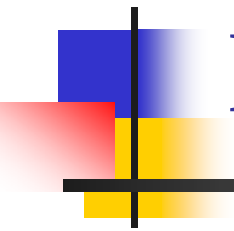
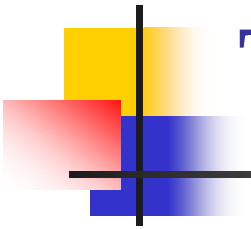


Introduction to the Linux Kernel

Hao-Ran Liu

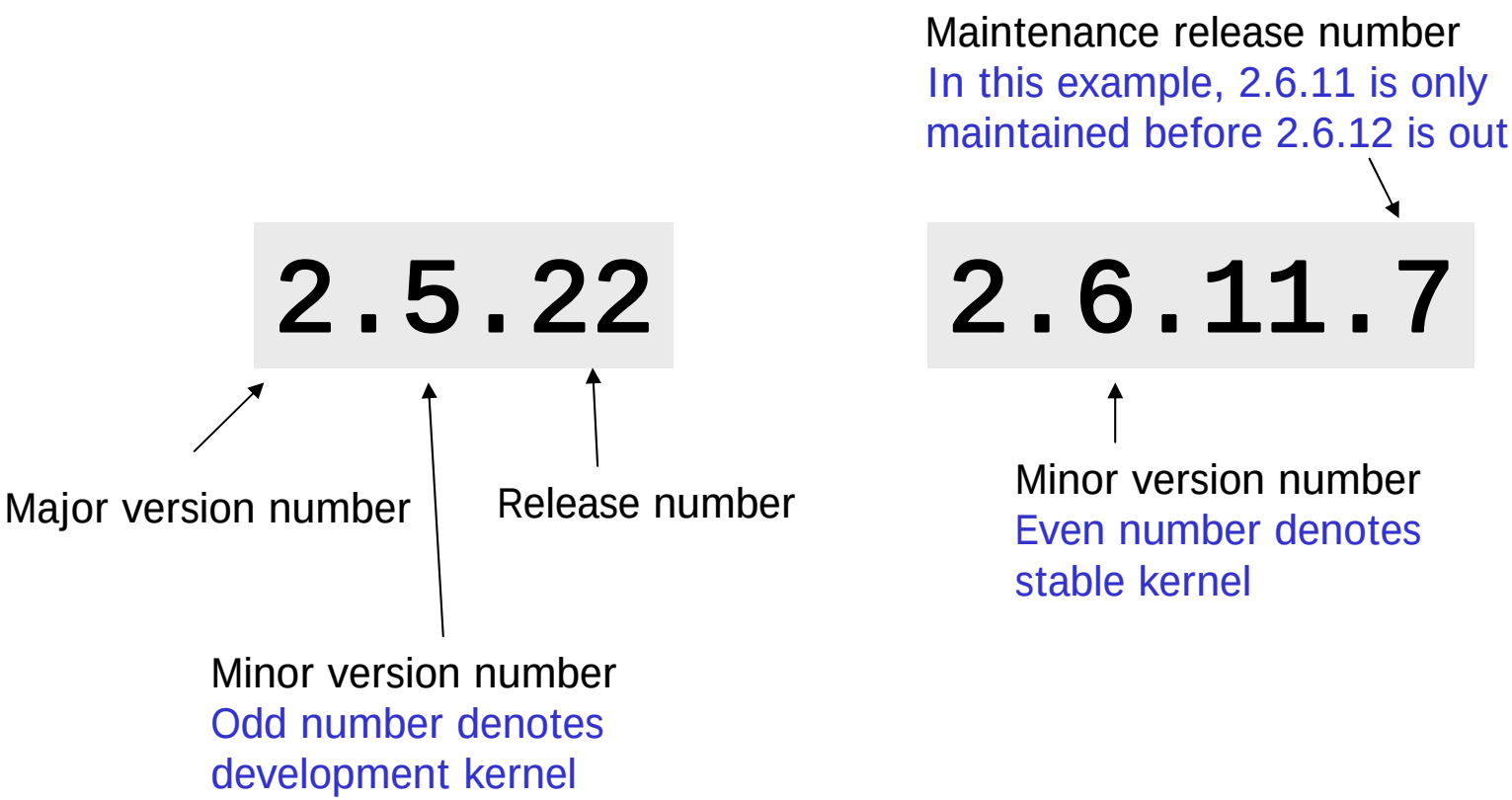


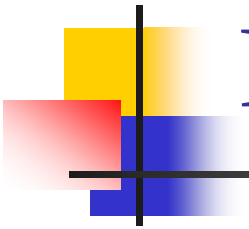
The history

- Initially developed by Linus Torvalds in 1991
- Source code is released under GNU Public License (GPL)
  - If you modify and release a program protected by GPL, you are obliged to release your source code

Version	Features	Release Date
0.01	initial release, only on i386	May 1991
1.0	TCP/IP networking, swapping	March 1994
1.2	more hardware support, DOSEMU	March 1995
2.0	more arch. support, page cache, kernel thread	June 1996
2.2	better firewalling, SMP performance, NTFS	January 1999
2.4	iptable, ext3, ReiserFS, LVM	January 2001
2.6	BIO, preemptive kernel, O(1) scheduler, I/O scheduler, objrmap, native POSIX thread library	December 2003

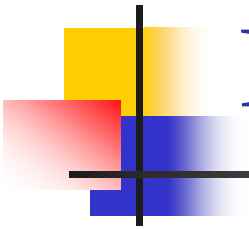
Rules of Linux versioning





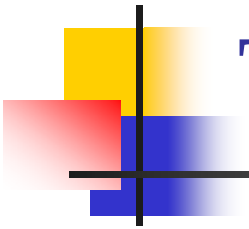
Features of the Linux kernel

- Monolithic kernel
  - Do everything in a single large program in a single address space
    - Allow direct function invocation between components
  - Microkernel, on the other hand
    - Modular design, the kernel is broken down into separate processes
    - Use message passing interface instead of direction function call
    - Example: Mach, Windows NT/2000/XP



Features of the Linux kernel (cont.)

