Operating Systems. Files

Operating Systems I
Viktor Iakovlev (Victor Yacovlev)

OS Purpose

- The Computers might have various processors and peripherals
- Various memory models
- You must know everything about I/O ports, interrupts and physical memory to make applications...

Really?

The OS components

- The Kernel
- Set of Standard Libraries
- At least minimal environment to interact users

API Layers

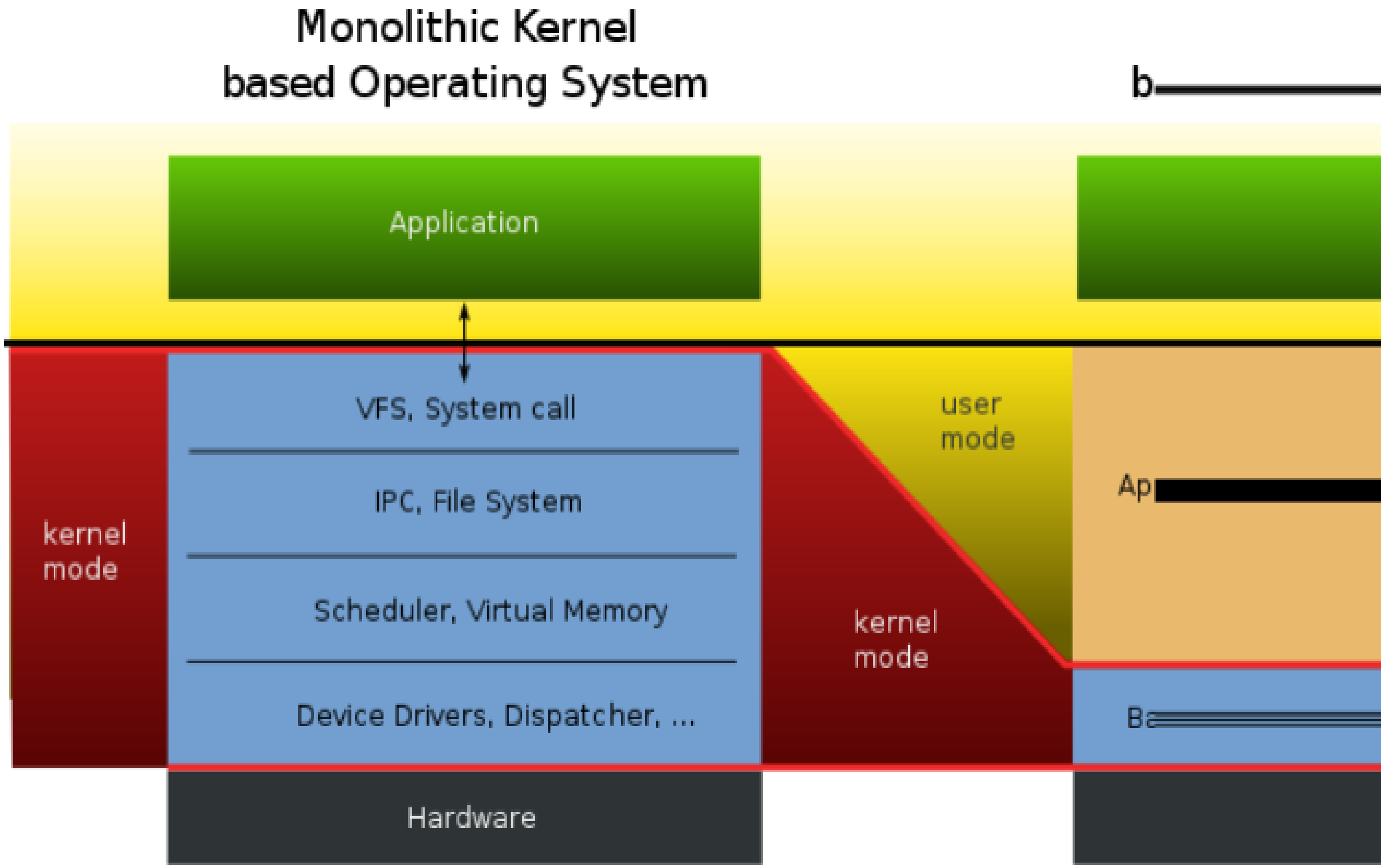
Programming Languages:

- Plain C Language: ISO/IEC 9899:2011
- C++ Language: ISO/IEC 14882:2017

API to interact operating systems:

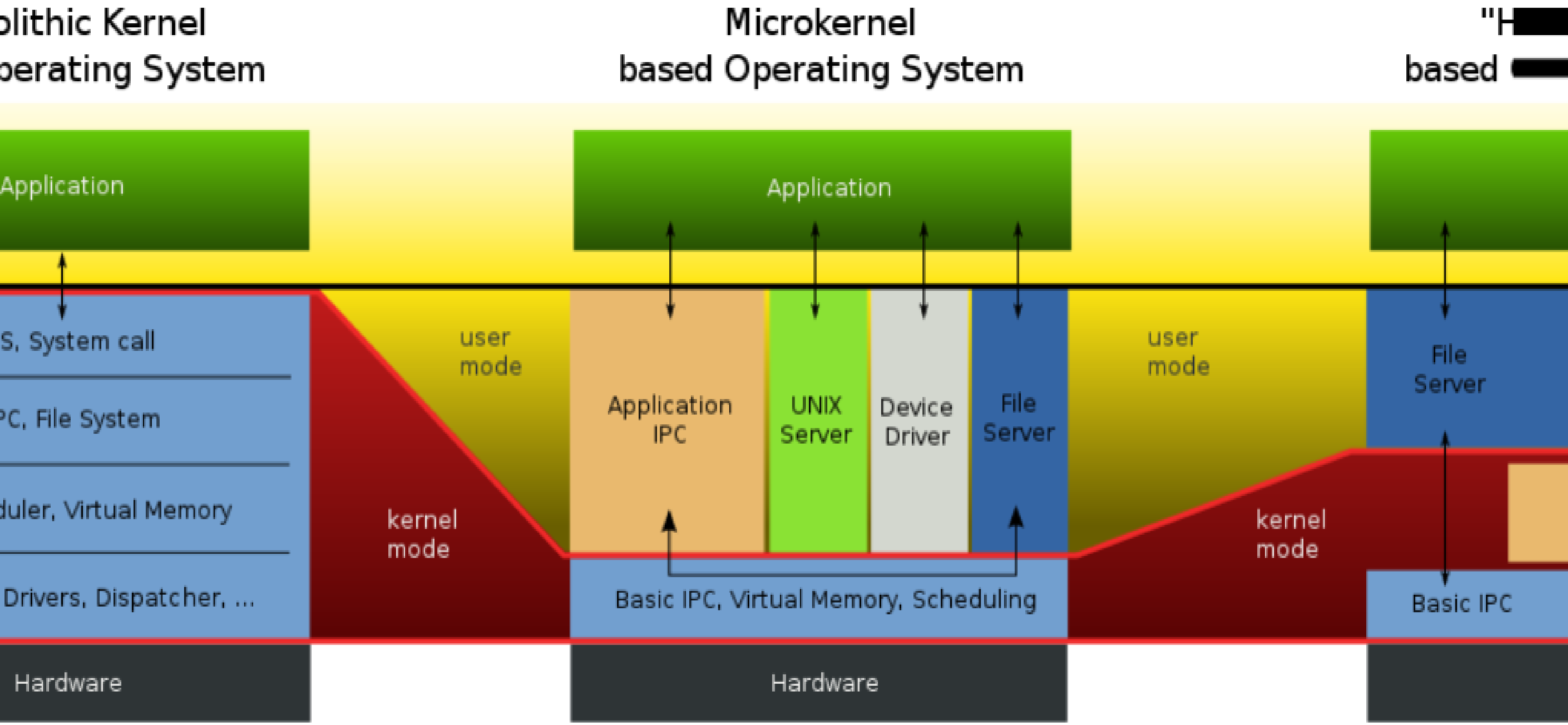
- ***P***ortable ***O***perating ***S***ystem ***I***nterface based on UNI***X*** (POSIX): IEEE 1003.1-2017
- WinAPI: internal Windows application programming interface

