Course "Operating System Security"

Prof. Dr.-Ing. Ahmad-Reza Sadeghi
Dipl.-Ing. Biljana Cubaleska

Chair for System Security
http://www.trust.rub.de
Ruhr-University Bochum

November 3, 2009

Organisational Issues

- **Lecture**: Prof. A.-R. Sadeghi
  - Tuesdays: 10.15 - 11.45 am
- **Exercises**: Team of System Security Group
  - Tuesdays: 8.30 - 10.00 (not every tuesday)
  - Start: 3. November 2009
  - Each exercise consists of
    - **theoretical part**: Homework (will also be discussed in the lab)
    - **practical part**: Should be done in the Lab for Operating System Security and Trusted Computing (IC 4/31)
- **Exam performance**:
  - **80%** from the written examination on the end of the semester
  - **20%** from the exercises

Organizational Issues

- Course home page: http://www.ei.rub.de/studierende/lehrveranstaltungen/et/232/
    - All relevant information (e.g., literature recommendation)
    - Files with slides, assignments
    - Lab infos and schedule
- Prof. Sadeghi will be supported by his assistants
    - Biljana Cubaleska
    - Hans Löhr
    - Marcel Winandy
    - several hiwis in the lab
- All contact data can be found on http://www.trust.rub.de

Rules of Behavior

- **Goals**
  - Unsolicited learning
  - Good and quietly lerning environment
- **Rules during the lecture and exercises**
  - Handys should be swiched-off
  - Laptops should be swiched-off
  - No privat conversations with the fellow students
  - No eating
  - No delays

Recommended Literature

- **Books**
  - [Jaeger] Trent Jaeger: **"Operating System Security"**
    Morgan & Claypool Publishers, 2008
    Examines past research that outlines the requirements for a secure operating system and research that implements example systems that aim for such requirements
  - [Gasser] Morrie Gasser: **"Building a Secure Computer System"**
    Available online as pdf-file
    Gives a very nice overview of the basic security concepts; Old, but still very valuable
  - [Silberschatz] Silberschatz and Galvin: **"Operating System Concepts"**
    Addison Wesley, 2006
    Gives a very nice overview of operating system concepts, but the focus is not on the security concepts

Recommended Literature (cntd.)

- **Books (cntd.)**
  - [Tanenbaum] Andrew S. Tanenbaum: **"Modern Operating Systems"**
    Prentice Hall International, 2001
    Gives a very nice overview operating systems design and functionalities, but the focus is not on the security concepts
  - [Palmer] Michael Palmer: **"Operating Systems Security"**
    Thomaon Course Technology, 2004
    Discussion based not on concepts, but more on administrative security measures in some operating systems
- **Some original papers**
  - Will be listed on the course web-site and later in the slides

Part I

Introduction to Copmputer Security

1. 1.1 What is Computer Security?
2. 1.2 Why Systems are not Secure?
3. 1.3 General Concepts
4. 1.4 Design Techniques

Computer Security

- **Traditionally focused on the physical machine**
  - To prevent theft of or damage to the hardware
  - To prevent theft of or damage to the information
  - To prevent disruption of service
- **Today, the value of data is greater than the value of hardware**
  - Thus, computer security focuses today on **information security**

Security Targets

1. **Secrecy** (or confidentiality)
   - Protecting information from unauthorized disclosure
2. **Integrity**
   - Protecting information from unauthorized modification or destruction
3. **Availability** (prevention of Denial of Service)
   - Prevention of temporary reduction in system performance, a system crash or a major crash with permanent loss of data

Security relates to secrecy first, integrity second, and denial of service a distant third.

- "I dont care if it works, as long it is secure"

"Secure System"

- **Any computer system can only be secure with respect to some specific policy that defines what is allowed in the system**

Evaluation Criteria for System Security

- **Orange Book**
  - Developed by the U.S. Department of Defense
  - The document employs the concept of a **Trusted Computing Base (TCB)**
    - A combination of computer hardware and an operating system that supports untrusted applications and users
  - The document gives his own definition of computer security by introducing layers of trust
- Today, internationally recognized standard is **Common Criteria**

1. 1.1 What is Computer Security?
2. 1.2 Why Systems are not Secure?
3. 1.3 General Concepts
4. 1.4 Design Techniques

Computer Security: Technology

- Major technological advances in computers raises **new security threats** that require new security solutions (e.g., Trusted Computing)
- New technologies should be accompanied by **integrated security strategies**!
- There has long been the perception that true computer security can never be achieved in practice
  - The reasons for the supposed failure are manifold
    - Programs intended for research have been wrongly criticized for not fulfilling needs of production systems
    - Researchers and developers promise more than they can deliver
    - Developments are often targeted to specific operating system, etc.
- Complexity problem

Computer Security: Implementation

- **Security vs. correctness**
  - The notions are not sysonymes
  - It is easier to build a secure system than to build a correct system
- **How many operating systems are correct and bug-free?**
  - For all operating systems vendors must periodically issue new releases
  - Bugs can usually be circumvented, but a single security hole can render all of the systems security controls
  - The important fact is not the likelihood of a flaw (which is high), but the likelihood that a penetrator will find one (which we hope is very low)

Computer Security: Functionality

- **Security seems to be a hurdle/annoying for users**
  - Security measures often interfere with an honest users normal job
  - Vendors often implement security enhancements in response to specific customer demands
  - Many customers take it upon to themselves to fix security problems at their own sites

Computer Security: Why it fails

- **Misconceptions of "security solutions"**
  - Misconceptions can have a serious negative effect on the overall progress towards achieving reasonable security
  - Example: Encryption is useful, but it does not address the general computer security problem
    - File encryption does nothing to increase the level of trust in the operating system (OS)
    - If you do not trust your OS to protect your files, you cannot trust it to encrypt your files or to protect the encryption keys properly!
- **Weakness in architecture and design** (both in hardware and software)
- **High complexity of common operating systems** (monolithic design)

