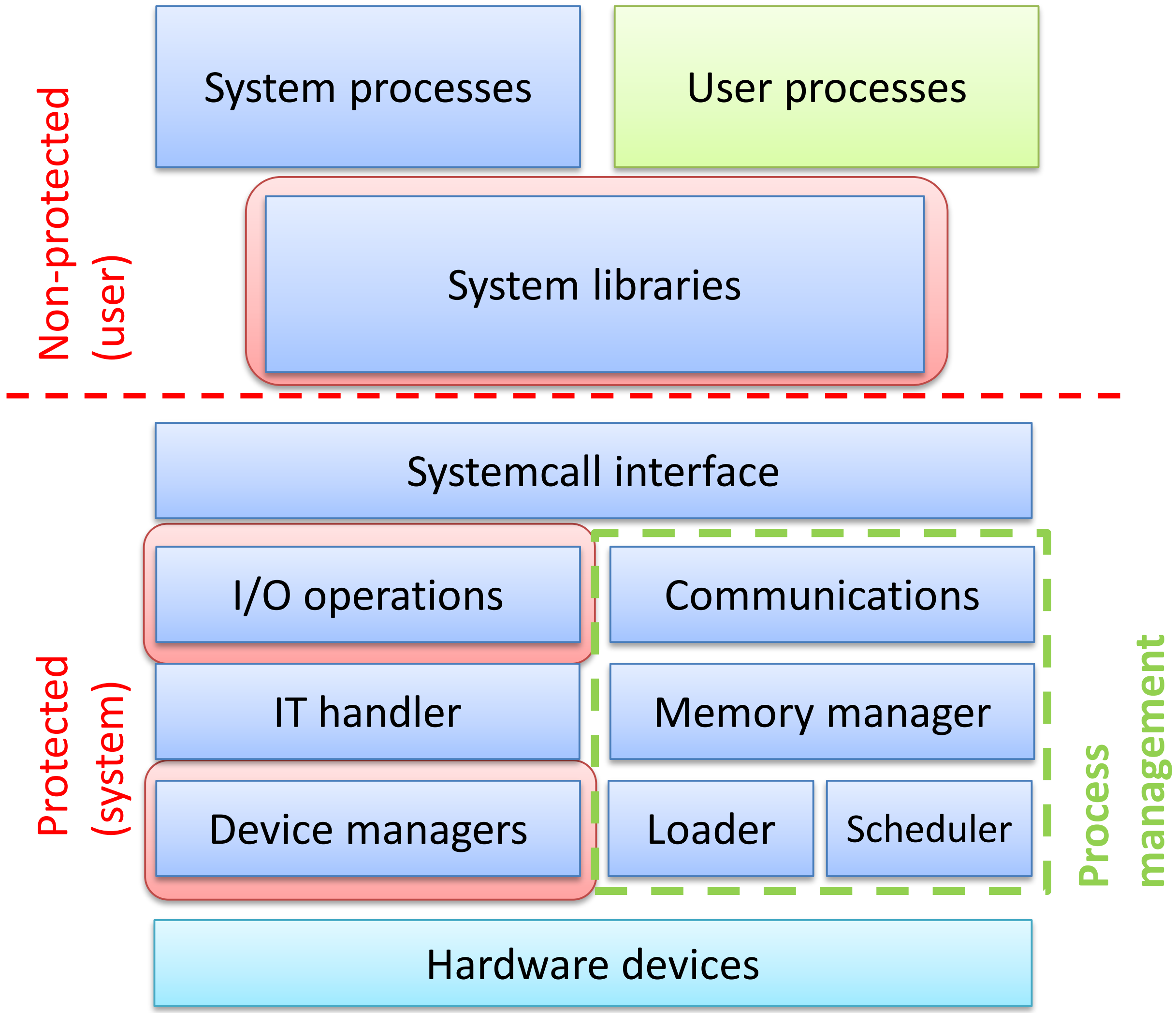
Operating Systems – File systems part 1

Péter Györke
http://www.mit.bme.hu/~gyorke/
gyorke@mit.bme.hu

Budapest University of Technology and Economics (BME)
Department of Measurement and Information Systems (MIT)

The slides of the latest lecture will be on the course page. (https://www.mit.bme.hu/eng/oktatas/targyak/vimiab00)

The main blocks of the OS and the kernel (recap)