- Dynamic loading of kernel modules
  - Runtime binding of Linux kernel and modules
- Multiprocessor support
  - SMP, NUMA
- Preemptive kernel
  - Since 2.6, the kernel is capable of preempting a task even if it is running in the kernel
- Threads are treated just like processes
  - The only difference is the sharing of memory resources
- Object-oriented device model, hotpluggable events, and a user-space device filesystem (sysfs)

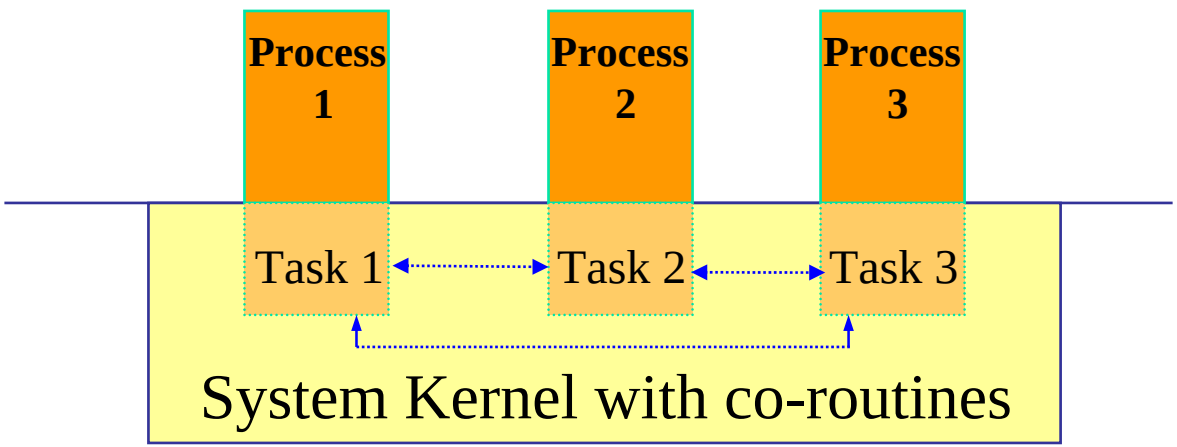


The concepts of processes

- Linux is a multi-user system, allowing multiple instances of programs to be executed at the same time
- Processes
  - An instance of a program in execution
  - Execution may be preempted at any time
  - Concurrency by means of context switching
  - Independency via the support of the CPU to prevent user programs from direct interacting with hardware components or accessing arbitrary memory locations
    - User mode and kernel mode (CPU ring level)
    - Memory protection (paging)

Processes and tasks

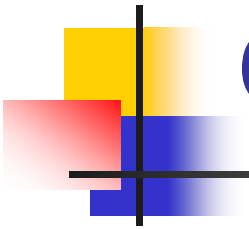
- Processes
  - seen from outside: individual processes exist independently
- Tasks
  - seen from inside: only one operating system is running



Process descriptor – `task_struct`

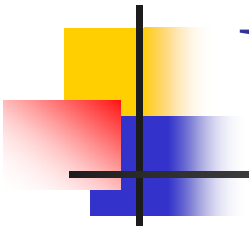
- Each process is represented by a process descriptor that includes information about the current state of the process

Type	Name	Description
volatile long	**state**	Current state of the process
int	**prio**	Priority of the process
unsigned long	**policy**	Scheduling policy (FIFO, round robin, normal)
unsigned int	**time_slice**	Time quantum of the process, decreased at every timer interrupt. If zero, scheduler activates other process
struct list_head	**tasks**	double linked list of all process descriptors
pid_t	**pid**	the process ID of the process
struct thread_struct	**thread**	CPU-specific state (registers) of the process



Context switching

- Context switching
  - Save the contents of several CPU registers into current process's process descriptor
  - Restore the contents of the CPU registers from next process's process descriptor
- Registers to be saved or restored
  - Program counter and stack pointer registers
  - General purpose registers
  - Floating point registers
  - Processor control registers (process status word)
  - Memory management registers (e.g. CR3 on x86)

