Introduction

1. Operating system and computer system
2. Operating systems: goals and interface
3. Operating system structures
4. Computer system operations

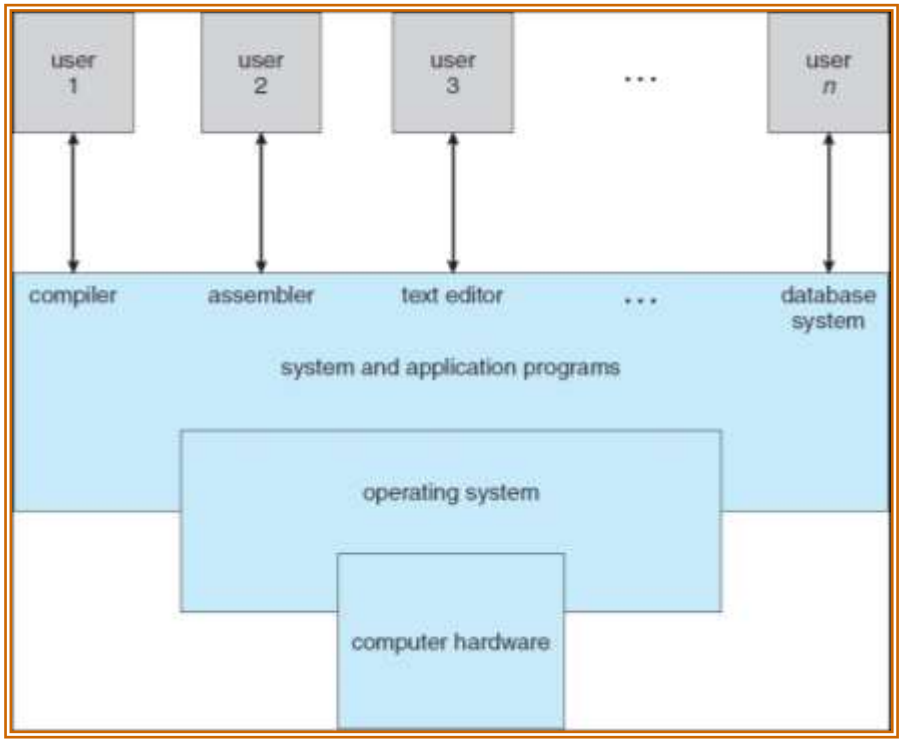
Last modification date: 20.09.2018

What is an operating system?

- No universally accepted definition. “Everything a vendor ships when you order an operating system” is a good approximation (but it varies wildly)
- A program that acts as an **intermediary** between a user of a computer and the computer hardware
- **Resource allocator** – manages computer system resources, decides between conflicting requests for efficient and fair resource use
- **Control program** - controls execution of programs to prevent errors and improper use of the computer
- **Kernel** – the one program running at all times on the computer
- Basic computer system software which:
  - executes user programs and makes solving user problems easier
  - makes the computer system convenient to use
  - uses the computer hardware in an efficient manner
- **Computer system software** which:
  - enables **management** of hardware and software resources of the system
  - Creates a process execution environment which is appropriate for the assumed mode of system operation

Computer System Structure

1. **Hardware** – provides basic computing resources (CPU, memory, I/O devices)
2. **Operating system** - controls and coordinates use of hardware among various applications and users
3. **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users
4. **Users** - people, machines, other computers



Modes of operations

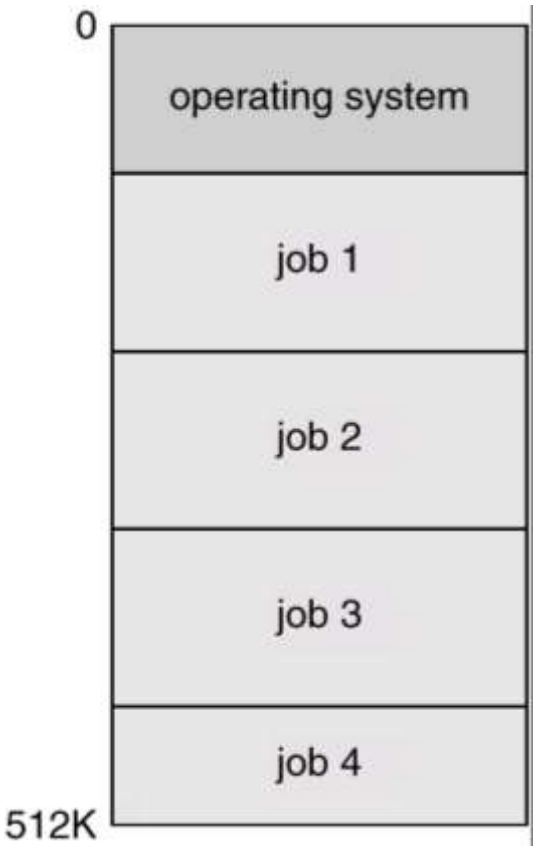
- Dominant mode of operation of computer system influences significantly operating system architecture.

- Main **modes of operation**:
  - *off-line*, *batch*
  - *on-line*, interactive
  - *real-time*

- Real systems typically support more than one mode of operation (although to a different degree).

Multiprogramming

- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so (each) CPU/core 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
- Multiprogramming requires:
  - Memory management – jobs have to have allocated/deallocated separate physical memory chunks, protected from unauthorized access
  - CPU scheduling – assignment of processor(s) to ready jobs is controlled with control commands (shell)
  - Shared devices are made available via a set of system provided I/O procedures (functions)



Time sharing and interactive systems

- **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 < 1 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 which are not loaded completely into memory
- **Interactive operation** uses direct communication between user terminal and a task which is driven by user commands. Because of slow computer-human interaction timesharing is needed to provide overall system efficiency.

Real-time systems

- **Real-time (RT) systems** are used when **rigid time requirements** have been placed on the operation of a processor or the flow of data.
- **Hard RT systems**.
  - Time requirements have to be respected absolutely (they are guaranteed). This is needed in time-critical industrial control, robotics, etc.
  - Strict time requirements of RT are **in conflict** with requirements of standard time-sharing ➔ hard real time operation cannot be strictly implemented within a universal (multi-mode) system
  - Rigid requirements prevent use of some cost-effective solutions like virtual memory, dynamic loading and other techniques which introduce unpredictable response time.
- **Soft RT systems***:*
  - Scheduling attempts to meet deadlines, but can occasionally violate them in the interest of cost-effectiveness of the whole system.
  - Soft RT can be useful in multimedia, virtual reality and other application when time response performance is important but not safety-critical.

Introduction

1. Operating system and computer system
2. Operating systems: goals and interface
3. Operating system structures
4. Computer system operations