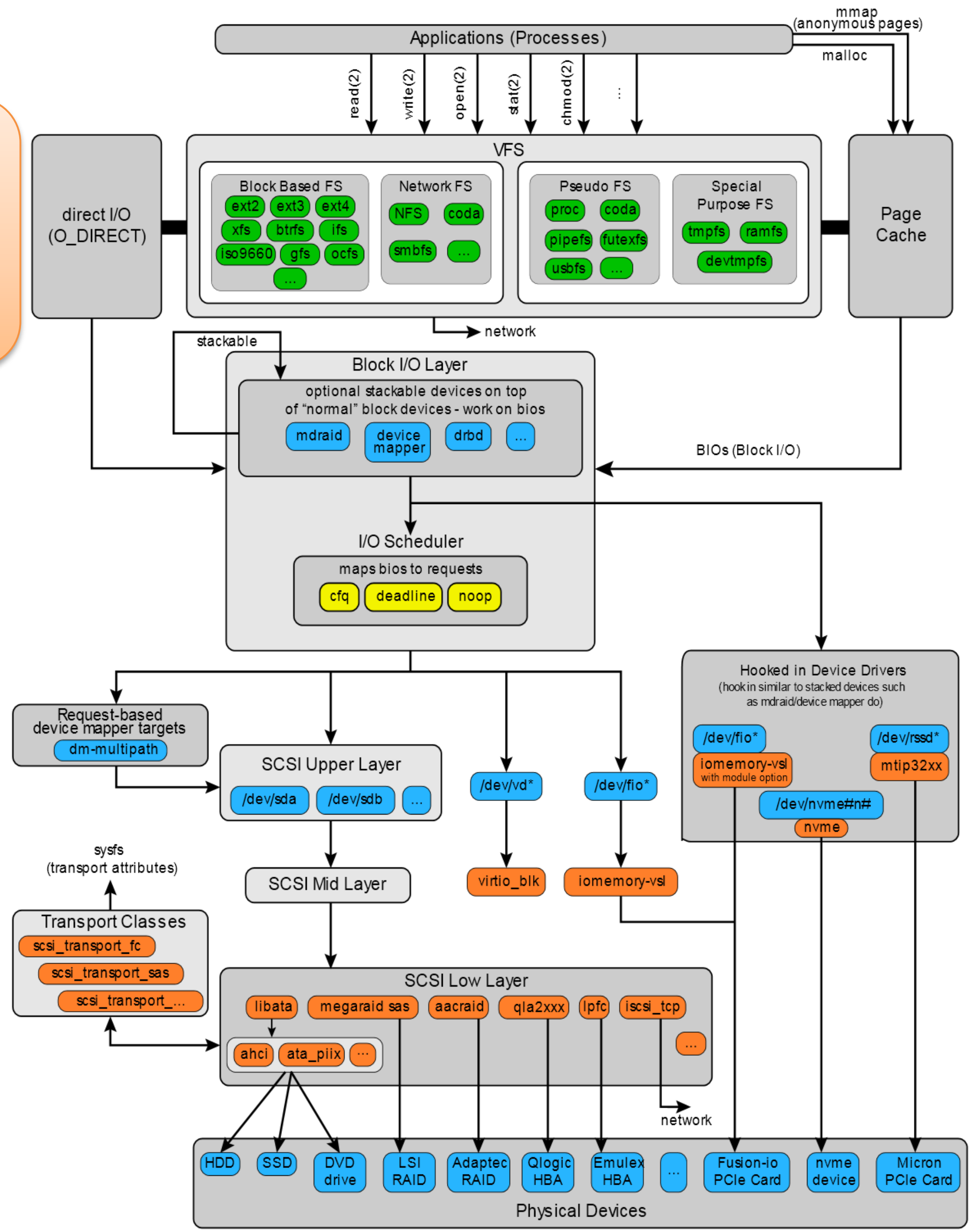


What we learned until now?

