Betriebssysteme

Overview

Lehrstuhl Systemarchitektur

WS 2009/2010

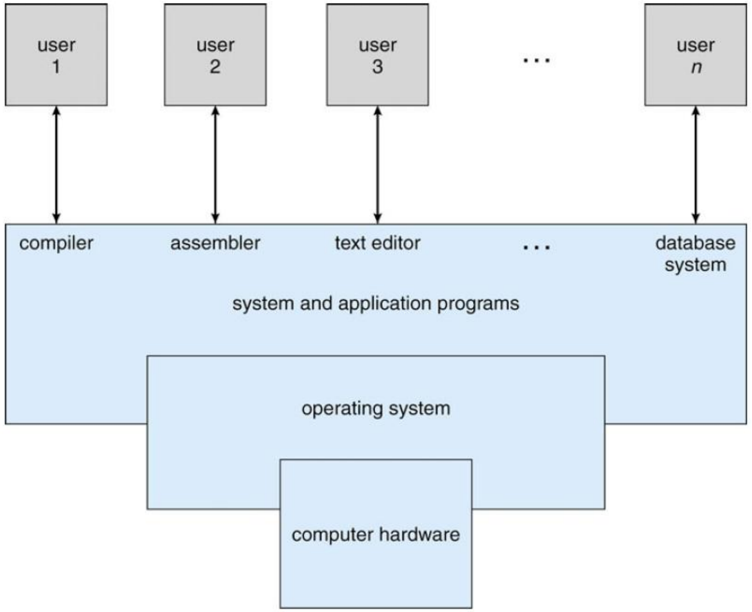
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

Computer System Structure

- Computer system can be divided into four components
  - Hardware - provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs - define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Four Components of a Computer System



Operating System Definition

- OS is a resource allocator
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the computer
- Everything a vendor ships when you order an operating system

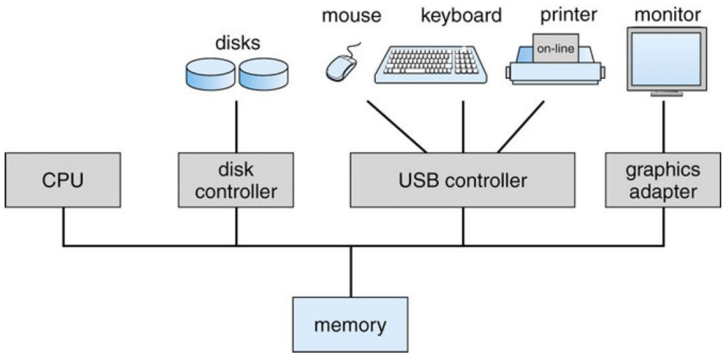
➜ No universally accepted definition

Computer Startup

- bootstrap program is loaded at power-up or reboot
  - Typically stored in ROM or FLASH memory, generally known as firmware
  - Initializes all (for the boot procedure) relevant HW components
  - Loads operating system kernel and starts execution

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

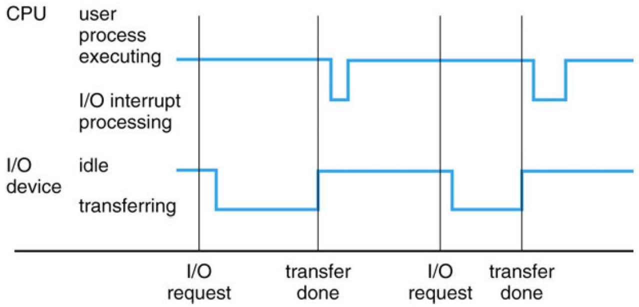
- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt

Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt
- A trap is a software-generated interrupt caused either by an error or a user request
- An operating system is interrupt driven
- Timer interrupt to prevent infinite loop / process hogging CPU
  - Trigger interrupt after specific period
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - polling
  - vectored interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt



I/O Semantics

- Synchronous I/O or blocking I/O
  - After the I/O request is submitted with a system call, control returns to user program only upon I/O completion
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- Asynchronous I/O or non-blocking I/O
  - After I/O request is submitted, control returns to user program without waiting for I/O completion
    - Polling
    - Signal
    - Callback function

