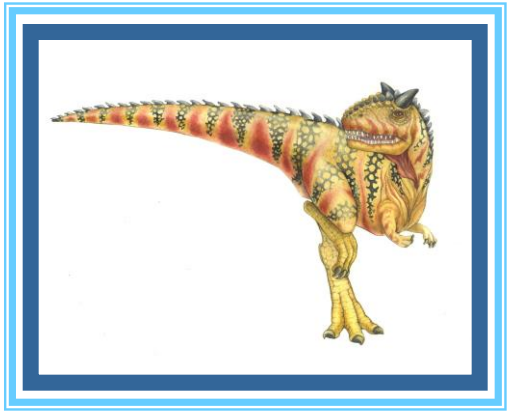
Chapter 2: Operating-System Structures



Silberschatz, Galvin and Gagne ©2009



Operating System Services

تختلف واجهة المستخدم، حيث ان هناك انواع :

1- CLI (Command Line Interface)
مثل Terminal في MacOS
تعتمد على كتابة الأوامر.

2- GUI (Graphical User Interface)
مثل الواجهة الرسومية في Microsoft Windows
تعتمد على النوافذ والأيقونات.

3- Batch Processing
تنفيذ مجموعة برامج بدون تدخل مباشر من المستخدم البشري.

- Operating systems provide an environment for execution of programs and services to programs and users
- One set of operating-system services provides functions that are helpful to the user:

كل أنظمة التشغيل تقريبًا تحتوي على واجهة مستخدم

  - **User interface** - Almost all operating systems have a user interface (UI).
    - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch processing** (the execution of a series of programs on a computer without human interaction.

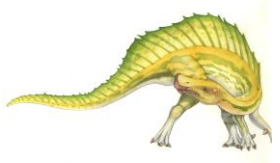
نظام التشغيل يجب أن يكون قادرًا على: تحميل البرنامج إلى الذاكرة (Load into memory)، تشغيله (Run)، و إنهاءه (بشكل طبيعي او غير طبيعي)

  - **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)

البرامج المشغله قد تحتاج الى اجهزة ادخال واخراج.

  - **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. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

البرامج تحتاج إلى:
إنشاء ملفات ومجلدات، حذفها، القراءة والكتابة، البحث عن ملفات، عرض معلومات الملف، إدارة الصلاحيات (Permissions)





Operating System Services (Cont.)

- **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)
- **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

نظام التشغيل يجب أن يكتشف الأخطاء باستمرار.
الأخطاء قد تحدث في:
CPU
الذاكرة
أجهزة الإدخال/الإخراج
برنامج المستخدم
وعند حدوث خطأ:
يتخذ النظام الإجراء المناسب
يحافظ على استقرار النظام
يوفر أدوات Debugging للمبرمجين





Operating System Services (Cont.)

خدمات نظام التشغيل للنظام نفسه:

- 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

من خدمات نظام التشغيل انه يتتبع:
من استخدم الموارد؟
كم استخدم؟
ما نوع الموارد المستخدمة؟

    - 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

التحكم في وصول العمليات للموارد حتى لا تؤثر على بعضها.

    - **Protection** involves ensuring that all access to system resources is controlled

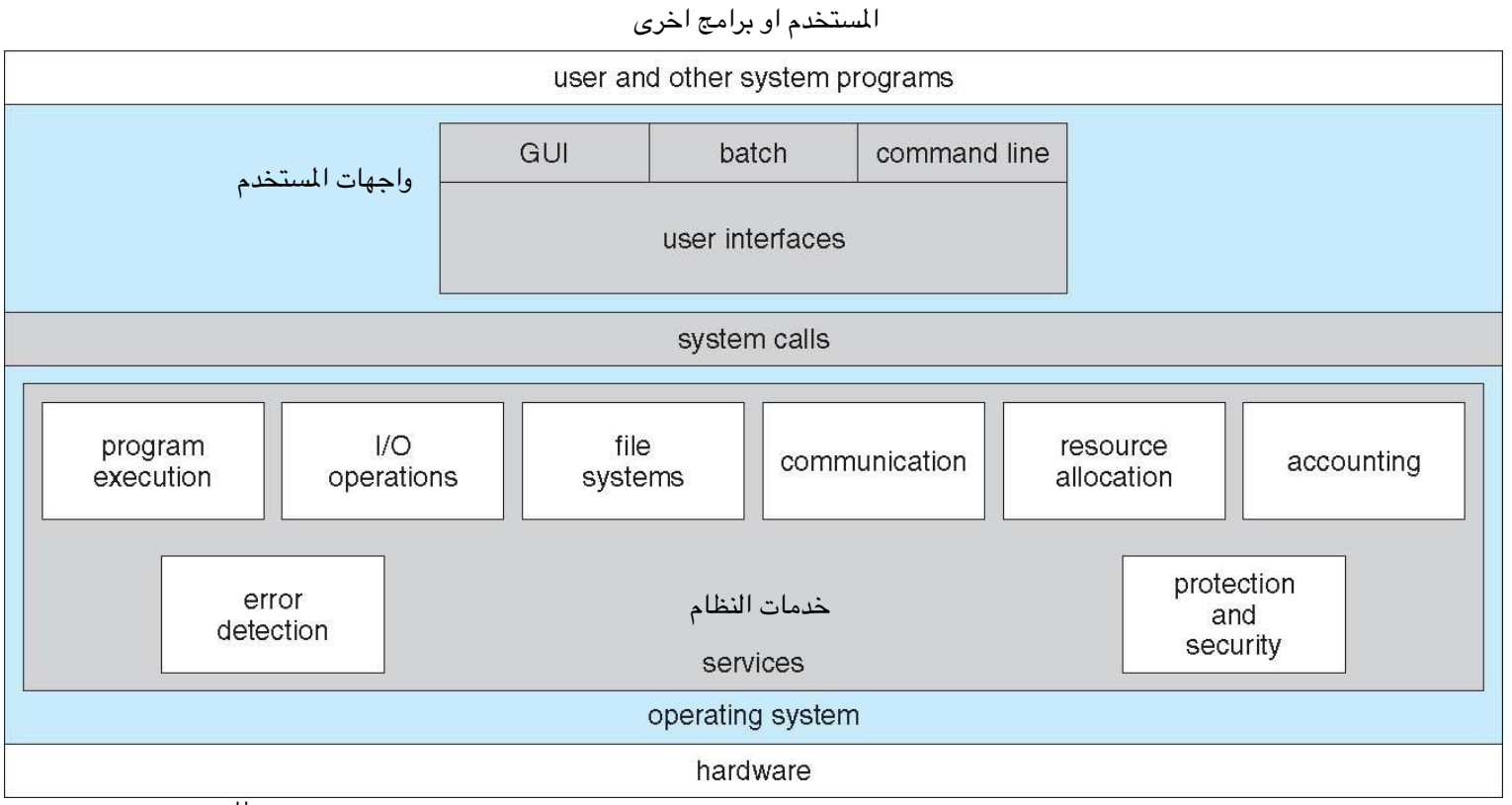
حماية النظام من المستخدمين غير المصرح لهم

    - **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 (series) is only as strong as its weakest link.