Monolitic Kernel



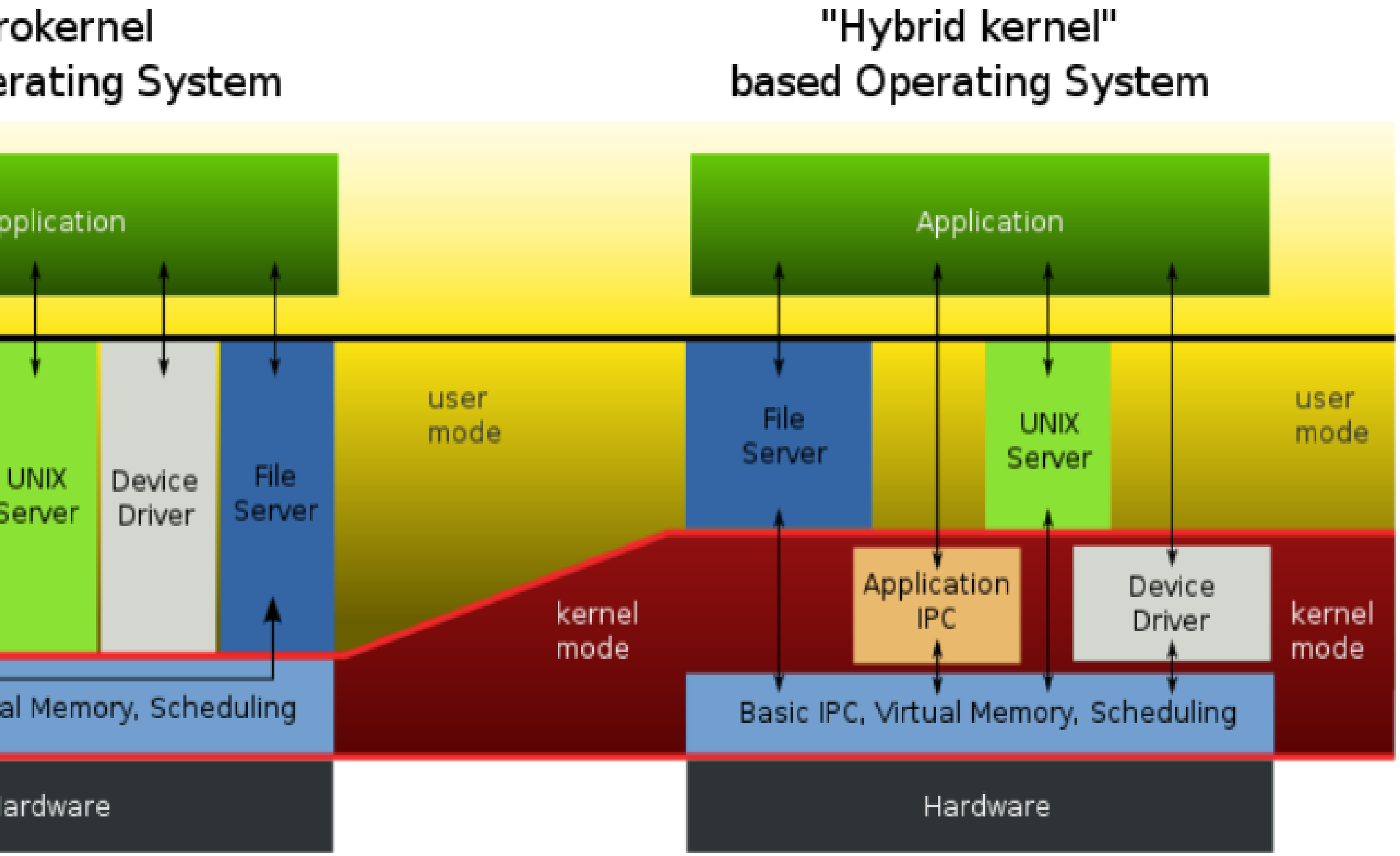
Linux, FreeBSD etc.

Microkernel



QNX, MINIX3

Hybrid Kernels



Windows, Haiku, MacOS X

Application Abstraction Layers

- Applications
- High-level Libraries
- libstdc++
- glibc -- POSIX
- Kernel -- POSIX

Interaction Using Libraries

- The Libraries are accessible from the Application memory space
- Just call a function (and don’t forget on PLT table long jump)
- The function arguments and data structures might be arbitary

Interaction Using The Kernel

- int 0x80
  - system call number stored at eax
  - arguments are stored at ebx, ecx, edx, ebp
- syscall / sysret
  - system call number stored at rax
  - arguments are stored at rdi, rsi, rdx, rcx

- Only integer arguments by size of Word:
  - Integer Values
  - Addresses (Pointers)

Interaction Using The Kernel

- vdso - virtual «library» that provides read-only kernel functions
- x86_64 examples:

```
  __vdso_clock_gettime
  __vdso_getcpu
  __vdso_gettimeofday
  __vdso_time
```

Kernel Subsystems

- Device Drivers
- Memory Management
- Task Scheduling
- Inter-Process Communication
- **Virtual File System**

Filesystem Tree for UNIX Systems
Virtual ***F***ile ***S***ystem

- The common naming space for all storage types and disks, even for remote
- Common API for interactions
- Tree-based structure

UNIX File Types

- Regular File
- Directory
- Devices: block and character
- Symbolic (but not hard) links
- Names channels (FIFO)
- Sockets

Demonstration: the /dev filesystem

Regular Files

- Just a Files
- Can seek over read/write position over the file contents
- The contents of file are not covered by filesystem

Directories

- Directory - is a file with a contents
- Directory content is a binary, that stores `struct dirent` entries

