

What is an OS?

- Ultimate control program
 - Applications can be cool but they can only do what the OS lets them do
 - If you control the OS, you control the system
- Two different views of an OS
 1. Extended machine view
 - Virtual machine that is easier to understand and program
 - Tool to make programmer's job easy
 2. Resource manager view
 - Tool to facilitate efficient operation of computer system
 - Provides services to users; processor, memory, I/O, system bus
- Resource allocator
 - Must be fair; not partial to any process, specially for process in the same class
 - Must discriminate between different class of jobs with different service requirements
 - Do the above efficiently
 - * Within the constraints of fairness and efficiency, an OS should attempt to maximize throughput, minimize response time, and accommodate as many users as possible
- Visualizing the place of operating system in layered architecture of computer systems

Banking system	Airline reservation	Adventure games
Compilers	Editors	Command Interpreter
Operating System		
Machine Language		
Microprogramming		
Physical Devices		

- UNIX view
 - Application environment – shell, mail, text processing package, SCCS
 - Operating system – support programs for applications

History with respect to operating systems; Punctuated Equilibrium¹

- Early Systems
 - 1945 – 1955
 - Bare machines – vacuum tubes and plug-boards
 - No operating system
 - No protection
 - ENIAC – Electronic Numerical Integrator And Computer
- Second Generation Systems

¹Steve Nigus

- 1956 – 1965
- Transistors and batch systems
- I/O channel
- Read ahead / spooling
- Interrupts / exceptions
- Minimal protection
- Libraries / JCL
- Third Generation Systems
 - 1965 – 1980
 - ICs and Multiprogramming
 - System 360 and S/370 family of computers
 - Spooling (simultaneous peripheral operation on-line)
 - Time sharing
 - On-line storage for
 - * System programs
 - * User programs and data
 - * Program libraries
 - Virtual memory
 - Multiprocessor configurations
 - MULTICS – Multiplexed Information and Computing Service
 - * Design started in 1965 and completed in 1972
 - * Collaborative effort between General Electric, Bell Telephone Labs, and Project MAC of MIT
 - * Aimed at providing
 - simultaneous computer access to large community of users
 - ample computation power and data storage
 - easy data sharing between users, if desired
- Fourth Generation and beyond
 - Personal computers and workstations
 - MS-DOS and Unix
 - Massively parallel systems
 - * Pipelining
 - * Array processing / SIMD
 - Sun UltraSPARC chips can process up to eight 8-bit data elements simultaneously using VIS instructions
 - Intel processors have a similar capability through MMX while PowerPC can achieve the same using AltiVec
 - * General multiprocessing / MIMD
 - * Symmetric multiprocessing / SMP
 - Any process and any thread can run on any available processor
 - Processors share the same memory and I/O, through a common communications bus
 - Shared memory is used to support communication and synchronization among processors
 - Getting more popular due to advantages such as performance, availability, incremental growth, and scaling from hardware perspective
 - Computer networks (communication aspect) – network operating systems

- Distributed computing – distributed operating systems
- Cluster computing and fault tolerance
 - * Group of individual computers equipped with their own OS and linked by a high speed interconnection
 - * A virtual server accepts requests and directs them to one in a series of servers
 - * High degree of availability
 - One node goes down, others can take over its work
 - * Provides load balancing, fault tolerance, and higher scalability
 - * Suitable for high-end, transaction processing applications
 - * Typical applications include firewalls, web servers, domain name servers, and mail servers
 - * Grid computing
 - Linking a number of separate fully-functional computers into a large cluster on demand
 - Applications run like a screen saver to take advantage of spare CPU cycles
 - SETI@home project
- Cloud Computing
 - * Software as a service, or Internet as a platform
 - * Migration of data and programs from desktop PCs and corporate server rooms to the *compute cloud*
 - * Google docs
 - * No central computer as required in time sharing topology
 - * No infrastructural investment in terms of OS updates and related changes to locally developed software; more mobility and collaboration
 - * From vendor perspective, no need to develop software for diverse arrays of platforms (Turbo Tax); quick deployment of bug fixes and updates
 - Server side software still needs to interact with a wide variety of clients
 - * Cloud OS
 - User interface resides in a single window in web browser
 - OS inside a browser – eyeOS
 - Bypass the web browser; more capable software runs as a separate application on client computer and communicates directly with servers in cloud – AIR from Adobe
 - * Application scalability
 - Coordinate information coming from multiple sources
 - Many-to-many communication – each server talks to multiple clients and each client may invoke programs on multiple servers
 - Platform and language issues; and culture of users
 - * Privacy, security , reliability, confidentiality
 - Who owns the documents stored in cloud?
 - Move from one cloud to another; what happens to old data?
 - Bill not paid; lose access to your data?
 - Delete unwanted documents?
 - Search warrant on your data; can you contest the order? will the cloud surrender it without your knowledge?
 - Which country’s laws govern the information privacy and access, yours or where it is hosted?

