Operating Systems

Slava Sidorov

23/03/2022

Computer resources

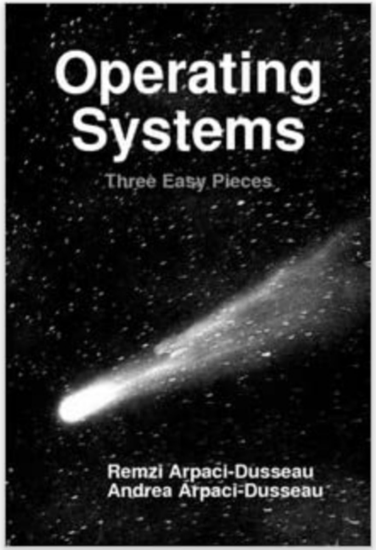
1. CPU (covered by Chris and Marc).
2. Memory and storage (covered by Mike).
3. Input / output devices, including network.

What is operating system?

Operating system is a software that meets the following needs:

- Resources need to be made easily usable **(virtualized)** and **securely allocated** to users' software.

- Users need to access software and system parameters via a **user interface**.



A great free textbook about operating systems!

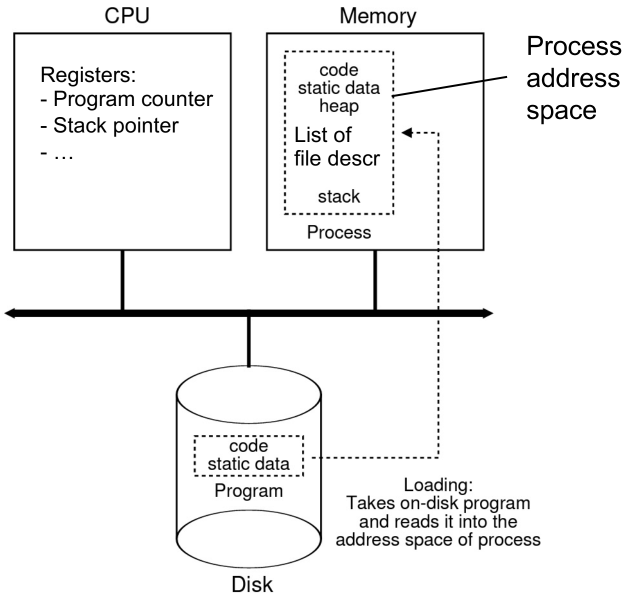
https://pages.cs.wisc.edu/~remzi/OSTEP/

Examples and details are from UNIX-like systems.

CPU

How to run many programs simultaneously on one or just a handful of CPUs? **Process** abstraction.

Process is a running program, and any program is managed by the OS.



Stack is a part of the process address space where process' local variables, function parameters and return addresses are stored.

'stack overflow'

Heap is dynamically allocated memory explicitly requested by a process from the OS (for example, with `malloc()` in C programs).

In UNIX-like systems, a process usually has 3 default file descriptors (file IDs): `stdin`, `stdout`, `stderr`.

CPU

How do the OS and processes communicate? Via **system calls** and **signals**.

System calls are functions that processes can access to ask the OS do something for them.

Categories of system calls [edit]

System calls can be grouped roughly into six major categories:[12]

1. Process control
   - create process (for example, `fork` on Unix-like systems,
   - terminate process
   - load, execute
   - get/set process attributes
   - wait for time, wait event, signal event
   - allocate and free memory
2. File management
   - create file, delete file
   - open, close
   - read, write, reposition
   - get/set file attributes
3. Device management
   - request device, release device
   - read, write, reposition
   - get/set device attributes
   - logically attach or detach devices
4. Information maintenance
   - get/set total system information (including time, date, computer name, enterprise etc.)
   - get/set process, file, or device metadata (including author, opener, creation time and date, etc.)
5. Communication
   - create, delete communication connection
   - send, receive messages
   - transfer status information
   - attach or detach remote devices
6. Protection
   - get/set file permissions

https://en.wikipedia.org/wiki/System_call

CPU

How do the OS and processes communicate? Via **system calls** and **signals**.

