PDF 1 & 2

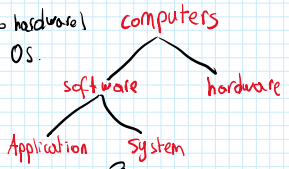
January 20, 2021 6:37 PM

* What is an Operating System?

. Operating system is a system software that manages the computer's hardware. Also considered as the ressource manager of the computer. It acts like a middleman between the user and hardware. Lowest software (closest to hardware)

System software: controls the system. Like compilers, interpreters, utilities and OS.

Application software: Serves a user's purpose. Like word processor, browser, games, etc.



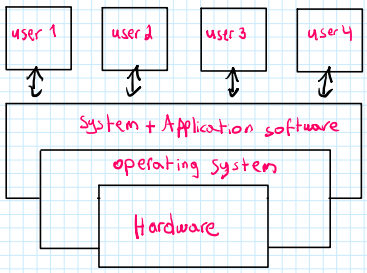
* A computer system is composed of 4 parts:

① Computer hardware: provides basic ressources (CPU, RAM memory, I/O devices, HDD, Network)
Can be viewed as a ressource allocator and a control program of I/O

② Operating system: Controls and coordinates use of hardware among applications, by delivering access to necessary devices

③ Application programs: Determine how ressources can be used to solve the user's problem. Allows the user to create a new product or provides a solution.

④ User: people or other computers. Users don't care about ressources managment or utilization they're only looking for ease of use and good performance



Example: User wants to print or save typed document. Application program (word) calls OS which calls printer or hard drive.

. A Kernel is at the core of an OS, its running at all times on the computer. It has complete control over everything in the system and facilitates interactions between software and hardware. it runs on start-up and translates requests to instructions for the CPU. It has its own protected space in memory thats secure and not accessible by any other software. (if it happens, the computer crashes)

. A bootstrap program is loaded during power-up or reboot, usually stored in ROM (read only memory) => Firmware. The program initializes all aspects of the system and loads the kernel and starts execution. Loads the kernel and BIOS (Basic input/output System) and Boot order.

. Daemon runs as long as kernel is running. its a name for programs loaded on memory during boot time. They provide services outside the kernel.

. Device controller: Specific to a device, every device has a controller and OS device manager handles them. 2 identical disks can have 1 controller.

After all of this is done and there's no processes to run or I/O devices to service, the OS sits there and waits for some signal or event to happen

↳ How does the OS know if some process or I/O needs further attention or ressources?

① Polling: Ask each if they need something. Thats a fair technique but it wastes CPU cycles doing that and most of the time the answer is no

② Interrupt approach: simply wait until a device tasked with something finishes, more efficient to signal the CPU only when that happens. (Hardware)

③ Trap/Exception: A type of interrupt (software) that happens when there's an error, a specific request (system call)) or infinite loop, etc...

* Every OS has 5 essential managers

① Device manager

↳ keeps track of all devices and the controllers responsible for them and monitors their status.

↳ optimizes the performance of every device

↳ Allocates and de-allocates the device in an efficient way. At the process level => when a command is executed and device is temporarily released vs Job level when a job is finished and the device is permanently released.

↳ Some devices require a special allocation schema where they can handle only one job at a time. (printers, tape drivers, plotters, etc.)

② Process manager

↳ Processes can be in 5 different states: hold, run, wait, ready, finish

↳ The manager decides the order in which processes have access the processor, and how much processing time each process has. (process scheduling)

↳ keeps track of the status of processes.

③ File manager

↳ Determines where and how each file should be stored in order to efficiently

↳ Allocates and de-allocates files, given that the user has the appropriate access rights + communicates availability of files.

↳ Keeps track of each file and directory that contains it.

④ Memory manager

↳ manages the primary memory and what processes are granted access to memory and for how long. Also checks how much memory is to be allocated to processes.

↳ Moves back and forth between main memory and disk during execution.

↳ Single user contingyous (first scheme): if a job was too large for memory it couldn't be handled. Only one job would occupy memory at a time (Not efficient)

↳ Fixed Partitions (second scheme): Memory divided into partitions recorded in a table. Partition sizes are static and only change upon rebooting.

↳ Dynamic partitions (current scheme): Jobs only recieve as much memory as they request upon loading find the most close-fitting partition to store jobs (slower performance but efficient)

⑤ Network manager (we don't cover much of it in this class)

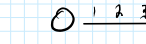
↳ keeps track of inter-process communication

↳ provides data transportation across the network through implementing protocols as drivers. Typically we need to reboot before loading or unloading drivers.

↳ provides file sharing across a network and the ability to send jobs for example a print job to remote printers

. RAM vs ROM

. No real relationship between both concepts. (The OS/Kernel/Bootstrap is in ROM part of the RAM



RAM like a tape, need to go through it to get the part we want vs ROM, only written once anyways we can only read.

- RAM requires power to store data, it temporarily stores files to be used by the computer, or temporary info produced by programs.
- ROM does not require power. Its a chip stored on the motherboard and data is read-only. (example is BIOS)

* Ressource Abstraction

- Ressource => hardware components that OS might use.
- Abstraction => hiding the details away. For example writing into memory, that can only be done by using the language that memory understands

Abstraction creates a method that allows us to do what we want easier. It's easier to use but less flexible cuz we're hiding 3 methods in one.

load (Block, length, device)
seek (device, track)
out (device, sector)
} abstraction of this shit → write (Block, length, device, sector).

* Ressource sharing

Concurrent execution of processes. They appear or really are two or more programs executing at same time.

① Space multiplexed: Share space in memory
② Time multiplexed: Each ressource is allocated a set amount of time.

