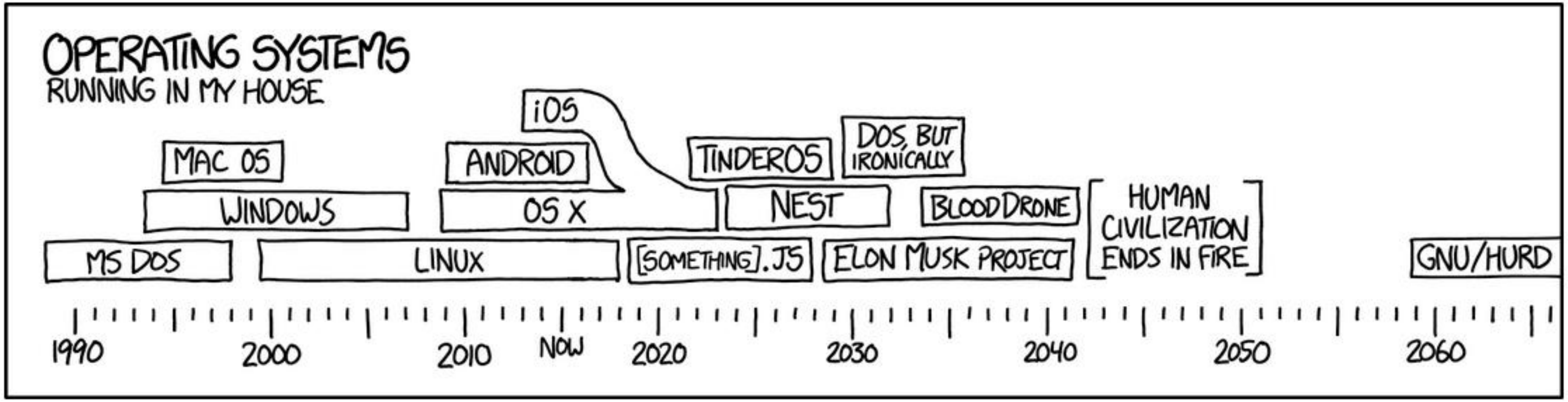
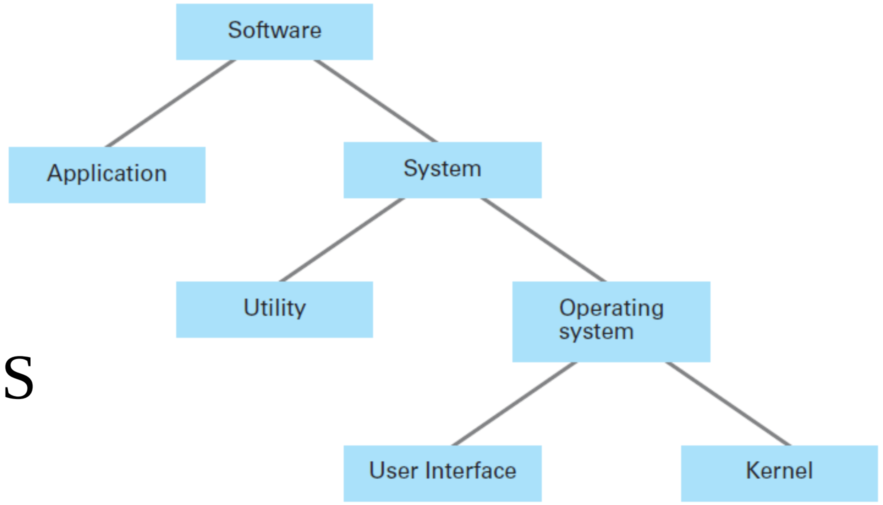
6. Operating system: tasks, process control and scheduling