Signals are messages / directives to running processes (for example, to pause or die).
They may also be sent via system calls ☺

Examples:

`kill -9 <pid>` or `Ctrl+C` to kill a process;

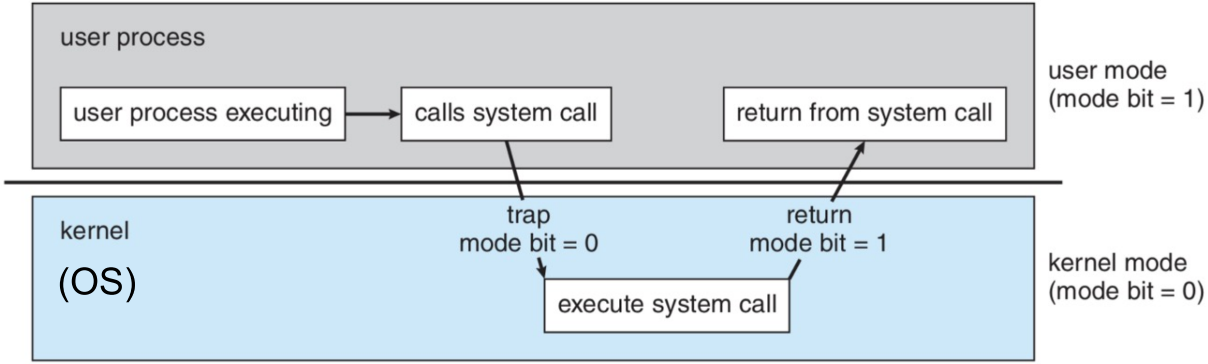
`Ctrl+Z` to suspend a process and the `fg` command to resume it.

The main system calls related to process management:

- Create a process (command in shell, click on an application icon, `fork()` / `exec()` in a program)
- Destroy a process (for example, `kill -9 <pid>` or `Ctrl+C`)
- Wait for a process to finish
- Suspend / resume a process

CPU

How are system calls performed? By the OS in the **kernel mode** of the *CPU*.

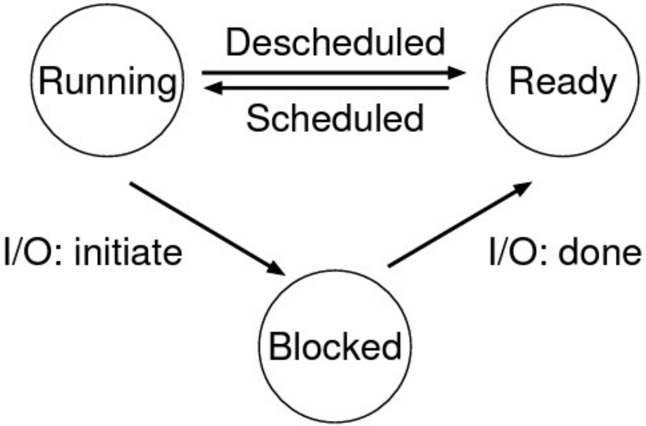


Adapted from https://medium.com/@er.samirshah/dual-mode-multimode-operation-bcdc93077f7a

CPU

How does OS manage processes? Mechanisms: **process states** and **context switch**

Process states (extremely simplified!)



Possible additional states:

- **Initial** – when a process is being created.
- **Final** – when a process has stopped but has not been cleared up yet.

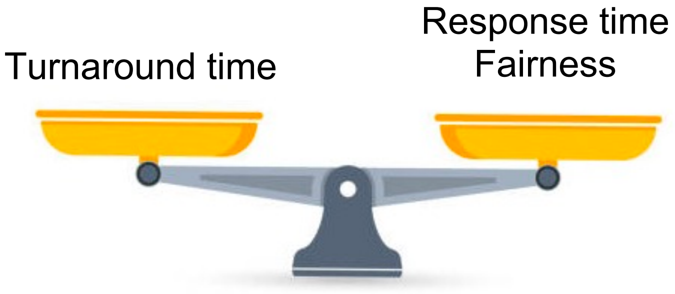
- OS shares the CPU(s) between many processes by **scheduling** and **descheduling** processes.
- When one process is descheduled or blocked, another can be scheduled via the **context switch**: The OS saves registers of the process that was descheduled into memory and loads the registers of the process that it is going to schedule.
- OS gains back the control of the CPU when a process does a system call or initiates an I/O operation, or by an **interrupt timer** set in the hardware.

CPU

How does OS manage processes? **Scheduling metrics**.

Some metrics to optimize for when scheduling processes:

- **Turnaround time** = T(completion) – T(arrival)
- **Response time** = T(first run) – T(arrival)
- **Fairness** – are all processes given equal chances to run?