Direct Memory Access Structure

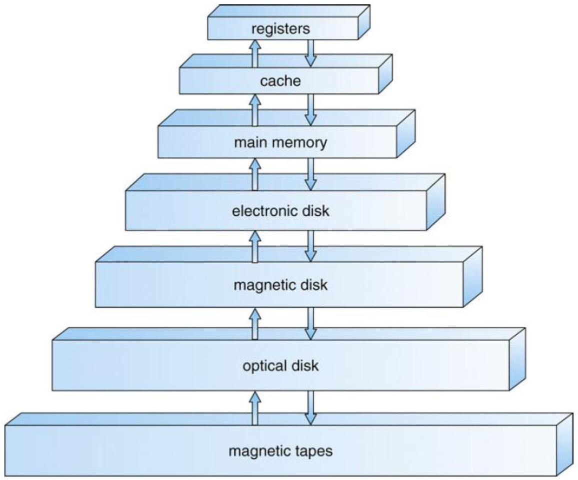
- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte/word

Storage Structure

- Main memory - only large storage media that the CPU can access directly
- Secondary storage - extension of main memory that provides large nonvolatile storage capacity
- Magnetic disks - rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into tracks, which are subdivided into sectors
  - The disk controller determines the logical interaction between the device and the computer

Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility



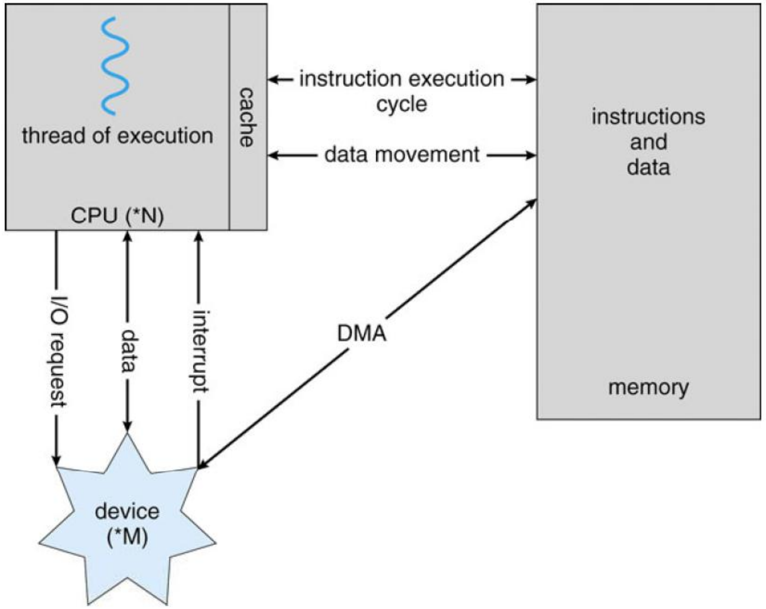
Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use is copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data is copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

Computer-System Architecture

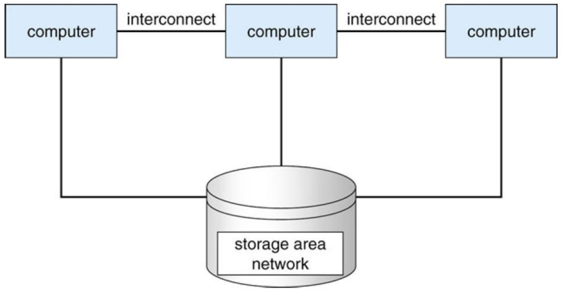
- Most systems use a single general-purpose processor
  - Most systems have special-purpose processors as well
- Shared-memory multiprocessor (MP) systems, also known as tightly-coupled systems, grow in use and importance
  - Types of MPs (often in combination)
    - Multi-socket systems
    - Multi-Chip Module (MCM) (=Multi-Core)
    - Chip Multiprocessor (CMP)(=Multi-Core)
    - Simultaneous MultiThreading Processor (SMT)
  - Advantages include
    - Increased throughput
    - Economy of scale
    - Increased reliability - graceful degradation or fault tolerance
  - Two types
    - Symmetric Multiprocessing (homogenous cores and functionality)
    - Asymmetric Multiprocessing (dedicated HW-/SW-functionality)

How a Modern Computer Works



Clustered Systems

- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a storage-area network (SAN)
  - Provides a high-availability service which survives failures
    - Asymmetric clustering has one machine in hot-standby mode
    - Symmetric clustering has multiple nodes running applications, monitoring each other
  - Some clusters are for high-performance computing (HPC)
    - Applications must be written to use parallelization