Need for Secure Hardware and Software

- **Hardware**
  - Even a secure operating system cannot verify its own integrity (another party is needed)
  - Secure storage
  - DMA (Direct Memory Access) control
    - Isolation of security-critical programs
  - Hardware-based random numbers
    - Fundamental to cryptography
- **Software (operating systems)**
  - Hardening, e.g., Secure Linux (SE Linux)
    - Still too complex and large TCB (Trusted Computing Base)
  - Complete new design
    - E.g., Trusted Mach, EROS (Extremely Reliable Operating System), Singularity (Microsoft)
    - Compatibility problem, less market acceptance
  - Secure Virtual Machine Monitors
    - Allow reuse of legacy software

Computer Security: User

- **Social engineering**
  - Many organizations believe that computer security technology is irrelevant to real-world problems, because all recorded cases of computer abuse and fraud are non-technical
  - But these organizations often fail to recognize that the computer can protect against flawed procedural controls
  - E.g., a computer system can restrict the access to information, so that the user only can access information he really needs to fulfill his task

1. 1.1 What is Computer Security?
2. 1.2 Why Systems are not Secure?
3. 1.3 General Concepts
4. 1.4 Design Techniques

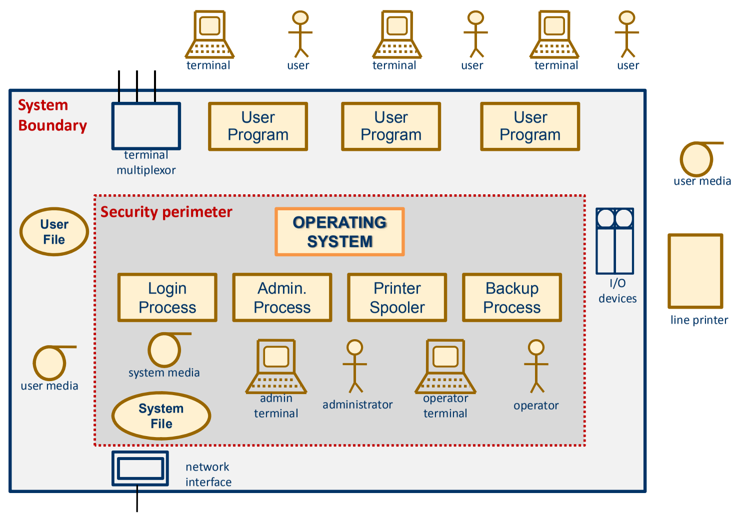
Internal and External Security

- **Internal security controls**
  - Implemented **within the hardware and software** of the system
  - Internal controls can only be effective if they are accompanied by adequate external security controls
- **External security controls**
  - Govern **physical access** to the system
  - Cover all activities for maintaining security of the system that the system itself cannot address

External Controls

- **Physical security**
  - E.g., locked rooms or guards
  - Physical security controls alone cannot address the security problems of **multiuser distributed systems**
- **Personnel security**
  - A level of **security clearance** is assigned to individuals
  - Employer decides whom to trust
- **Procedural security**
  - Covers the processes of granting people access to machines, input/out handling (e.g., printouts), installing system software, attaching user terminals, etc.

System Boundary and Security Perimeter (1) [Gasser]



System Boundary and Security Perimeter (2)

- **System boundary**
  - Everything inside the system is protected by the system, and everything outside is unprotected
  - Specifying the interface between the system and the outside world
- **Security perimeter**
  - In a system there exists security-relevant components and non-security-relevant components
  - The **security perimeter is an imaginary boundary** between the two types of components inside a system
  - Within the security perimeter lies the OS and the hardware, outside the perimeter user programs, data, and terminals

Handling Users

- The **user** is the entity whose information the system protects and whose access to information the system controls
  - User can in general be a program
- The system must assume that the user is **trusted** not to disclose (own data) willfully to unauthorized entities
- Making reasonable decisions on whether to grant or deny access to an entity the system must **identify and authenticate** users
  - The act of associating a unique user ID with a program is called **identification** (e.g., names which are usually public)
  - The act of associating the real user with the user ID is called **authentication** (proving the identity of the user, e.g., by passwords which must be kept secret)
  - Within the system, a user gets an **authorization** (according to a policy) what he is allowed to do and what not

Subjects and Objects

- The term **subject** is used to identify a running process (a program in execution)
  - Each subject assumes the identity and the privileges of a single principal
- The term **object** generally refers to a passive entity (file or a record in a database)
  - However, object may indicate an active device from the systems resource pool (network printer or a programmable service that is managed as a resource)

Access Control and Security Policy

- **Access control**
  - Consists of three tasks
    - Authorization: Determining which subjects are entitled to have access to which objects
    - Determining the access rights (read, write, execute, etc.)
    - Enforcing the access rights
- **Security policy**
  - Consists of a precise **set of rules** for determining authorization as a basis for making access control decisions

Categories of Trust

- Computer programs are not trustworthy in general
- **Software may be grouped into three categories of trust**
  - **Trusted**: The software is responsible for enforcing security
  - **Benign**: The software is not responsible for enforcing security but uses special privileges, so it must be trusted not to violate the rules intentionally
  - **Malicious**: The software is of unknown origin and must be treated as malicious and likely to attempt actively to subvert the system
- Classifying software
  - Most software is benign, whether it is written by a good or an incompetent programmer
  - Software is often not trusted, because it is not responsible for enforcing security of the system
  - In environments with extremely sensitive information the benign and malicious software are assigned to a single group: untrusted

Example of Malicious Software

- **Trojan horses**
  - A program that **masquerades as a friendly program** and is used by trusted people to do what they believe is legitimate work
  - Most peoples model of how malicious programs do their damage involves a user (the **penetrator**)
  - The Trojan horse is another type of malicious program, which requires no active user
  - Since the Trojan horse runs with the privileges of a trusted user, the malicious program does not violate against the security rules of a system