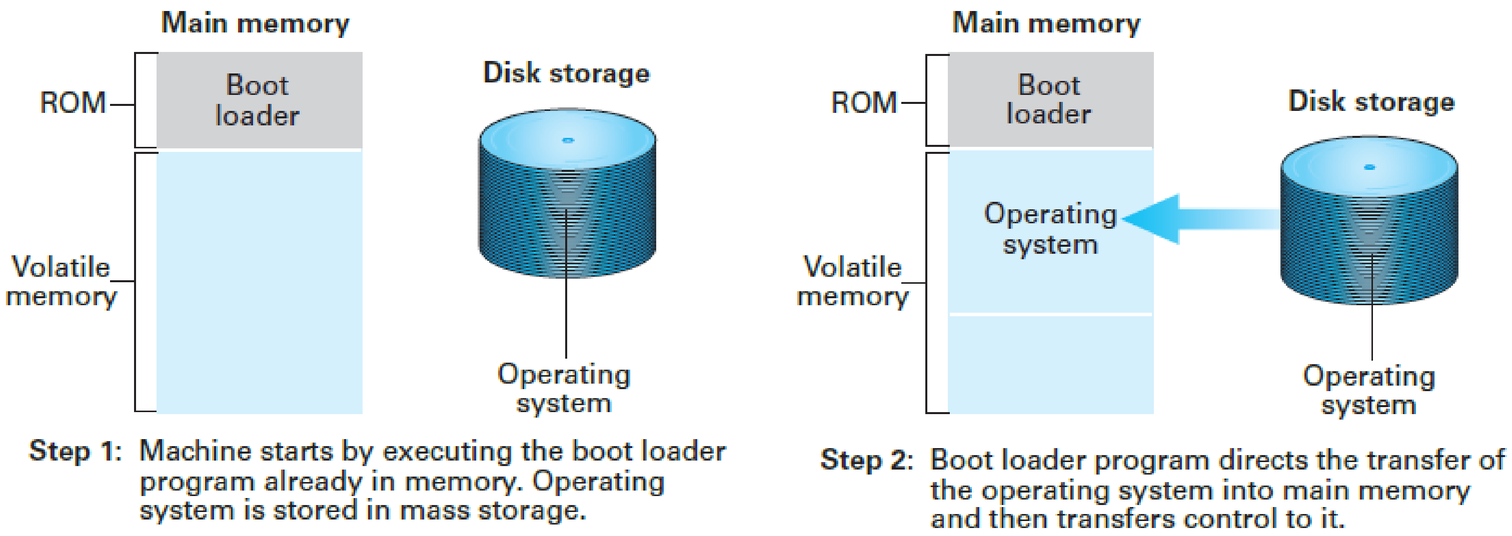
Types of software

- Computer software can be divided into two categories: system software and application software
- Application software = "what the computer is used for"
  - Office programs (text processing, spreadsheets)
  - Calculation and analysis programs
  - Production control programs etc.
- System software provides the infrastructure for these
  - Operating system (OS) and utility software
- Operating system controls the use of resources
  - Kernel contains the basic functions of OS
  - Users communicate with kernel through user interface
- Utility software extends/customizes the features of OS
  - Defragmentation tools, network communications etc.



Booting

- Each time when the computer is started, it has to be able to start the OS
- Starting the OS is known as booting
  - When a PC is started, BIOS checks the system and then initiates the boot loader
  - Boot loader starts and loads the OS from disk storage to RAM
  - Strictly speaking, boot loader is not in ROM – it can be altered under special circumstances



Operating system

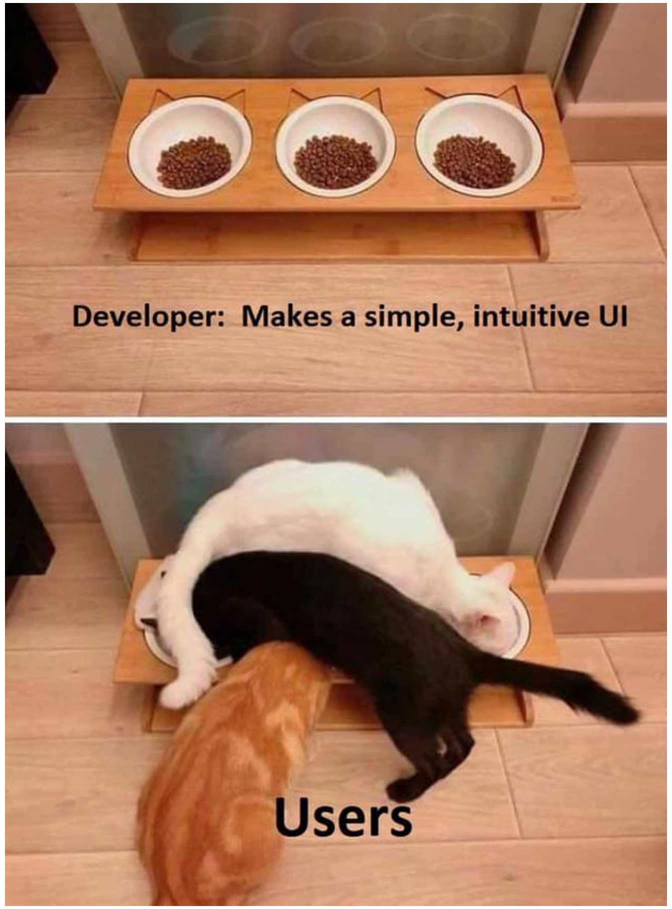
- Basically all computers are delivered with some kind of an operating system
  - DOS, Windows, macOS, Linux, Chrome OS…
- Operating system is the software used for controlling the hardware of the computer as well as execution of application software
  - Application software is run via the operating system
- Executing a program under control of an OS is called a *process*
- Processes are coordinated by controlling the allocation of resources:
  - CPU time scheduling
  - Memory allocations in different level memories
- Input/output (I/O) is controlled via handling I/O requests and interrupts