A View of Operating System Services



User
↓
Program (Word)
↓
API
↓
System Call
↓
Operating System
↓
Hardware

نظام التشغيل هو نقطة الوصل بين المستخدم (او البرنامج، او حتى اجهزة اخرى) والهاردوير

مثال لو المستخدم ضغط زر Save في برنامج Word هذا ما يحدث :



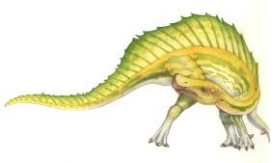


User Operating System Interface - CLI

- Command Line Interface (CLI) or command interpreter allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Sometimes multiple flavors implemented – shells
  - Primarily fetches a command from user and executes it
    - Sometimes commands built-in, sometimes just names of programs
      - If the latter, adding new features doesn't require shell modification

واجهة تعتمد على كتابة الأوامر يدويًا.
يسمى أيضًا: Command Interpreter, Shell
كيف يعمل؟
يستقبل أمر من المستخدم، ثم يفسر هذا الامر، ثم ينفذه.

أين يتم تنفيذه؟
أحيانًا يكون جزءًا من الـ Kernel
أحيانًا يكون برنامج مستقل (Shell)
Multiple Shells (أكثر من نوع Shell)
قد يوجد أكثر من Shell في نفس النظام.





User Operating System Interface - GUI

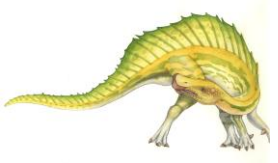
واجهة رسومية سهلة الاستخدام
تعتمد على:
الماوس
لوحة المفاتيح
الشاشة

- 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) المجلدات تسمى Folders او Directory
  - Invented at Xerox PARC (Company)

أيقونات تمثل:
ملفات
مجلدات
برامج
الضغط بالماوس ينفذ عمليات:
فتح ملف
عرض معلومات
تنفيذ برنامج

اغلب الانظمة تحتوي على كلًا من واجهات CLI و GUI

- 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 GUI interfaces (Java Desktop, KDE)