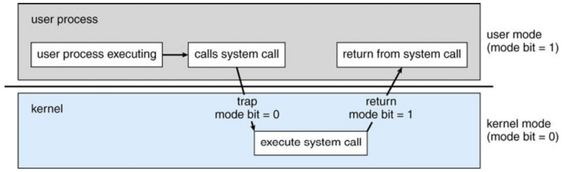


Operating System Structure

- Multiprogramming needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via job scheduling
  - When it has to wait (for I/O for example), OS switches to another job
- Timesharing (multitasking) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
  - Response time should be < 0.2 second
  - Each user has at least one program executing in memory ➜process
  - If several jobs ready to run at the same time ➜CPU scheduling
  - If processes don't fit in memory, swapping moves them in and out to run
  - Virtual memory allows execution of processes not completely in memory

Operating-System Operations

- Interrupt driven by hardware
- Software error or request creates exception or trap
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- Dual-mode operation allows OS to protect itself and other system components
  - User mode and kernel mode
  - Mode bit provided by hardware
    - Mode bit allows to distinguish when system is running user code or kernel code
    - Some privileged instructions are only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user



Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a passive entity, process is an active entity.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one program counter specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes/threads, some user, some operating system, running concurrently on one or more CPUs
  - ➜ Concurrency by multiplexing the CPUs among the processes/threads

Process Management Activities

- The operating system is responsible for the following activities in connection with process management:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a long period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

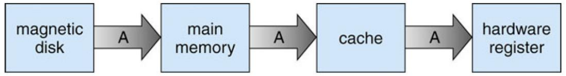
Performance Levels of Storage

- Implicit/explicit movements between levels of storage hierarchy

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1KB	< 64 MB	< 64 GB	> 100 GB
Implementation technology	custom memory multiport CMOS	on-/off-chip CMOS SRAM	DRAM PRAM STT-RAM	FLASH magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	5000 5.000.000
Bandwidth (MB/sec)	20.000 - 100.000	5000 - 10.000	1000 - 5000	20 - 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	DVD/tape

Migration from Disk to Registers

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 17

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Drivers for specific hardware devices
  - General device-driver interface
  - Memory management of I/O including
    - buffering (storing data temporarily while it is being transferred)
    - caching (storing parts of data in faster storage for performance
    - spooling (the overlapping of output of one job with input of other jobs)

Protection and Security

- Protection: any mechanism for controlling access of processes or users to resources defined by the OS
- Security: defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
  - Privilege escalation allows user to change to effective ID with more rights

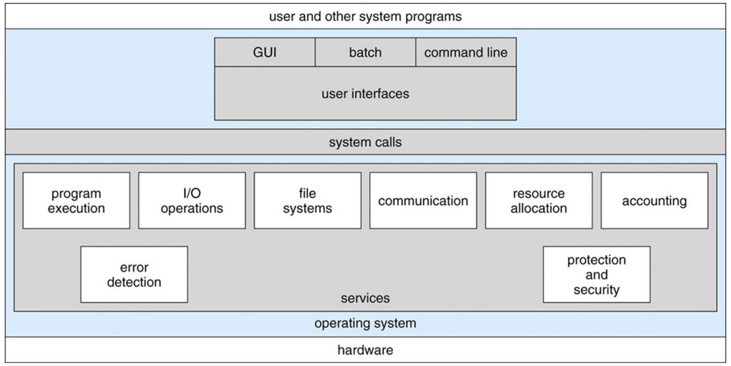
System Structures

- System Services
- System Calls
- System Programs
- Operating System Design

Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - User interface - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - I/O operations - A running program may require I/O, which may involve a file or an I/O device
  - File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

A View of Operating System Services



Operating System Services

- One set of operating-system services provides functions that are helpful to the user:
  - Communication: Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the OS)
  - Error detection: OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services