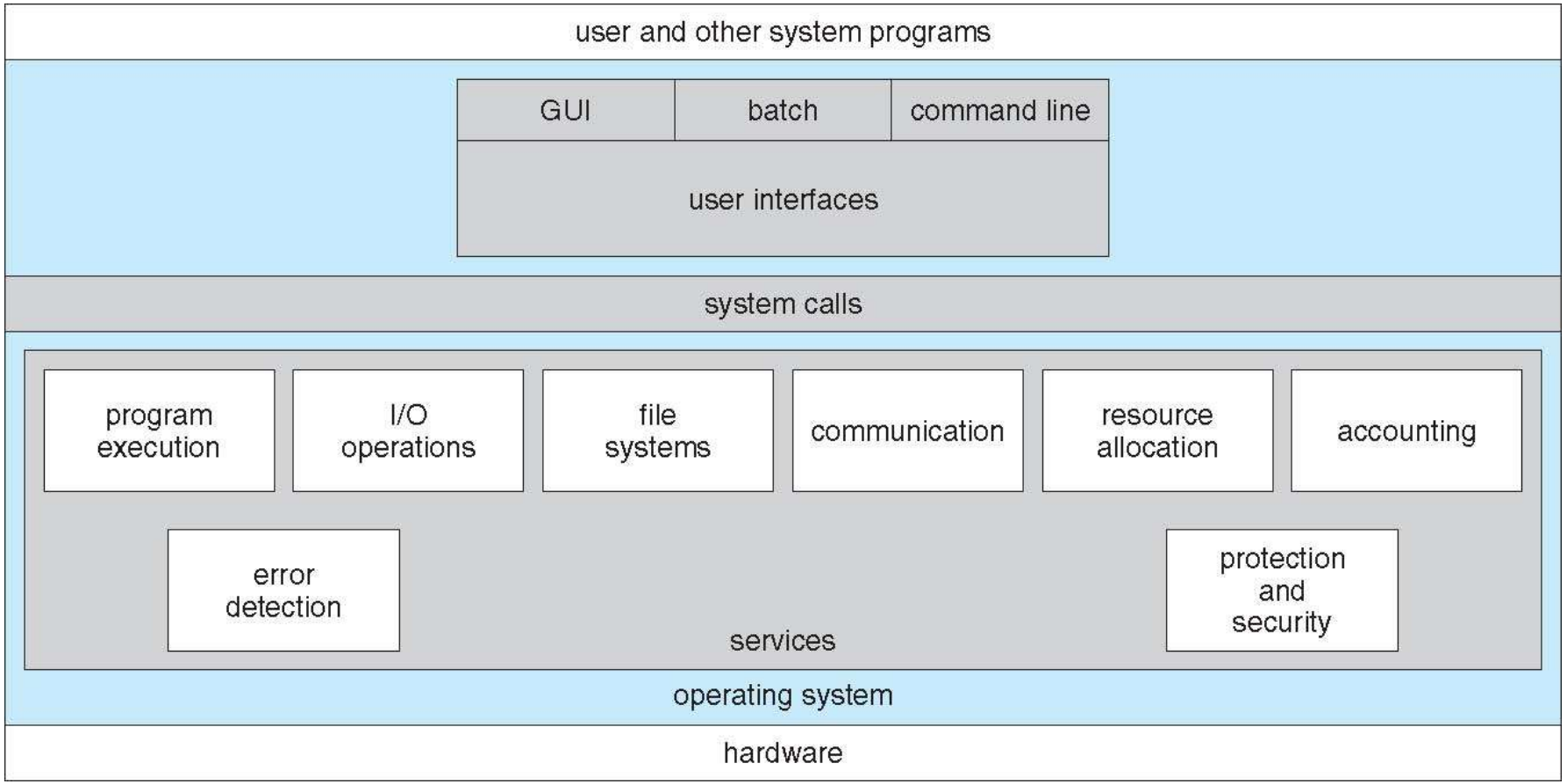
Operating system goals

- Execution of user and system programs and protection of user data
  - Users want convenience, **ease of use** and **good performance** (but might not care about **resource utilization**)
  - Shared computer (**mainframe** or **minicomputer**) must keep all users happy ☺
  - Handheld computers are resource poor ➔ optimized for **usability** and **battery life**
- Efficient computer system resource **management**
  - resource allocation/reclaiming
  - planning resource use
  - protection and security
  - resource use accounting
  - error handling

Computer system resources

- Resources managed by operating system
  - Processor(s), cores
  - Memory and other devices of the computer system
  - Information kept in the system
- Composition of system resources depends on the computing environments the system belongs to
  - Generic single user system
  - Portal
  - Network computer (thin client))
  - Distributed computations (and network operating systems)
  - Client-server computing (asymmetric relationship)
  - Peer-to-peer (P2P) computing (symmetric relationship of peers)
  - Computer system hosting virtual machines
  - Cloud computing (computing, storage, apps as a service on a net)
  - Real-time embedded systems

Operating system services

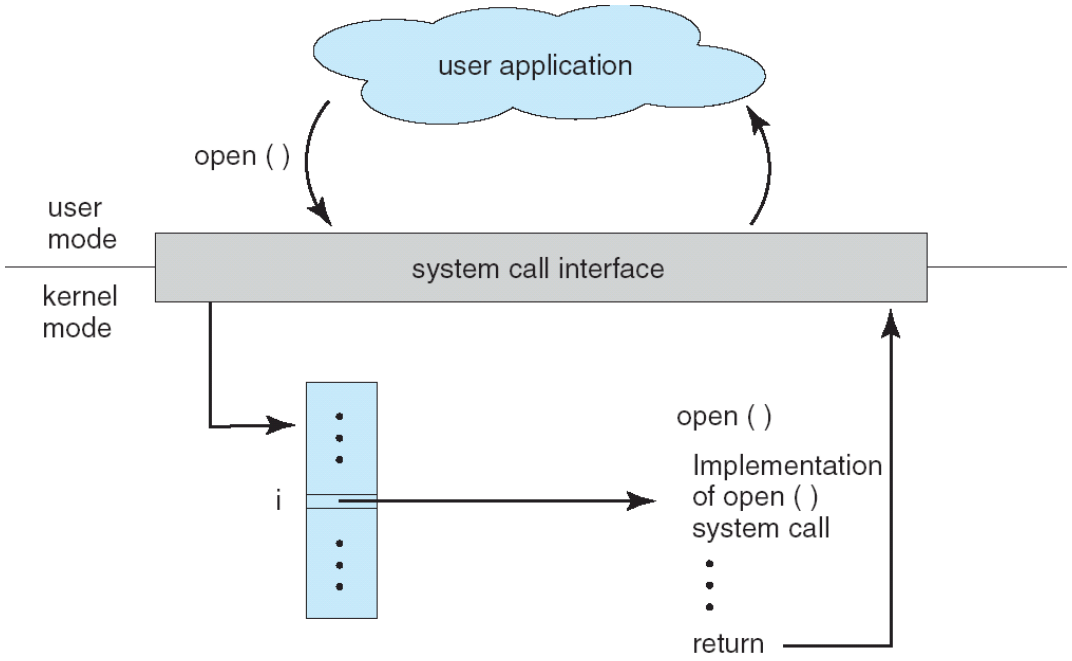


Concurrent resource use

- **Multiprogramming** and **time-sharing** demand safe and efficient sharing of computer system resources among users and their jobs.
- Concurrent resource use is implemented with
  - interrupts
  - system means of communication and synchronization
  - Multi-processor/multi-core architectures
- Degree of concurrency depends not only on hardware but also on OS properties:
  - **Symmetric Multiprocessing** (SMP) – CPUs share all system and user job related activities
  - **Asymmetric Multiprocessing** – one master and many slave (worker) CPUs
- Access to shared resources is possible for user jobs **only indirectly** - by making requests to the operating system via **system calls**.
- System functions form an interface between operating system and a running program.

API function vs system function

- System functions can be directly called from assembly language code.
- High-level languages enable access to shared resources with
  - calls of language specific functions which make appropriate system function call(s)
  - System function wrappers, which are usually called **Application Program Interface (API)** Popular APIs: Win32, POSIX API, JAVA API

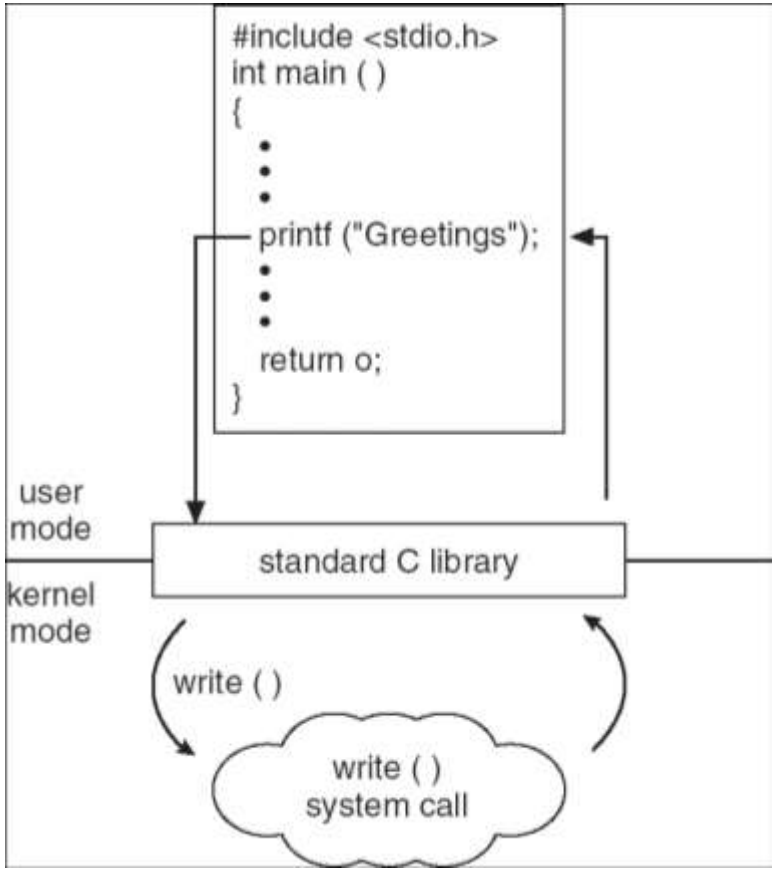


System function call from C

A C language program call standard output formatting function **printf**() ➔

The function stores formatted output to a local buffer associated with the standard output stream (*stdout*).

When the buffer is about to be full (or on demand: **fflush**()) a call to a system function is made (**write**() for POSIX API) to move buffer content to the destination device.