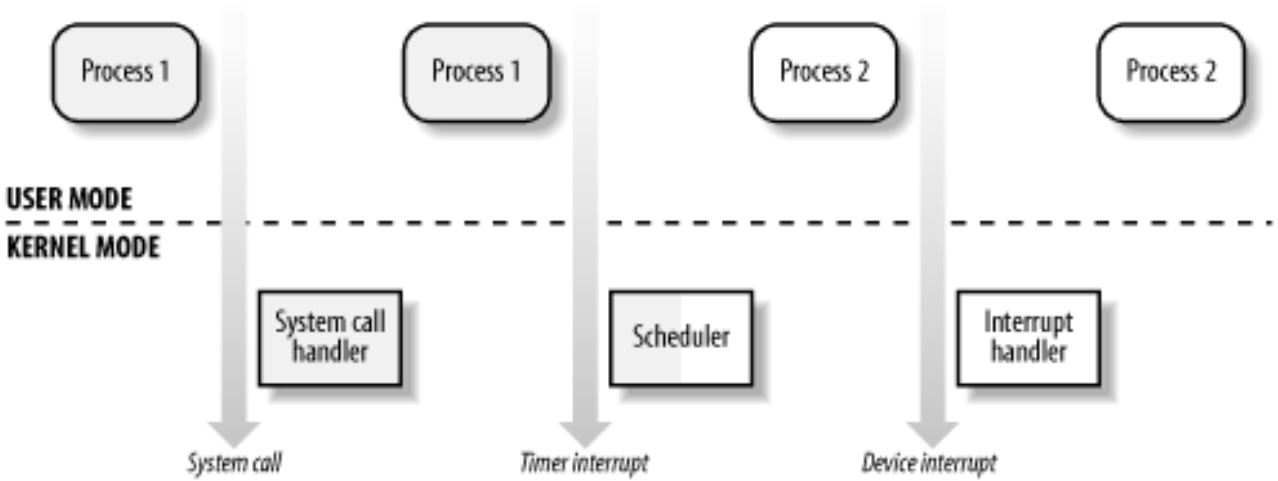


User mode and kernel mode

- CPU runs in either user mode or kernel mode
- Programs run in user mode cannot access kernel space data structures or functions
- Programs in kernel mode can access anything
- CPU provides special instructions to switch between these modes

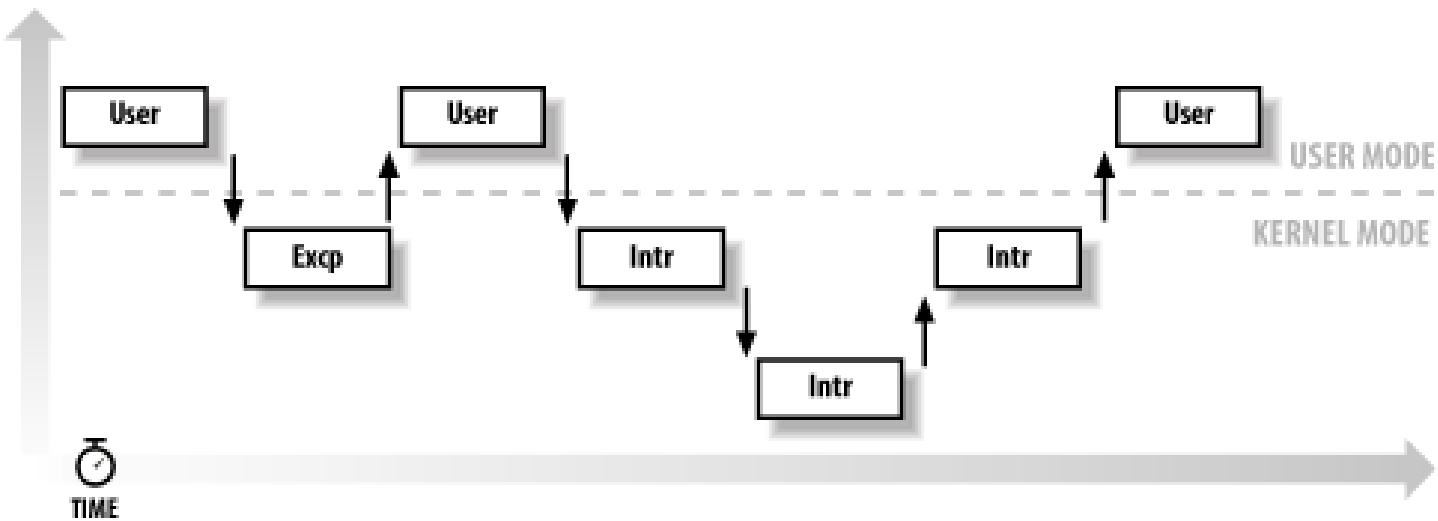
Switching into kernel mode

- CPU may enter kernel mode when:
  - A process invokes a system call
  - The CPU executing the process signals an exception
  - A peripheral device issues an interrupt signal to the CPU to notify it of an event
  - A kernel thread is executed

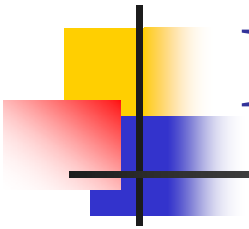


Reentrant kernel

- Reentrant -- several processes may be executing in kernel mode at the same time