Operating System Concepts

- Program
 - Collection of instructions and data kept in ordinary file on disk

- Executable program, complete code output by linker/loader, with input from libraries
 - * Generated from source program and object code
- The file is marked as executable in the i-node
- File contents are arranged according to rules established by the kernel

- Processes

- Created by kernel as an environment in which a program executes
- Program in execution
- May be stopped and later restarted by the OS
- Core image
 1. Instruction segment
 2. User data segment
 3. System data segment
 - * Includes attributes such as current directory, open file descriptors, and accumulated CPU times
 - * Information stays outside of the process address space
- Program initializes the first two segments
- Process may modify both instructions (rarely) and data
- Process table – records information about each process
Program code + data + stack + PC + SP + registers
- Process may acquire resources (more memory, open files) not present in the program
- Child and parent processes
- Communication between processes through messages
- uid and gid
- Process id
 - 0 swapper
 - 1 /sbin/init
 - 2 pagedaemon

- Threads

- Stream of instruction execution
- A dispatch-able unit of work to provide intraprocess concurrency in newer operating systems
- A process may have multiple threads of execution in parallel, each thread executing sequentially

- Files

- Services of file management system to hide disk/tape specifics
- System calls for file management
- Directory to group files together
- Organized as a hierarchical tree
- Root directory
- Path name
- Path name separator
- Working directory
- Protection of files (9-bit code in Unix – rwx bits)

- File descriptor or handle – small integer to identify a file in subsequent operations, error code to indicate access denied
 - * 0 – standard input
 - * 1 – standard output
 - * 2 – standard error
 - I/O device treated as a special file
 - * Block special file
 - * Character special file
 - Pipe – pseudo file to connect two processes
- System calls
 - Interface between user program and operating system
 - Set of extended instructions provided by the operating system
 - Applied to various software objects like processes and files
 - Invoked by user programs to communicate with the kernel and request services
 - Access routines in the kernel that do the work
 - Library functions corresponding to each system call
 - * Machine registers to hold parameters of system call
 - * Trap instruction (protected procedure call) to start OS
 - * Hide details of trap and make system call look like ordinary procedure call
 - * Return from trap instruction
 - Broadly divided into six classes: filesystem (`open`), process (`fork`), scheduling (`prctl`), IPC (`semop`), socket/networking (`bind`), and miscellaneous (`time`)
 - * The scheduling related system calls on p. 100 of the text are actually library functions (section 3RT in Solaris)
 - * The socket/networking is implemented as library functions in Solaris but as system calls in FreeBSD
 - System calls vs library functions
 - * System calls run in kernel mode on user's behalf; provided by kernel itself
 - * Library functions run completely in user space and provide a more convenient interface for the programmer to the functions that do the real work – system calls
 - * System calls corresponding to `printf`
 - Shell
 - Unix command interpreter
 - * Interprets the first word of a command line as a command name
 - Is a user program and not part of the kernel
 - Prompt
 - Redirection of input and output
 - Background jobs
 - For most commands, the shell `forks` and the child `execs` the command associated with the name, treating the remaining words on the command line as parameters to the command
 - Allows for three types of commands:
 1. Executable files
 2. Shell-scripts
 3. Built-in shell commands