Sample Win32 i POSIX API functions

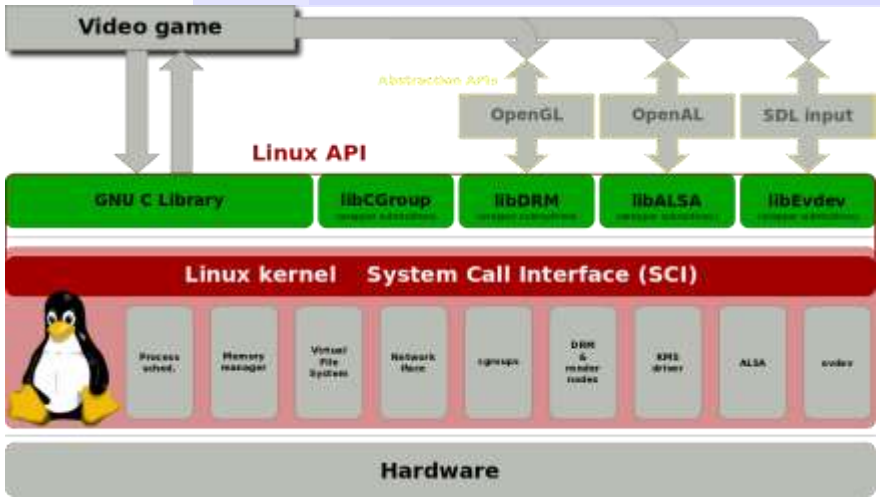
	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information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

Selected operating system API standards

- UNIX type systems:
  - SVID (System V Interface Definition)
  - 4.x BSD (Berkeley Software Distribution)
    - POSIX (Portable Operating System Interface) – IEEE, ISO, The Open Group. Status as of 2018: **POSIX.1-2017** (IEEE Std 1003.1-2017)
      - **Base Definitions (XBD)**
      - **Shell and Utilities (XCU)**
      - **System Interfaces (XSH)**
      - **Rationale (XRAT)**

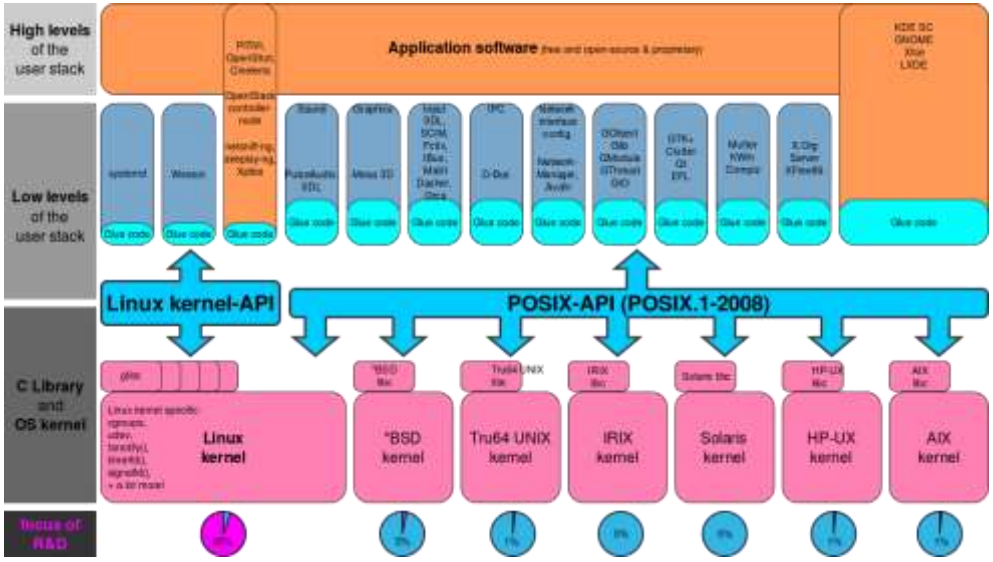
      Separate: **POSIX Conformance Test Suite**
    - Single Unix Specification (SUS) – family of standards for systems qualified for the name „Unix” (93, 95, 98, 03). Basic components of POSIX plus **POSIX Certification Test Suite** and the terminal interface standard: **CURSES**. In all: 1742 interfaces.
- win16/win32/win64/winCE – Microsoft Windows APIs (WinAPIs) for its OSs

Linux kernel interfaces

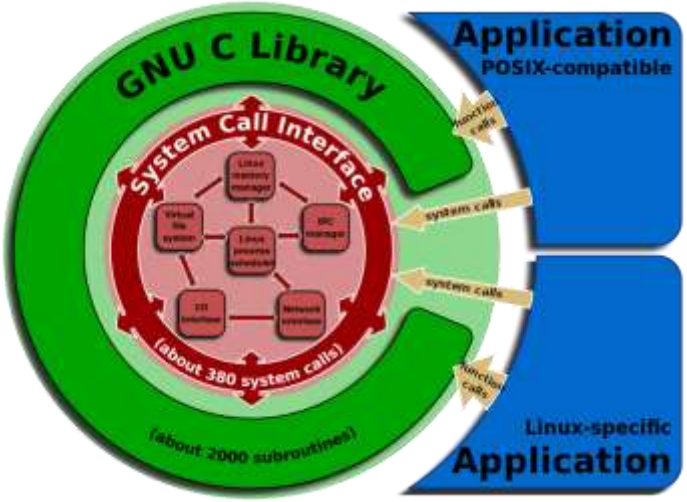


Linux API:

- Kernel internal API
- Kernel-user space API:
  - System Call Interface +
  - GNU C Library (`glibc`) wrapper



Linux API vs POSIX API



https://en.wikipedia.org/wiki/Linux_kernel_interfaces

Introduction

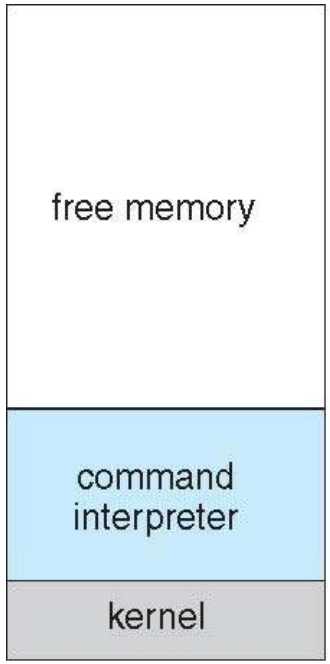
1. Operating system and computer system
2. Operating systems: goals and interface
3. Operating system structures
4. Computer system operations

Operating systems structure

- Operating system code is huge, and so structuring is important.
- Structuring of systems evolved in time.
  - Simple system (hardly structured)
  - Monolithic kernel + plethora of system programs
  - Layered design
  - Microkernel + system code in userspace
  - Hybrid systems

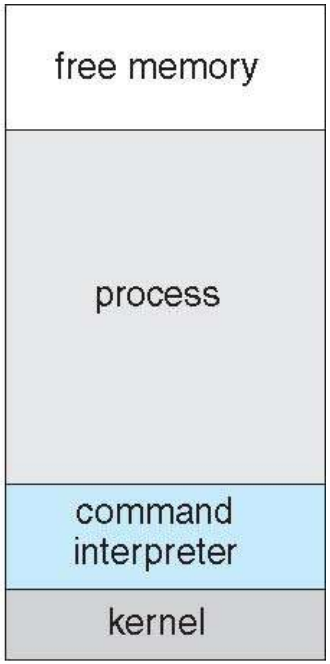
Simple system: MS-DOS

- Single-tasking
- Shell invoked when system booted
- Simple method to run program (no process created)
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



(a)

at system startup



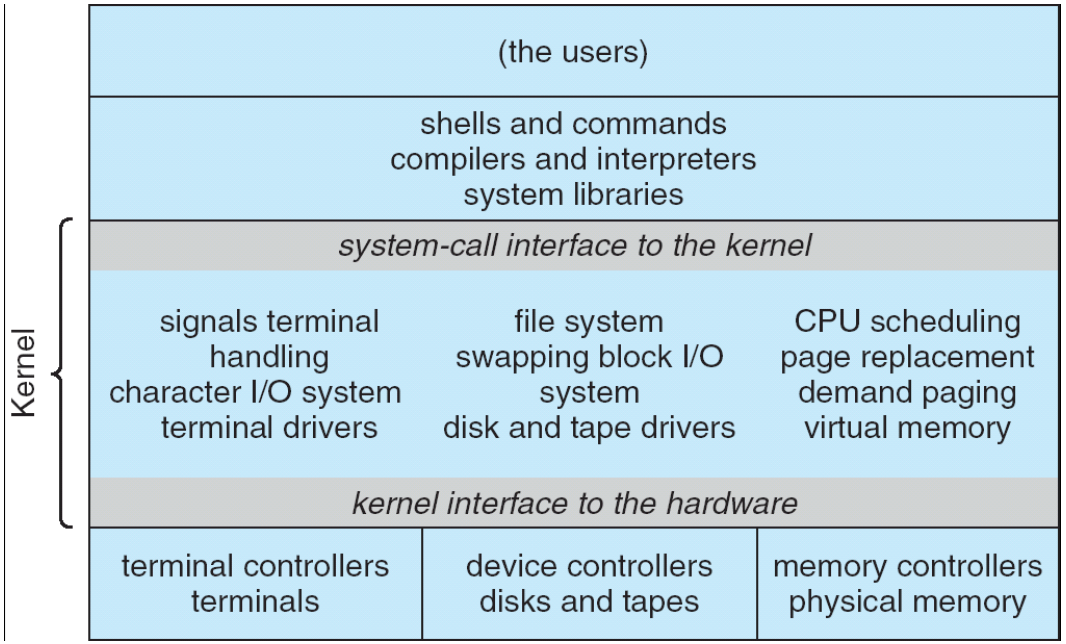
(b)