* More details on strategies used by OS

① Batch systems (Multi-programming): program = process = job

- Run programs concurrently in a single computer system, A single program cannot normally keep the CPU or I/O devices busy forever
- Keeps users satisfied (cuz they want to run many programs at same time typically) and increases CPU utilization by organizing jobs in a way that the CPU always has something to execute.
- Keeps multiple processes in memory. So instead of waiting on additional ✗ to finish a process, OS tells the CPU to switch to another process.
- Security issue cuz running on same machine
- fairness issue cuz gotta decide how to distribute ressources fairly.

UNI-programming

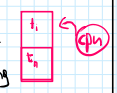
tasks: t_1, t_2, t_3, t_n

The time it takes to run this is the sum of them all together

- One program at a time
- No security concern
- consider ressource Abstraction

Multi-programming

CPU runs multiple prog at "same time", pretty fast so it looks like they're running at same time.



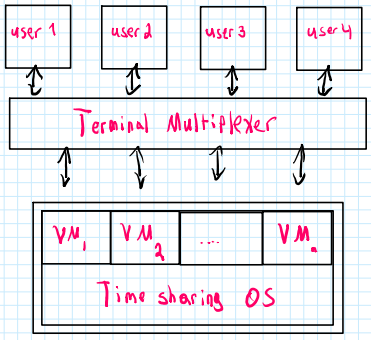
time multiplexed CPU + space multiplexed RAM

- programs running at same time sharing the memory can access each other's so need some type of security.
- So we use ressource isolation abstraction
- Improve overall system performance but not single process. max($t_1, t_2, t_3 \ldots$) instead of $t_1 + t_2 + t_3 \ldots$

So in other words: A Batch processing system handles jobs a batch at a time. An input spooler groups jobs into a batch, then sends them to the CPU. The CPU executes the jobs and writes output to the output spool. Then all the results are returned to the user. This is good for large jobs with little interactions

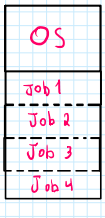
② Time sharing (multi-tasking)

- This is the logical extension of multi-programming. The CPU executes multiple processes by continuously switching between them. The switches happen so frequently it provides the user with a fast response time, usually under 1 second. Ressorces are not divided, one process can use all ressources for a set period of time.
- An interactive process will usually be slower (cuz it depends on user input, how fast you can type) so instead of having the CPU sit idle while waiting for I/O, it will switch to another job (CPU scheduling, jobs in memory)
- A time-sharing system must ensure a reasonable response time and that jobs don't step all over each other.
- In order for a job to run it has to be loaded into memory. Since the CPU always needs something in queue to be loaded, what if we run out of memory? The OS is now designed to not need to load the whole program in memory, that's called virtual memory. This technique allows execution of jobs not completely loaded in memory and allows the user to run programs that are longer than the actual physical memory of the computer.

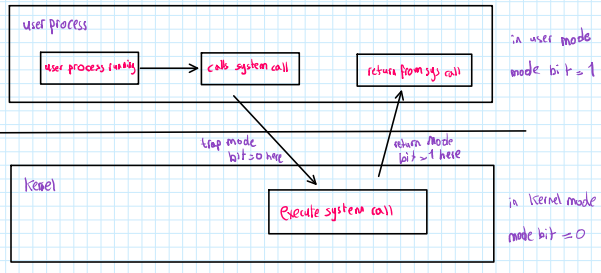


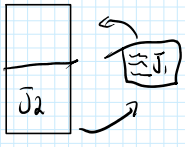
Ressource isolation prevents unauthorized access of ressources by processes when they're allocated to another one.

- In order to ensure the proper execution of the OS, we need to distinguish between its own code and a user-defined code. In order to do that, computers use a strategy called Dual-mode which allows to have a bit that dinstinguishes between Kernel mode (0) and User mode (1), this bit is provided by the hardware. Systems that support virtualization will have more than 2 modes.



- The OS is loaded in its own protected place in memory and granted priveleged access to execute priveleged instructions in Kernal mode.
- It ensures that an improperly designed program doesn't cause other programs, or the OS itself to execute incorrectly
- The idea is that the the OS software is trusted and executes with the CPU in Kernel mode and all other software executes in user mode, that means that I/O instructions are protected so an application program can't perform its own I/O it has to request that the OS performs the I/O operation for it. So then when that system call happens, the system has to transition between user and Kernel modes.
- Context_switching is needed to save the state (address) of doing Job 1 before running Job 2, so the CPU can restore its state later. This happens because switching the CPU to another Job requires a save state when the interrupt happens, whether in Kernal or user mode.
- Time spent on context-switching is pure overhead because the system does not do anything useful at that time.





Briefly put, this system supports interactive users because it ensures that the response time is fast enough. It uses multi-programming and CPU scheduling to stay economic and busy. Instead of having a spool each program is loaded into memory and executed + rapidly switches so no idle time.

* There are 2 ways for a program in user mode to request Kernel's services.

① System call:

next class

② Message passing:

next class

③ Personal computers and workstations

- The entire machine is dedicated to a single user, usually PC is more suitable for reasonably small jobs.
- workstations were more powerful with complex hardware providing more ressources and requiring more complex OS to handle them.

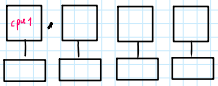
④ Process control/Embedded system/Real time systems

- whole machine is dedicated to serve one purpose.
- Must guarantee response within a maximum time. This is a rigid time constraint not a suggestion. System halts if deadline is not met.

⑤ Network systems

- provide operating system services features across a network. Such as file sharing or network printing

a) Distributed system: Systems are scattered among different processors. The processors do run at same time making it seem like a parallel but its not because each processor has its own memory and clock, its not shared



b) Parallel system: Some but shared memory