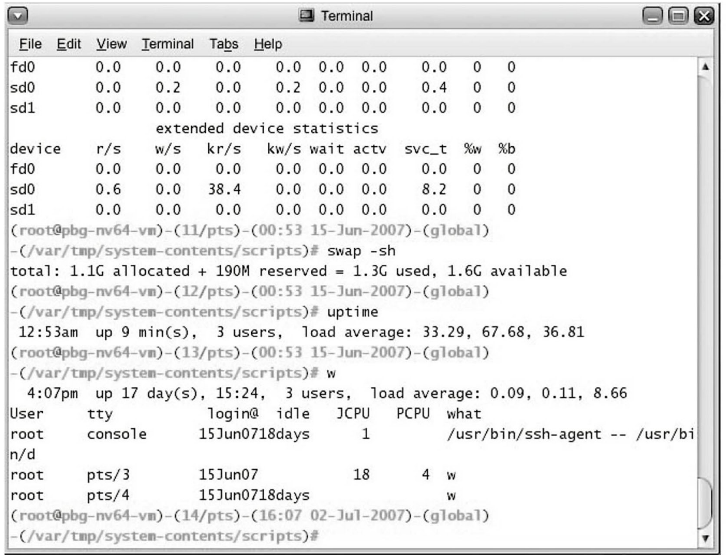
- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - Resource allocation - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
  - Accounting - To keep track of which users use how much and what kinds of computer resources
  - Protection and security - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - Protection involves ensuring that all access to system resources is controlled
    - Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
    - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

User Operating System Interface - CLI

Command Line Interface (CLI) or command interpreter allows direct command entry

- Implemented in kernel
- Implemented by systems program
  - Sometimes multiple flavors implemented - shells
- Primarily fetches a command from user and executes it
  - Commands built-in
  - Names of programs
    - ➜ adding new features doesn't require shell modification

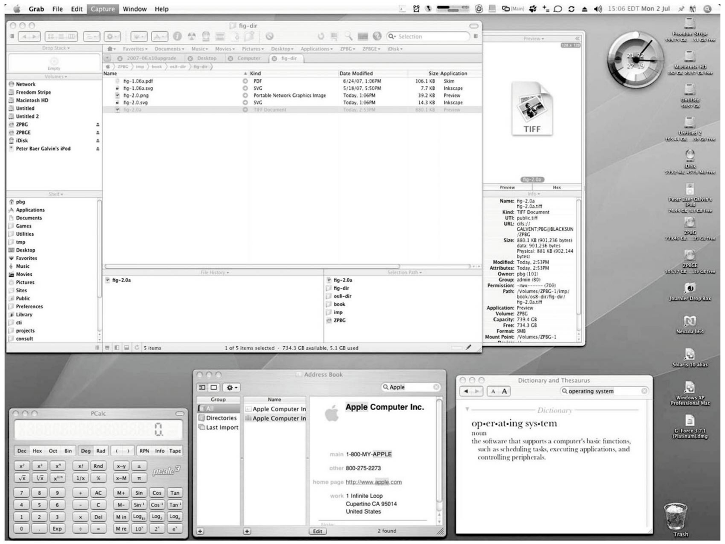
Bourne Shell Command Interpreter



User Operating System Interface - GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X as "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional X11 GUI interfaces (Java Desktop, KDE)

The Mac OS X GUI



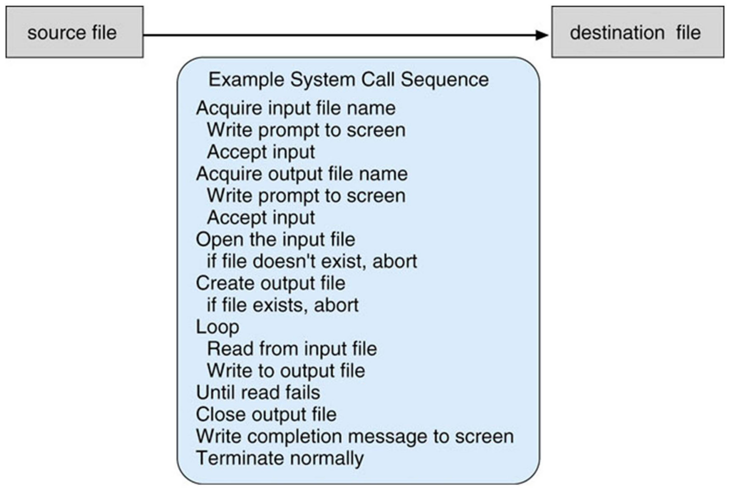
System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

(Note that the system-call names used throughout this text are generic)