Bourne Shell Command Interpreter

مثال

Terminal

File Edit View Terminal Tabs Help

```
fd0       0.0     0.0     0.0     0.0  0.0  0.0     0.0    0   0
sd0       0.0     0.2     0.0     0.2  0.0  0.0     0.4    0   0
sd1       0.0     0.0     0.0     0.0  0.0  0.0     0.0    0   0
                  extended device statistics
device    r/s     w/s    kr/s    kw/s wait actv  svc_t  %w  %b
fd0       0.0     0.0     0.0     0.0  0.0  0.0     0.0    0   0
sd0       0.6     0.0    38.4     0.0  0.0  0.0     8.2    0   0
sd1       0.0     0.0     0.0     0.0  0.0  0.0     0.0    0   0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
 12:53am  up 9 min(s),  3 users,  load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
  4:07pm  up 17 day(s), 15:24,  3 users,  load average: 0.09, 0.11, 8.66
User     tty           login@  idle   JCPU   PCPU  what
root     console      15Jun0718days      1         /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07           18      4  w
root     pts/4        15Jun0718days                w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```





The Mac OS X GUI

مثال







System Calls

مناداة النظام : واجهة برمجية تسمح للبرامج بطلب خدمات من نظام التشغيل

- Programming interface to the services provided by the OS

  مكتوبة بلغات عالية المستوى مثل C او C++

- Typically written in a high-level language (C or C++)

  الوصول اليها يتم من خلال API

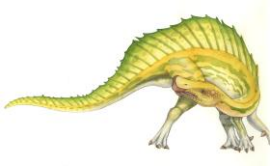
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use

- 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)

ما هي System Calls؟
هي واجهة برمجية تسمح للبرامج بطلب خدمات من نظام التشغيل.
البرنامج لا يتعامل مباشرة مع الهاردوير، بل يطلب من نظام التشغيل عبر System Call.
كيف تُكتب؟ غالبًا تُستخدم لغات عالية المستوى مثل: C, C++
لكن المبرمج عادةً لا يستدعي System Call مباشرة، بل يستخدم API.
ما هو API؟
دوال جاهزة تسهّل على المبرمج استخدام خدمات النظام بدون الدخول في التفاصيل المعقدة = API.