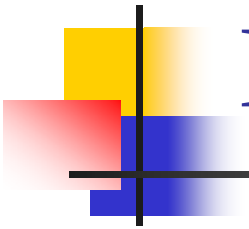


Interleaving of kernel control paths



Kernel control path

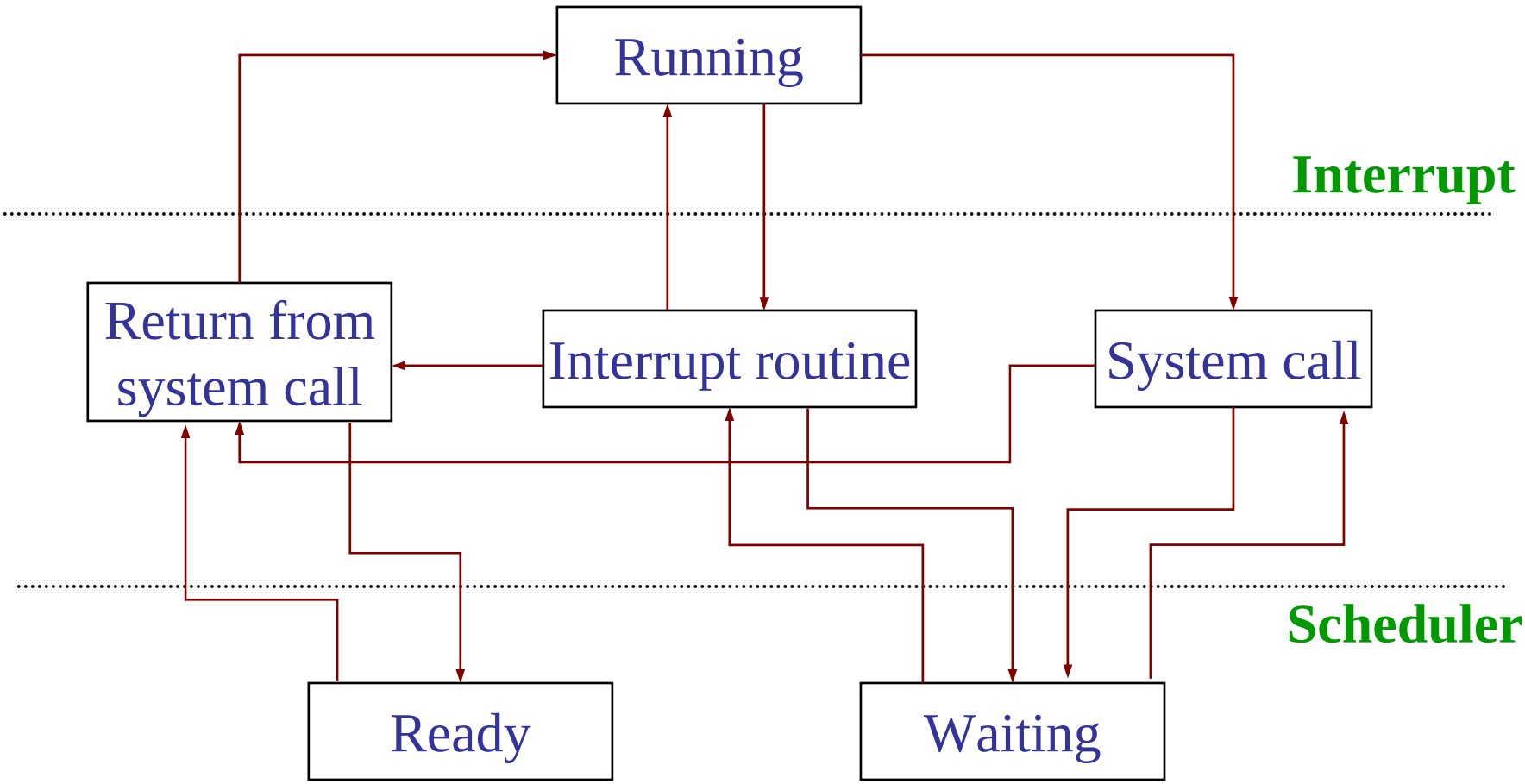
- Kernel control path – the sequence of instructions executed by the kernel to handle a system call, an exception, or an interrupt
- At any given moment, CPU may be doing one of the following things
  - In kernel space, in process context, executing on behalf of a specific process (system call or exception)
  - In kernel space, in interrupt context, not associated with a process, handling an interrupt
  - In user space, executing user code in a process



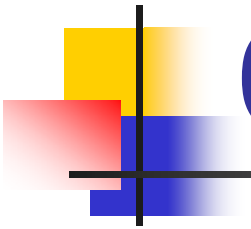
Kernel mode stack

- In user mode, each process runs in its private address space
  - User-mode stack, data, code
- In kernel mode, each kernel control path refers to its own private kernel stack
  - A kernel mode stack per process
  - A interrupt stack for all interrupts