- Kernel

- Permanently resides in the main memory
- Controls the execution of processes by allowing their creation, termination or suspension, and communication
- Schedules processes fairly for execution on the CPU
 - * Processes share the CPU in a time-shared manner
 - CPU executes a process
 - Kernel suspends it when its time quantum elapses
 - Kernel schedules another process to execute
 - Kernel later reschedules the suspended process
- Allocates main memory for an executing process
 - * Allows processes to share portions of their address space under certain conditions, but protects the private address space of a process from outside tampering
 - * If the system runs low on free memory, the kernel frees memory by writing a process temporarily to secondary memory, or SWAP device
 - * If the kernel writes entire processes to a swap device, the implementation of the Unix system is called a *swapping* system; if it writes pages of memory to a swap device, it is called a *paging* system.
 - * Coordinates with the machine hardware to set up a virtual to physical address that maps the compiler-generated addresses to their physical addresses
- File system maintenance
 - * Allocates secondary memory for efficient storage and retrieval of user data
 - * Allocates secondary storage for user files
 - * Reclaims unused storage
 - * Structures the file system in a well understood manner
 - * Protects user files from illegal access
- Allows processes controlled access to peripheral devices such as terminals, tape drives, disk drives, and network devices.
- Services provided by kernel transparently
 - * Recognizes that a given file is a regular file or a device but hides the distinction from user processes
 - * Formats data in a file for internal storage but hides the internal format from user processes, returning an unformatted byte stream
 - * Allows shell to read terminal input, to spawn processes dynamically, to synchronize process execution, to create pipes, and to redirect I/O
- Kernel in relation to other processes
 - * In older operating systems, kernel executed outside of any process
 - * This definition is also followed in Unix and Linux
 - * Kernel has its own address space and its own system stack to control procedure calls and returns
 - * In such systems, the concept of process only applies to user programs; OS code executes in privileged mode
- Micro-kernel architecture
 - * Smaller set of operations in a limited form assigned to kernel
 - * Interprocess communication, limited process management and scheduling, and some low-level I/O
 - * Other OS services provided by processes, or servers, running in user mode
 - * Less hardware specific because many system-specific operations are pushed into user space
 - * Simplifies implementation, provides flexibility, and is better suited for distributed environment
- Kernel in UNIX
 - * Traditionally, the operating system itself

- * Isolated from users and applications
 - Unix philosophy – Keep kernel as small as possible
 - A bug in an application should not crash the OS
 - * At the top level, user programs invoke OS services using system calls or library functions
 - * At the lowest level, kernel primitives directly interface with the hardware
 - * Kernel itself is logically divided into two parts:
 1. *File subsystem* to transfer data between memory and external devices
 2. *Process control subsystem* to control interprocess communication, process scheduling, and memory management
- Kernel in Linux
- * Monolithic in nature
 - One large block of code
 - Runs as a single process within a single address space
 - All functional components have access to all internal data structures and functions
 - If anything is changed, all modules and functions must be recompiled and relinked and system rebooted
 - Difficult to add new device driver or file system functions
 - * Structured as a collection of modules that can be automatically loaded and unloaded on demand (loadable modules)
 - Loadable modules overcome some of the problems of monolithic kernel
 - Dynamically linked
 - Stackable modules – arranged as a hierarchy
 - * Runs in its own memory space (memory is divided into kernel space and user space)
 - * Operations include scheduling, process management, signaling, device I/O, paging, and swapping
 - * Being monolithic, it includes low-level interaction with the hardware, and hence is specific to a particular architecture
 - * Linus Torvalds chose monolithic architecture for Linux because
 1. Micro-kernels were experimental at the time Linux was designed
 2. Micro-kernels were more complex than monolithic kernels
 3. Micro-kernels executed notably slowly than monolithic kernels
- Kernel in Windows NT
- * Traditional kernel operations provided by *executive*; Microsoft calls it a *modified micro-kernel architecture*
 - Unlike pure micro-kernel, many of the system's functions outside the micro-kernel run in kernel mode for performance reasons
 - * Kernel works in cooperation with executive
 - * Manages thread scheduling, process switching, exception and interrupt handling, and multiprocessor synchronization; rest in executive
 - Kernel's own code does not run in threads
 - Only part of the OS that can not be preempted or paged
 - * As with UNIX, it is isolated from user programs, with user programs and applications allowed to access one of the protected subsystems
 - Each system function is managed by only one component of the OS
 - Rest of the OS and all applications access the function through the responsible component using a standardized interface
 - Key system data can be accessed only through the appropriate function
 - In principle, any module can be removed, upgraded, or replaced without rewriting the entire system or its standard application programming interface (API)
 - * Two programming interfaces provided by a subsystem

1. *Win32 interface* – for traditional Windows users and programmers
 2. *POSIX interface* – to make porting of UNIX applications easier
- * Subsystems and services access the executive using system services
 - * Executive contains object manager, security reference monitor, process manager, local procedure call facility, memory manager, and an I/O manager