Trust: Notion and Meanings

- **Trust**
  - Complicated notion studied and debated in different areas (social sciences, philosophy, psychology, computer science,)
  - Notion relating to belief in honesty, truthfulness, competence, reliability etc. of the trusted entity
  - Social trust - belief in the safety or goodness of something because of reputation, association, recommendation, perceived benefit
- **Meanings (an attempt)**
  - **Secure**: system or component will not fail with respect to protection goals
  - **Trusted**: system or component whose failure can break the (security) policy (Trusted Computing Base (TCB))
  - **Trustworthy**: the degree to which the behavior of the component or system is demonstrably compliant with its stated functionality
- Trusted Computing Group (TCG) defines a system as trusted **"[...] if it always behaves in the expected manner for the intended purpose."**

Trustworthy Systems: Primary Goals

- **Improve security of computing platforms**
- **Reuse existing modules**
  - E.g., GUI, common OS
- **Applicable for different OS**
  - No monopoly, space for innovation (small and mid-sized companies)
- **Open architecture**
  - Use open standards and open source components
  - Trustworthiness/costs/reliability/compatibility
- **Efficient portability**
- **Allow realization of new applications/business models**
  - Providing multilateral security needed for underlying applications (based on various sets of assumptions and trust relations)
  - Avoiding potential misuse of trusted computing functionalities

Trustworthy Systems: Desired Primitives

1. **Metric for code configuration**
   - I/O behavior of a machine based on an initial state
   - e.g., represented by the hash value of the binary code
     - Problematic when functionality depends on other codes not included in hashing (e.g., shared or dynamically linked libraries)
   - Sometimes the notion of code identity is used
2. **Integrity verification (attestation)**
   - Allows a computing platform to export verifiable information about its properties (e.g., identity and initial state)
   - Comes from the requirement of assuring the executing image and environment of an application located on a remote computing platform

Trustworthy Systems: Desired Primitives (cntd.)

3. **Secure storage**
   - To persist data securely between executions using traditional untrusted storage like hard drives
   - To encrypt data and assured to be the only capable of decrypting it
4. **Strong process isolation**
   - Assured (memory space) separation between processes
   - Prevents a process from reading or modifying another processs memory
5. **Secure I/O**
   - Allows application to assure the end-points of input and output operations
   - A user can be assured to securely interact with the intended application

Trust Model in the OS Development

- **The developer of secure OS must provide that their systems have a viable trust model**. This requires [Jaeger]
  - that the system TCB must mediate all security-sensitive operations
  - that the correctness of the TCB software and its data must be verified
  - verification that the softwares execution cannot be tampered by process outside the TCB
- **Problems**
  - Identifying the TCB Software itself is not a trivial task
  - Verifying the correctness of the TCB software is a very complex task
  - The system must protect the TCB software and its data

1. 1.1 What is Computer Security?
2. 1.2 Why Systems are not Secure?
3. 1.3 General Concepts
4. 1.4 Design Techniques

System Development Process

- **Steps to develop a system**
  - Define **requirements**
    - Establishing generic needs
  - Write **specification**
    - Defining precisely what the system is supposed to do, including **specification verification**, which involves demonstrating that the specification meets the requirements
  - **Implementation**
    - Designing and building the system, including **implementation verification**, which involves demonstrating that the implementation meets the specification

Notion of Architectures

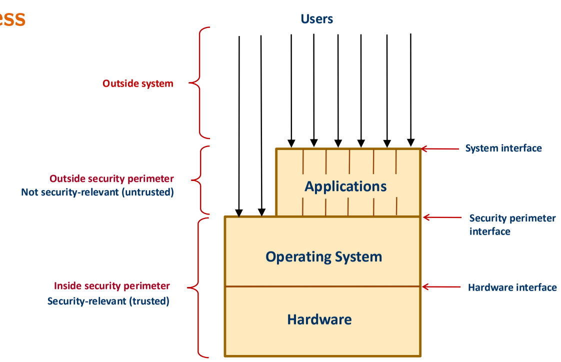
- **System architecture**
  - Description of the processes by which the system is built
  - Many of the desired characteristics that help dictate the architecture (such as reliability, maintainability, and performance) have a profound impact on the development strategy
- **Security architecture**
  - Describes how the system is put together to satisfy the security requirements

IT System Structures

- **We group systems into two types**:
  - **Computer system**, consisting of a single machine or closely coupled multiprocessors
    - Such a system must protect itself and does not rely on assistance from other systems
  - **Distributed system**, that resembles a single computer system from the outside but actually consists of multiple computer systems
- **The difference is reflected in the internal system structure and may not be apparent to users outside the system**

Generic Computer System Structure (1)

- A computer system consists of **hardware**, an **operating system**, and **application programs**
- Users generally interact only with the applications and not directly with the operating system
- Each application running on behalf of a user can be thought of as a **process**

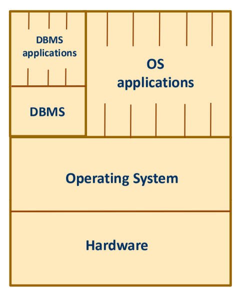


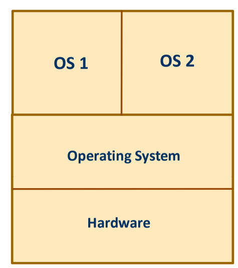
Generic Computer System Structure (2)

- **Distinction hardware and operating system**
  - In past the distinction was obvious: the operating system was implemented with **bits in memory** and the hardware with fixed **circuits**
  - However, now many machines contain **microcode** and **firmware**
- **Distinction applications and operating system**
  - More obvious
  - An operating system can be distinct clearly from the system applications or processes that are needed to support it

