk thrashing*

Memory Management

Segmentation

Old-fashioned hardware support to separate code from data, kernel from user memory, etc.

... Supported by x86 but totally unused by Linux/UNIX

Paging

Hardware memory protection and address translation by the MMU

Implementation

- Associated with specific processor control registers
- The kernel reconfigures the page table at each context switch by assigning to a control register

  Note: this effectively flushes the TLB (cache for address translations), resulting in a severe performance hit in case the physical memory pages are scattered around

Memory Management

Memory Allocation

- Often the most complex part of a kernel
  - Appears in every aspect of the system
  - Major performance impact → highly optimized
- *Buddy system* to allocate contiguous pages of physical memory
  - Coupled with free list for intra-page allocation
  - Contiguous physical pages improve performance
  - But not required except for kernel memory
- *Slab allocator* (first implemented in Sun Solaris)
  - Cache of special purpose, fixed size, pool of memory regions
  - Learn from previous allocations/deallocations
  - Anticipate future requests
  - Well suited for short-lived memory needs
    E.g., `fork(); exec();` or kernel internal buffer management

Low-Level Process Implementation

Hardware Context

- Saved and restored by the kernel upon context switches
- Mapped to some hardware thread when running (with affinity policies when caches are distributed among cores/processors)

Lightweight Processes

- `clone()` system call
- Supports both threads and processes, selecting which attributes are shared/separate