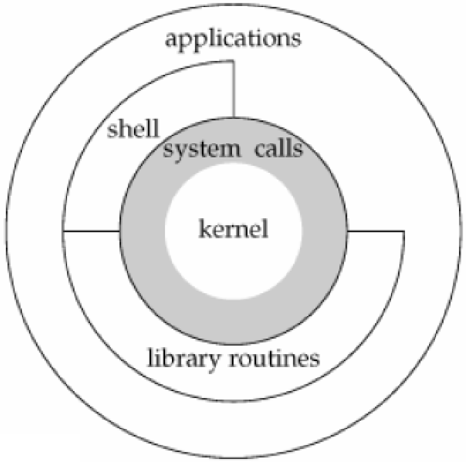
Operating system tasks

- Resource allocation
  - Memory, processor cycles, I/O devices
- Dispatching
  - Exchanging the process currently running in CPU
- Scheduling
  - Keeping track of processes (in queue or in execution) via process table
  - Decision on which process will be selected for execution next
  - Multiple possible decision criteria for selection (known as scheduling methods)
- Resource protection
  - Making sure that a process can’t access a resource it hasn’t reserved
- Interrupt handling
- Handling of I/O requests

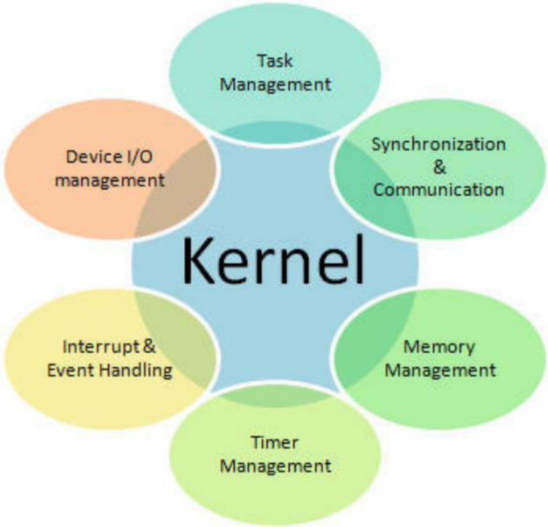
Layers of an operating system

- Kernel is a program set that contains the basic functions
  - Kernel instructions are run in privileged state
- Shell & library routines, or user interface (UI) in general
  - Intermediary between users and the kernel
  - Shell is an old-fashioned, text-based UI
    - Extended by library routines
  - Nowadays the vast majority of operating systems use some kind of Graphical User Interface (GUI)