Kernel control path of a process

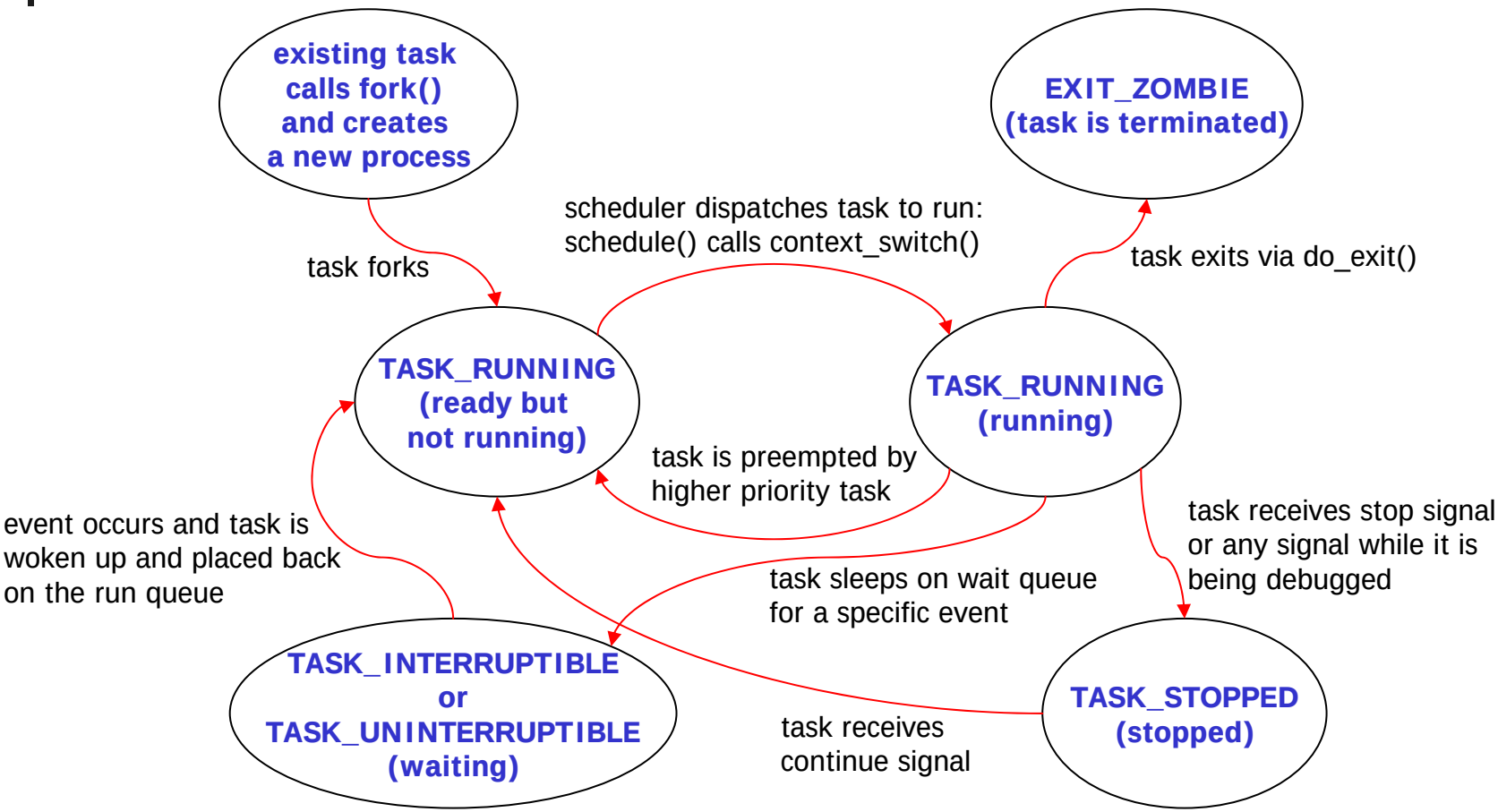


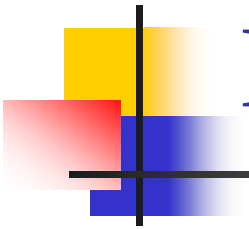
Kernel control path of a process (cont.)



- Running
  - Task is active and running in the non-privileged user mode.
  - If an interrupt or system call occurs, the processor is switched to the privileged system mode and the appropriate interrupt routine is activated
- Interrupt routine
  - hardware signals an exception condition
  - E.g. page fault, keyboard input or clock generator signal every 1 ms
- System call
  - System calls are initiated by software interrupts
- Waiting
  - The process is waiting for an external event (e.g. I/O complete)
- Return from system call
  - When system call or interrupt is complete
  - Check if a context switch is needed and if there are signals to be processed
- Ready
  - The process is competing for the processor

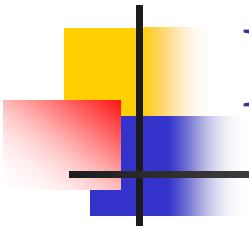
Transition of process states





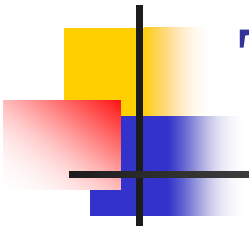
Interrupts

- Interrupts allows for hardware to communicate with operating system asynchronously
  - Remove the need of polling from OS
- Type of interrupts
  - Hardware generated interrupts (IRQ)
    - It is asynchronous! (the exact time of the delivery of an interrupt is unpredictable)
    - Example: interrupt from timer or network card
  - Software generated interrupts (exception or trap)
    - It is synchronous! (generated by CPU)
    - Example: Page fault, divide by zero, system call



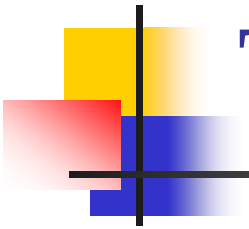
Designing interrupt handlers

- Limitations that must be aware of
  - Interrupt handlers may interrupt other important tasks (e.g. multimedia player) or other interrupt handlers
  - Runs with current interrupt level disabled or worst, all local interrupts are disabled
    - Delaying the interrupt processing of other devices (think about sharing interrupt lines)
  - Time critical since they deal with hardware (e.g. NIC)
  - Cannot block since they do not run in process context
- Design goal
  - Interrupt handlers should execute as quickly as possible



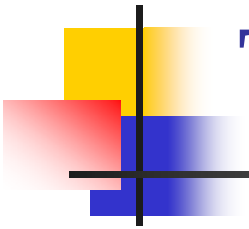
Top halves and buttom halves

- Interrupt handler may need to perform a large amount of work
  - conflict with the goal of quickness
- Divide an interrupt handler into two parts
  - Top half
    - Run immediately upon receipt of the interrupt
    - Perform only the work that is time critical
  - Bottom half
    - Runs in the future at a convenient time with all interrupts enabled



Timers and time management

- System timer (i.e. timer interrupt)
  - Program the hardware timer to issue interrupts periodically
  - Works must be performed periodically
    - Update the system uptime and the time of day
    - Check if the current process has exhausted its timeslice and, if so, causing a reschedule
    - Run any dynamic timers that have expired
    - Update resource usage and processor time statistics
- Dynamic timer
  - schedule events that run once after a specified time has elapsed (ex. Flush an I/O request queue after some time)