- I/O operations – usually file operations
- The nature of tasks
  - There are I/O intensive tasks (memory intensive tasks may become I/O intensive, see virtual memory)
  - Most of the tasks on a client machine are I/O intensive
- Scheduling
  - Tasks usually spent a lot of time in waiting state, because I/O operations are slow
- Memory management
  - The physical memory is extended with swap space on disk (much slower)
  - Background data can be loaded into physical memory (mmap)
- Synchronization
  - Waiting for others isn't a good thing, especially the busy waiting

Overview of the topic

- User interfaces
  - User
  - Administrator
  - Programmer
- File systems
  - Kernel data structures
  - File system interfaces
  - Data arranged in blocks on disks
- Storing the data
  - Physical storages (HDD, SSD)
  - I/O scheduling
  - Local storage system virtualization (RAID, LVM)
  - Network and distributed file systems



File systems from the user's point of view

- Standard user of the OS
  - Command line and GUI file managers
    - Windows explorer, Nautilus, Dolphin, Total commander, mc
  - Volumes, folder structure, special folders/directories
  - Managing files and folders, owner and group, permissions, attributes
- Administrator
  - Managing file systems (creation, maintenance, deletion)
  - Mounting local or remote file systems
  - Performance tuning
  - Managing disk usage
  - Performing back-ups
- Programmer
  - Application programming interfaces (system libraries, system calls)
  - File descriptors, handles: handling open file objects
  - File operations: open, create, write, read, seek, close, delete, …
  - Locking files for exclusive usage

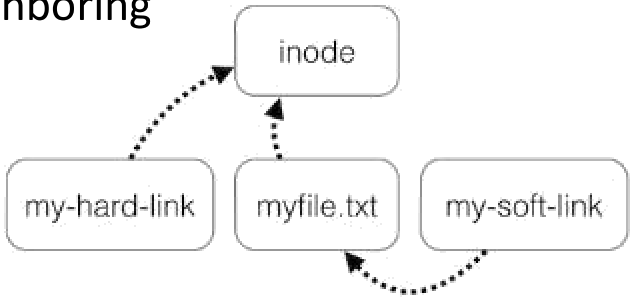
Physical and logical units (definitions)

- **File**
  - Logical unit of storage
  - It is referenced by its name (by user)
  - Some systems use extensions to define the type of the data (*.abc)
- **Directory**
  - Logical organization structure for files
  - It can contain files and other directories
  - A file or directory may be accessed from different paths (OS dependent)
- **Volume**
  - A set of related files and directories
  - It is assigned to a physical storage unit (e.g.: partition)
  - On windows it is also called „drive”

- **File system**
  - Physical storage unit of files and directories, organization system of them
- **Partition**
  - Organization unit of the disk, it can contain one file system

Directory structures, volumes and drives

- Files and directories can be assigned in different ways
- The basic structure is a directed tree
  - A directory can contain files and other directories
  - The direction of the edges is determined by the containment relation
  - Path: a place of a file or a directory in the tree
    - Absolute: the path from the root of the tree
    - Relative: the path from a specific node in the tree
      - Usually the actual working directory of the user
- Some systems (e.g. UNIX) use further edges
  - These edges can connect nodes which are not neighboring
  - With the introduction of these edges,
  - the tree becomes a graph (directed)
  - Hard link
    - More nodes (files) linked to the same data
  - Symbolic link (symlink, soft link, shortcut)
    - It references a file or directory which is linked to the physical data (it's another file)
  - How can we delete the link or data? What happens if there is directed circle in the graph?



- Typically there are more than one trees in a system
  - There can be more volumes in the system, each one contains one tree
  - On Windows, the drives are named with C, D, E, etc. letters

Overview of the Windows 10 folder structure

- More than one folder structures (trees)
  - Physical storages are assigned with logical units, drives
- The boot drive (usually `C:`) is the starting point (`C:\`)
  - `\Program Files` – installed applications
  - `\Program Files (x86)` – installed applications (32-bit)
  - `\ProgramData` – user independent data of the applications
  - `\Users` – user folders (files, folders, user dependent application data, …)
  - `\Windows` – the OS files and directories

- Further drives (`D:, E:`, …)
  - CD/DVD/USB drives
  - Further partitions on the disk
  - Network file systems

- Versions, trends
  - In the newer Windows systems the physical storages can be assigned to folders also (not just to volumes), but it isn't a widely-used feature

Overview of the UNIX directory structure

- It is organized into one structure (tree)
- The root directory is the starting point (`/`)
  - `/bin` – binary files for the system
  - `/sbin` – similar to /bin, usually programs with root permissions
  - `/dev` – hardware devices
  - `/etc` – system and application configuration files
  - `/home` – user directories and files
  - `/lib` – basic shared system libraries
  - `/mnt` – the mount point of physical partitions
  - `/tmp` – temporary files (for apps. and users)
  - `/usr` – user programs and libraries, documentation, etc.
  - `/var` – dynamic files of the system, logs, databases, …
- More details: `man hier`
- Disk usage: `df, du, xdu, baobab, kdiskstat, filelight`
- File system „standards", changes
  - Between the different UNIX systems, there are significant differences in the detailed operation
  - Filesystem Hierarchy Standard (FHS) is just a recommendation
  - UsrMove: the `/bin, /sbin` is moved under `/usr` (Solaris11, Fedora)