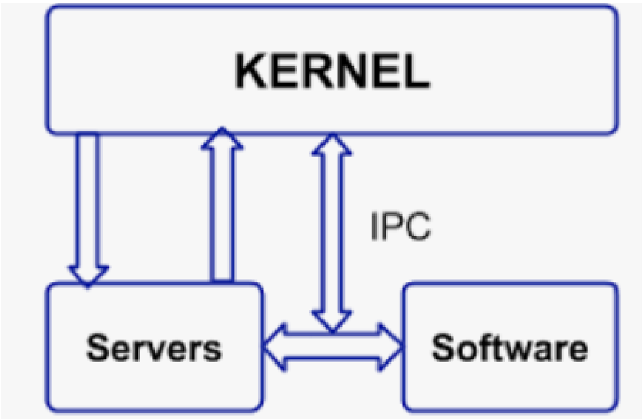


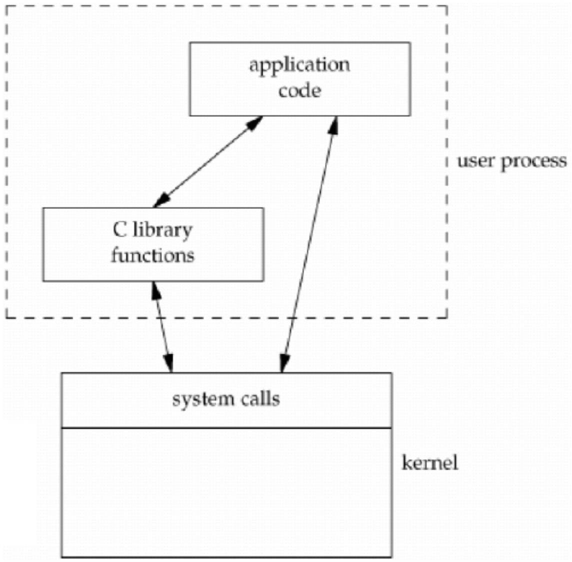


Tasks of a kernel



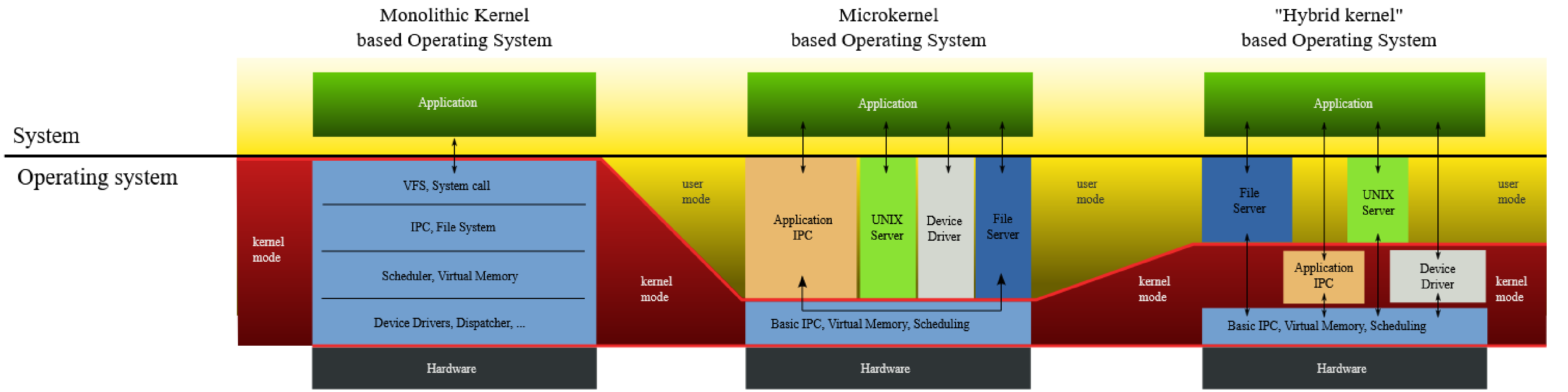
- System call interface
- Process control
  - Creating and removing processes
  - Scheduling between processes
  - Conveying messages between processes (Inter-Process Communications, IPC)
  - Memory control (allocation)
- I/O control
  - File system
  - Buffering
  - Device management





Kernel architectures

- There are two basic kernel architectures: monolithic and microkernel
  - In monolithic kernel design, basically the whole OS is placed in kernel
  - In microkernel, the kernel only contains the bare minimum
- Hybrid kernels combine features from both



(Excellent illustration by Golftheman from Wikipedia)

Microkernel architecture

- Only most essential functions to kernel
- Instructions started by kernel are executed in privileged state
  - Initiation of interrupt handling (what caused the interrupt?)
  - Dispatching functions (usually just copying registers)
  - Memory control functions (memory control unit settings, protection)
  - Inter-process communications (conveying requests, copying data)
  - I/O functions (use of disk drives)
- Other OS instructions are normal processes, which are executed in user state
  - Device drivers, file system
- Benefits of microkernel architecture:
  - Modularity and flexibility of OS (adding new modules requires no changes to kernel)
  - Stability and reliability are easier to attain (user state processes can’t crash the computer)

Comparison and use of kernel architectures

- Monolithic kernels have better performance, because communication with applications and devices are done using system calls
  - Downside is that system stability is harder to reach (for example, one bad device driver can cause the whole computer to crash)
  - Also, adding new features is limited, because it always requires changes to the kernel
  - Examples: Linux, DOS, Windows 9x
- Microkernels offer better stability and security, but performance is worse due to slower communication method (message passing)
  - Popular choice in small devices due to minimal size
  - Examples: Horizon (Nintendo Switch), L4 (embedded systems)
- Most modern OS use a hybrid-type kernel
  - Examples: Windows 7/10, macOS

Processes

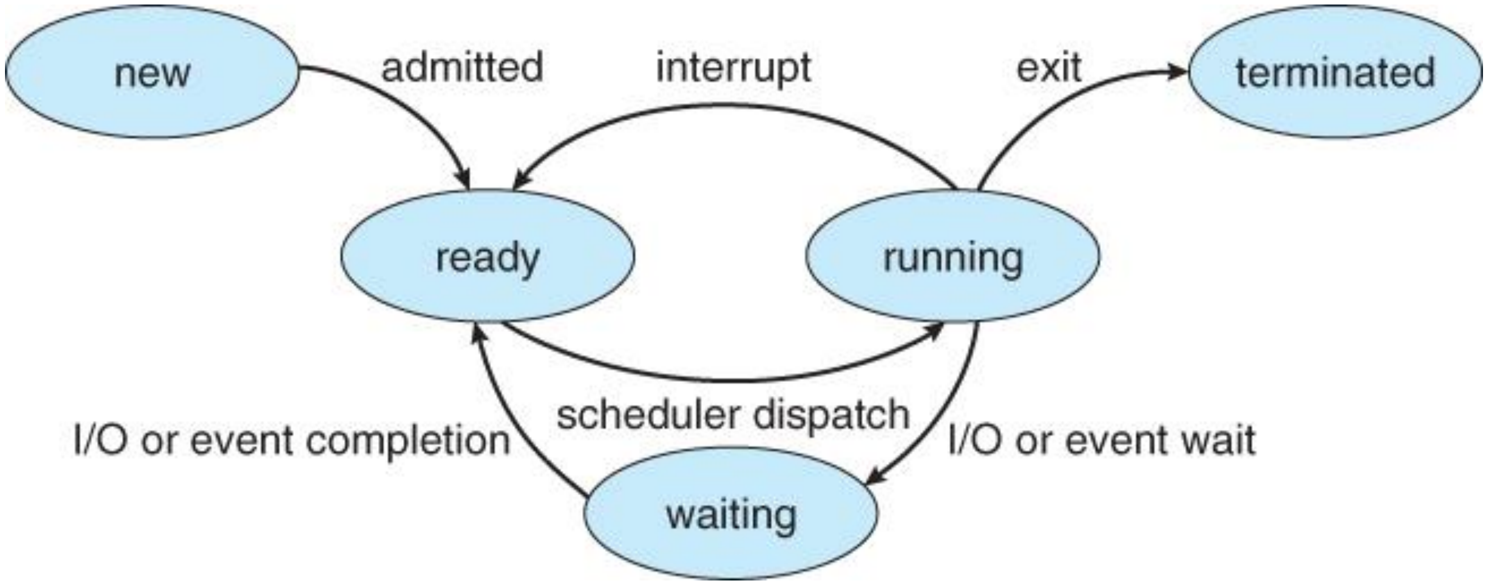
- Program = a series of commands, stored on hard disk
- Process = execution of a selected program in selected environment
  - Analogy: sheet music of a song vs. a song performed by an artist on a gig
- Process state = current (momentary) state of execution
  - Ready / running / waiting for resources
- In a computer, there are usually dozens of processes going on
  - Some initiated by the user, some by the OS
- Processor time and other resources must be divided between processes
  - Rapid switching of running process creates the user a sense of multitasking, even though there is always only one process (per core) actually running