CPU

How does OS manage processes? **Schedulers**.

MLFQ (Multi-Level Feedback Queue): Move processes in a hierarchy of queues.

- **Rule 1:** If Priority(A) $>$ Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) $=$ Priority(B), A & B run in round-robin fashion using the time slice (quantum length) of the given queue.
- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.

Versions of MLFQ are used in BSD, Solaris, Windows. Linux uses **CFS** (Completely Fair Scheduler).

CPU

Summary:

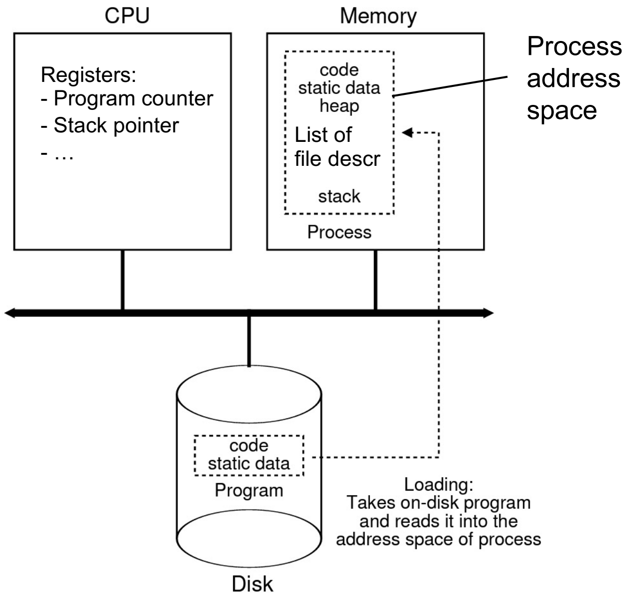
- OS virtualizes CPU for programs by creating an *abstraction* of an unlimited number of available CPUs and of parallel program execution on the same CPU (core).

- OS gives processes a secure *abstract* interface to hardware and other processes via system calls.

Memory

How to share physical memory between many processes but make it easily usable for each of them?

Via the **address space** and **virtual addresses**.



Virtual address is any address in memory generated by a process. All virtual addresses must lie in the process' **address space**.

It is **translated** into **physical** (real) **address** by the **MMU (memory management unit)** of the CPU.

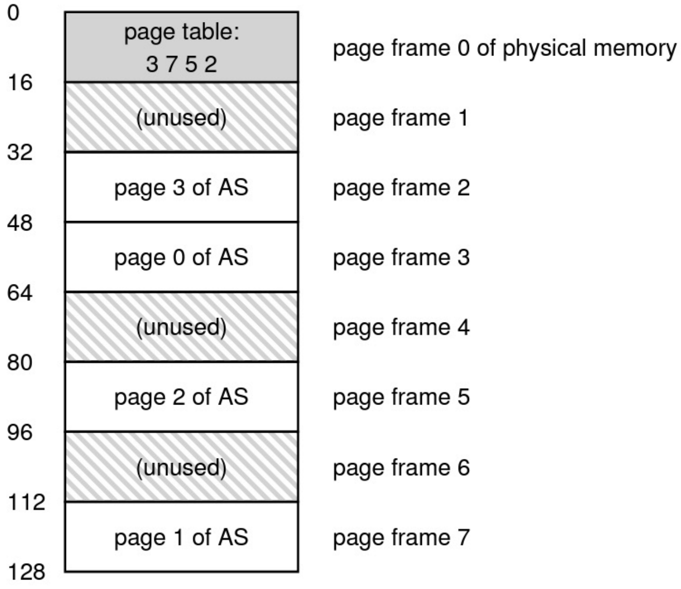
```
physical address = virtual address + base
```

OS Requirements	Notes
Memory management	*Need to allocate memory for new processes; Reclaim memory from terminated processes; Generally manage memory via **free list***
Base/bounds management	*Must set base/bounds properly upon context switch*
Exception handling	*Code to run when exceptions arise; likely action is to terminate offending process*

Memory

How to share physical memory between many processes *efficiently*?

Via **paging: A virtual page** is a fixed-sized part of a process' address space.



Physical memory is hence divided into **page frames**, each of which can accommodate one **virtual page**.

A **segmentation fault** happens when a process tries to access memory that is not allocated to it.