Overview of the Android directory structure

- To a certain point it has inherited the UNIX structure, additional directories
  - `/cache` – cache for applications
  - `/data` – user programs and data
  - `/data/app` – applications installed by the user
  - `/data/anr` – app-not-responding: error logs
  - `/data/tombstones` – memory dumps of the terminated apps.
  - `/data/dalvik-cache` – optimized binary files of the apps.
  - `/data/misc` – user configuration files
  - `/data/local` – temporary files
  - `/mnt` or `/storage` – mounted file systems, e.g. SD card
  - `/mnt/asec` – unsecured copies of the apps. running from SD card
  - `/system` – preinstalled apps., system libraries, configuration files
- Remarks
  - Full access to file is system is limited, only root user has full access, the vendors are limiting this. Becoming root is not part of the normal usage.
  - The apps. stored on the SD card are encrypted (`.android_secure`), these are mounted under the `/mnt/asec` directory when running

File properties (with UNIX examples)

- List the content of the actual directory (`ls -la`)

```
drwx------  6 root root   4096 Feb 23 14:20 .
drwxr-xr-x 22 root root   4096 Nov 21  2014 ..
-rw-r--r--  1 root root    570 Jan 31  2010 .bashrc
-rw-r--r--  1 vps  vps   71103 Nov  5  2013 package.xml
-rwxrwxrwx  1 root root     35 Feb 23 14:21 test.sh
lrwxrwxrwx  1 root root      8 Nov 24  2014 www -> /var/www
```

- What is in the list?
  - Type of the entry: `- d p l b c s`
  - POSIX permissions (see next slide)
  - Number of links
  - Owner and group
  - Size
  - Timestamp (ctime: change of the metadata, mtime: data modification, atime: access time)
  - Name of the entry
- The OS also stores
  - Unique identifier (for internal identification)
  - Location (where the file is stored on the disk)

The UNIX permission systems

- POSIX access permissions
  - 3x3 bits: owner, group, others X read, write, execute
  - Values: read-4, write-2, execute-1, no access-0
    - E.g.: 740 = owner: RWX, group: R, others: no access
  - In the case of directories, the execute means „list”
  - Setting: `chmod <permissions> <file/directory>`
    - E.g.: `chmod 750 /home/me` `chmod u+rwx,g+rx,o-rwx /home/me`
- Special permissions: SETUID, SETGID, StickyBit
  - SETUID/GID: **set u**ser **ID** upon execution" and "**set g**roup **ID** upon execution
    - The executed file will have the same permission as the owner (not the user which executed the file)
    - It is usually set to files which require root permissions
  - StickyBit: only the owner (and root) can delete/rename the files or directories

Administration of file systems

- Creating and configuring a file system
  - Select a type
  - Configure the data storage properties
  - The name of the volume (for users)
  - Selecting the partition and disk for the physical storage (determines the size)
  - Set up encryption (if the system supports it)
- Mounting a file system to a drive or directory
  - **Mount** and unmount
    - Mounting the physical storage to a given point of the logical structure
  - Mount point
    - a directory (typically an empty one) in the currently accessible filesystem on which an additional filesystem is **mounted**
- Checking, modifying, tuning the file system
  - Checking status and repair errors
  - Modify the size (not every file system makes this possible)
  - Performance tuning (accommodation for the storage device, compression, …)
- Sharing file systems on the network and mounting network file systems
- Back-ups

An overview of the widely used file systems

- FAT32
  - Typically used on portable storage devices because the compatibility
  - Originally 8+3 character file names extended to 255 characters, maximum file size: 4GiB (!)
- NTFS
  - Default file system in Windows
- UFS/ Berkeley FFS
  - Traditional UNIX file system, currently rarely used
- ext2,3,4 (cased on UFS)
  - Currently used file systems in Linux systems
- XFS
  - Default in RedHat Linux 7
- HFS+
  - Default in MacOS
- Integrated file + virtual storage systems (see later)
  - ZFS: Designed for Solaris, later it become open source, popular in BSD-s also
  - Linux btrfs: newer, currently under development
- Many more file systems
  - CD/DVD file systems
  - ISO9660 and extensions: filename and sizes are limited