Dispatching

- Single task of a program can be comprised of several processes
- Only one process can be in running state per processor core, so the CPU resources (clock cycles) have to be dealt between processes
- Dispatcher picks up the selected process from process queue, moves it to running state and gives it a permission to use the CPU
  - Analogy: TV singing contest – processes are contestants, dispatcher is the production assistant who fetches the next contestant from backstage and gives him/her the tools needed (mic etc.)
- Dispatching can be pre-emptive (process halted when still running) or not pre-emptive (process is changed only after the previous one has terminated)
- Dispatcher is activated when
  - Currently running process terminates or reaches the time limit it was given
  - Currently running process performs an I/O request

Process states

- Because I/O requests take some time to complete, a process performing such a request is put on waiting mode
- After I/O request is responded to, the state of the process is changed to ready
- Dispatcher takes ready processes from queue to running when scheduler sees fit



Scheduling of resources

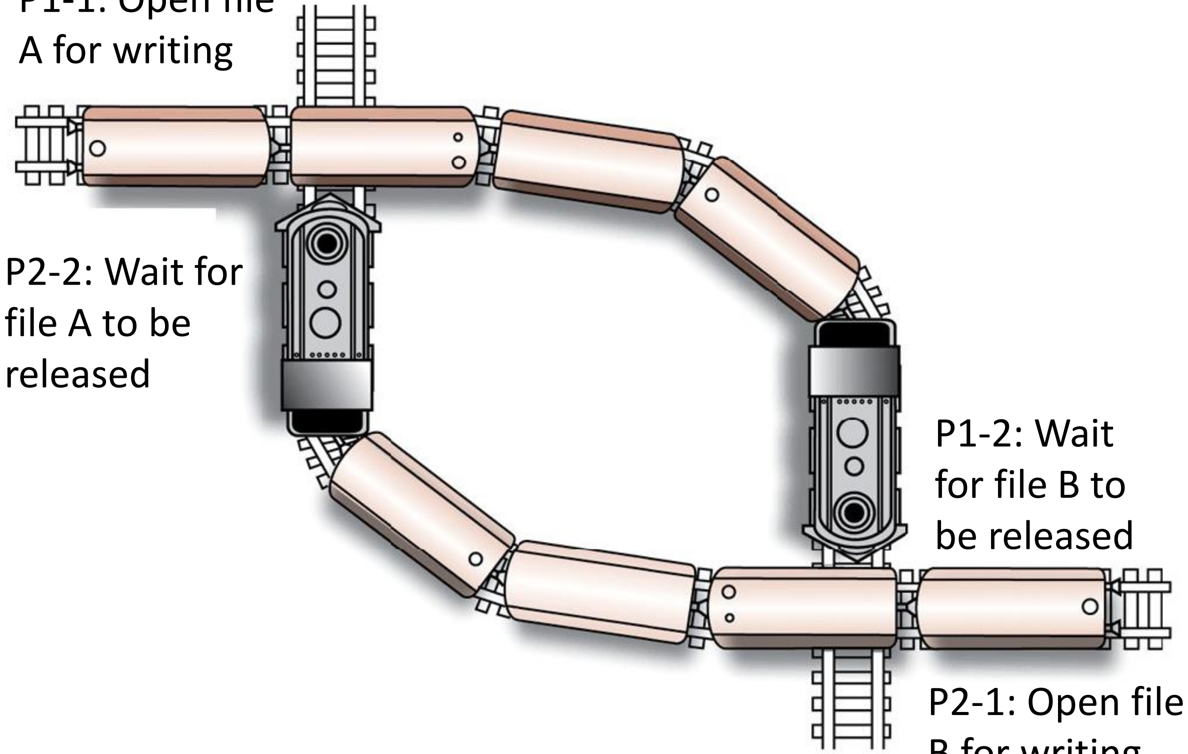
- The order in which queued processes are taken to running state is specified by the scheduler
  - TV singing contest analogy: broadcast director is the scheduler (decides who goes on stage next)
- Scheduler has to take into account:
  - Resources needed
  - Resources currently available (free)
  - Priority level of task
  - Expected waiting time before execution
- Allocation of resources can be done in static or dynamic fashion:
  - Static: all resources ready before process starts to run
  - Dynamic: resources will be reserved during the run
- Dynamic allocation can lead to a deadlock situation