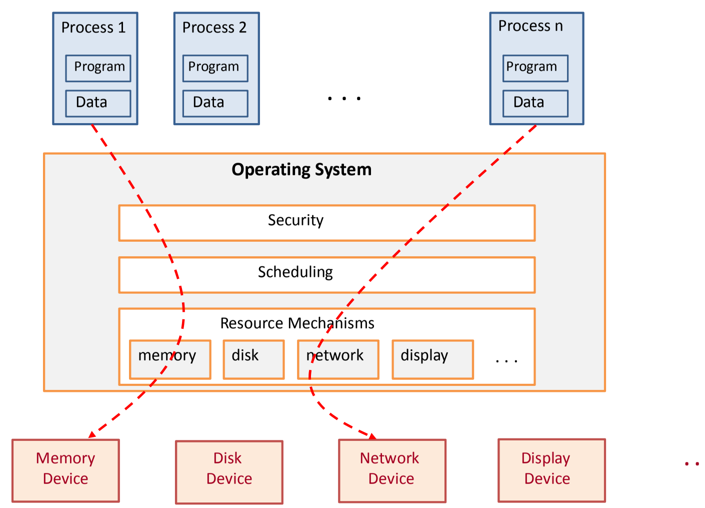
Example for Subsystems

- **Subsystem: Database Management Systems (DBMS)**
  - Often constitutes mini-operating systems of their own
  - From the perspective of the operating system, the DBMS is just another application
  - The DBMS may be responsible for enforcing its own security polic
- **Subsystem: Multi OS**





Whot does an Operating System do? [Jaeger]



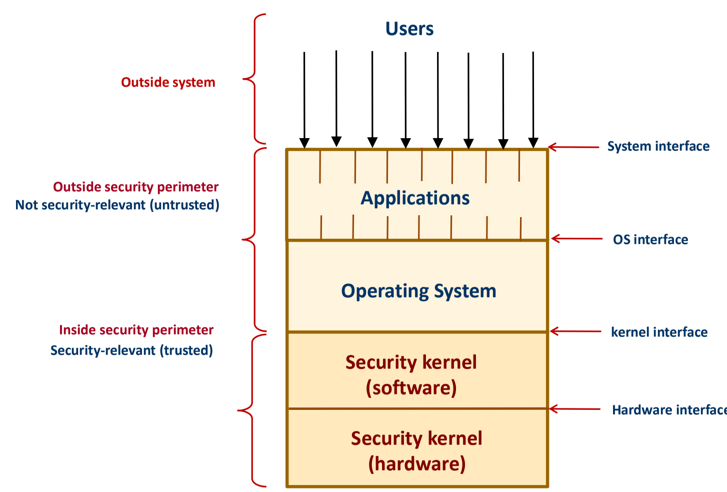
System States

- **Privileged mode**
  - May also be called **"executive"**, **"master"**, **"system"**, **"kernel"**, or **"supervisor"** mode
  - Software can execute any machine instruction and can access any location in memory
- **Unprivileged mode**
  - May also be called **"user"**, **"application"**, or **"problem"** mode
  - Software is prevented from executing certain instructions or accessing memory in a way that could cause damage to the privileged software

Reference Monitor and Security Kernel

- **Security kernel approach**
  - Building an operating system that avoids the security problems inherent in conventional design
  - Uses the concept of a reference monitor
    - **Combination of hardware and software responsible for enforcing the security policy of the system**
  - Access decisions are based on the information in an access control database

Reference Monitor and Security Kernel (cntd.)



Literature

- [Jaeger] Trent Jaeger: **"Operating System Security"**
  Morgan & Claypool Publishers, 2008
- [Gasser] Morrie Gasser: **"Building a Secure Computer System"**
  Available online as pdf-file

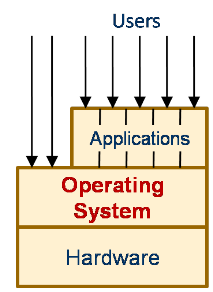
Part II

Basic Concepts of Operating Systems

1. 2.1 Computer System Operation
2. 2.2 Operating System Responsibilities
3. 2.3 Operating System Services
4. 2.4 System Calls and System Programms
5. 2.5 Operating System Architectures
    - OS Design and Implementation
    - Monolitic OS Structure
    - Modular OS Structure
    - Microkernel OS Structure

The Operating System in the Computer System

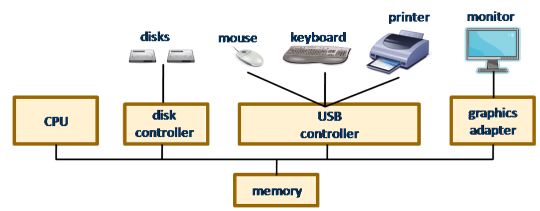
- **Components of a computer system**
  - Hardware
    - Provides basic computing resources as CPU, memory, I/O devices
  - Operating system (OS)
    - Controls and coordinates use of hardware among various applications and users
  - Application programs
    - Use system resources to solve computing problems
  - Users
    - People, machines, other computers



Operating System Definition

- **Operating system is a program that acts as an intermediary between a user of a computer and the computer hardware**
- **The Operating System is a control program and a resource allocator of the computer system**
  - OS as a **control program**: Controls execution of programs to prevent errors and improper use of the computer
  - OS as a **resource allocator**: Manages all resources and decides between conflicting requests for efficient and fair ressource use
- **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 Operation



- I/O devices and the CPU can execute **concurrently**
- 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**
  - Each device controller is in charge of a particular device type (e.g., disk driver)
  - Each device controller has a local buffer

Interrupts

- **Common functions of interrupts**
  - When CPU is interrupted, it transfers control to the interrupt service routine generally, through the **interrupt vector** (IV), 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
- **Trap (Unterbrechung)**
  - Software-generated interrupt caused either by an error or a user request
- **Modern operating systems are interrupt driven**

Input/Output (I/O) Structure

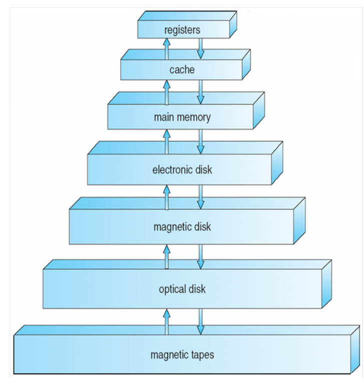
- **Synchronous I/O: After I/O starts, control returns to user program only upon I/O completion**
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- **Asynchronous I/O: After I/O starts, control returns to user program without waiting for I/O completion**
  - System call request to the operating system to allow user to wait for I/O completion
  - Device-status table contains entry for each I/O device indicating its type, address, and state (needed to keep track of many I/O requests at the same time
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