running a program

Old-style (monolithic) UNIX system

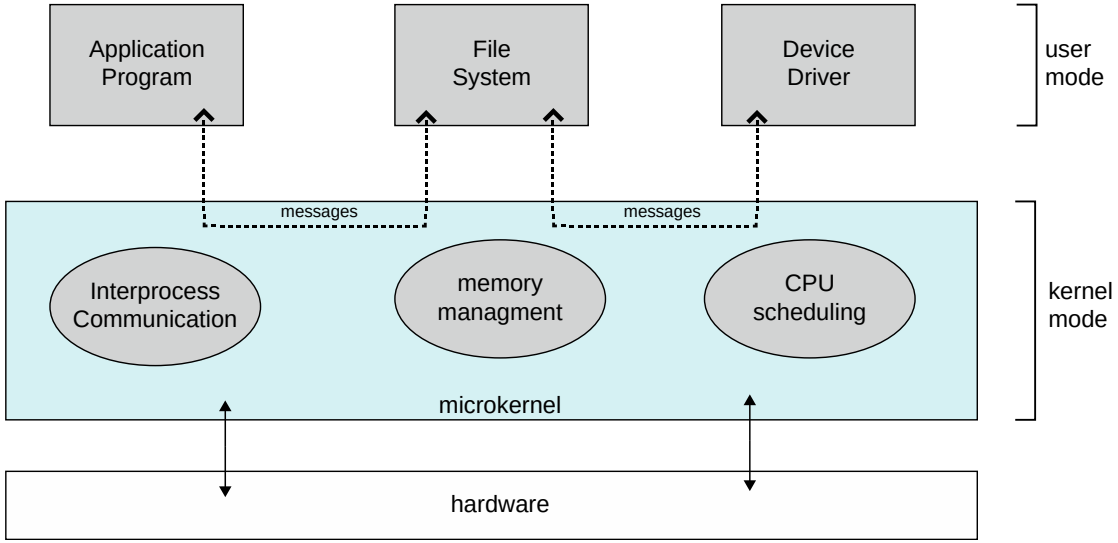
Two distinct parts of the system.

- Monolithic 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
- System programs (frequently connected via pipes)



Microkernel System Structure

- Moves as much from the kernel into user space
- Communication takes place between user modules using **message passing**
- Features:
  - **+** Easier to extend a microkernel (to add new functionality)
  - **+** Easier to port the operating system to new architectures
  - **+** More reliable (less code is running in kernel mode) and secure
  - **-** performance overhead of user space to kernel space communication

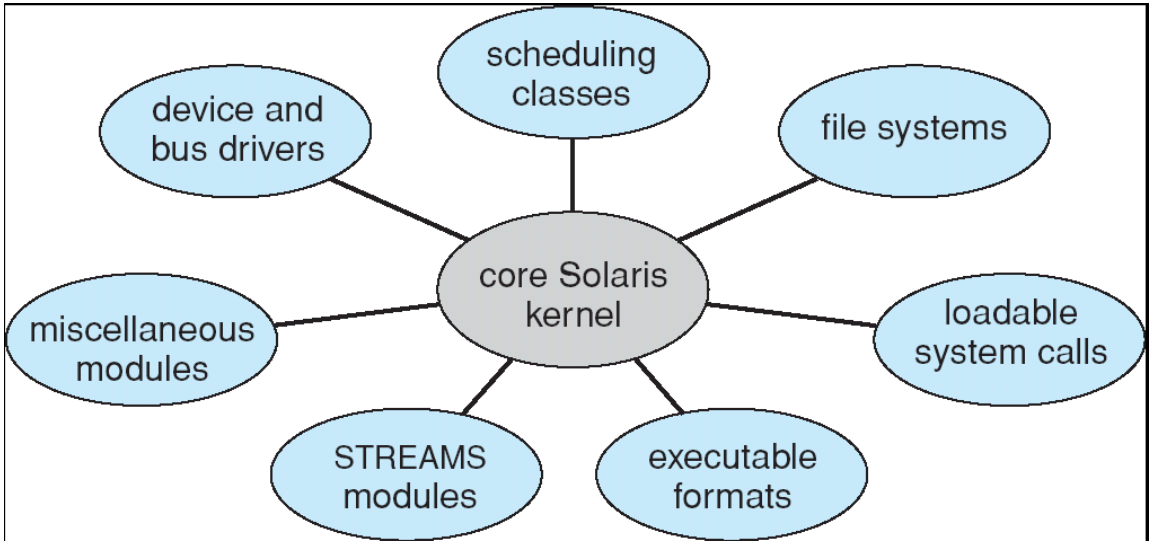


Kernel modules

Many modern operating systems implement **loadable kernel 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