Deadlock

- In a deadlock, two processes are waiting for resources that are reserved for each other
- Deadlock can occur, if the following conditions are satisfied:
  - There is competition for non-sharable resources.
  - The resources are requested on a partial basis (so, not all at once)
  - Once a resource has been allocated, it can’t be forcibly retrieved
- Prevention: kill commands, spooling

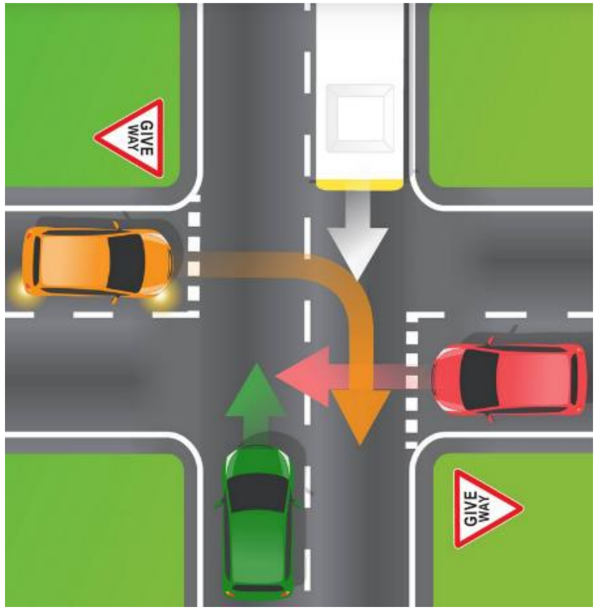
Example:

- Process 1 tries to copy contents of B to A
- Process 2 tries to copy contents of A to B
- Scheduler deals CPU cycles in order P1, P2, P1, P2



Starvation

- Another possible problem that might occur when dealing resources is starvation
- This means a situation, where some process is constantly denied necessary resources
- Real-life example: busy major road
  - Cars coming from side roads have to give way
  - Theoretically, if major road traffic is constant, it’s never their turn
- Usually a passing problem, doesn’t continue forever
- Good scheduling algorithms are starvation-free
- Exploited by hackers
  - Some denial-of-service (DoS) attack types aim at starvation

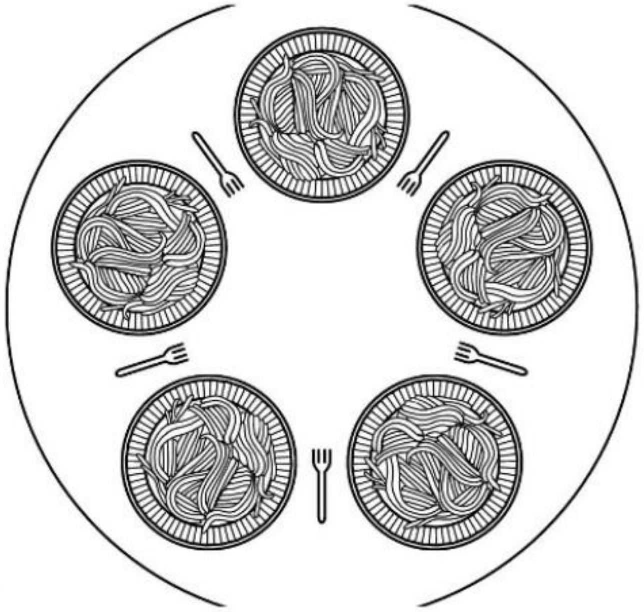


Example: Dining philosophers (Tanenbaum, 2015)

- Premises:
  - A philosopher always either thinks or eats
  - A philosopher needs two forks for eating spaghetti*
  - There are 5 philosophers, each has a plate and one fork
- Eating algorithm: does this work?