Example of System Calls

System call sequence to copy the contents of one file to another file



Example of Standard API

- Consider the ReadFile() function in the Win32 API

return value

```
BOOL   ReadFile c  (HANDLE        file,
                    LPVOID        buffer,
                    DWORD         bytes To Read,
                    LPDWORD       bytes Read,
                    LPOVERLAPPED  ovl);
```

function name

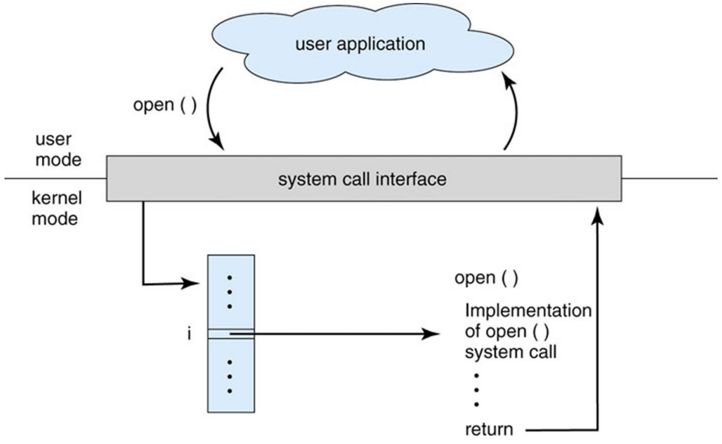
parameters

- A description of the parameters passed to ReadFile()
  - HANDLE file - the file to be read
  - LPVOID buffer - a buffer where the data will be read into
  - DWORD bytesToRead - the number of bytes to be read into the buffer
  - LPDWORD bytesRead - the number of bytes read during the last read
  - LPOVERLAPPED ovl - indicates if overlapped I/O is being used

System Call Implementation

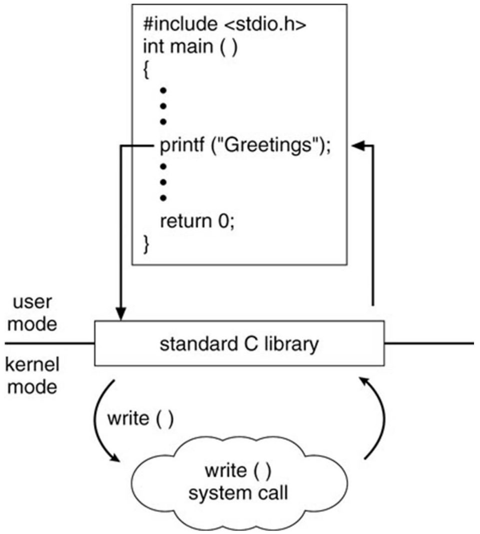
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
  - Managed by run-time support library (set of functions built into libraries included with compiler)

API - System Call - OS Relationship



Standard C Library Example

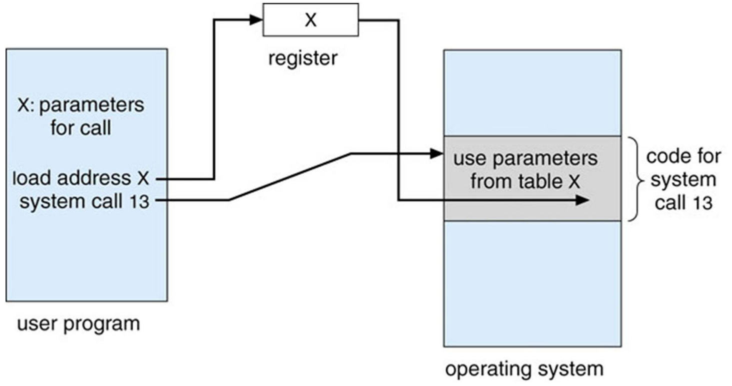
- Managed by run-time support library (set of functions



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed*, onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Examples of Linux System Calls

	Linux
Process Control	fork() execve() wait()
Memory Control	sbrk() brk() mmap()
File Manipulation	open() read() write() close()
Device Manipulation	mount() read() ioctl()
Information Maintenance	getpid() getimeofday()
Communication	pipe() socket() connect()
Protection	chown() chmod() setuid()

Linux Manual: "bellosa@i30s5: > man 2 fork"

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - Program loading and execution
  - File manipulation
  - Status information
  - Device configuration
  - Communications
- Most users' view of the operation system is defined by system programs, not the actual system calls

Linux Manual: "bellosa@i30s5: > man 1 bash"