API = Application Programming Interface
هي طبقة وسيطة بين البرنامج و الـ System Calls.
بدل أن يستدعي البرنامج System Call مباشرة، يستخدم API جاهزة.
أشهر الـ APIs
1- Win32 API
تستخدم في أنظمة Microsoft Windows
2- POSIX API
تستخدم في الأنظمة المبنية على UNIX مثل:
Linux
macOS
3- Java API
تعمل داخل Java Virtual





API vs. System Calls

مميزات الـ API
مقارنة بين الـ API و System Calls

تختلف من جهاز الى جهاز System calls
بينما الـ API يمكن نقل البرامج بين الانظمة بسهولة

1) System calls differ from platform to platform. By using a stable API, it is easier to migrate your software to different platforms.

2) The operating system may provide newer versions of a system call with enhanced features. The API implementation will typically also be upgraded to provide this support.

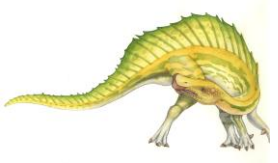
نظام التشغيل ممكن ان يوفر تحديثات للـ System calls
بينما الـ API سيتم تحديثها تلقائيًا لدعم هذة المتغيرات

3) The API usually provides more useful functionality than the system call directly.

الـ API يوفر وظائف اعلى واسهل من الـ System calls

4) The API can support multiple versions of the operating system and detect which version it needs to use at run time.

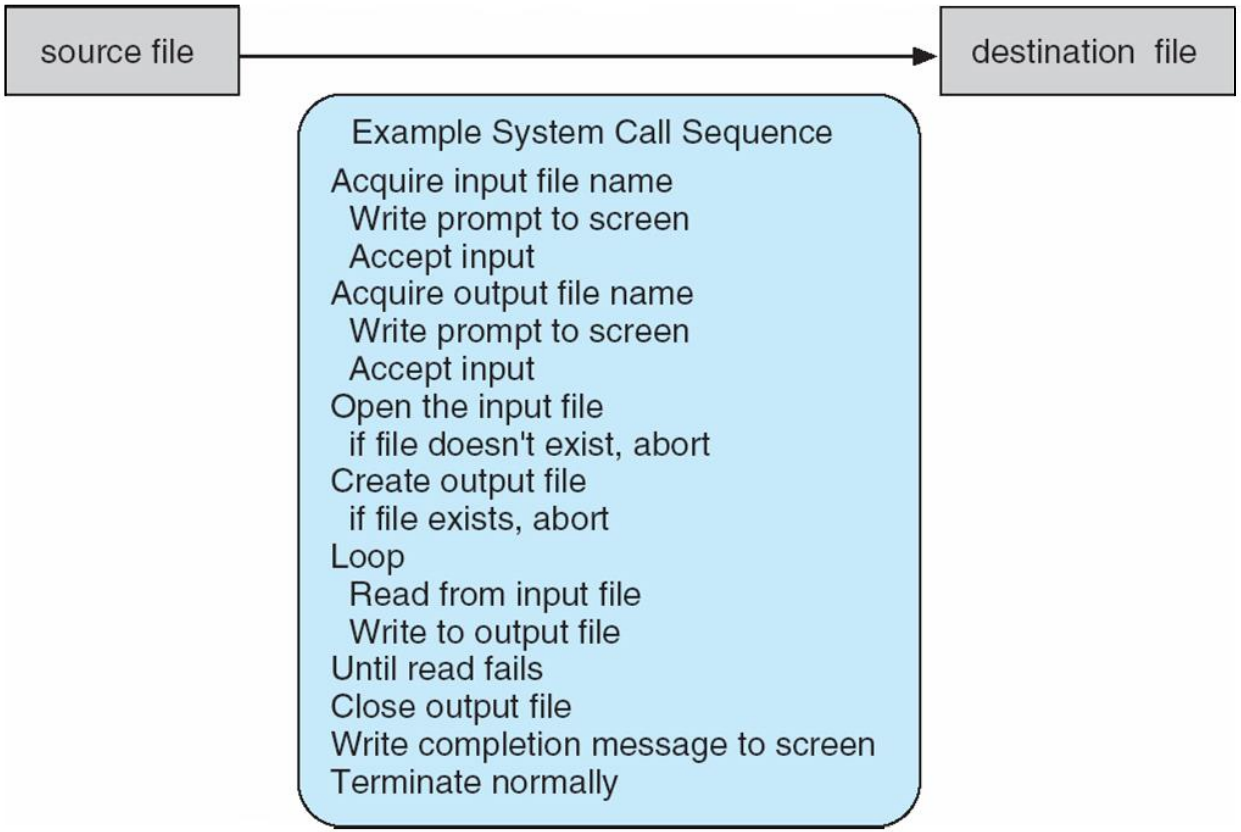
الـ API يدعم عدة نسخ للنظام، ويحدد اي نسخه تحتاج ان تستخدم في وقت التنفيذ

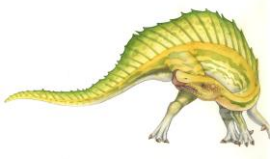




Example of System Calls

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



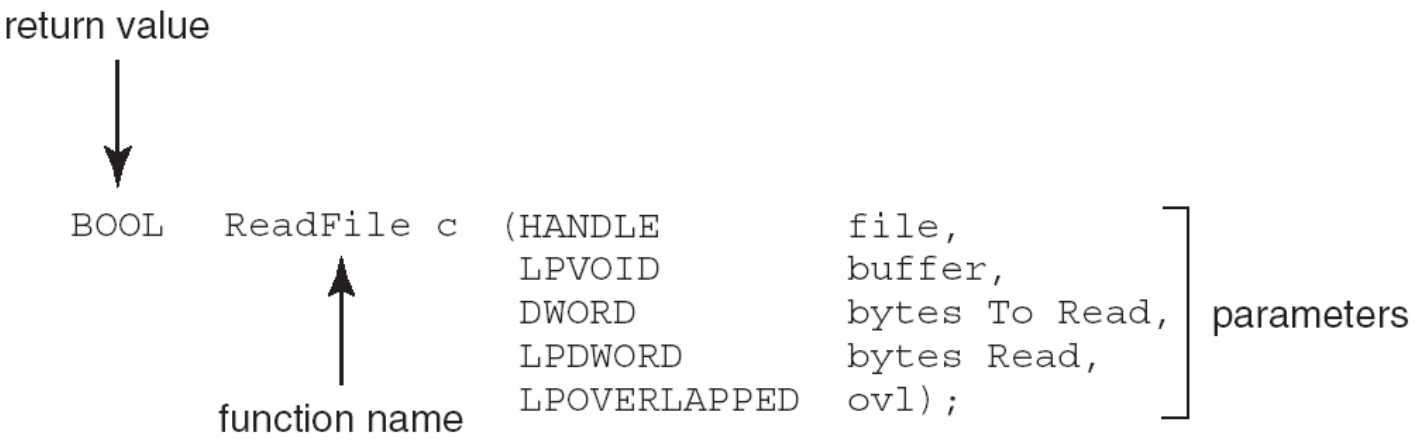




Example of Standard API

مثال

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



- 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 and written from
  - 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 (extension) 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)

لكل System Call رقم خاص بها.
يوجد داخل الـ Kernel جدول يسمى System Call Table يتم الوصول إليه باستخدام هذا الرقم.
عند استدعاء System Call:
البرنامج يستدعي API
يتم الانتقال إلى Kernel Mode
النظام ينفذ العملية المطلوبة
يعيد النتيجة أو حالة الخطأ
المبرمج لا يعرف تفاصيل التنفيذ الداخلية (هذة وظيفة الـ API).

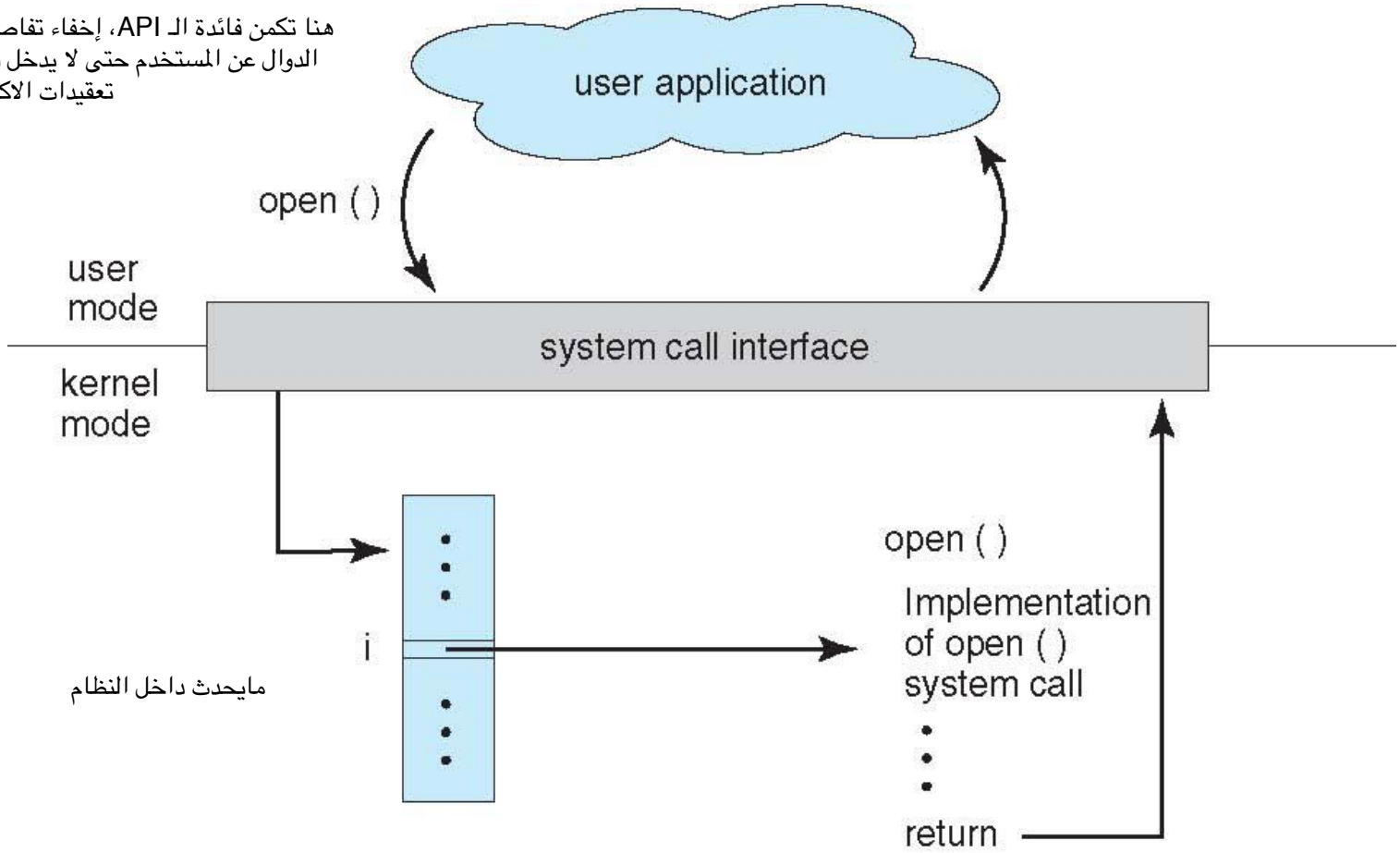




API – System Call – OS Relationship

العلاقة بين النظام، الـ System call، والـ API

هنا تكمن فائدة الـ API، إخفاء تفاصيل الدوال عن المستخدم حتى لا يدخل في تعقيدات الاكواد



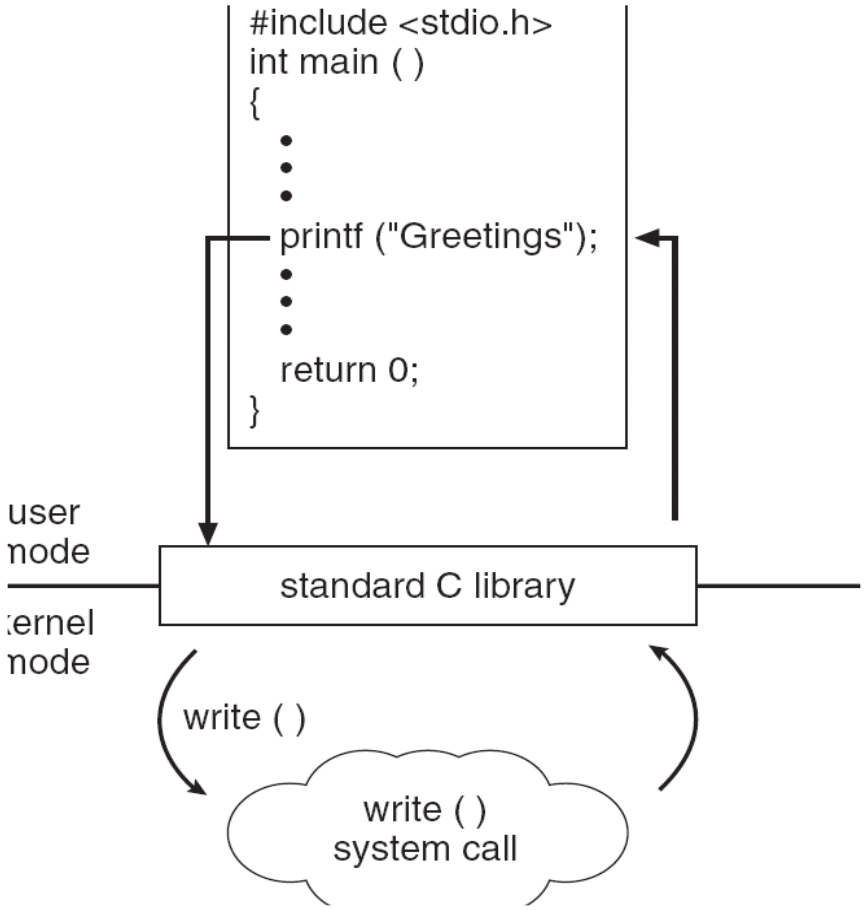
مايحدث داخل النظام





Standard C Library Example

- C program invoking printf() library call, which calls write() system call



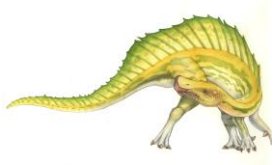




System Call Parameter Passing

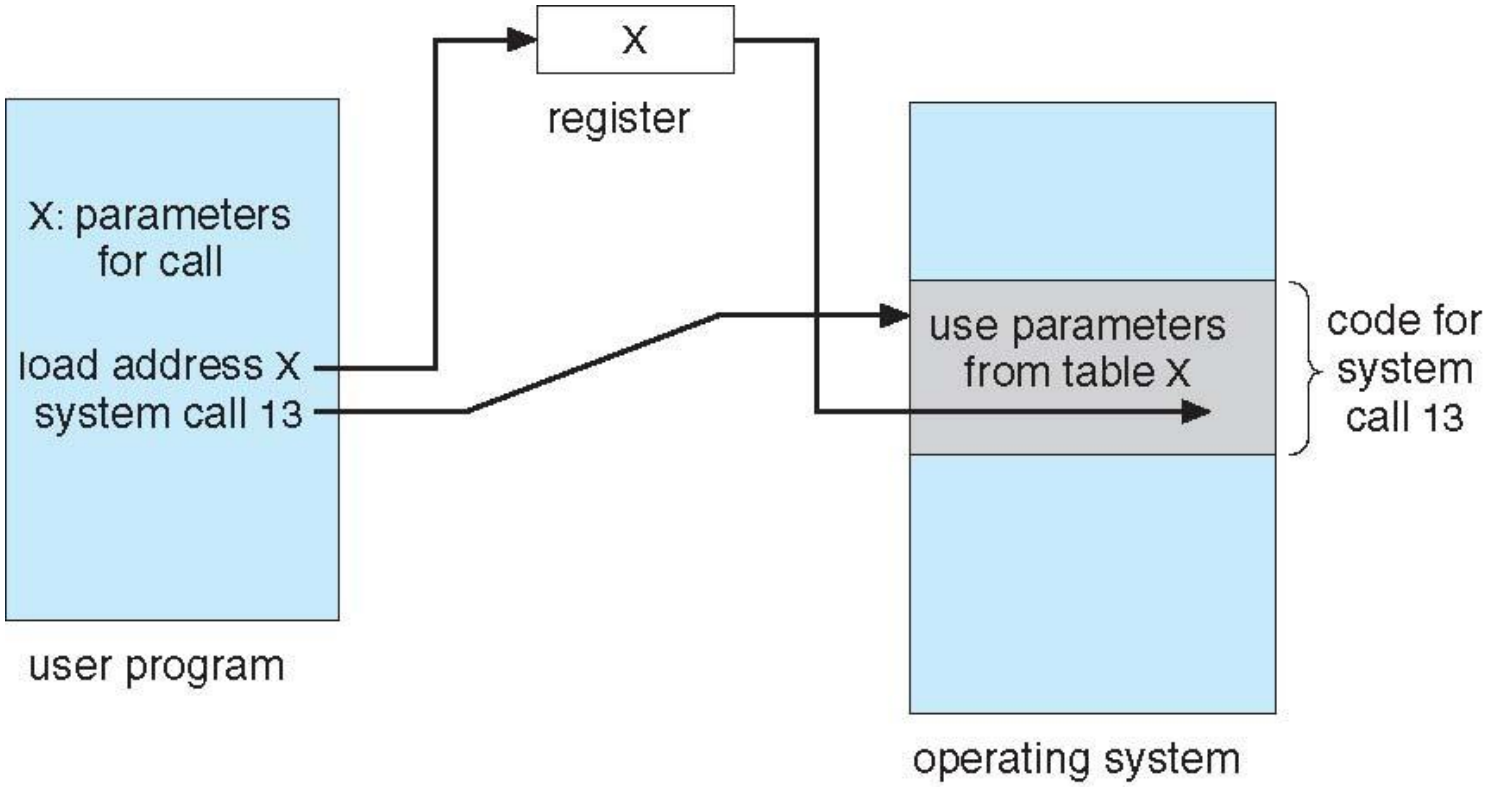
- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters 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

عند استدعاء System Call نحتاج أحيانًا إرسال Parameters.
الطرق الثلاث:
1 Registers
توضع القيم في السجلات — سريعة وسهلة لكن عددها محدود.
2 Memory Block
توضع القيم في Block بالذاكرة ويُمرر عنوانه (مثل Linux) — لا يوجد حد للعدد.
3 Stack
توضع القيم في الـ Stack — أيضًا بدون حد عملي.





Parameter Passing via Table







Types of System Calls

System calls انواع الـ

- Process control
  - 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
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

1 Process Control (التحكم بالعمليات)
إنهاء أو إيقاف عملية (end, abort)
تحميل وتشغيل برنامج (load, execute)
إنشاء أو حذف عملية (create, terminate)
قراءة أو تعديل خصائص العملية
الانتظار لوقت معين أو لحدث معين
تخصيص وتحرير الذاكرة

2 File Management (إدارة الملفات)
إنشاء أو حذف ملف
فتح وإغلاق ملف
قراءة وكتابة
تغيير موقع القراءة داخل الملف (reposition)
قراءة أو تعديل خصائص الملف





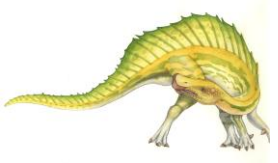
Types of System Calls (Cont.)

- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach and detach remote devices

3 Device Management (إدارة الأجهزة)
طلب جهاز أو تحريره (request / release)