Practice in Linux

- Basic file and directory operations
  - `cp, mv, cd, pwd, mkdir`
    - How to rename a file?
- File attributes: `ls -la`
- Managing file systems: `mount, umount, df, mkfs, fsck`
- Example: create a file system in a file

```
dd if=/dev/zero of=filesystem.img    bs=1k count=1000
losetup /dev/loop0 filesystem.img
mke2fs /dev/loop0
mount /dev/loop0 /mnt
```

  - A typical annoying error: device is busy
    - While unmounting a currently used file system (e.g.: unmounting portable drives)
    - Check what is used: `lsof /mnt`
- What’s happening in the file system?
  - `iotop, sar, dstat, vmstat,` …

Backing up and restoring data

- Multiple causes of data loss
  - Uncorrectable fault in the file systems
    - The error in the physical storage (disk error)
    - Inconsistency caused by power failure or other HW error
  - User mistakes (not rare)
    - Accidental deleting of files or whole file systems, partitions
  - Malwares (sadly these are also not rare)
    - Deleting or encrypting data (ransomware)
- The type of data loss
  - Limited (e.g.: disk error, user mistakes, …)
  - Total (e.g.: SSD sudden death)
- Creating a backup
  - How: automated (regular), manual (casual)
  - What: files or whole file system
    - A consistent state has to be backed up – problematic when the FS is in use
  - Where: high capacity disks, CD/DVD, tape systems
- Restoring the system from a backup (recovery)
  - Bare metal recovery: restoring the whole system
  - Data recovery: only recovering specific files

Programming interfaces

- Opening (creating) files
  - `open()` system call and its arguments
  - File descriptor and the opened file object (next slide)
  - File opened by multiple processes?
- Read, write, seek: `read()`, `write()`, `fseek()`
  - Sequential access: the data is accessed in the stored order
  - Direct access: given sized blocks can be read in any order
- Close files: `close()`
- Managing directories:
  - `opendir()`, `readdir()`, `rewinddir()`, `closedir()`

What happens when a program opens a file?

- Calling the open() system call…
  - A session is started to manage the file operations
  - The kernel locates the file on the disk
  - The location and metadata are loaded into a kernel object
  - A kernel data structure is created: **open file object**
    - Opening mode (read, write, append)
    - The pointer of the next read or write operation (**file pointer**)
    - The address of the related kernel object
    - The set of operations which can be performed on this file
    - The kernel returns the address of this object: the **file descriptor**
- During the further operations the file is accessed with the file descriptor
- When the session is closed, the kernel liquidates the date structures

Locking files

- Locking files
  - It is also a synchronization problem, to conserve the consistency of the file (as a shared resource)
  - It can be managed with classic synchronization methods
  - But it is more simple and safe on kernel level (level of file op.-s)
  - Deadlocks are also possible
- Advisory locking
  - OS provides tools for implementation
  - The file should be only accessed with using these tools
  - Usually system libraries contain the tools
- Mandatory locking
  - Kernel level mechanism
  - The system calls (e.g.: open()) checks the lock states, the lock is mandatory for every task
- The scope of locking: The whole file or just a part of it

Shared access to files through memory (mmap)

- Communicate through a file
  - It is problematic with the standard op.-s (`read(), write(), fseek()`)
  - Can we use a file like the shared memory?
- UNIX mmap (Windows: [CreateFileMapping]())
  - An open file object (`open()`) can assigned to an address: `mmap(addr, size, prot, flags, fd, offset)`
    - `addr`: the assigned address, 0: the kernel choses
    - `size`: the accessed data range
    - `prot`: the mode of access: R, W, X
    - `flags`: own or shared file, etc.
    - `fd`: file descriptor returned by the `open()` systemcall
    - `offset`: the start position
  - Return value: the assigned virtual memory address
  - Close the assignment: `munmap(addr, len)`
- Multiple access, consistency, mutual exclusion
  - Using the shared file is based on the PRAM model
- It is usable for simple file operations when there are many readers

I/O operations without waiting

- If the program can perform other instructions, it don’t has to enter into waiting state
  - There are two approaches: non-blocking, asynchronous
- Non blocking I/O operations
  - When calling the read() system call, there is an option: non-blocking
  - In this case, the call will return immediately
    - With the data
    - Or with „no data” error code
  - If there are no data the program can perform other instructions and later retry the read()
- Asynchronous I/O operations
  - The program initiates the I/O operation and set a buffer for the data
  - The asynchronous I/O request is sent
    - In the background the I/O operation is performed
    - The system call returns immediately
  - Meanwhile the program can perform other instructions
  - When the I/O op. is done, the kernel notifies the caller
    - E.g.: with a signal with custom handler