The usual policy in this case is to kill the process.

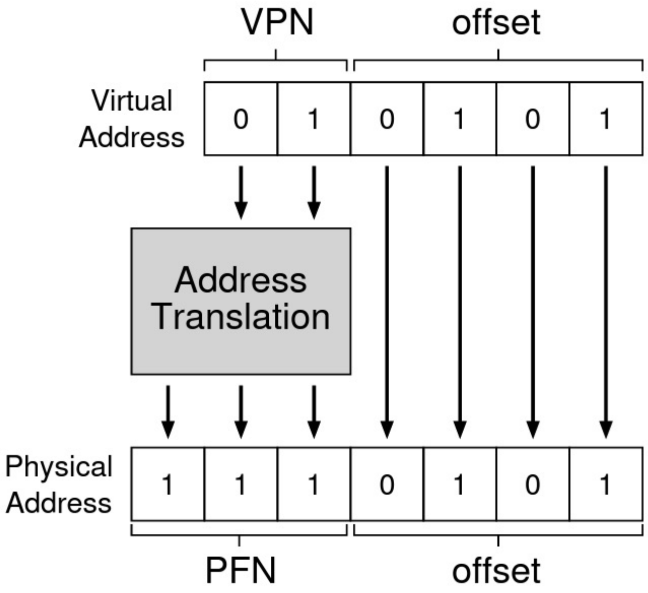
Highly optimized **page tables** describe the correspondence between virtual and physical pages.

TLB (translation-lookaside buffer) contains most frequent address translations.

Memory

How to share physical memory between many processes *efficiently*?

Via **paging: A virtual page** is a fixed-sized part of a process' address space.



Physical memory is hence divided into **page frames**, each of which can accommodate one **virtual page**.

A **segmentation fault** happens when a process tries to access memory that is not allocated to it.

The usual policy in this case is to kill the process.

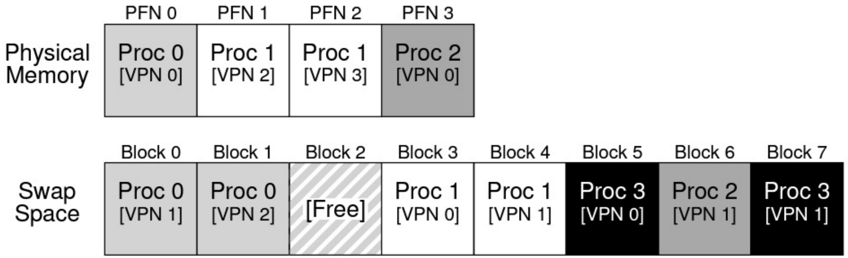
Highly optimized **page tables** describe the correspondence between virtual and physical pages.

TLB (translation-lookaside buffer) contains most frequent address translations.

Memory

How to free up memory without killing processes?
How to support many large address spaces in a limited physical memory?

Via **swapping:** OS **swaps** (copies) pages from physical memory to the disk and back.



- OS swaps pages to the disk to make space in the physical memory.
- **Page fault** occurs when a process tries to access a page which in currently on the disk.
- OS **handles page faults** by blocking the process, swapping the page back to RAM and resuming the process.
- While the process is blocked another process is running!

Memory

How to free up memory without killing processes?
How to support many large address spaces in a limited physical memory?

Via **swapping:** OS **swaps** (copies) pages from physical memory to the disk and back.

- Which pages to swap to disk (**page out**)? Policy: (Number of page faults) → min.
- Swapping is slow! ⟹ Optimizations: (a) **prefetching** of pages; (b) **grouping of writes** to the disk.
- **Thrashing** is constant swapping of pages to disk in order to swap other pages to back to RAM.
- OS may use **out-of-memory (OOM) killer** to kill memory-heavy processes in case of thrashing.