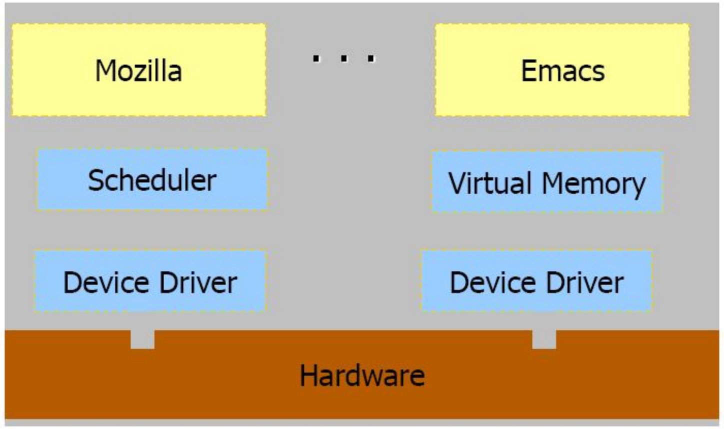
Design Objectives

- User goals and System goals
  - User goals - operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals - operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

➜ Important principle to separate
Policy: What will be done? How to reach a goal?
Mechanism: How to do it?
Mechanisms determine how to do something, policies decide what will be done.

- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

Monolithic Systems

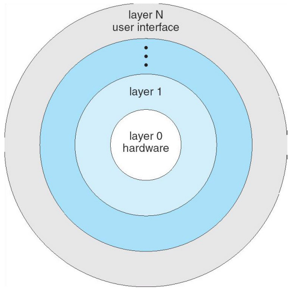


Monolithic Systems: Pros and Cons

- Advantages
  - Well understood
  - Easy access to all system data (they are all shared)
  - Cost of module interactions is low (procedure call)
  - Extensible via interface definitions
- Disadvantages
  - No protection between system and application
  - Not stable or robust
- Examples
  - uCLinux, PalmOS, VxWorks, OSEK/VDX, eCos

Layered Systems

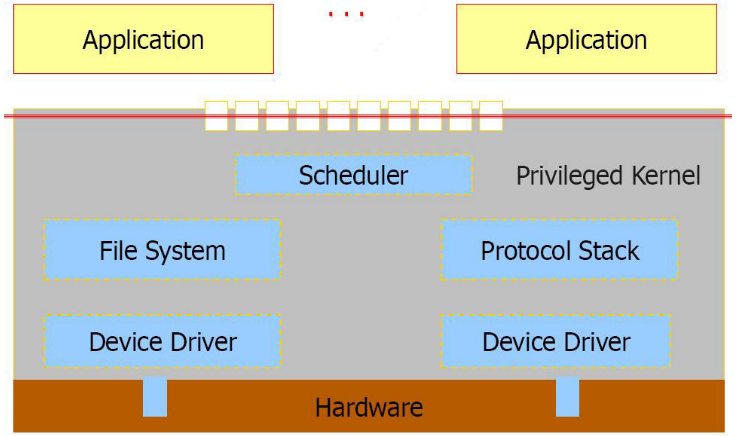
- System is divided into many layers (levels)
  - Each layer uses functions (operations) and services of lower layers
  - Bottom layer (layer 0) is hardware
    - Easier migration between platforms
    - Easier evolution of hardware platform
  - Highest layer (layer N) is the user interface
  - Lower layers implement mechanisms
  - Upper layers implement policies (mostly)



Layered Systems: Pros and Cons

- Advantages
  - Each layer can be tested and verified independently
  - Correctness of layer N only depends on layer N-1
  - ➜ Simpler debugging/maintenance
- Disadvantages
  - Just unidirectional protection
  - Mutual dependencies (e.g., calls between process, memory and file management) prevent strict layering
    - Need to reschedule processor while waiting for paging
    - May need to page in information about tasks
    - Memory would like to use files for its backing store
    - File system requires memory services for its buffers
- Examples
  - THE (Dijkstra), Multics(GE), VOCOS(EWSD)