Implementation of file systems (overview)

- Operation from the user's point of view (already discussed)
  - Files, directories, tree/graph structure
  - Format, mount, unmount
  - Check, repair, create, modify, tune
- Operation on the disk (data organization in the storage system)
  - The logical units are assigned to physical devices
  - The data is stored in **blocks**
  - Beside the file contents, metadata is also stored
  - Managing the free (unused) blocks in the storage device
- Operation in the memory (during runtime)
  - File system descriptors (metadata of the mounted file systems)
  - Descriptors (metadata) of the files
    - Access to opened files
  - Managing the data in the memory, buffering

Storing file system data on disk

- Recap: the boot process
  - Level 0 (ROM) loader: loads the RAM loader from the disk
  - Level 1 (RAM) loader: loaded from the master boot record (MBR), loads the OS loader
  - Level 2 (OS) loader: it loaded from partition boot record, knows the file system
  - Kernel loader: initiates the kernel
    - It mounts the root file system (read-only in Linux)
  - User mode OS start: starting services and sessions
    - Mounting user file systems

- Many types of data are stored on the disk
  - Metadata
    - Partition types and location on the disk
    - File system descriptors (type, size, usage, etc.)
    - File (directory) descriptors (name, location, etc.)
  - Data
    - Bootloaders
    - File data (the actual data)

Organization of the file systems on the disk

- The stored data
  - File system metadata (superblock, master file table, partition control block)
  - File metadata (inode, file control block, on Windows: it is part of master file table)
  - Stored data

superblock	file metadata	data blocks

- The file system metadata
  - On disk
    - Type and size
    - List of free blocks
    - The location of the file metadata
    - State
    - Modification information
    - …
  - In the memory
    - Everything from the disk
    - Mounting information
    - Dirty bit
    - Locking state
    - …
- The file system is sensitive to metadata loss (e.g. block error)
  - Therefore backups are made
  - See: `dumpe2fs /dev/sda1 | grep -i superblock`

Location of the file metadata

- **On disk**
  - Authentication information (UID, GID)
  - Type
  - Permissions
  - Timestamps
  - Size
  - Data block locations
  - Example: UNIX inode (index node), Windows Master File Table entry

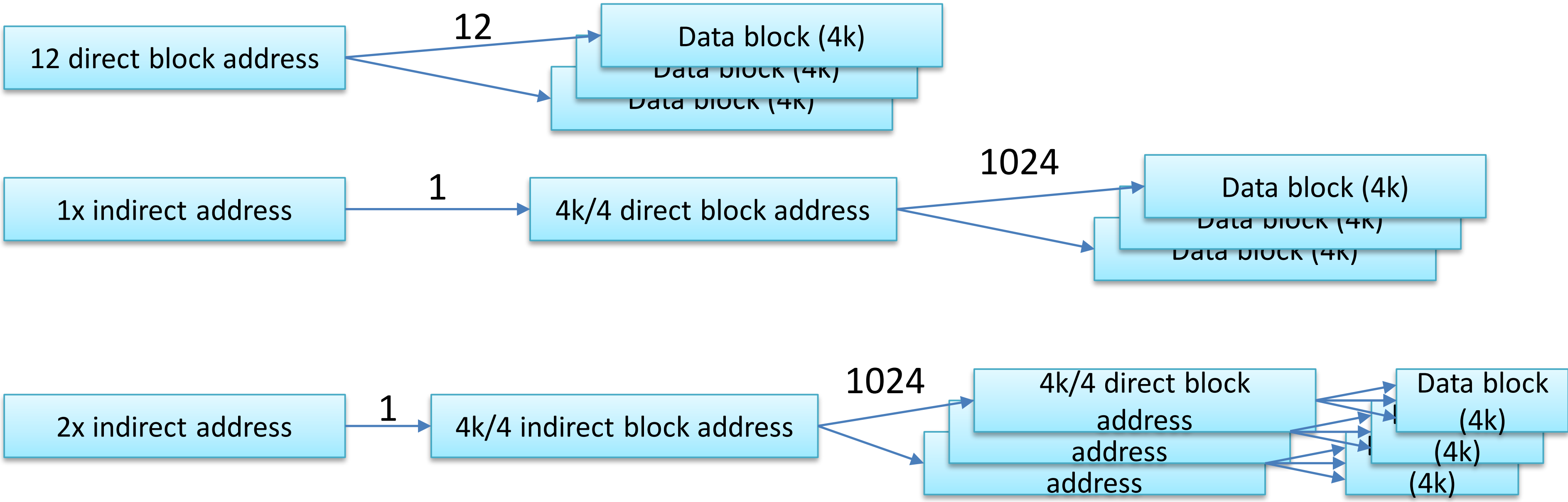
- **In memory** – runtime extensions
  - The contents of the open file object – which is created by the `open()` system call
    - State (locked, modified, etc.)
    - Disk/file system identifier
    - Reference counter (file descriptors)
    - Mounting point descriptor