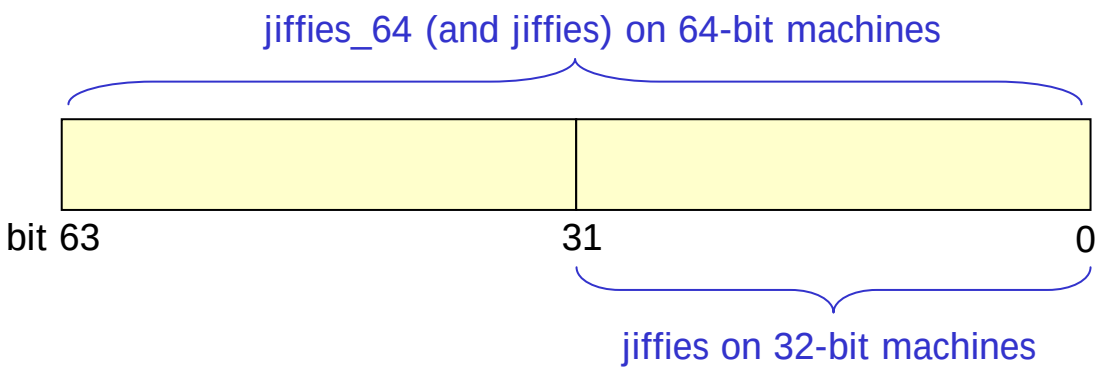


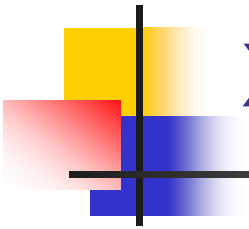
The tick rate: HZ

- HZ macro defines the frequency of the timer interrupt in Linux
  - If HZ = 100, you have 100 timer interrupts per second
  - On i386, HZ is 100 for 2.4 kernel and 1000 for 2.6 kernel
- The pros and cons for a higher HZ
  - Pros: improve the accuracy of timed events and preemption of process
  - Cons: less processor time available for real work, less battery time for laptop

`jiffies` variable

- The number of ticks that have occurred since the system booted
- `jiffies` variable is 32 bits or 64 bits in size depends on the architecture
- With `HZ` = 1000, it overflows in 49.7 days
  - Use macro provided by the kernel to compare tick counts correctly

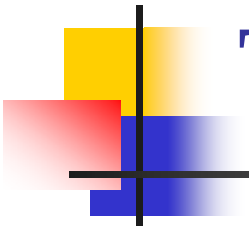




`xtime` variable

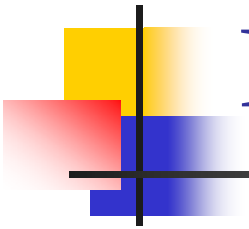
- The current time of day (the wall time)
  - the number of seconds that have elapsed since midnight of Jan. 1, 1970
- On boot, the kernel reads the RTC (real-time clock) and uses it to initialize `xtime`

```
struct timespec {
        time_t tv_sec;          /* seconds */
        long tv_nsec;           /* nanoseconds */
} xtime;
```



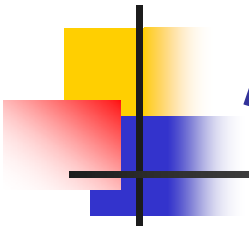
The purposes of system calls

- The only interfaces through which user-space applications can access hardware resources
- The benefits
  - An abstracted hardware interface for user-space
    - Nearly all kinds of devices are treated as files
  - Enhancement of system security and stability
    - Properly use of CPU time, memory
  - Virtualization of hardware resources
    - Multitasking and virtual memory



POSIX, C library and system calls

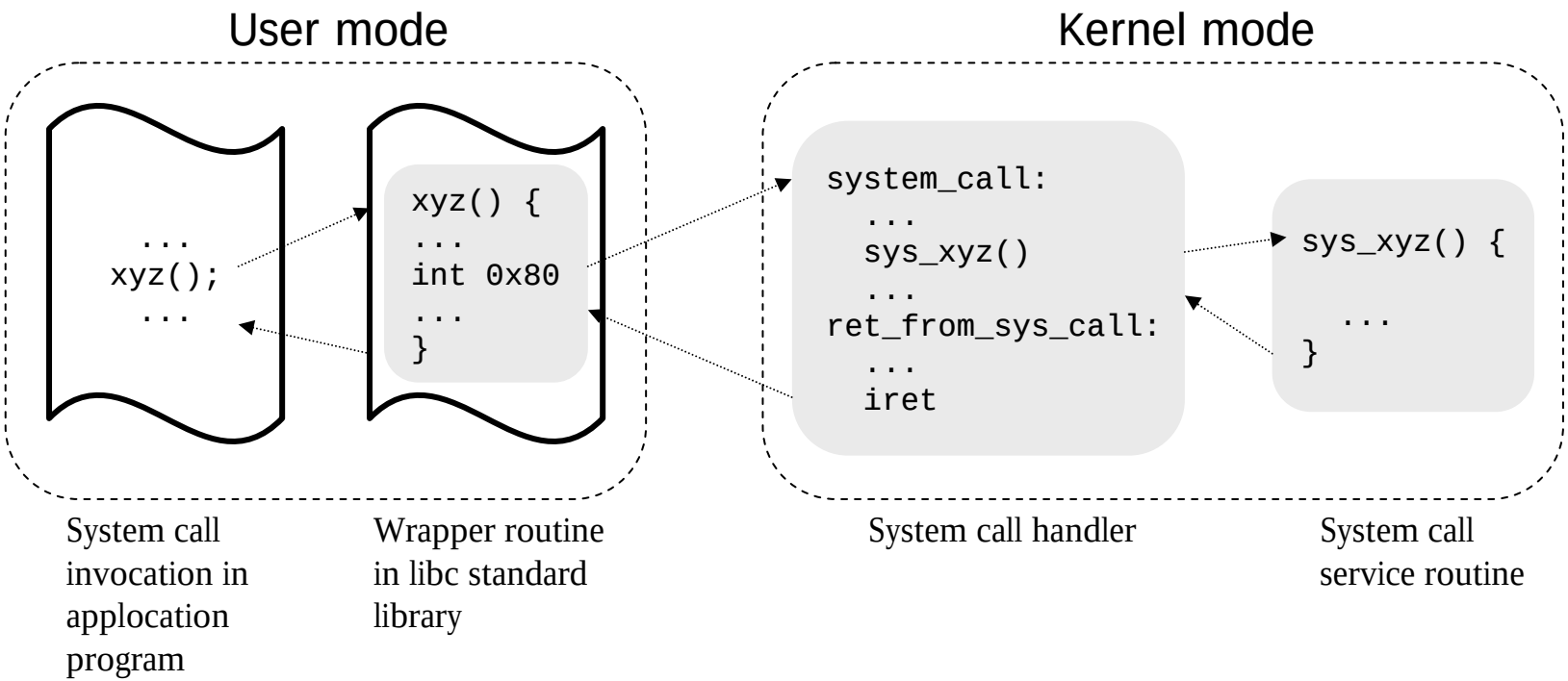
- POSIX (Portable Operating System Interface)
  - A single set of APIs to be supported by every UNIX system to increase portability of source codes
- C library implements the majority of UNIX APIs
- A C library function can be
  - just a wrapper routine of a system call
  - implemented through several system calls
  - not related to any system calls