Memory

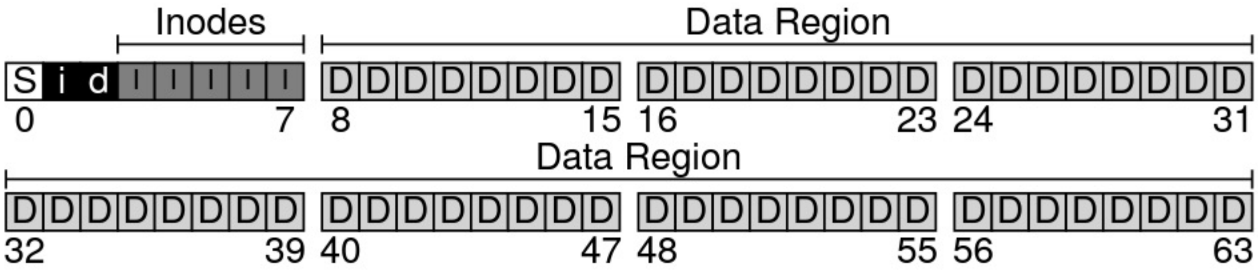
Summary:

- OS virtualizes memory for programs by creating an *abstraction* of a large private address space for each process while sharing physical memory between many processes and swapping data between the RAM and disk.

Storage

How does OS manage long-term storage ("disk")? Via **file systems**.

File system is an on-disk structure plus a **driver** in OS.



- The disk is split into **blocks** of an equal size.
- Most blocks store user data, but some are reserved for the file system own data structures.
- **Inodes** ("index nodes") store information about each file and its data blocks.
- **Allocation structures** (for example, **bitmaps**) indicate which inodes (i) and data blocks (d) are allocated.
- A **superblock** stores information about the file system itself: its type, number of inodes and data blocks, etc.

Storage

How does OS manage long-term storage (“disk”)? Via **file systems**.

A simplistic **inode**:

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists

Figure 40.1: **Simplified Ext2 Inode**

Storage

How does OS manage long-term storage ("disk")? Via **file systems**.

- **Directory** is stored on disk as a list of tuples `<inode, filename, any_additional_meta>`.
- Directories are often just a type of file!
- To delete a file means to delete a link to it, not the contents.

Storage

How does OS manage long-term storage (“disk”)? Via **file systems**.

Many file systems exist, for example:

- To access the local disk: **EXT3**, **EXT4**, **XFS**, **NTFS**, …
- For network access: **NFS**, **SMBFS**, …
- For “indefinitely scalable storages”: **ZFS**

Virtual file system:

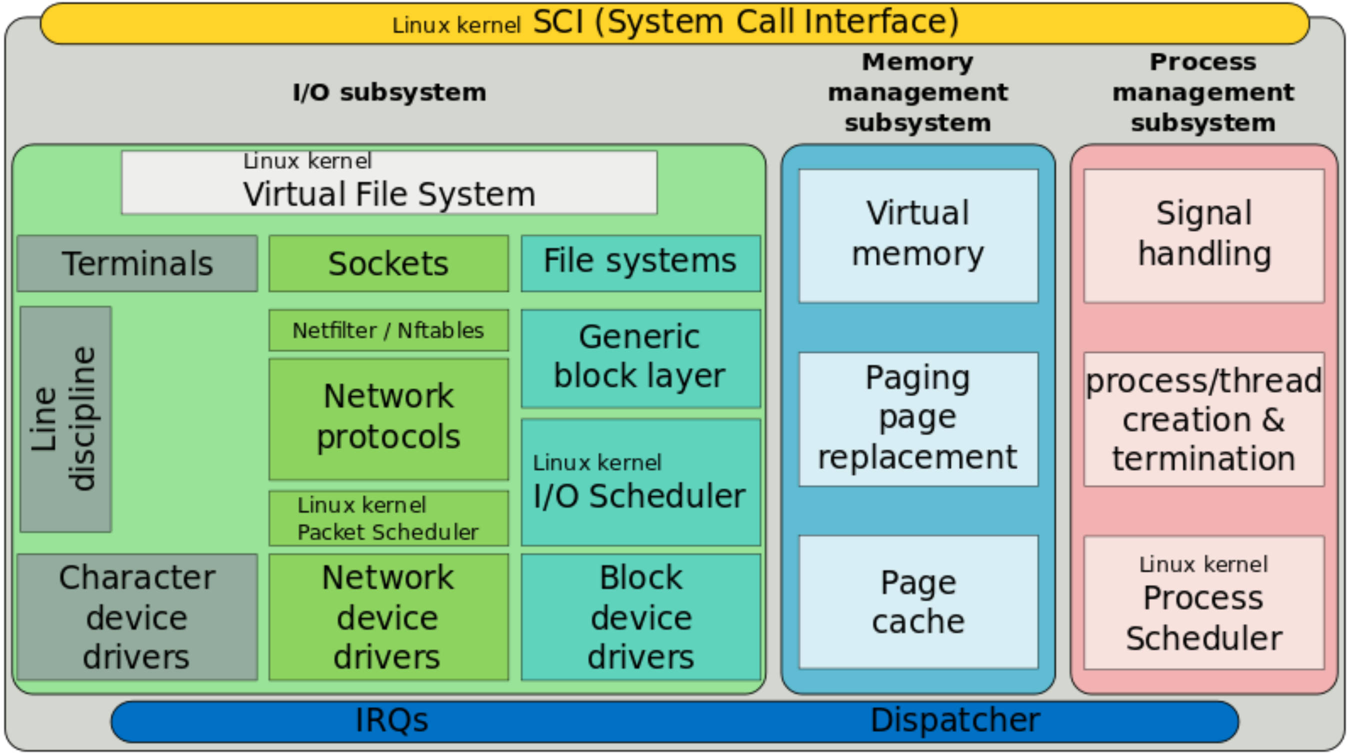
- Combine (**mount**) different file systems on one machine.
- Possibly, represent devices and some internal structures as files too.