قراءة وكتابة وإعادة تموضع
قراءة أو تعديل خصائص الجهاز
ربط أو فصل جهاز منطقيًا

4 Information Maintenance (إدارة المعلومات)
قراءة أو ضبط الوقت والتاريخ
قراءة أو تعديل بيانات النظام
قراءة أو تعديل خصائص: عملية، ملف، جهاز

5 Communications (التواصل)
إنشاء أو حذف تواصل
إرسال واستقبال رسائل
نقل معلومات الحالة
ربط أو فصل أجهزة بعيدة





Examples of Windows and Unix System Calls

	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()



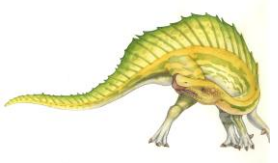


System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs

🔷 ما هي System Programs؟
برامج توفر بيئة مريحة لتطوير وتشغيل البرامج.
وهي التي يتعامل معها المستخدم غالبًا.
تنقسم إلى:
1 File Manipulation
التعامل مع الملفات (نسخ، حذف، إعادة تسمية، عرض محتويات).
2 Status Information
عرض حالة النظام (الذاكرة، المعالج، العمليات).
3 File Modification
تعديل محتوى الملفات (مثل محررات النصوص).
4 Programming Language Support
دعم لغات البرمجة (Compilers, Assemblers).
5 Program Loading & Execution
تحميل وتشغيل البرامج.
6 Communications
التواصل بين المستخدمين أو الأنظمة (مثل الشبكات).
7 Application Programs
برامج يستخدمها المستخدم مباشرة (متصفح، محرر، إلخ).

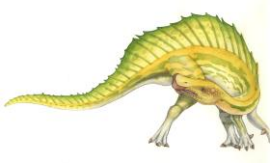
- Most users' view of the operation system is defined by system programs, not the actual system calls





System Programs

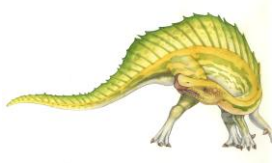
- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
  - Some ask the system for info - 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





System Programs (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **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 another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another



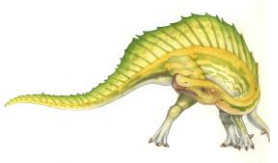


Operating System Design and Implementation

- 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
- Affected by choice of hardware, type of system
- *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