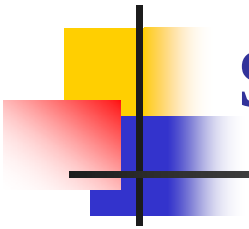
syscalls in Linux

- Each system call is assigned a ***syscall number***, which is a unique number used to refer to a specific system call
- Kernel keeps a list of all registered system calls in the `sys_call_table`
- A special CPU instructions is used to switch into kernel mode and execute the system call in kernel-space
  - On i386, the special instructions can be `int 0x80` or `sysenter`

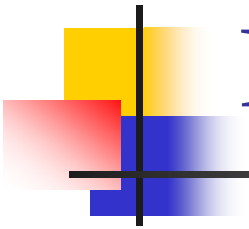
Invoking a system call



Consideration of implementing a system call



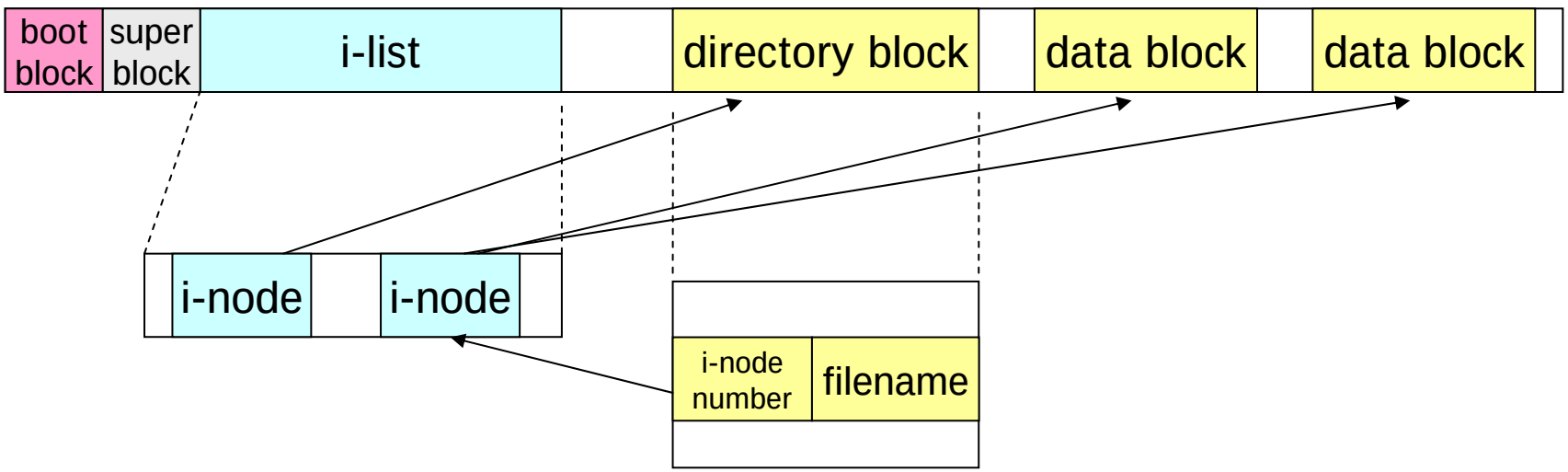
- You need a syscall number, officially assigned to you during a developmental kernel series
- When assigned, the number and the system call interface cannot change
  - or else compiled applications will break
  - likewise, if a system call is removed, its system call number cannot be recycled
- The alternatives
  - Implement a device node and use `read()`, `write()` or `ioctl()`
  - Add the information as a file in procfs or sysfs