Memory

- Memory hierarchy based on storage capacity, speed, and cost
 - As capacity increases, applications grow to use it
 - Need for speed to keep up with the improvement in CPU speed
 - Higher the storage capacity, lesser the speed, and lesser the cost
- Different memory levels, in decreasing cost per byte of storage

Registers	Few bytes	Almost CPU speed
Cache memory	Few kilobytes	Nanoseconds
Main memory	Gigabytes	Microseconds
Magnetic disk	Terabytes	Milliseconds
USB/Optical disk	No limit	Off-line storage

- Use hierarchical memory to transfer data from lower memory (away from CPU) to higher memory (closer to CPU) to be executed
 - Smaller/expensive/faster memory is supplemented by larger/inexpensive/slower memory
 - Hit ratio and access time
 - * Hit ratio H is the fraction of all memory accesses that are found in the faster memory
 - * For higher values of H , the access time is much closer to that of higher memory
 - * Consider two levels of memory with access times of $0.1\mu s$ and $1.0\mu s$
 - Average access time is given by

$$\begin{aligned}
 & 0.95 \times 0.1\mu s + 0.05 \times (0.1\mu s + 1.0\mu s) \\
 = & 0.095 + 0.055\mu s \\
 = & 0.15\mu s
 \end{aligned}$$

- Locality of reference
 - Most of the references in the memory are clustered and move from one cluster to the next
 - Volatility
 - Cache memory
 - * Use of very fast memory (a few kilobytes) designated to contain data for fast access by the CPU
 - * May not be visible to programmer or system
 - * Used to temporarily hold data for transfer between main memory and registers to improve performance
 - Virtual memory or extension of main memory
 - Disk cache
 - * Designating a portion of main memory for disk read/write
 - * Improves performance by
 - Clustering disk writes; perform fewer large data transfers instead of many small transfers

- Data destined for disk can be read back before it is written

- Memory management

- Memory management is one of the most important services provided by the operating system
- An operating system has five major responsibilities for managing memory:
 1. Process isolation
 - * Should prevent the independent processes from interfering with the data and code segments of each other
 2. Automatic allocation and management
 - * Programs should be dynamically allocated across the memory depending on the availability (may or may not be contiguous)
 - * Programmer should not be able to perceive this allocation
 3. Modular programming support
 - * Programmers should be able to define program modules
 - * Programmers should be able to dynamically create/destroy/alter the size of modules
 4. Protection and access control
 - * Different programs should be able to co-operate in sharing some memory space
 - * Contrast this with the first responsibility
 - * Make sure that such sharing is controlled and processes should not be able to indiscriminately access the memory allocated to other processes
 5. Long-term storage
 - * Users and applications may require means for storing information for extended periods of time
 - * Generally implemented with a file system
- OS may separate the memory into two distinct views: physical and logical; this division forms the basis for virtual memory

Process execution modes in Unix

- Two modes of process execution

1. User mode
 - Normal mode of execution for a process
 - Execution of a system call changes the mode to kernel mode
 - Processes can access their own instructions and data but not kernel instructions and data
 - Cannot execute certain privileged machine instructions
2. Kernel mode
 - Processes can access both kernel as well as user instructions and data
 - No limit to which instructions can be executed
 - Runs on behalf of a user process and is a part of the user process

Operating System Structure

- Minimal OS
 - CP/M or DOS
 - Initial Program Loading (Bootstrapping)
 - File system

- Monolithic Structure
 - Most primitive form of operating systems
 - No structure
 - Collection of procedures that can call any other procedure
 - Well-defined interface for procedures
 - No information hiding
 - Services provided by putting parameters in well-defined places and executing a *supervisor call*
 - * Switch machine from *user mode* to *kernel mode*
 - Basic OS structure
 - * Main program that invokes requested service procedures
 - * Set of service procedures to carry out system calls
 - * Set of utility procedures to help the service procedures
 - User program executes until
 - * program terminates
 - * time-out signal
 - * service request
 - * interrupt
 - Difficult to maintain
 - Difficult to take care of concurrency due to multiple users/jobs
- Layered Systems
 - Hierarchy of layers – one above the other
 - THE system (1968), MULTICS
 - Six layers
 1. Allocation of processor, switching between processes
 2. Memory and drum management
 3. Operator-process communication – process and operator console
 4. I/O management
 5. User programs
 6. Operator
 - MULTICS
 - * organized as a series of concentric rings
 - * inner rings more privileged
- Virtual machines
 - Basis for developing the OS
 - Provides a minimal set of operations
 - Creates a virtual CPU for every process
 - IBM System 370 – CMS, VM
 - * Allowed a number of *guest* operating systems under the control of VM
 - * Useful for cross platform development of software
 - * A similar task is accomplished by VMware in recent times
 - Provides support for Linux, FreeBSD, DOS 6.0, and Windows 95/98/NT/2000 on the same machine
 - Allows the different systems to run concurrently