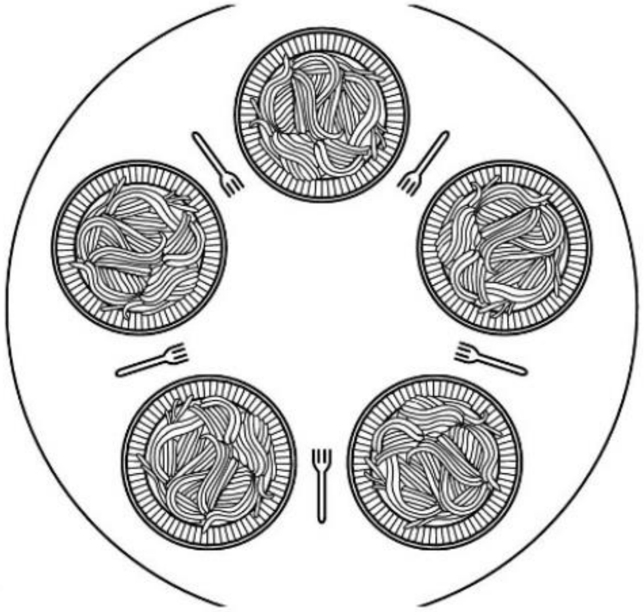
```
#define N 5

void philosopher(int i)              /* i: philosopher number, from 0 to 4 */
{
      while (TRUE) {
            think( );                /* philosopher is thinking */
            take_fork(i);            /* take left fork */
            take_fork((i+1) % N);    /* take right fork; % is modulo operator */
            eat( );                  /* yum-yum, spaghetti */
            put_fork(i);             /* put left fork back on the table */
            put_fork((i+1) % N);     /* put right fork back on the table */
      }
}
```



*Actually, this Tanenbaum's version is a clumsy translation: in original version it was noodles and chopsticks, which makes a lot more sense.

Example: Dining philosophers (Tanenbaum, 2015)



- If the philosophers run the whole algorithm one person at a time (no interrupts), then yes
  - One eats, others just wait; inefficient
- If the philosophers run the algorithm simultaneously – or one instruction at a time for each, this results in deadlock
  - Everybody will sit idle with a left fork in hand
- If the philosophers don’t run the algorithm simultaneously but stagger the starting time by an interval of t, the algorithm works (if t is different for everyone)
  - If t is the same for all, two pairs of philosophers can starve the fifth one, because they will alternate eating turns
- Many ways of solving the problem (Google if you’re interested)

Scheduling algorithms

- As we noticed from the previous example, scheduling resources for processes is not an easy task but contains several pitfalls (mostly regarding efficiency).
- Numerous algorithms have been designed for scheduling:
  - FIFO (First-In-First-Out) or FCFS (First-Come-First-Served)
  - LIFO (Last-In-First-Out)
  - RR (Round Robin) – uses division to time slices (quanta)
  - SJF (Shortest Job First) or SPN (Shortest Process Next)
  - HRRN (Highest Response Ratio Next)
  - Feedback
  - FSS (Fair-Share Scheduling)
- Algorithms differ in efficiency and priority criteria
  - Efficiency can be evaluated using several metrics

LUT University

Scheduling algorithms

- Comparison table of most common scheduling algorithms
- “Efficiency” is relative to application of the computer (what to favor?)

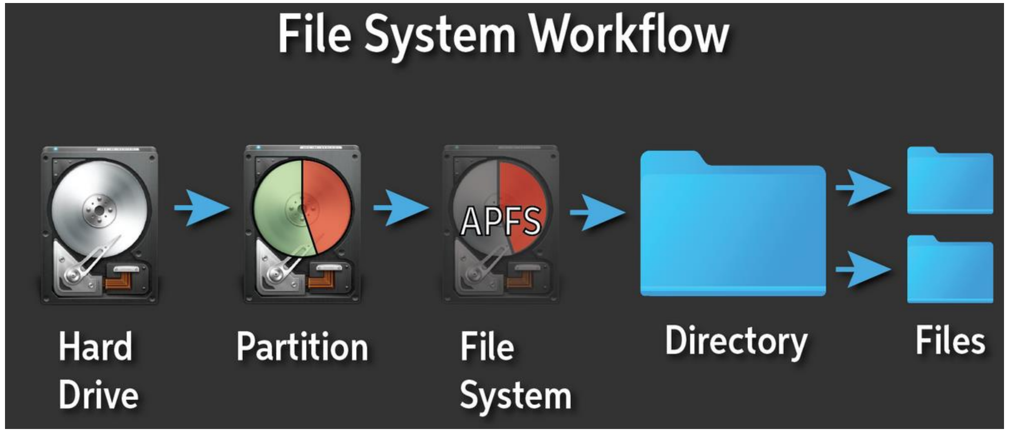
	FCFS	Round robin	SPN	SRT	HRRN	Feedback
Selection function	max[w]	constant	min[s]	min[s – e]		(see text)
Decision mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Through-Put	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Fair-share scheduling

- FSS divides processor resources to time quanta just like Round Robin
- The difference lies in equal division of resources between users instead of processes (different users may have differing number of processes in queue)
- For example, if user X has processes A, B, C and D and user Y has process E in queue
  - RR: A, B, C, D, E, A, B, C, D, E, A, B,…
  - FSS: A, E, B, E, C, E, D, E, A, E,…
- Possibility to grant also different share of resources to users – if in previous example resources are divided in 2:1 share between users X and Y
  - FSS: A, B, E, C, D, E, A, B, E, C, D, E, A,…
- Prevents greedy users from hogging all resources
- Used in Linux (especially due to server use)