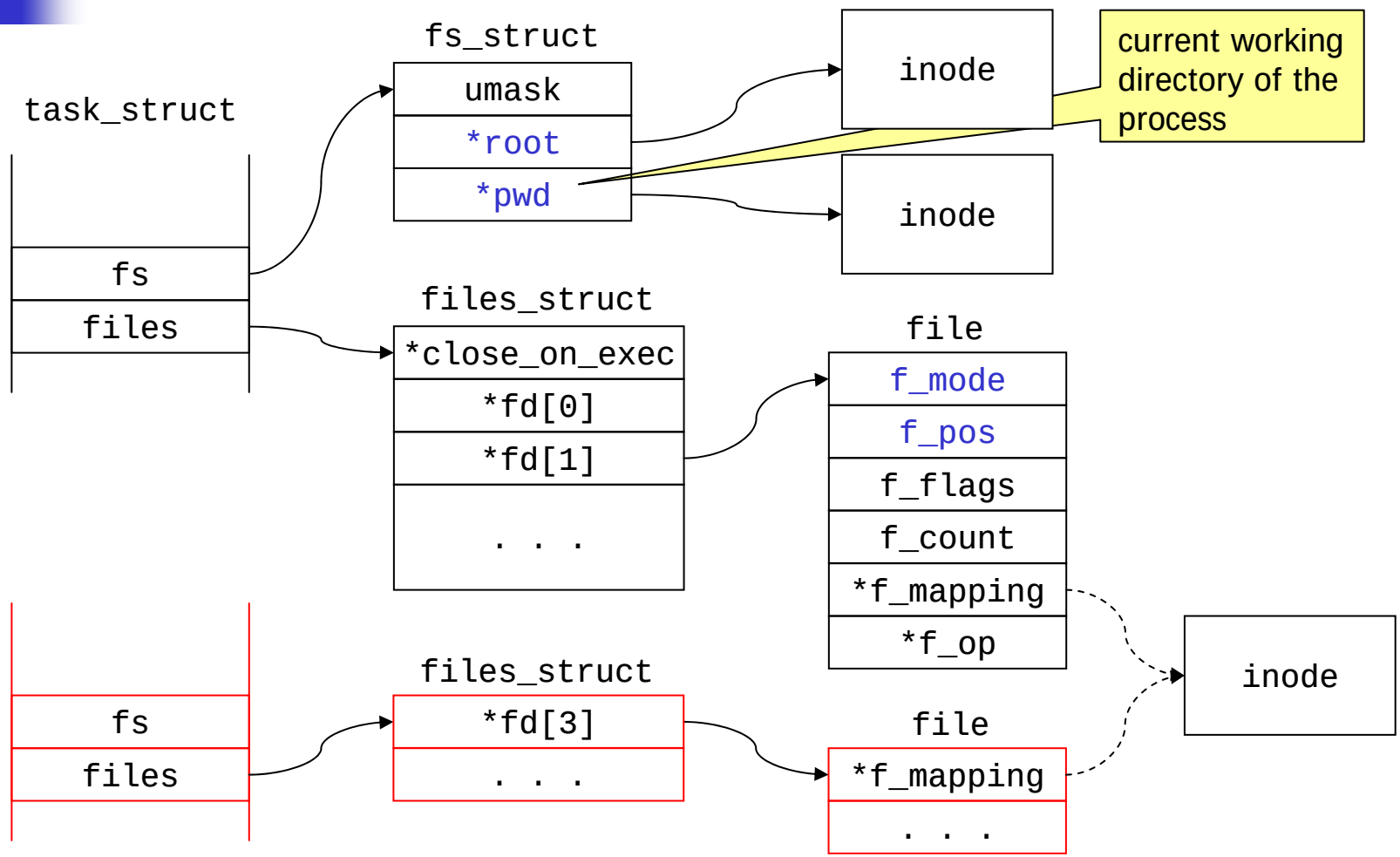
Files and inodes

- Inode has a number of meanings
  - The inode structure in the kernel memory
  - The inode structure stored on the hard disk
  - Both describe files from their own viewpoint
- File structures is the view of a process on files represented by inodes
  - File is opened for: read, write or read+write
  - Current I/O position

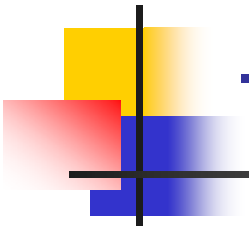
The structure of a traditional UNIX file system



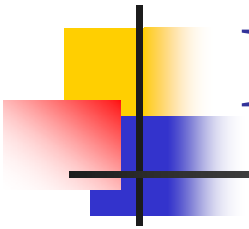
Files and inodes (cont.)
-- two processes open the same file



Linux kernel programming -- a different world



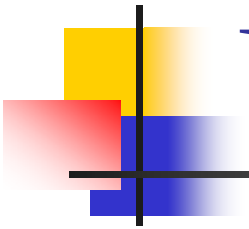
- No access to the C library
- The kernel code uses a lot of ISO C99 and GNU C extensions
  - Inline assembly
  - Inline functions
  - Branch optimization with macros: `likely()` and `unlikely()`
- No memory protection
- No (easy) use of floating point
- Small, fixed size stack
- Kernel is susceptible to race conditions because of
  - Multi-tasking support, Multiprocessing support, Interrupts and preemptive kernel



Kernel books

- Linux Kernel Development 2nd Edition, Robert Love, Novell Press, 2005
- Understanding the Linux Kernel 2nd Edition, Bovet & Cesati, O'REILLY, 2002
- Linux Device Drivers 3rd Edition, Corbet, Rubini & Kroah-Hartman, 2005

Useful sites about Linux kernel



- Linux Weekly News, http://lwn.net
  - A great news site with an excellent commentary on the week's kernel happenings
- KernelTrap, http://www.kerneltrap.org
  - This site has many kernel-related development news, especially about the Linux kernel
- Kernel.org, http://www.kernel.org
  - The official repository of the kernel source
- Linux Kernel Mailing List, http://vger.kernel.org
  - The main forum for Linux kernel hackers