Monolithic Kernels



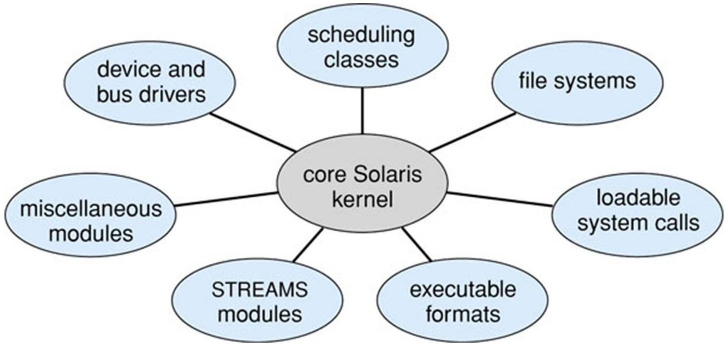
Monolithic Kernels: Pros and Cons

- Advantages:
  - Well understood
  - “Good performance
  - Sufficient protection between applications
  - Extensible via interface definitions and static/loadable modules
    - Uses object-oriented approach
    - Each core component is separate
    - Each talks to the others over known interfaces
    - Each is loadable as needed within the kernel
- Disadvantages:
  - No protection between kernel components
  - Side-effects by undocumented interfaces
  - Complexity due to high degree of interdependency
- Examples
  - Linux, Solaris

1969 First Unix by Ritchie & Thompson @ Bell Labs

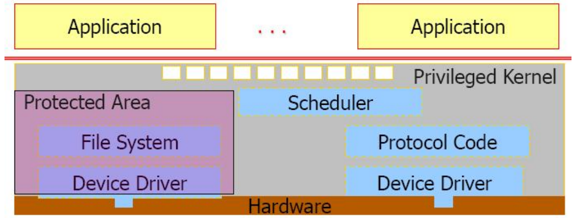
- DEC PDP-7
  - 18-Bit processor
  - $ 72000
  - 8 KB for Unix OS
  - 16 KB user memory for applications

Solaris Modular Approach



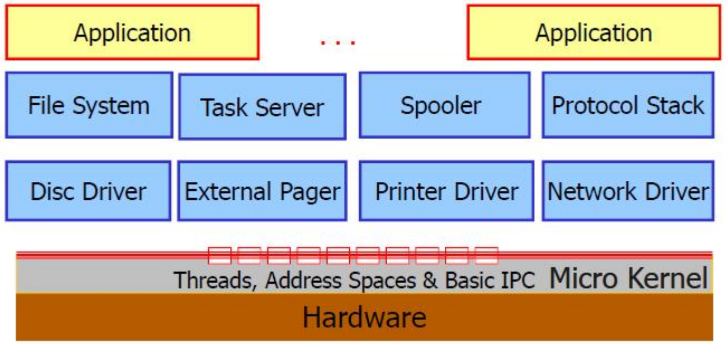
Approaches tackling Complexity and Fault Isolation

- Safe kernel extensions
  - SPIN - safe programming language (Modula 3) @ U of Washington
  - Spring - OO design @ SUN Microsystems
  - VINO - sandboxing @ Harvard

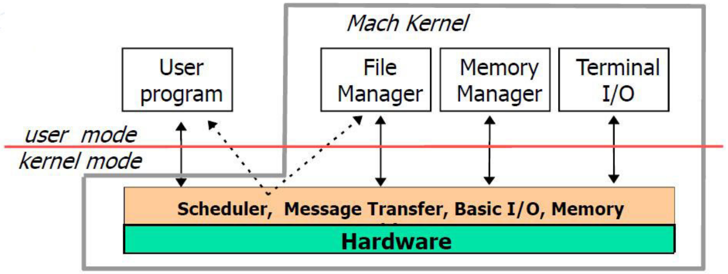


- Exokernel@MIT
  - Kernel offers multiplexing of raw HW
  - All other control is done at application level
- Microkernels
  - MACH @ CMU, L4 @ KIT, EROS, Pebbles, QNX Neutrino