Storage

Summary:

- OS virtualizes storage for programs by creating an *abstraction* of a tree of directories and files and by hiding the actual model of the storage (of a hard drive / SSD / …) behind the file system driver, virtual file system and system calls.

Linux kernel: Combining pieces together



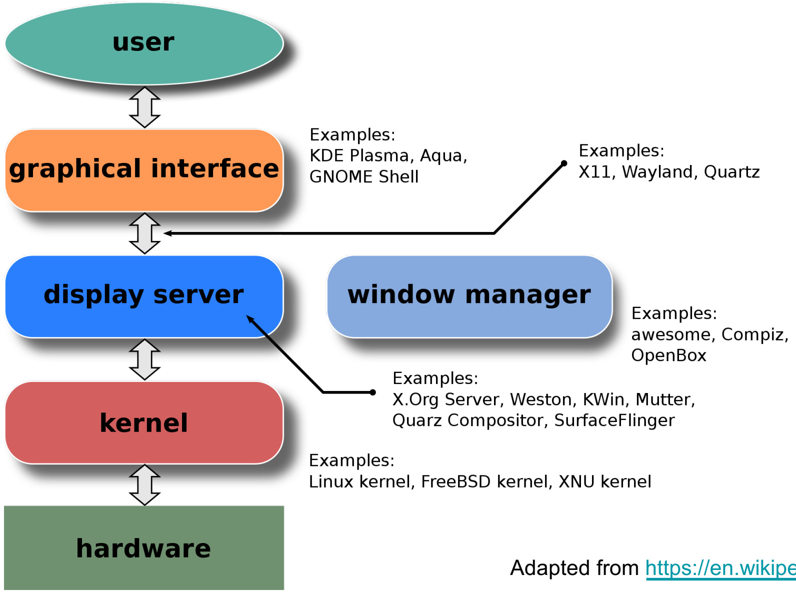
https://de.m.wikipedia.org/wiki/Datei:Simplified_Structure_of_the_Linux_Kernel.svg

There is much more to operating systems than this!

- **Multithreading** and mechanisms to control **concurrency**.