1 User Goals (أهداف المستخدم)
النظام يجب أن يكون:
سهل الاستخدام
سهل التعلم
سريع
آمن
موثوق

2 System Goals (أهداف النظام نفسه)
النظام يجب أن يكون:
سهل التصميم والتنفيذ
سهل الصيانة
مرن (Flexible)
كفء (Efficient)
خالي من الأخطاء قدر الإمكان
موثوق





Operating System Design and Implementation (Cont.)

- Important principle to separate

السياسة **Policy:** What will be done?

الآلية **Mechanism:** How to do it?

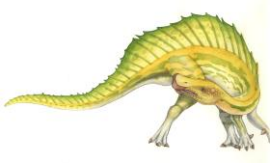
- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

◆ Principle: Separation of Policy and Mechanism

Policy = ماذا سيتم عمله؟

Mechanism = كيف سيتم عمله؟

الفصل بين السياسة (Policy) والآلية (Mechanism) يسمح بتغيير السياسات لاحقًا بدون تعديل الآليات، ويزيد المرونة.





Simple Structure

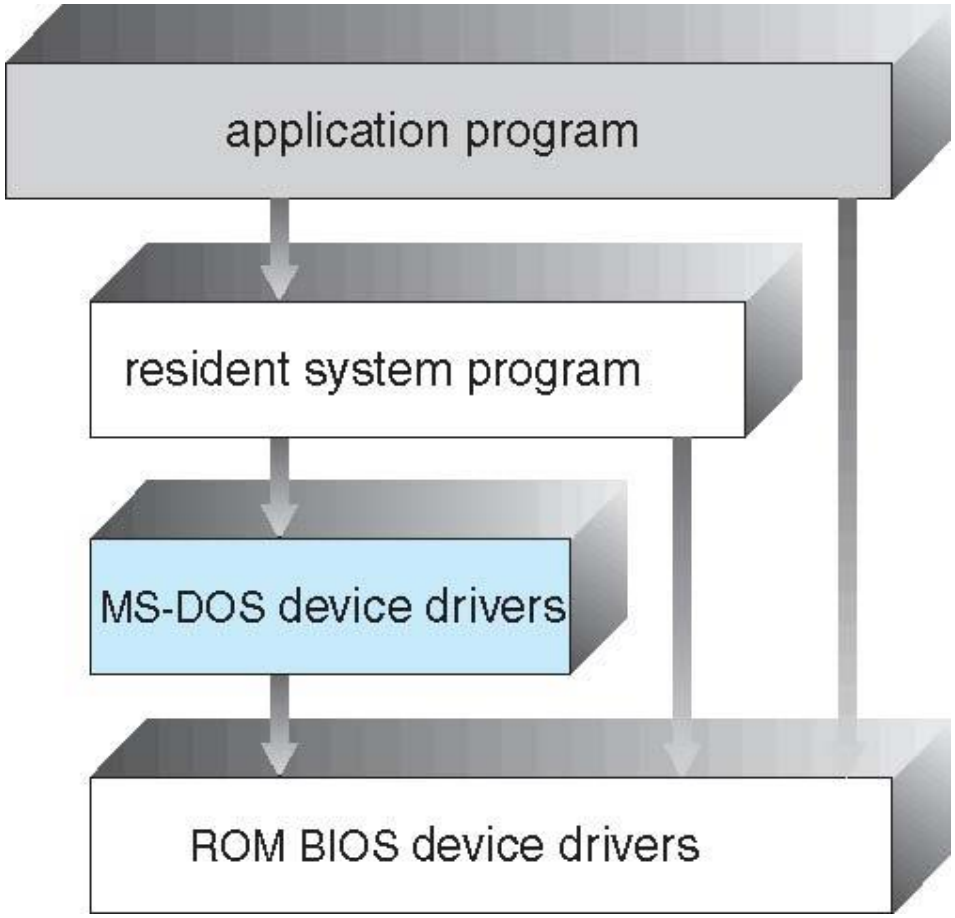
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - E.g., MS-DOS – written to provide the most functionality in the least space
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

MS-DOS
من امثلة البنية البسيطة Simple Structure
هدفه: توفير أكبر قدر من الوظائف في أقل مساحة ممكنة.
مميزاته:
ليس مقسمًا إلى وحدات واضحة (Not modular)
واجهاته ووظائفه غير مفصولة بوضوح
رغم ان لديه البنية البسيطة، إلا أن الفصل بين المستويات ضعيف.



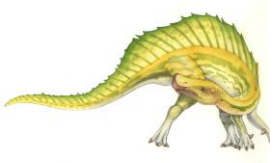


MS-DOS Layer Structure



يمكن الانتقال بين المستويات بسهولة ولكن هذا يقلل من الاعتمادية على النظام







Layered Approach

- 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.

- With modularity (independent), layers are selected such that each uses functions (operations) and services of only lower-level layers

Layered Approach يعني أن نظام التشغيل يُقسم إلى طبقات (Layers).

الطبقة السفلى (Layer 0) → الهاردوير (Hardware)

الطبقات الوسطى → أجزاء نظام التشغيل

أعلى طبقة (Layer N) → واجهة المستخدم (User Interface)

كل طبقة تعتمد فقط على الطبقة التي تحتها.

الفكرة الأساسية

كل طبقة تستخدم خدمات الطبقة الأدنى منها، ولا تتعامل مباشرة مع الطبقات البعيدة