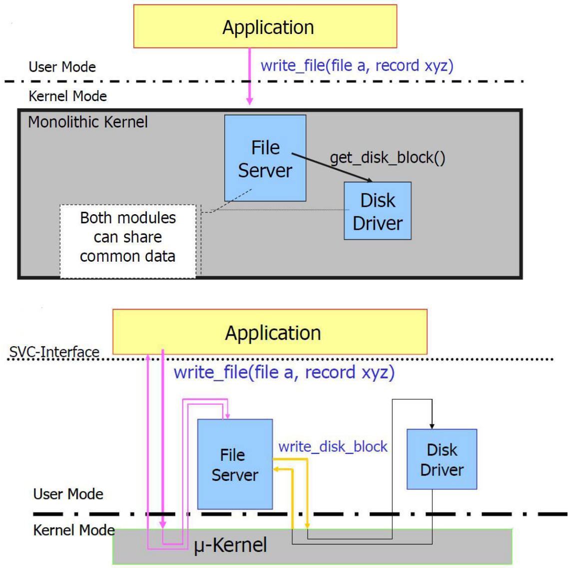
Microkernel Systems



MACH Microkernel



Architectural Cost Monolithic vs. Micro-Kernel

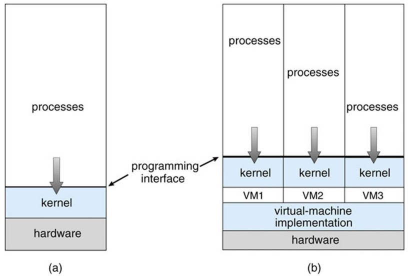


Microkernels: Pros and Cons

- Advantages:
  - Easier to test/prove/modify
  - Improved robustness & security
    (each system component in user level is protected from itself)
  - Improved maintainability
  - Coexistence of several APIs
  - Natural extensibility
    (add a new server, delete a no longer needed old server)
- Disadvantages:
  - Additional decomposing
  - Expensive to re-implement everything using a new model
  - Communication (IPC-) overhead ➜low performance
  - Bad experiences (2 B$ loss) with IBMs Workplace OS (1991-1995)
    1 kernel based on Mach 3.0 for OS/2, OS/400, AIX, Windows, ···

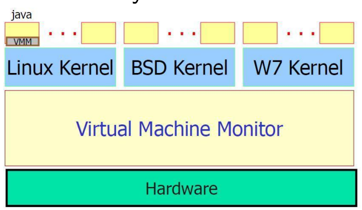
Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system host creates the illusion that a process has its own processor and (virtual memory)
- Each guest is provided with a (virtual) copy of the underlying computer

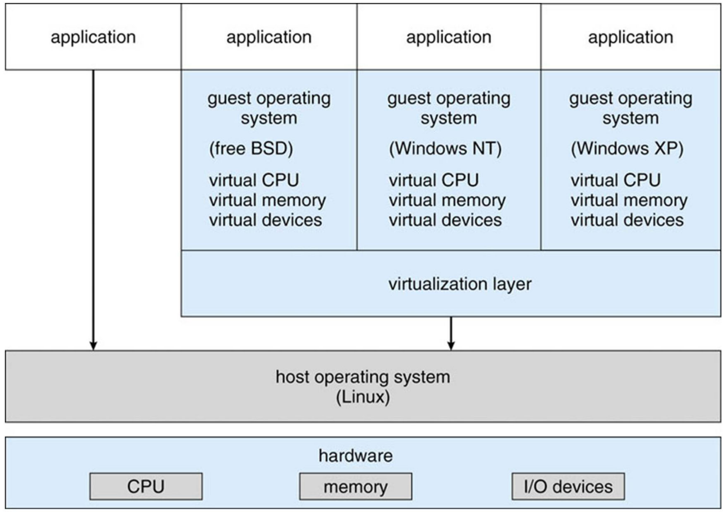


Virtual Machines Benefits

- Multiple execution environments (different operating systems) can share the same hardware
- Protect from each other
- Some sharing of file can be permitted &controlled
- Communmicate with each other & other physical systems via networking
- Useful for development, testing
- Consolidation of many low-resource use systems
- "Open Virtual Machine Format"(OVF), allows a VM to run within many different virtual machine (host) platforms

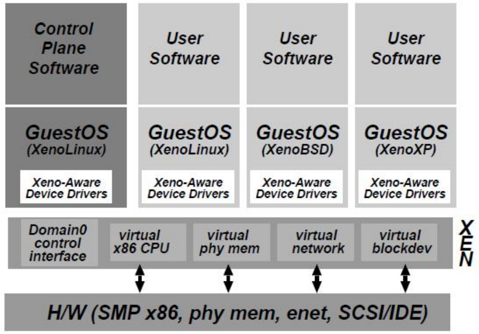


Example: VMware Architecture



Para-Virtualization

- Presents guest with system similar but not identical to hardware
- Guest must be modified to run on paravirtualized hardware (e.g., XEN)



- Guest can be an OS, or in the case of Solaris 10 applications

Solaris 10 with 2 Containers