Storage Structure

- **Main storage: Main memory (Random Access Memory, RAM) and the registers built into the processor**
  - Computer programs must be in the main memory in order to be executed
  - This is the only storage that the CPU can access directly
- **Secondary storage: Extension of main memory that provides large nonvolatile storage capacity**
  - Magnetic disks, magnetic tapes, CDs, USBs, etc.
  - 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

- **The storage systems for computers can be organized in a hierarchy, according to the speed and cost**
  - The higher levels are expensive, but they are fast
  - As we move down, the cost per bit decreases, but the access time increases
- **In addition to having different speed and cost, the various storage systems can be**
  - **Volatile** (losses its contents when the power to the device is removed), and
  - **Nonvolatile** (does not loose its contents when to the device is removed)



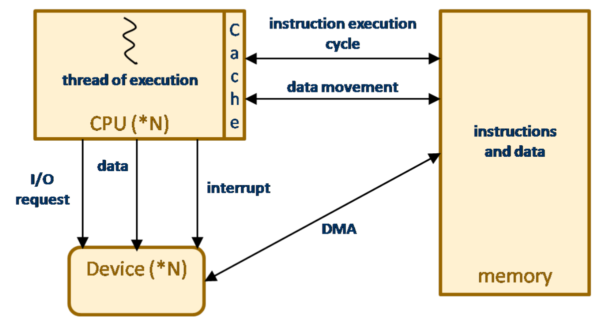
Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- **Caching: Information in use 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 copied to cache and used there
- Cache is smaller than storage being cached
- Cache management is important design problem
  - E.g., Cache size, replacement policy

Direct Memory Access (DMA)

- **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
- **Might lead to security problems**

How a Modern Computer Works



Computer Start Up (Boot)

- **Bootstrap program: Initial program which is run for a computer to start running, e.g., when it is powered up or rebooted**
  - Initializes all aspects of the system (CPU registers, device controllers, memory contents)
  - Typically stored in read-only memory (ROM) or in electronically programmable ROM (EPROM), generally known as **firmware**
  - Tends to be simple
- **Computer boot process**
  - The bootstrap program loads the operating system and starts executing that system
  - OS then starts executing the first process, such as init, and waits for some event to occur
  - An event is usually signaled by an interrupt from either the hardware or the software
    - Hardware may trigger an interrupt at any time by sending a signal to the CPU
    - Software may trigger a trap by executing a system call (Systemaufruf)

1. 2.1 Computer System Operation
2. 2.2 Operating System Responsibilities
3. 2.3 Operating System Services
4. 2.4 System Calls and System Programms
5. 2.5 Operating System Architectures
   - OS Design and Implementation
   - Monolitic OS Structure
   - Modular OS Structure
   - Microkernel OS Structure

Operating System Responsibilities

- **Management of the computer resources**
  - Process management (CPU time)
  - Memory management
  - Storage management
  - Input/Output management

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
- **Typically system has many processes**
  - Concurrency by multiplexing the CPUs among the processes
- **Responsibilities of OS in the 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

- **Memory = main memory**
  - All data are in the main memory before and after processing
  - All instructions come into the main memory in order to execute
- **Responsibilities of OS in the memory management:**
  - Keeps track of which parts of memory are currently being used and by whom
  - Decides which processes (or parts thereof) and data to move into and out of memory
  - Allocates and deallocates memory space as needed
  - Determines what is in memory when optimizing CPU utilization and computer response to users

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)
- **Storage management = file system management**
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
- **Responsibilities of OS in the storage management:**
  - Creating and deleting files and directories
  - Primitives to manipulate files and directories
  - Mapping files onto secondary storage
  - Backup files onto stable (non-volatile) storage media
  - Free space management
  - Storage allocation

I/O Management

- **I/O subsystem**
  - Hardware devices for input and output of information to the computer system
- **Responsibilities of OS in the I/O management:**
  - Buffering (storing data temporarily while it is being transferred),
  - Caching (storing parts of data in faster storage for performance), and
  - Spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices
  - Sometimes, hiding peculiarities of hardware devices from the user is one purpose of OS

1. 2.1 Computer System Operation
2. 2.2 Operating System Responsibilities
3. 2.3 Operating System Services
4. 2.4 System Calls and System Programms
5. 2.5 Operating System Architectures
   - OS Design and Implementation
   - Monolitic OS Structure
   - Modular OS Structure
   - Microkernel OS Structure

Different Views of OS

- **Since an OS provides the environment within programs are executed, we can view an OS from several vantage points**
  - One view focuses on the **services that the system provides to users, processes, and other systems**
  - Another view focuses on the interfaces that it provides to the users and programmers
  - Third view focuses on the system components and their interconnections
- **In the following we will briefly describe two sets of services provided by an operating system:**
  - Services providing functions that are helpful to the user
  - Services providing functions for ensuring the efficient operation of the system itself via resource sharing

OS Services Providing Functions Helpful to the User

1. **User interface**
   - Almost all operating systems have a user interface (UI)
   - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
2. **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)
3. **I/O operations**
   - A running program may require I/O, which may involve a file or an I/O device
4. **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.

OS Services Providing Functions Helpful to the User (cntd.)

5. **Communications**
   - 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)
6. **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 users and programmers abilities to efficiently use the system

OS Services providing functions for ensuring the efficient operation of the system itself

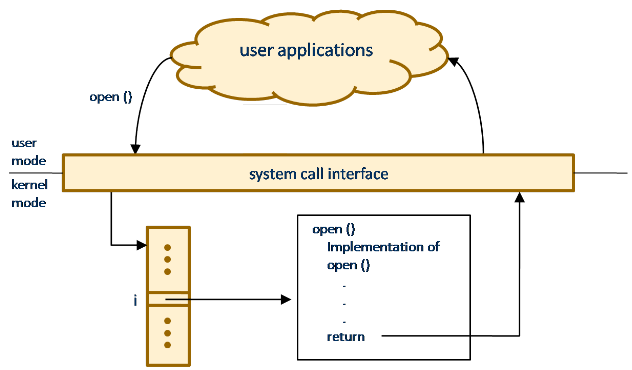
1. **Resource allocation**
   - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
   - Many types of resources; Some (e.g., CPU cycles, main memory, and file storage) may have special allocation code, others (e.g., I/O devices) may have general request and release code
2. **Accounting**: Keep track of which users use how much and what kinds of computer resources
3. **Protection and security**
   - The owners of information stored in a system 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
   - 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