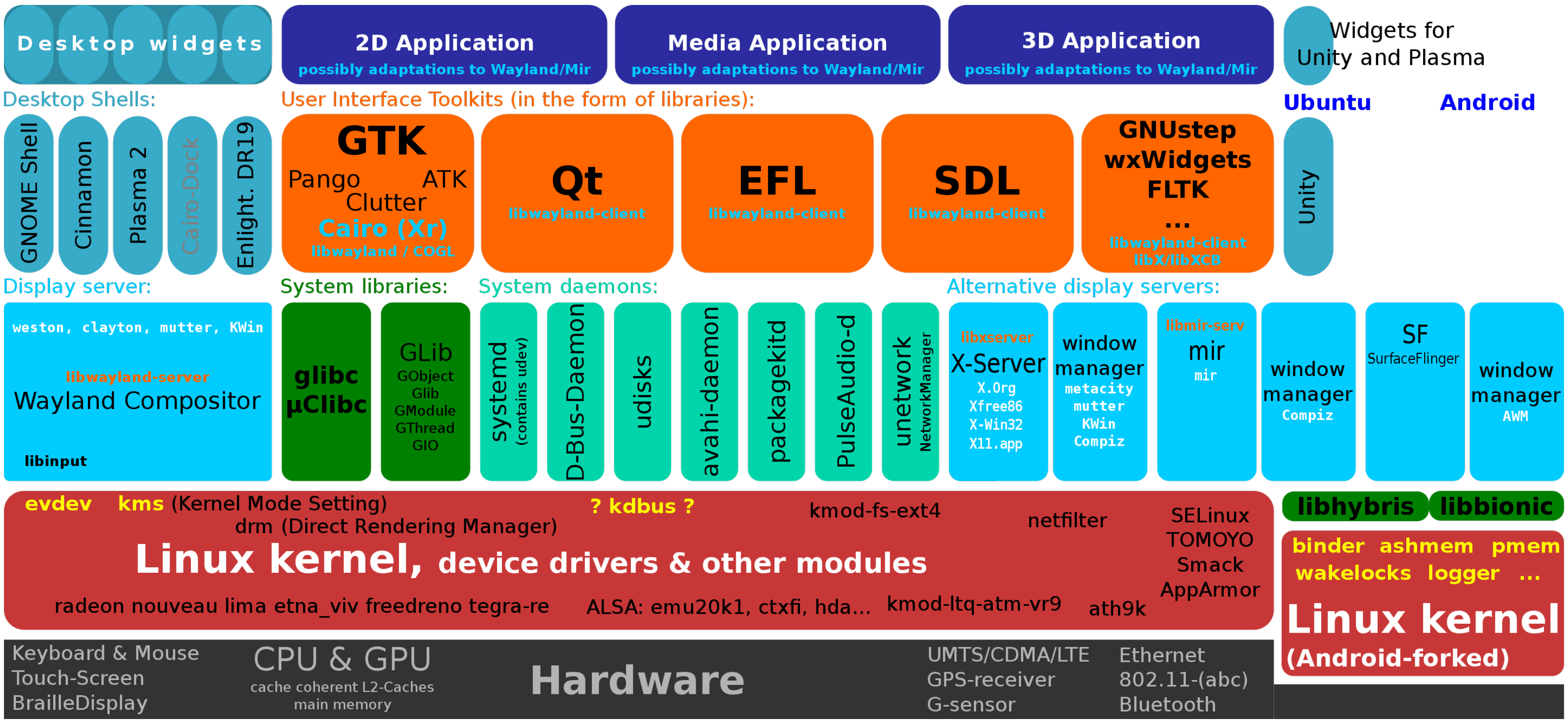
There is much more to operating systems than this!

- **Multithreading** and mechanisms to control **concurrency**.
- **Command line**, **display server** and **graphical interface**:



Adapted from https://en.wikipedia.org/wiki/Windowing_system#Display_server

GNU/Linux OS: Possible media components above the kernel



https://en.wikipedia.org/wiki/Udev

There is much more to operating systems than this!

- **Multithreading** and mechanisms to control **concurrency**.
- **Command line**, **display server** and **graphical interface**.
- Managing all the **periphery** (including **network connections**):

There is much more to operating systems than this!

- **Multithreading** and mechanisms to control **concurrency**.
- **Command line**, **display server** and **graphical interface**.
- Managing all the **periphery** (including **network connections**).
- Managing **users** and system **security**.

GNU/Linux distributions

GNU/Linux distribution = Linux kernel + GNU tools and libs + applications + package manager + display server + window manager + desktop environment + shell + docs.

Some general-purpose distributions:
Debian, **Ubuntu**, **Arch**, openSUSE (**Leap** & **Tumbleweed**), **Fedora**, **Mint**, …

Some specialized distributions:

- **Gentoo** – machine-specific optimization.
- **Rocks** – for high-performance computer clusters.
- **OpenWrt** – for embedded devices, such as network routers and smartphones.

Phylogenetic tree of a plethora of GNU/Linux distributions:
https://en.wikipedia.org/wiki/Linux_distribution#/media/File:Linux_Distribution_Timeline_21_10_2021.svg

Different kinds of operating systems

- **Real-time operating systems** (often, for embedded devices): Meet stringent time constraints.

  Examples: **ThreadX** (Microsoft), **FreeRTOS** (currently developed by Amazon).

- **Distributed operating systems:** Unite a set of computers into an illusion of one big computer.

  Examples: **Plan9**, **Inferno**.