System Solaris

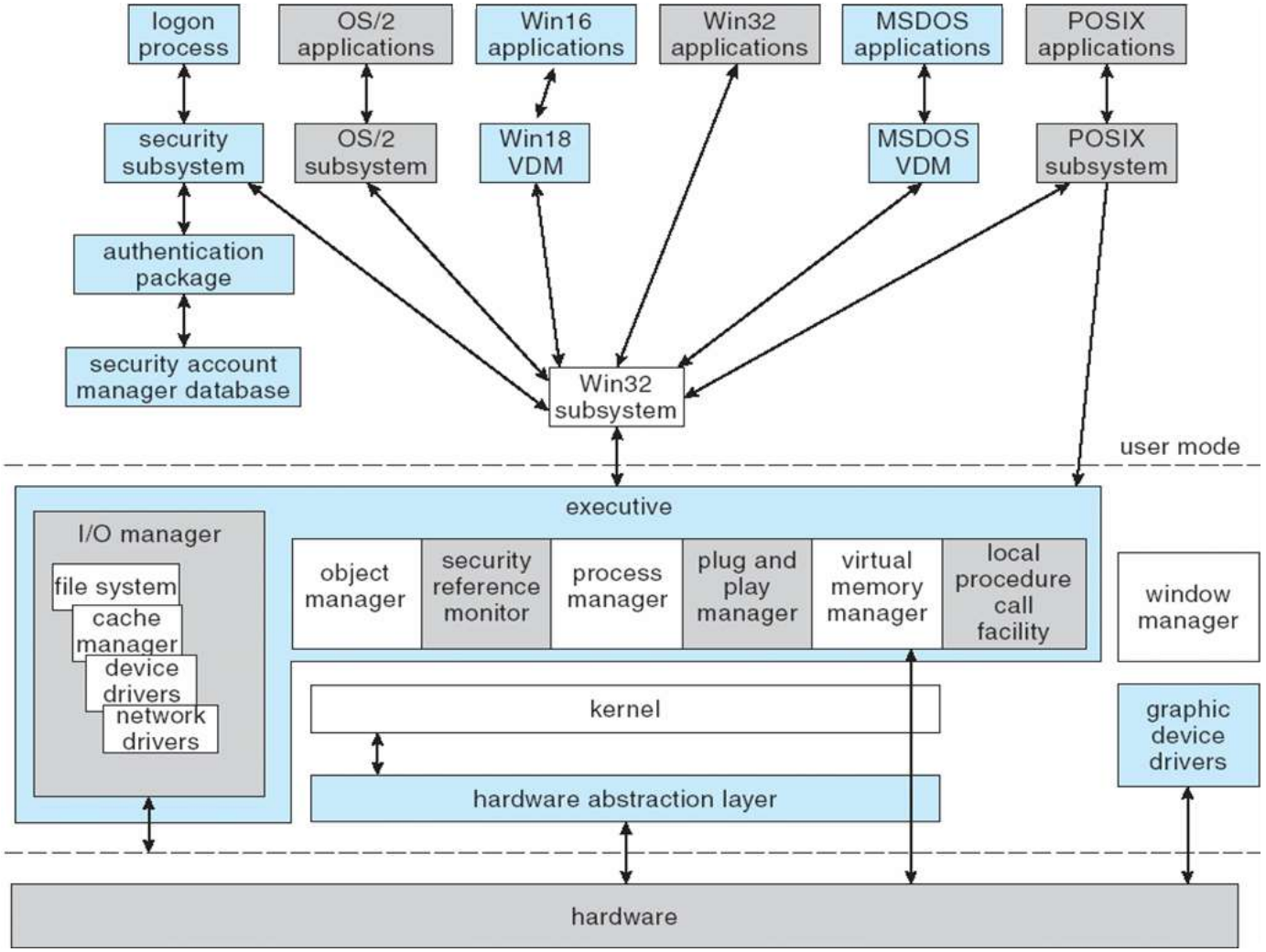


Hybrid systems

Most modern operating systems are actually not one pure model

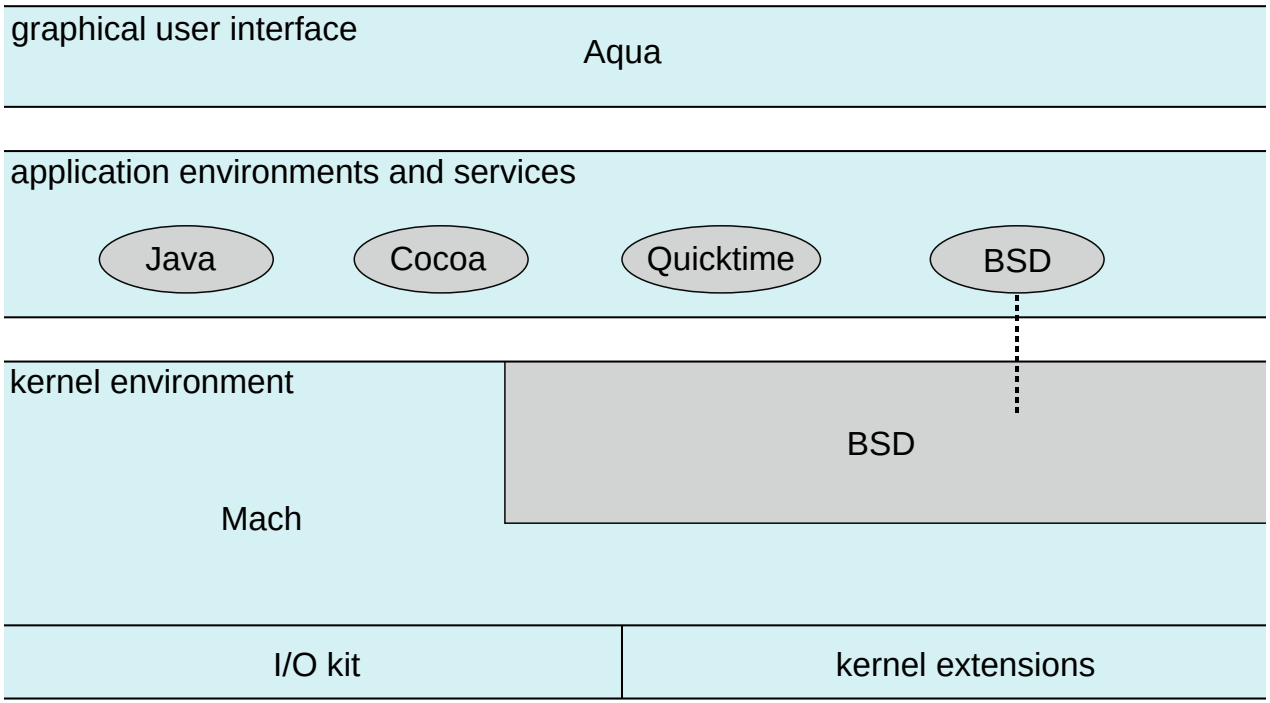
- Hybrid combines multiple approaches to address performance, security, usability needs
- Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
- Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment. Below there is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

Microsoft Windows 7,… architecture



From Silberschatz, Galvin, Gagne, Operating System Concepts, 9th ed.,, 2013

Mac OS X Structure

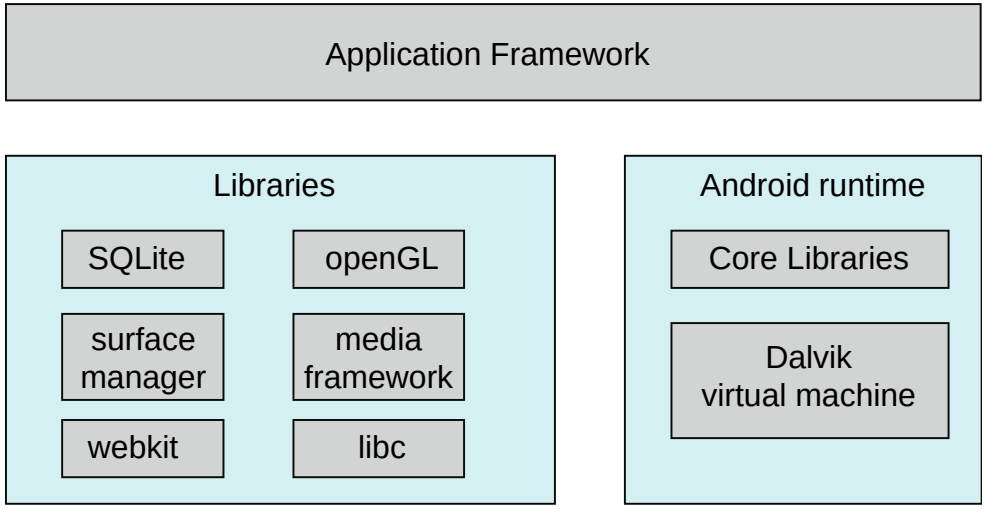


Hybrid XNU kernel:

- microkernel Mach 3: CPU scheduling (also RT), threads, virtual memory
- BSD UNIX kernel (POSIX API): models of processes and threads, protection mechanisms, file systems (including HFS/HFS+), IPC, network protocols, sockets, NFS,…

Android OS

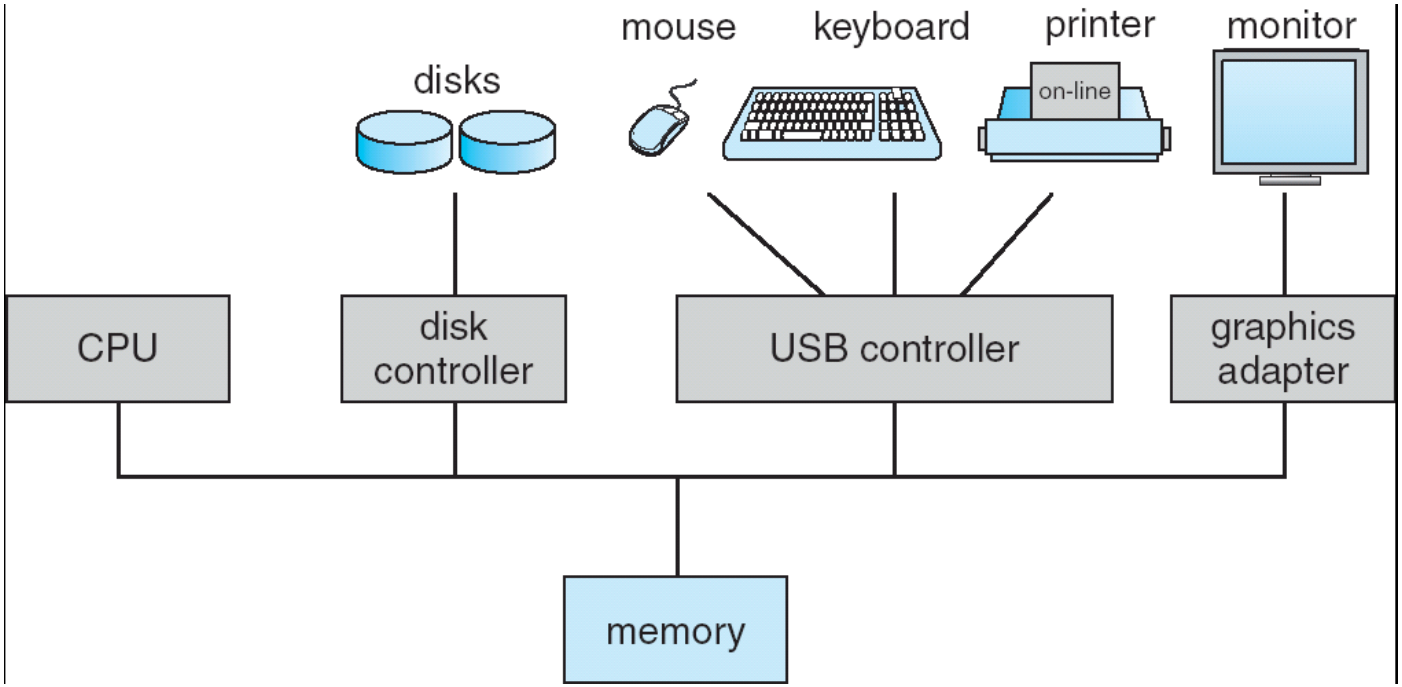
- Developed by Open Handset Alliance (mostly Google). Open Source
- Based on Linux kernel but modified. Provides process, memory, device-driver management. Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine. Apps developed in Java plus Android API. Java class files compiled to Java bytecode then translated to executable which runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc



Introduction

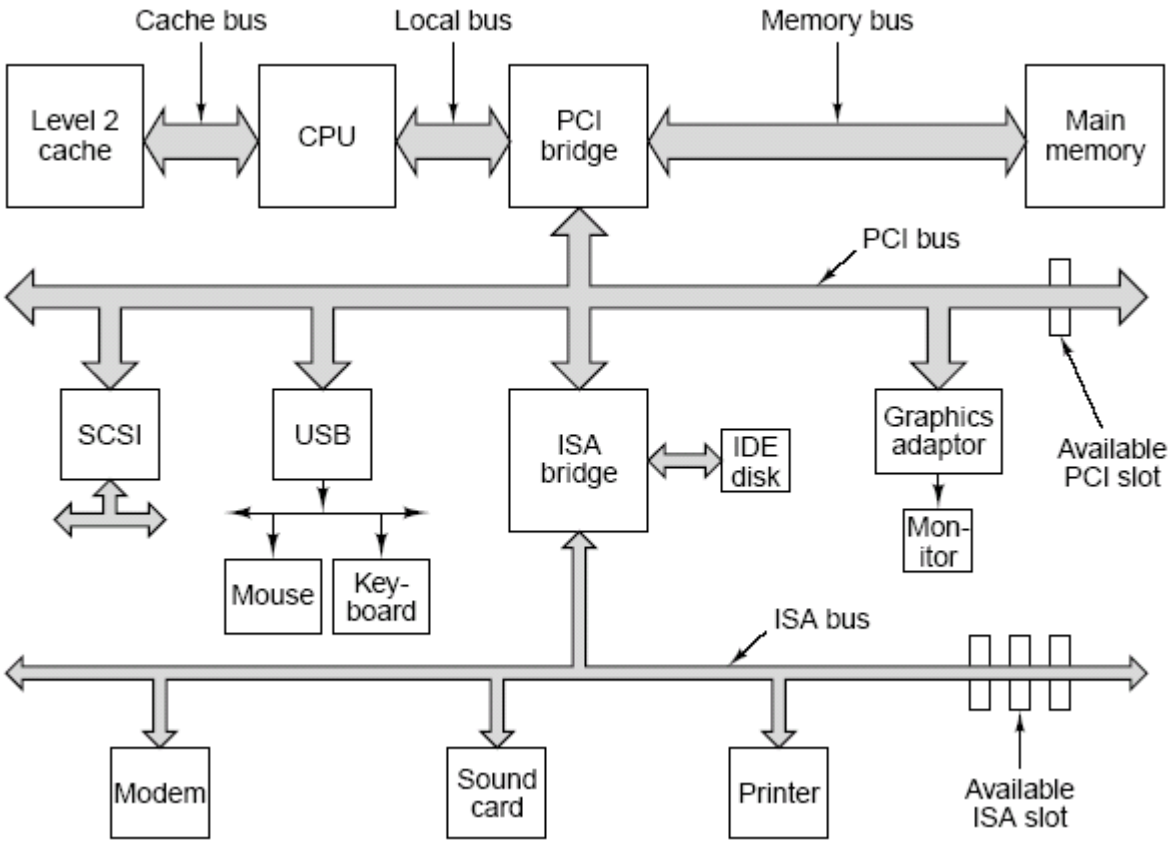
1. Operating system and computer system
2. Operating systems: goals and interface
3. Operating system structures
4. Computer system operations

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

Example: „classic PC” architecture



Computer-System Operation

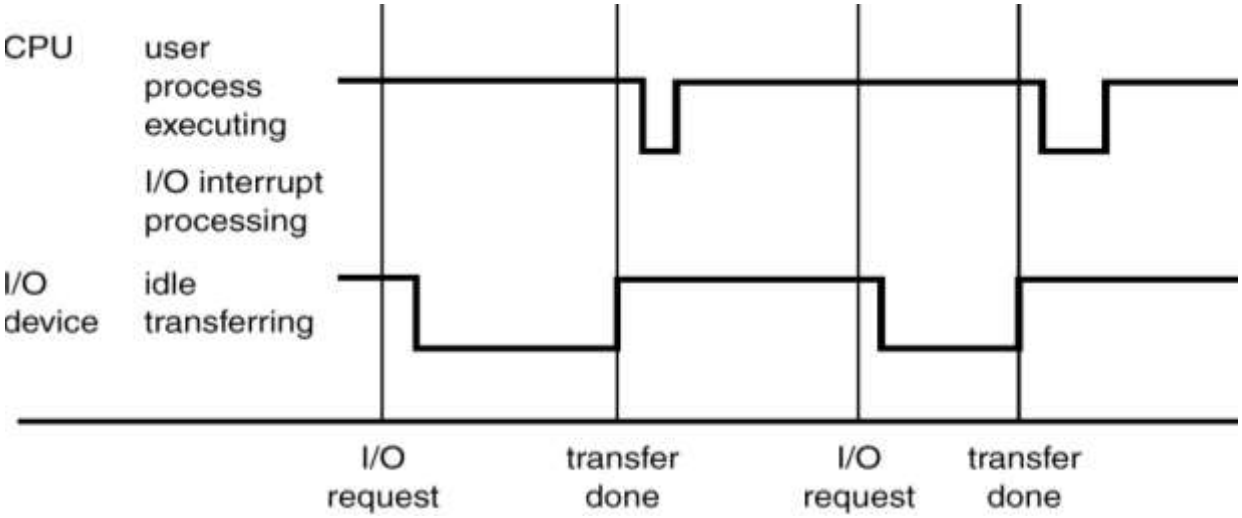
- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs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

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**

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



Przebieg zdarzeń przy obsłudze przerwań

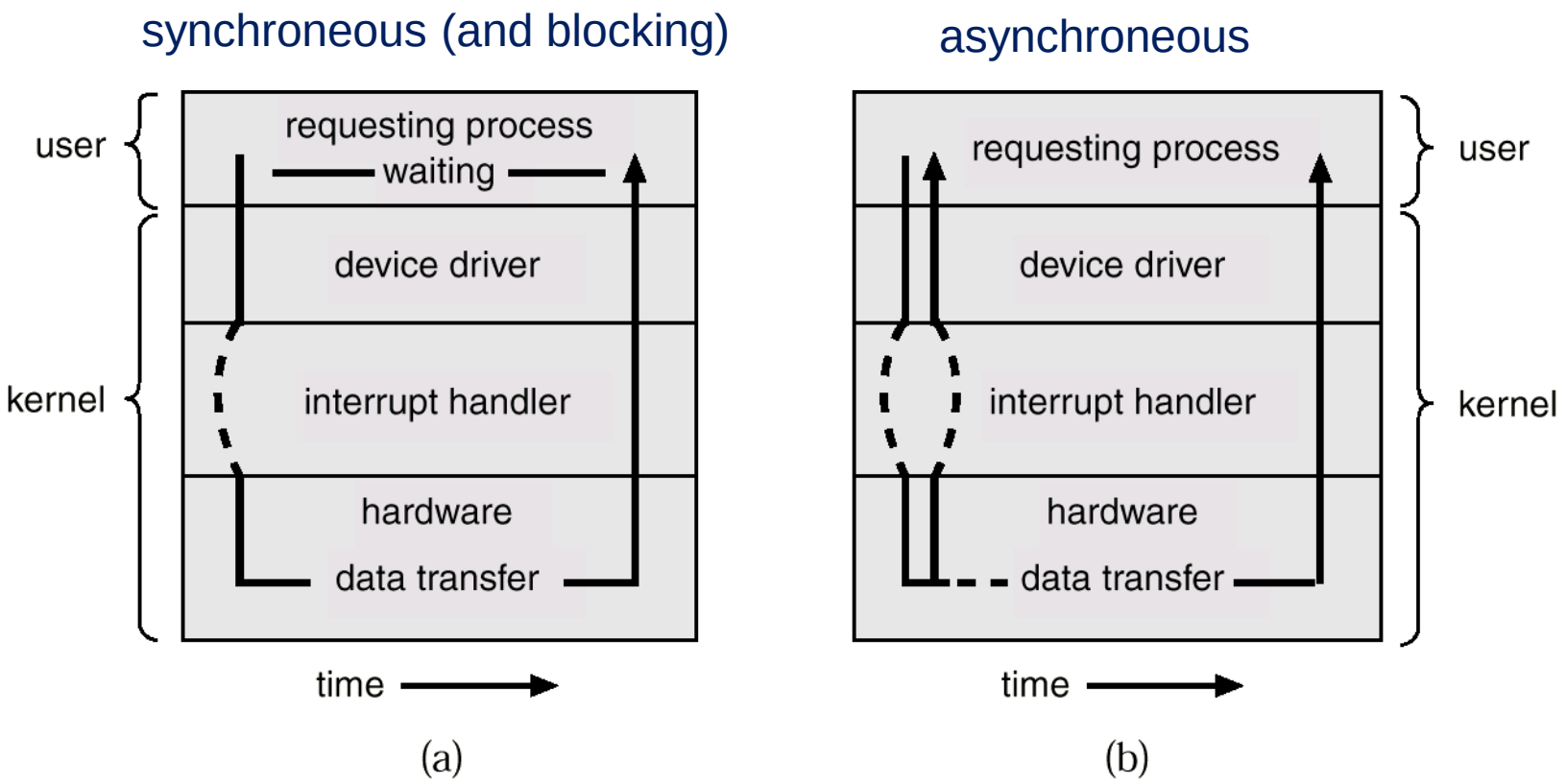
I/O operations

Scenarios of I/O operations

- **Synchronous I/O operation**. After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt or wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing

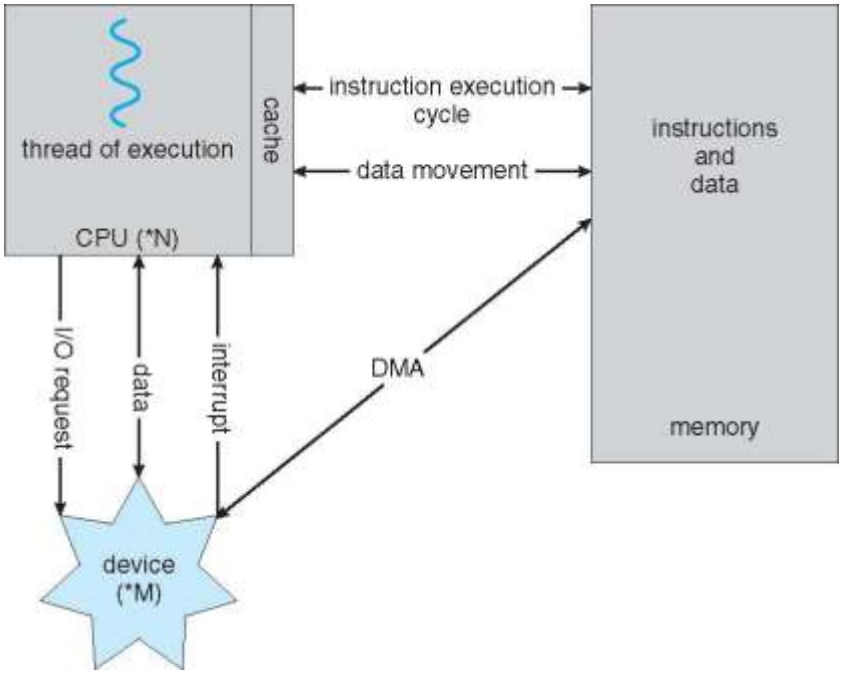
  Note: synchronous operation can be also performer in **non-blocking mode**. Control returns to the caller immediately, but if the operation could not be performer – the caller is notified with an appropriate error code (so the attempt can be retried).
- **Asynchronous I/O operation.** After I/O starts, 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
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

Two scenarios of I/O operations



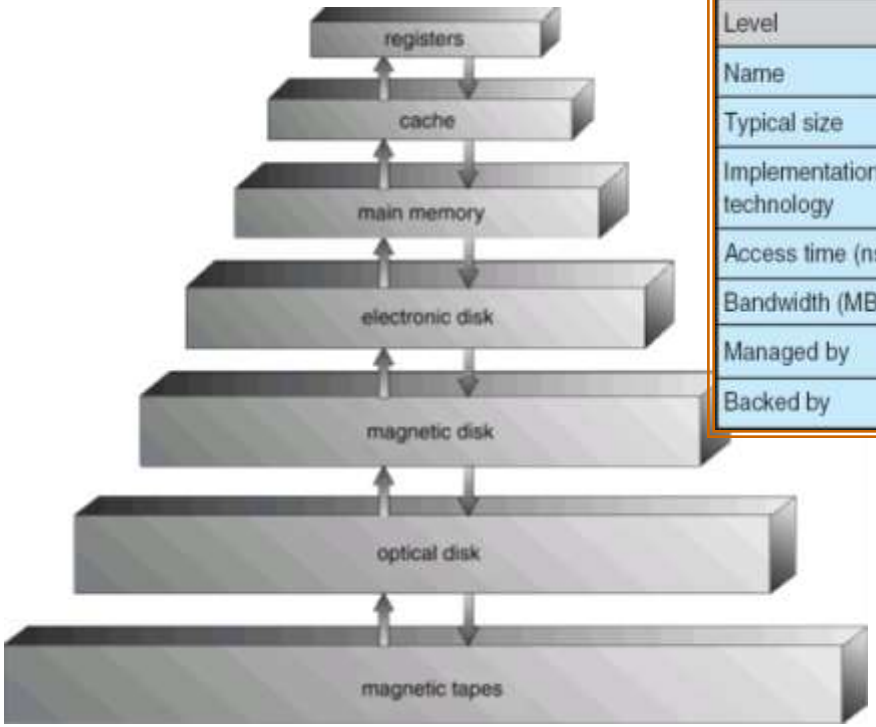
Direct Memory Access - DMA

- 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
- DMA and CPU compete for memory – so the net speedup may vary.



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

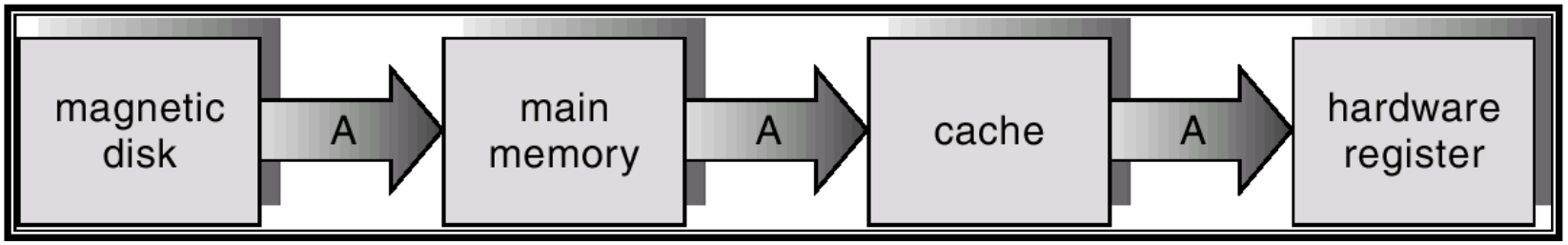


Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

← **Memory hierarchy**

Memory hierarchy - cont.

Path of data item **A** from magnetic disc to a CPU register



Caching – copying information into faster storage system temporarily; main memory can be viewed as a last *cache* for secondary storage

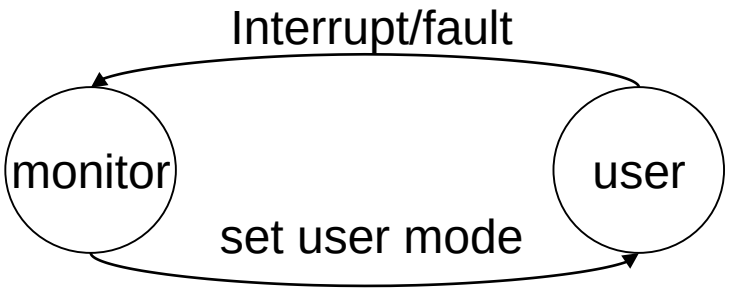
- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller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

Hardware protection

- Dual mode CPU operations (user and kernel modes).
- I/O protection
- Memory protection
- CPU protection

Dual-mode CPU operations

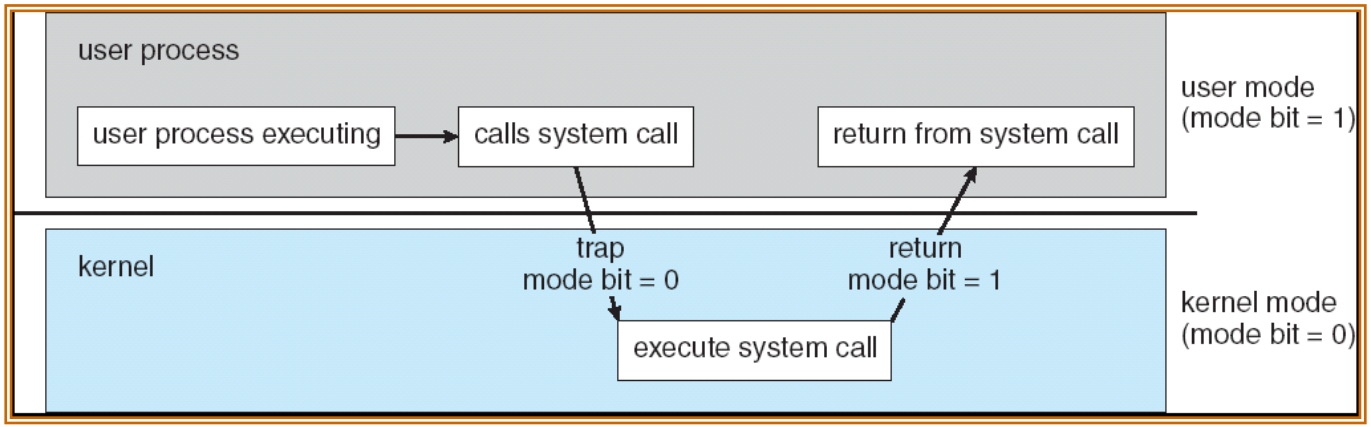
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user



- **Privileged CPU instructions** can be executed only in privileged CPU mode of operation