1. 2.1 Computer System Operation
2. 2.2 Operating System Responsibilities
3. 2.3 Operating System Services
4. 2.4 System Calls and System Programms
5. 2.5 Operating System Architectures
   - OS Design and Implementation
   - Monolitic OS Structure
   - Modular OS Structure
   - Microkernel OS Structure

System Calls

- **System calls are 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**
  - Most common APIs
    - Win32 API for Windows
    - POSIX API for POSIX-based systems (including all versions of UNIX, Linux, and Mac OS X)
    - Java API for the Java virtual machine (JVM)
- **System call implementation**
  - 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 are hidden from programmer by API

API - System Call - OS Relationship

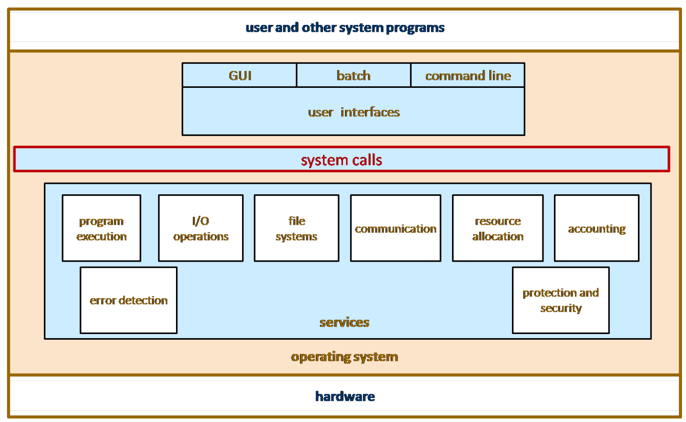


Typically, a number associated with each system call.
System-call interface maintains a table indexed according to these numbers.

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, there may be more parameters than registers
  - Parameters are 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 OS
- **Block and stack methods do not limit the number or length of parameters being passed**

System Calls in the System Architecture



Types of System Calls

- **Process control**
  - Examples: End, abort, load, execute, create process, terminate process, get process attributes, set process attributes, wait for time, wait event, signal event, allocate and free memory
- **File management**
  - Examples: Create file, delete file, open, close, read, write, reposition, get file attributes, set file attributes
- **Device management**
  - Examples: Request device, release device, read, write, reposition, get device attributes, set device attributes, logically attach or detach devices
- **Information maintenance**
  - Examples: Get time or date, set time or date, get system data, get system data, get process (file, or device) attributes, set process (file, or device) attributes
- **Communications**
  - Examples: Create or delete communication connection, send or receive messages, transfer status information, attach or detach

Examples of Windows and Unix System Calls

	Windows	Unix
Process control	CreateProcess ()	fork ()
	ExitProcess ()	exit ()
	WaitForSingleObject ()	wait ()
File manipulation	CreateFile ()	open ()
	ReadFile ()	read ()
	WriteFile ()	write ()
	CloseHandle ()	close ()
Device manipulation	SetConsoleMode ()	ioctl ()
	ReadConsole ()	read ()
	WriteConsole ()	write ()
Information maintenance	GetCurrentProcessID ()	getpid ()
	SetTimer ()	alarm ()
	Sleep ()	sleep ()
Communication	CreatePipe ()	pipe ()
	CreateFileMapping ()	shmget ()
	MapViewOfFile ()	mmap ()
Protection	SetFileSecurity ()	chmod ()
	InitializeSecurityDescriptor ()	umask ()
	SetSecurityDescriptorGroup ()	chown ()

System Programms

- **Programs in the system**
  - **System programs**: Provide a convenient environment for program development and execution
  - **Application programs** (also known as system utilities): Programs that are useful in solving common problems or performing common operations, e.g., web browsers, compilers, database systems, etc.
- **In the logical computer hierarchy, system programs are between the OS and application programs**
  - Most users view of the OS is defined by system programs, not the actual system calls
- **System programs have different complexity**
  - Some of them are simply user interfaces to system calls
  - Others are considerably more complex

Types of System Programms

- **File management**
  - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
  - Some ask the system for info, e.g., date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information
- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text

Types of System Programms (cntd.)

- **Programming-language support**
  - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**
  - Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications**
  - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one anothers screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

1. 2.1 Computer System Operation
2. 2.2 Operating System Responsibilities
3. 2.3 Operating System Services
4. 2.4 System Calls and System Programms
5. 2.5 Operating System Architectures
   - OS Design and Implementation
   - Monolitic OS Structure
   - Modular OS Structure
   - Microkernel OS Structure

OS Design and Implementation

- **Design and implementation of OS not solvable, but some approaches have proven successful**
- **Internal structure of different Operating Systems can vary widely**
- **Start by defining goals and specifications**
  - User goals
    - OS should be convenient to use, easy to learn, reliable, safe, secure, and fast
  - System goals
    - OS should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- **Affected by choice of hardware, type of system**

Policies and Mechanisms

- **Important principle to separate:**
  Policy: What will be done? (design)
  Mechanism: How to do it? (implementation)
  - Mechanisms determine how to do something, policies decide what will be done
- **The separation of policy from mechanism is a very important principle**
  - This allows maximum flexibility if policy decisions are to be changed later

Implementation of OS

- **Once an OS is designed, it must be implemented**
- **Traditionally, OS have been written in assembly language**
- **Now, however, they are most commonly written in higher-level languages as C or C++**
  - First, MCP (Master Control Program for Burroughs computers) was written in ALGOL
  - MULTICS was written mainly in PL/1
  - Linux and Windows XP are written mostly in C, although there are some small sections of assembly code (device drivers, register operations)