- Physical raw disk or virtual disk
- Physical raw disk allows you to use data directly for the native OS, if installed
- Virtual disk gives you up to 2GB file system in a single file
- * *Virtual Machine Monitor*
- Virtual CPU.** Virtualize CPU for all processes
- Virtual memory.** Virtualize memory for all processes
 - Single virtual memory shared by all processes
 - Separate virtual memory for each process
- Virtual I/O devices.**
- * Performs functions associated with CPU management and allocation
- * Provides synchronization and/or communication primitives for process communication

- Process Hierarchy

- Structured as a multilevel hierarchy
- Parent-child model

- Client-Server Model

- Remove as much as possible from the OS leaving a minimal kernel
- User process (client) sends a request to server process
- Kernel handles communications between client and server
- Split OS into parts – file service, process service, terminal service, memory service
- Servers run in user mode – small and manageable

I/O communication

- Programmed I/O

- Simplest and least expensive scheme
- CPU retains control of the device controller and takes responsibility to transfer every bit to/from the I/O devices
- Instruction set need I/O instructions for
 - * Control: To activate and operate external devices (rewind tape)
 - * Status: Test status conditions for I/O module and peripherals
 - * Transfer: Read/write data
- Bus
 - * Address bus: To select a memory location or I/O device
 - * Data bus: To transfer data
- Handshaking protocol
- Disadvantages:
 - * Poor resource utilization
 - * Only one device active at a time
 - * Gross mismatch between the speeds of CPU and I/O devices

- Interrupt-driven I/O

- CPU still retains control of the I/O process
- Sends an I/O command to the I/O module and goes on to do something else
- I/O module reads/writes data on the specified peripheral and places it on the data bus when instructed by CPU

- I/O module interrupts CPU when it is ready to transfer more data
- Possible to have multiple I/O modules on a machine
- Direct memory access
 - CPU trusts the DMA module to read from/write into a designated portion of the memory
 - DMA module (also called I/O channel) acts as a slave to the CPU to execute those transfers
 - DMA module takes control of the bus, and that may slow down the CPU if the CPU needs to use the bus

CPU and I/O overlap

- Hardware flag
 - CPU is blocked if device is busy
- Polling by test-device-flag
- Memory-mapped I/O
 - Uses memory address register (MAR) and memory buffer register (MBR) to interact with I/O devices
- I/O-mapped I/O
 - Uses I/O address register and I/O buffer register to communicate with the I/O devices

Multiprogramming

- CPU-bound system
- I/O-bound system
- Maintain more than one independent program in the main memory
- Sharing of time and space

Multiprogramming OS

- Requires addition of new hardware components
 - DMA Hardware
 - Priority Interrupt Mechanism
 - Timer
 - Storage and Instruction Protection
 - Dynamic Address Relocation
- Complexity of operating system
- Must hide the sharing of resources between different users
- Must hide details of storage and I/O devices
- Complex file system for secondary storage

Tasks of a Multiprogramming OS

- Bridge the gap between the machine and the user level
- Manage the available resources needed by different users

- Enforce protection policies
- Provide facilities for synchronization and communication

Operating Systems as Virtual Machines

- Allows each user to perceive himself as the only user of the machine
- Fair share of available resources
- Time sharing (for CPU time)
- Abstraction
 - Availability of higher level operations as primitive operations
 - Virtual command language as the machine language of virtual machine
 - Virtual memory

But what about user interface?

- Not a concern for OS
- The only purpose of user interface is to shield the user from the system
- Unix takes the users inside the system, leading them through labyrinthine logic trails, labeling itself “unfriendly” in the process
- Is it GUI, or is it CUI?
 - Creeping featurism
 - * Designing for beginners vs designing for experts
 - * Getting into peoples’ head (playing dumb charades or pictionary)
 - Interface with other programs
 - Effects on complexity
 - Adaptability
 - Scalability (adding 1 user vs 100 users)
 - Who is in charge? User or computer?