File system

- One important task of an OS is to control and maintain the file system
- Hard drive: physical disk where information is “permanently” stored
- Partition: logical section of the hard drive
- File system: method how data on the partition is stored and organized
  - FAT = File Allocation Table; simple and compatible, but limited features
  - NTFS = Newer replacement; supports larger files and enhanced security
  - Several others (APFS, SquashFS, HPFS,…)
- Directory (folder): collection of files
- Path: location of file in folder hierarchy
- File descriptor: unique file identifier
  - “Keys” to use the file

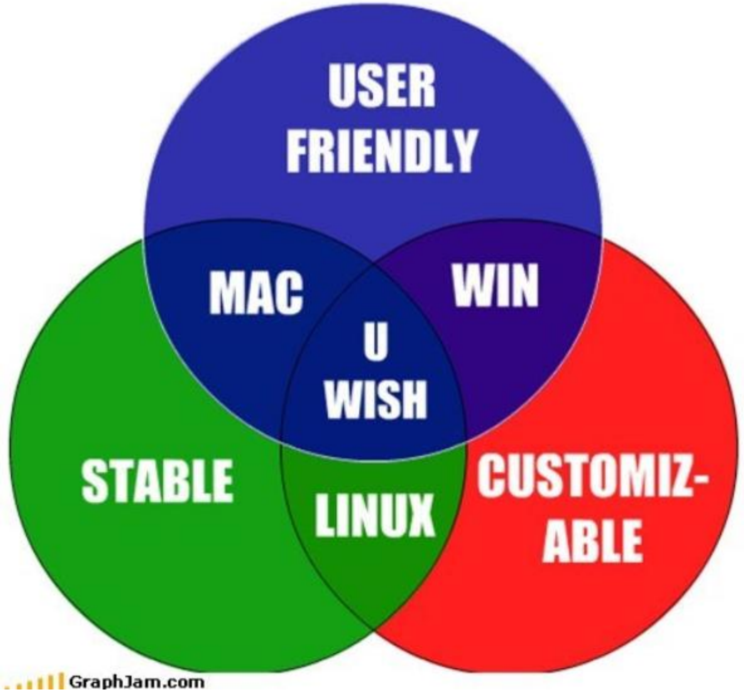


Information security and data protection

- Identification of users
  - Traditionally, only users who know the username and password are allowed access
  - Nowadays also other means of identification (fingerprint, face recognition, retina scanning, mobile identification applications)
- Every process has an owner, and it uses resources only by owner's permission
  - Either some user or OS ("System" in Windows PCs)
- Rights to access resources
  - Files have owners, who specify permissions
  - Only the owner of a file can alter the permissions
- Programs and their data must be sheltered from other programs
  - Especially critical to shelter the OS from applications
- Mutual use of resources must be allowed in certain cases

Desired features of an OS

- Good performance (short response time, high throughput)
- Stability (MTBF, mean time between failures)
- Data protection and security (resistance to data breaches)
- Scalability to different environments
- Extensibility (possibility to add new features easily)
- Portability to multiple devices
- Safety and reliability (low chance of user errors)
- Interactivity (ease of communication with users)
- Usability (user-friendliness)



OS development and administration

- No operating system is "finalized" at any point
- Computers and devices evolve
  - Switches, punch cards, magnetic tapes, disks, solid state memory…
  - From Text User Interface (TUI) to Graphical User Interface (GUI)
  - Massively increased amounts of memory (all types), improved bus speeds
  - Support for virtual memory
  - Increased clock speeds, multiple core processors
- Information processing methods evolve
  - Interactive real-time systems (require massive data transfer speed)
  - Graphical windowing environments
  - Image processing and video editing
  - Local area networks and Internet, dealing with large user amounts
  - Machine learning, Big Data, pattern recognition, neural networks…

OS development and administration

- Due to continuous need for development, prefer
  - Modular structure
  - Clear interfaces between modules
  - Possibly object-based implementation
  - Internal vs. public data (internal data not visible to users)
- All operating systems contain deficiencies and mistakes, so they need to be updated as these get fixed
  - Patches and service packages
  - New OS versions
- Completely new OS, when it's time
  - Need for new module structure
  - Code inefficiency due to multiple patches

Present and the future

- Trends in hardware development
  - Multiprocessor systems, increased use of GPU (Graphics Processing Unit)
  - Fast telecommunications networks
  - Increased number of cores in processors, optimization
  - Improved memory, quicker disk storage (M.2 SSD, thousands of megabytes per second)
- Trends in software usage
  - Customer/server-model; IaaS (Infrastructure as a Service; programs and databases are located on a server leased from a server company)
  - Mining of cryptocurrencies (Proof-of-Work vs. Proof-of-Stake)
  - Streaming services (4K or even 8K quality)
  - Metaverse?
- Mobile OSs and their developer ecosystems
  - Basically a duopoly between Android and iOS – are there no challengers?

Thank you for listening!