Implementation of OS Using a Higher-Level Language

- **Advantages**
  - The code can be written faster
  - The code is more compact
  - It is easier to understand and debug
  - Improvements in compiler technology will improve the generated code for the entire OS by simple recompilation
  - An OS is easier to port (to move to some other HW)
    - MS-DOS was written in Intel 8088 assembly language; So, it is available only on the Intel family of CPUs
    - Linux OS is written mostly in C and it is available on a number of different CPUs, including Intel 80x86, Motorola 680x0, SPARC, and MIPS RX00
- **Disadvantages**
  - Reduced speed
  - Increased storage requirement

Operating System Architectures

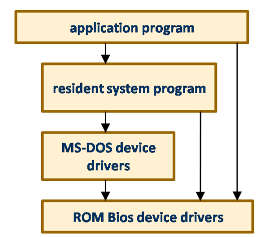
- **Modern OS are large and very complex**
  - They must be engineered carefully in order to function properly and be modified easily
  - Common approaches:
    - Monolithic system
    - Modular system (partition into small components)
- **OS architecture means how the components of an OS are interconnected and melded** into a kernel
- **Common OS architectures**
  - Simple structure
  - Layered structure
  - Microkernels
  - Modules

Simple Structure of OS

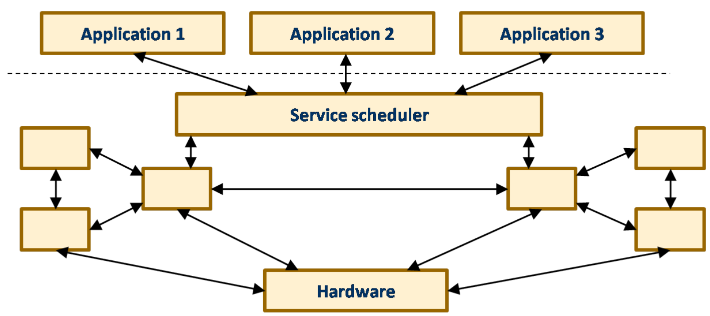
- **Many commercial systems do not have well-defined structures**
  - Frequently, such OS started as small, simple, and limited systems and then grew beyond their original scope
- **The monolithic structure is difficult to implement and maintain**
  - Security problems
- **Examples:**
  - MS-DOS
  - Original UNIX (initially limited by hardware functionality)

Structure of MS-DOS

- **MS-DOS was originally designed and implemented by a few people who had no idea that it would become so popular!!!!!**
  - It was written to provide the most functionality in the least space
  - It was not divided into module carefully
- **The interfaces and levels of functionality are not well-separated**
  - For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives
  - Because of this freedom MS-DOS is vulnerable to errant or malicious programs, causing entire system crashes when user programs fail
  - Provides no hardware protection, no dual mode (the base hardware was accessible)



Example for Monolithic Structure

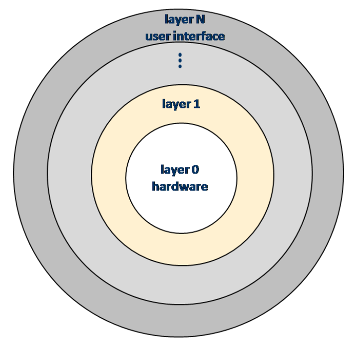


Modular Structure of OS

- **Modular systems**
  - With proper HW support, OS can be broken into pieces that are smaller and more appropriate for some tasks
  - The overall functionality and features are determined and are separated into components
- **Advantages of modular systems**
  - The OS can then retain much greater control over the computer and over the applications that make use of that computer
  - Information hiding possible: It leaves programmers free to implement the low-level routines as they see fit, provided that
    - The external interface by the routine stays unchanged, and
    - The routine itself performs the advertised task
- **A system can be made modular in many ways**
  - One method is the layered approach

Layered Approach of OS Structure

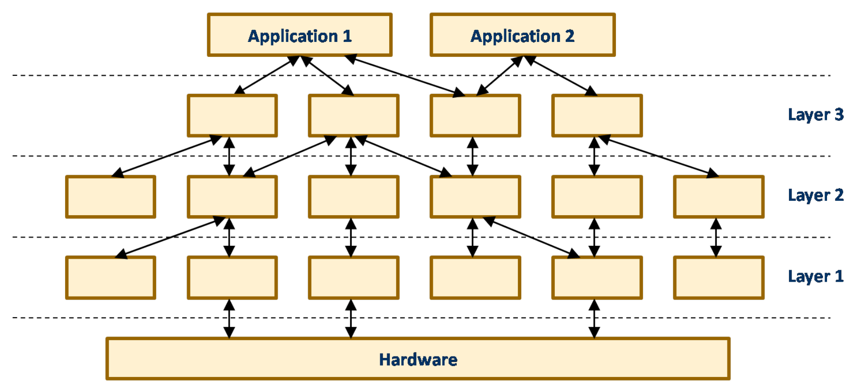
- **Operating system layer**
  - an implementation of an abstract object made up of data and the operations that can manipulate those data
- **The operating system is divided into a number of layers (levels), each built on top of lower layers**
  - The bottom layer (layer 0), is the hardware
  - The highest (layer N) is the user interface
- **How it works:**
  - Layer M can invoke functions (operations) and services of only lower-level layers
  - Functions and services of layer M can be invoked by the higher level layers



Layer Implementation

- **Each layer is implemented with only those operations provided by lower-level layers**
  - A layer does not need to know how these operations are implemented; It needs to know only what these operations do
  - Hence, each layer hides the existence of certain data structures, operations and hardware from higher-level layers

Example for Layered Structure



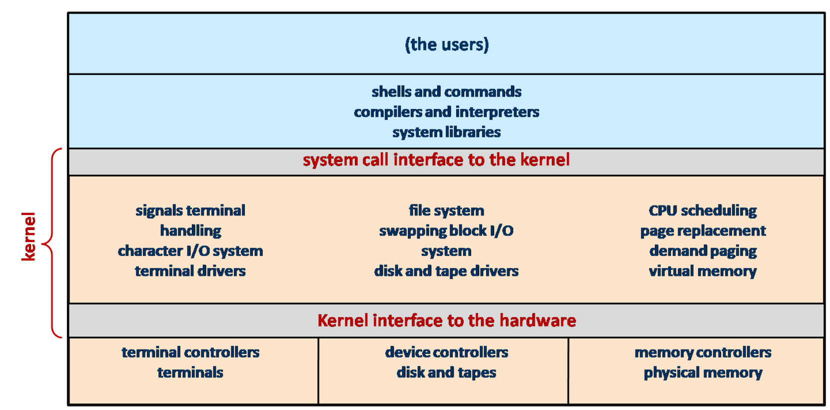
Advantages and Limitations of the Layered Approach