Storing data blocks (allocation methods)

- It would be simple to store the blocks (files) continuously on the disk…
  - But when files are deleted, different sized „holes” are created – like memory fragmentation
  - With many small holes, storing large files are impossible
- Chained list allocation (sequential access storage)
  - The file data is stored in smaller parts
  - The specific parts are linked to the next part
  - Simple chained list
    - The address of the first part is in the metadata
    - Every part contains the address of the next part
    - The parts can be located anywhere – slow to access the umpteenth part
  - Efficient for sequential access, sensitive to errors
  - Multiple variants, e.g.: FAT
- Indexed storage (direct access storage)
  - The file data are stored in equal sized block (determined by the FS or the HW)
  - The location/map of the blocks: the index
  - If it’s possible, the blocks are located in a sequential order (it can accessed in sequential or direct way)
  - If the index is too big, it can be stored in multiple blocks with the chained list allocation

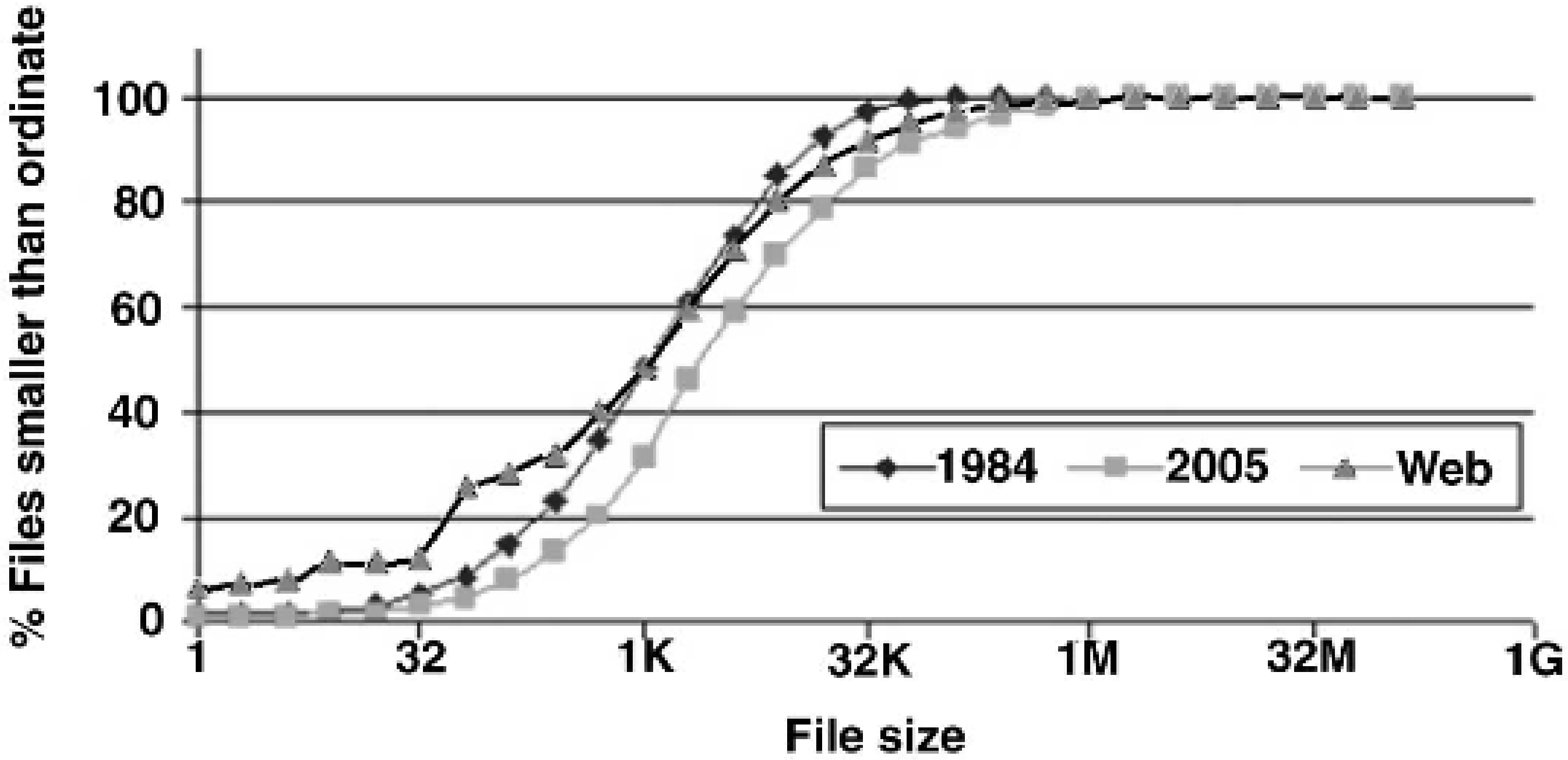
Example: Multiple indexed data block address table