Links

- Symbolic links - just a files that stores paths to other files
- Hard links - just additional names to existing files in filesystem structure

- The functions for operating files deals links like they are real files

Devices

- Character devices: allows to read and write stream of data
- Block devices: allows to read, write and seek pointer

- Be careful!

  ```
  dd if=/dev/zero of=/dev/sda # facepalm
  ```

Physical Devices and VFS

- System root VFS - directory /
- Filesystem mounting:
  - commands mount/umount
  - configuration file /etc/fstab

Physical File Systems

- For disk usage - fat, ntfs, ext2/3/4
- Fileless disk usage - swap
- Network - smbfs, nfs
- Virtual - tmpfs, sshfs, overlayfs

Files to VFS Mapping

- Each phisycal FS has it's own unique *session number* by size of 32 bits (st_dev):
  major (24 бита) - device type
  minor (8 бит) - enumeration number
- Each file on disk has it's own unique number on filesystem calles *inode* (st_ino)
- If filesystem does not support inodes (alien file systems) - inodes are assigned by filesystem driver
- The tuple of (st_dev, st_ino) are unique keys to find any file

File Descriptors

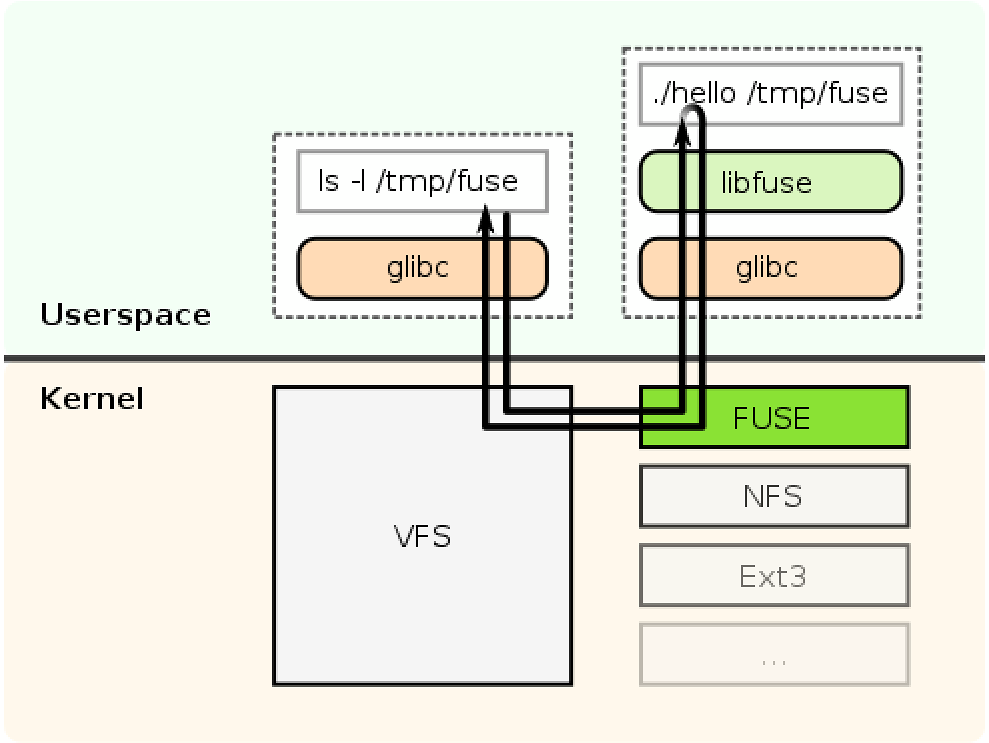
- The tuple (st_dev, st_ino) maps to file at VFS
- The file must be in open state to be accessed by user process
- File Descriptor - is an unique key for every opened file for user process
- Each user process has it’s own set of file descriptors
- The count of file descriptors is limited

File Descriptors

- Three standard file descriptors:
  - stdin (0)
  - stdout (1)
  - stderr (2)
- File open operation uses the lowest unused integer number for file descriptor

Demo: arbitary file descriptor number

FUSE - user space file system



- The library to write custom filesystem
- Not required to user C lanugage, the Python is good too

Data Integrity

- The Linux Kernel tries to use as much free memory as possible for disk cache
- Data are guaranteed to be flushed to disk on unmount or `sync`
- There might be additional buffers, like USB internal buffer of on-disk internal buffer

Data Integrity

- Files Intergrity
  - hard problem to be solved at OS level
  - duplication using RAID-1+
- Filesystem Integrity
  - transaction based: each change is written **to journal** as a sequence of operations
  - might be rolled back to make filesystem consistent