- **Advantages**
  - Simplicity of construction
  - Simple debugging and system verification
    - One layer can be debugged without any concern of the rest of the system
- **Limitations**
  - Major difficulty: How to define the various layers appropriately
  - Layered implementations tend to be less efficient than other types
    - E.g., when a user program executes an I/O operation, at each layer, the parameters may be modified, data may need to be passed, and so on.
    - Each layer adds overhead to the system call that takes longer than does one on a non-layered system
- **The limitations have caused small backlash against layering in recent years**
  - Fewer layers with more functionality are being designed
  - This approach provide most of the advantages of modularized code while avoiding the difficult problems of layer definition and interaction

Example: UNIX OS

- The original UNIX operating system had **limited structuring** caused by limited hardware functionality
  - We can view the traditional UNIX OS as being layered
- The UNIX OS consists of two separable parts:
  - **The kernel**
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions (a large number of functions for one level)
  - **Systems programs**
- As UNIX expanded, the kernel became large and **difficult to manage**

Traditional UNIX System Structure



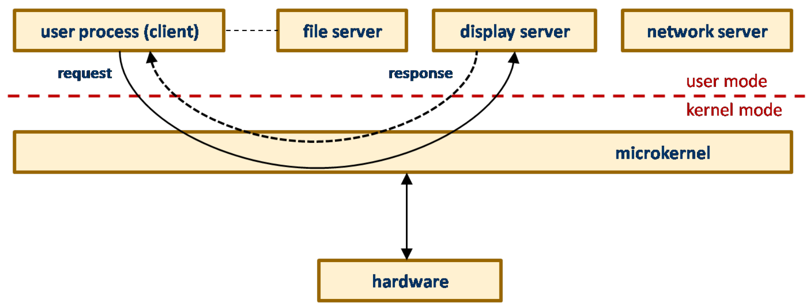
Microkernel System Structure

- **Problems with large kernels consisting of many functionalities**
  - UNIX example: As the system expanded, the kernel became large and difficult to manage
  - In the mid-1980s, at Carnegie Mellon University an OS called Mach was developed, that modularized the kernel using the microkernel approach
- **Microkernel method: Structures the OS by removing the nonessential components from the kernel and implementing them as system and user-level programs**
  - Result: Smaller kernel (microkernel)
  - Question: Which services should remain in the kernel and which should be implemented in the user space?

Microkernel Functionalities

- **Functions of the microkernel**
  - Main function: Provide a communication facility between the client program and the various services that are running in the user space
  - Other functions: Minimal process and memory management
- **Communication takes place between user modules using message passing**
- **Examples for microkernel-based systems**
  - Tru64 UNIX (formerly Digital UNIX)
  - QNX (a real-time OS)
  - Windows NT (first release)
  - Windows NT 4.0 partially redressed the performance problem by moving layers from user space to kernel space and integrating them more closely

Example for Microkernel Structure



Advantages and Limitations of Microkernel System Structure

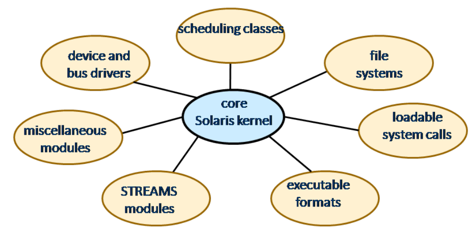
- **Advantages**
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (if a service fails, the rest of the OS remains untouched)
  - More secure (since more services are run as user process rather than a kernel process)
- **Limitations**
  - Performance overhead of user space to kernel space communication

Modular Approach of OS Architecture

- Most modern operating systems implement **modular kernel**: **The kernel has a set of core components and dynamically links in additional services either during the boot time or during the run time**
  - 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
- **Overall, similar to layers but more flexible**
- **Examples:**
  - Modern implementations of UNIX (Such as Solaris, Linux)
  - Mac OS

Solaris Modular Approach

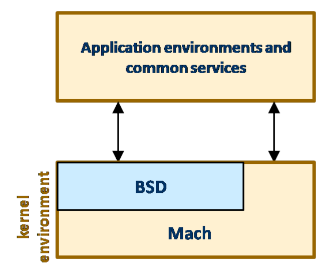
The Solaris OS is organized around a core kernel with seven types of loadable kernel modules



- Such a design allows the kernel to provide core services and also allows certain features to be implemented dynamically
  - E. g., device and bus drivers for specific HW can be added to the kernel, and support for different file systems can be added as loadable modules
  - The overall result resembles a layered system in that each kernel section has defined, and protected interfaces, but it is more flexible than a layered system in that any module can call any other module
  - The approach is like the microkernel approach in that the primary module has only core functions and knowledge how to load and communicate to other modules, but it is more efficient, because modules do not need to invoke message passing in order to communicate

Mac OS X Structure

- The Apple Macintosh Mac OS X uses a **hybrid structure**
  - The OS is structured using a layered technique
    - One layer is the **kernel environment**
    - Another layer are the **application environments**
- The kernel environment consists of
  - **Mach kernel** (providing memory management, support for remote procedure calls (RPCs), interprocess communication facilities, and thread scheduling)
  - **BSD component** (providing a BSD command line interface, support for networking and file systems and implementation of APIs)
  - **Kernel extensions** (an I/O kit for development of device drivers and dynamically loadable modules)



Literature

- [Silberschatz] Silberschatz, Galvin and Gagne: **"Operating System Concepts"**
  Addison Wesley, 2008
- [Tanenbaum] Andrew S. Tanenbaum: **"Modern Operating Systems"**
  Prentice Hall International, 2001