System functions

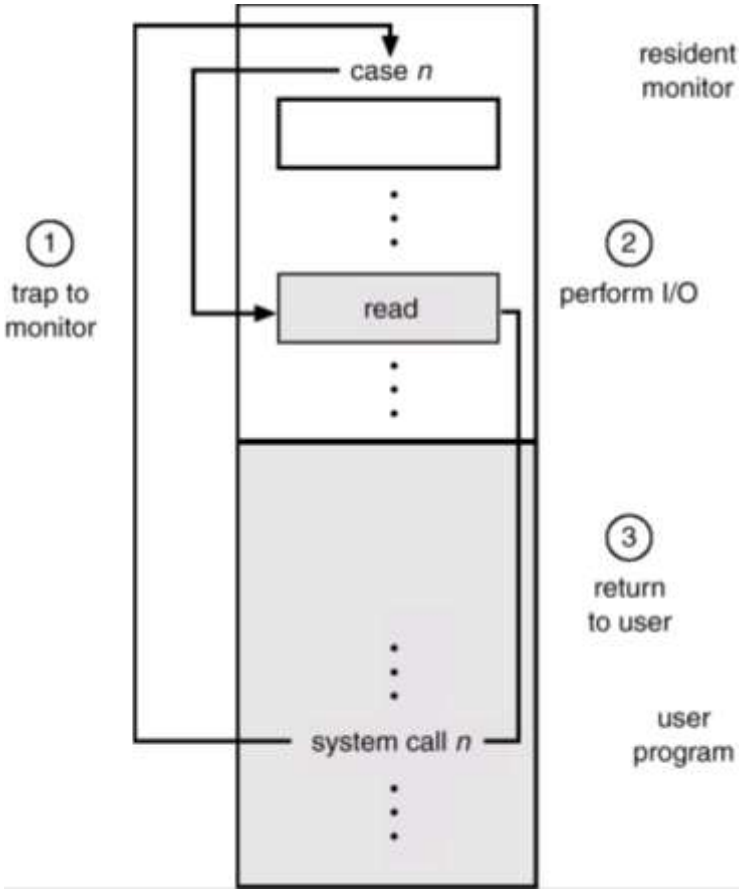
- System call is typically made by execution of a special CPU instruction (**system trap**), which is handled similarly to interrupts (except interrupts are asynchronous events).
- CPU changes its mode to **privileged** and then uses **interrupt vector** to find address of the system trap handler (**ISR**).
- ISR takes arguments of the system function call (CPU registers | system stack | memory block addressed with registers).
- If parameters are correct and acceptable some kernel code is invoked that implements system function functionality. Otherwise the ISR sets error code and returns.
- The ISR returns exit code, CPU changes its mode and return is made.



Note: in real systems implementations of system calls can be more complex, e.g. to increase degree of concurrency (typically **CPU scheduler** is invoked before function return – so a **context switch** can occur before return to the caller process.

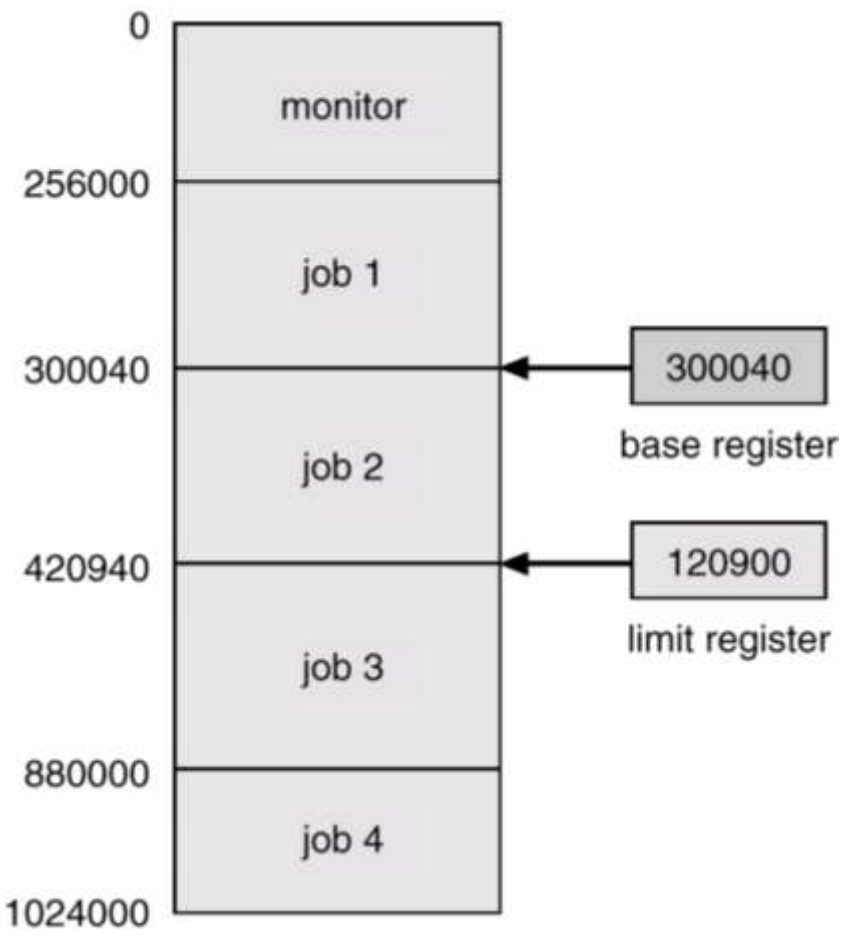
I/O protection

- All **I/O instructions** are **privileged**
- For full protection of I/O devices it is necessary to **protect interrupt vector** – because device drivers use interrupts
- Userspace code can access I/O only through calls to system functions (which change processor mode appropriately).

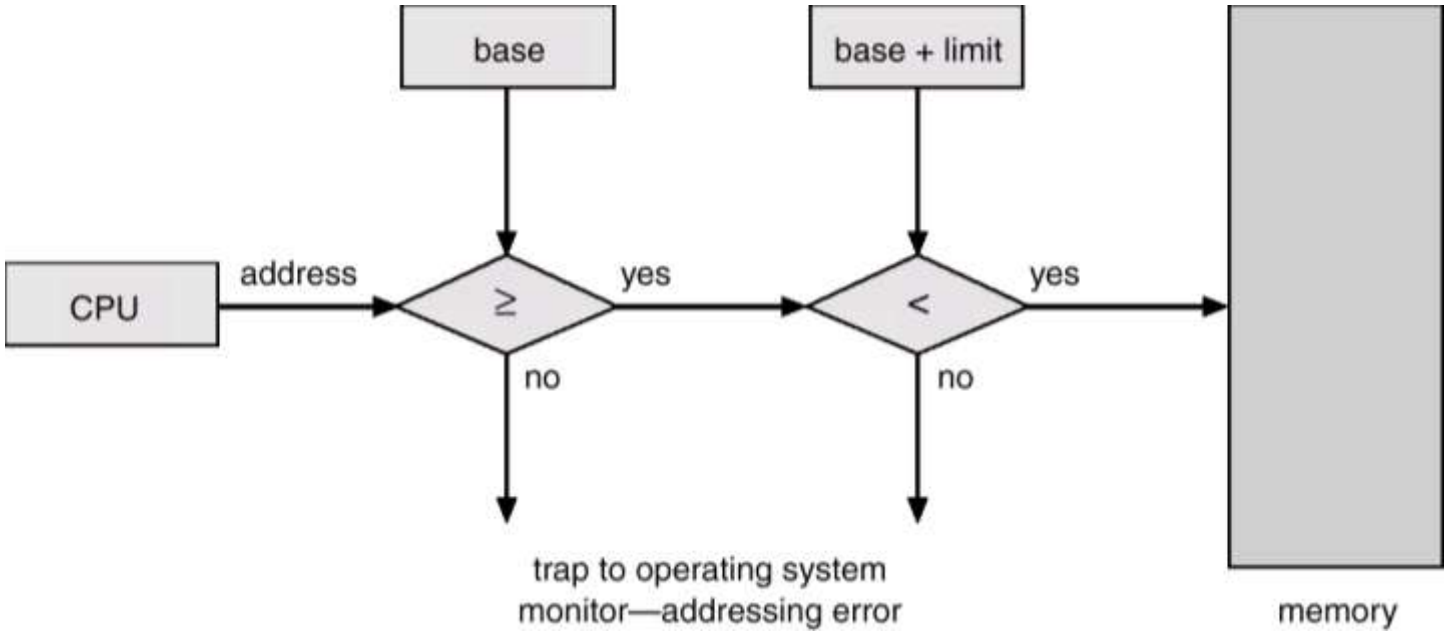


Hardware memory protection

- Hardware protection of interrupt vector and handler is necessary for system sanity.
- Other areas of memory can be protected (from unauthorized use) with at least two registers, which determine range of valid addresses for a job (and kernel).
  - **Base register** – holds the low-end address.
  - **Limit register** – keeps the high-end address of a memory region
- Hardware is to make impossible access of memory outside the range specified with the two registers.



Hardware memory protection



- While in kernel mode the CPU can access any memory location.
- CPU instructions that modify base and limit register are **privileged.**
- Hardware generates a trap when invalid memory reference is detected. In turn a system Interrupt System Routine (ISR) is called after switching CPU to the kernel mode of operation.

CPU protection

- **Timer** is to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero -> generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time
- Timer can be used to determine current time in the system.
- Timer setting instructions are **privileged**.