- Address table for a file
  - 12 direct block address
  - Single and double indirect block address
  - 4 kB block size
  - 4 byte address



What is the maximal file size?

How to determine the block size?



Source: Andrew S. Tanenbaum, Jorrit N. Herder, Herbert Bos
File size distribution on UNIX systems: then and now. Operating Systems Review 40(1): 100-104 (2006)

Managing the free blocks

- Registering free blocks for new allocations
- Bitmap, bit-vector description
  - Every block is represented by a bit
  - 1=free, 0=used
  - Simple method, easy to find a free block
    - The map can be stored in the memory for smaller FS
    - Typically there is a CPU instruction for getting the first non zero bit location
  - It uses more memory for a larger file system
- Chained list storage
  - The free blocks are marked and the address of the next free block is written there
  - Only the address of the first free block has to be stored
  - Simple, but not so efficient method
  - It can be combined with the chained list block allocation method
- Hierarchical methods
  - Managing the group of (free) blocks
  - The groups can be created based on the size of the FS
  - Within a group, a simpler structure can be used (e.g.: bitmap)

Accelerating data access

- Recap: the virtual memory management (VM) extends the memory with the disk
- From the opposite side: load the file system data to the physical memory
  - To accelerate the access to frequently used data
  - This is called **disk buffering**
  - The frames which are used for this is called **buffer cache** (see `free` Linux command)
- The organization of the buffer cache
  - Basic idea: the VM and the FS can use the same mechanisms
    - Virtual addresses makes it simple
    - The data is loaded into frames by the VM mechanism
    - This can be beneficial for mmap also
    - This is called the unified buffer cache (Linux: page cache)
  - Accelerating the reads: read ahead
  - Deleting buffered blocks from the RAM: the standard page replacement algorithms do it
  - Managing write operations (when to write the modified data to the disk)
    - Write through cache: it writes immediately (slow)
    - Buffered write: it writes the data periodically (flush, sync) (faster)

Consistency of metadata and journaling file systems

- Disc buffering may introduce consistency problems
  - It can cause file data loss also, but the inconsistency in the metadata can lead to larger scale data loss (storage leak)
  - Solutions
    - Write through cache can solve the problem, the price is the slower operation
    - Use it only for the metadata
- Journaling file systems
  - The changes are saved to a journal, which is always stored on the disk
    - The operations on the metadata is grouped into transactions
    - The transaction is finished when the data is also stored in the journal (commit)
    - The journal is sequential access circular buffer
  - If the operation is performed on the file system, the journal entry can be deleted
  - What happed if the system crashes? At the re-boot the journal is processed
- Log-structured file system: the FS is the log (e.g. BSD LFS)
  - The data and metadata are written sequentially to a circular buffer (log)
- Copy-on-write file system (ZFS, btrfs)
  - The write operation is performed on a copy of the original data, then the metadata is updated

Overview of the topic

- User interfaces
  - User
  - Administrator
  - Programmer
- Operation of the file systems
  - Kernel data structures
  - File system interfaces
  - Data arranged in blocks on disks
- Storing the data
  - Physical storages (HDD, SSD)
  - I/O scheduling
  - Local storage system virtualization (RAID, LVM)
  - Network and distributed file systems