مثال بسيط

User Interface

↓

System Programs

↓

Operating System Services

↓

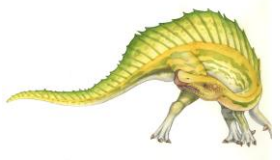
Hardware

ميزة هذا التصميم

✔ تنظيم أفضل للنظام

✔ أسهل في التطوير والصيانة

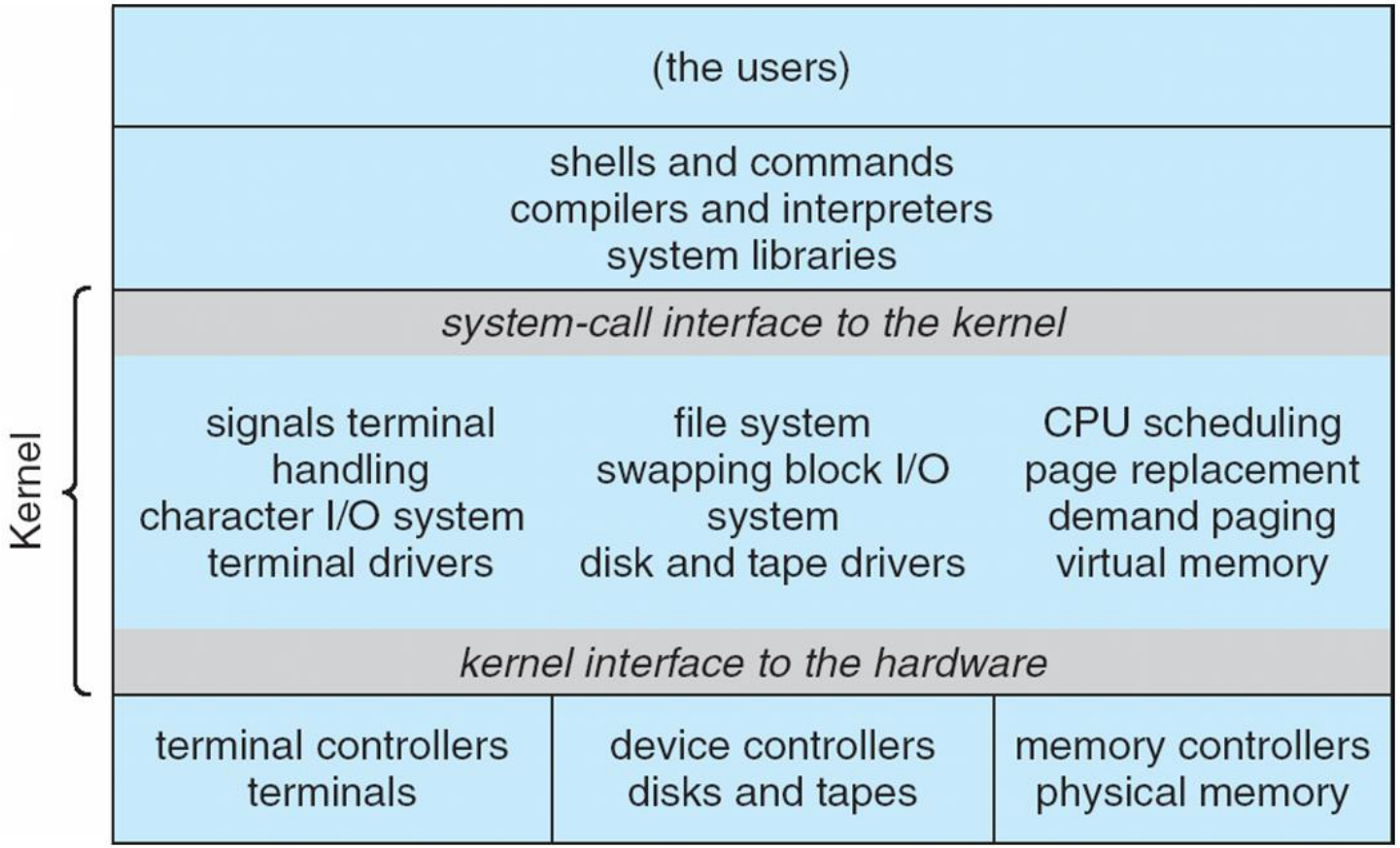
✔ كل طبقة مستقلة نسبيًا (Modular)





Traditional UNIX System Structure

مثال توضيحي



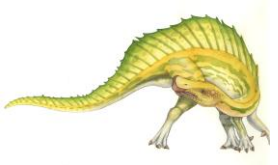




UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - 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

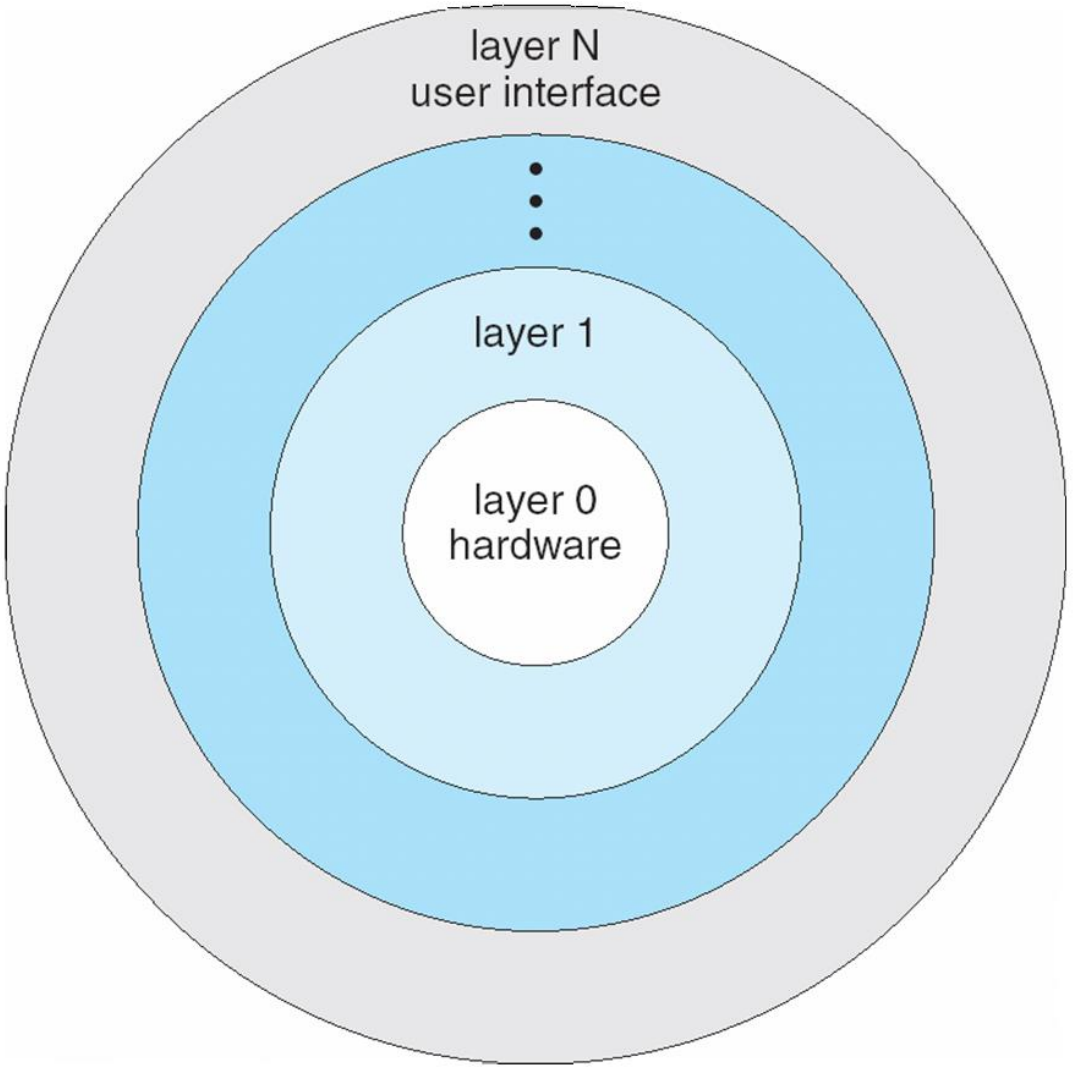
بسبب محدودية الهاردوير في ذلك الوقت، لم يكن نظام UNIX منظمًا بطبقات كثيرة
كان يتكون أساسًا من جزئين رئيسيين:
1- System Programs
برامج يتعامل معها المستخدم مثل:
إدارة الملفات
تشغيل البرامج
أدوات النظام
2- Kernel
النواة وهي قلب نظام التشغيل.
تقع بين:
System Call Interface
Hardware
وتقوم بوظائف أساسية مثل:
(File System) إدارة الملفات
(CPU Scheduling) جدولة المعالج
Memory) إدارة الذاكرة
(Management
وظائف نظام التشغيل الأخرى

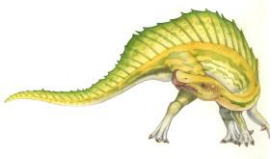




Layered Operating System

مثال توضيحي







Microkernel System Structure

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

في Microkernel يتم نقل معظم وظائف نظام التشغيل من الـ Kernel إلى User Space.
يبقى في الـ Kernel فقط الوظائف الأساسية جدًا.

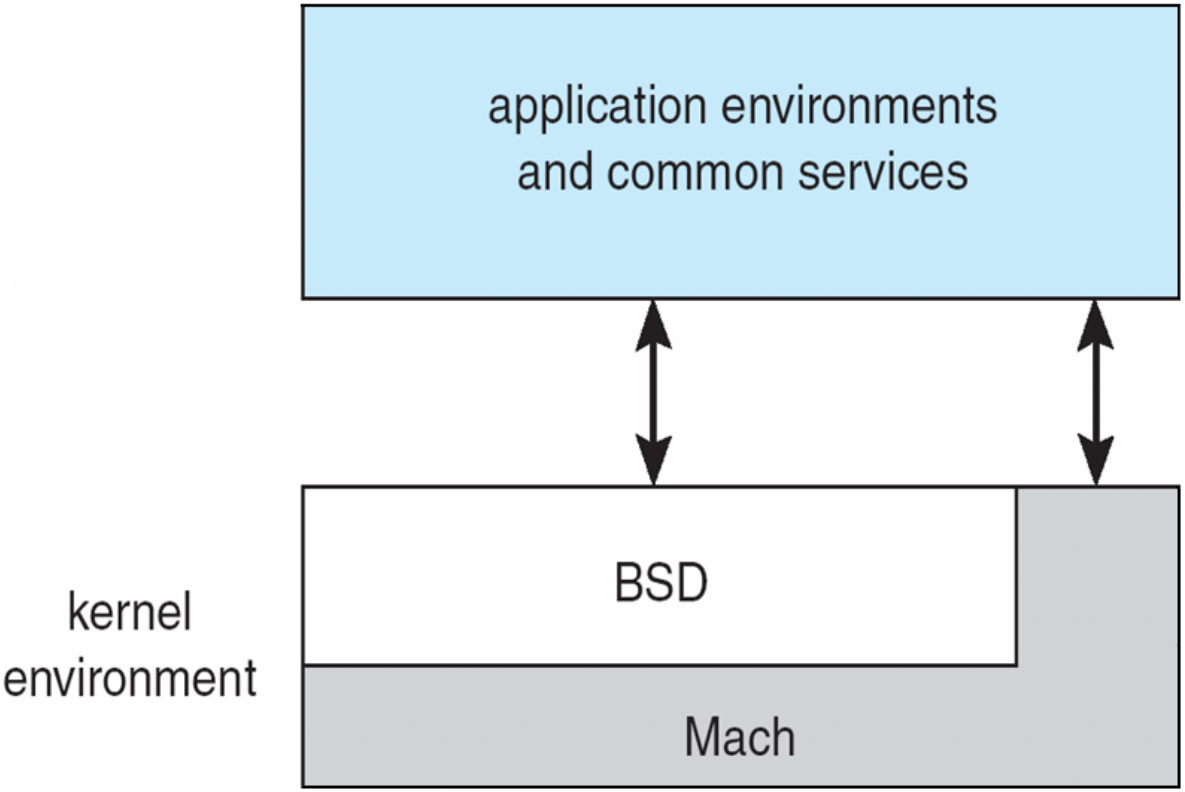
كيف تتواصل الأجزاء؟
التواصل بين الوحدات يتم عبر Message Passing (إرسال رسائل).
المميزات:
✔ أسهل في التوسيع (Extend)
✔ أسهل نقل النظام إلى أجهزة مختلفة (Portability)
✔ أكثر موثوقية لأن الكود داخل Kernel أقل
✔ أكثر أمانًا
العيوب:
بطء نسبي بسبب كثرة التواصل بين
Kernel Space و User Space

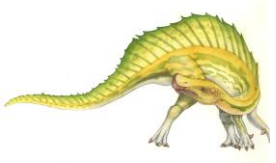




Mac OS X Structure

نظام macOS (Mac OS X سابقًا) يستخدم بنية هجينة (Hybrid Structure) تجمع أكثر من أسلوب تصميم.







Modules

- Most modern operating systems implement 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

- Overall, similar to layers but with more flexible

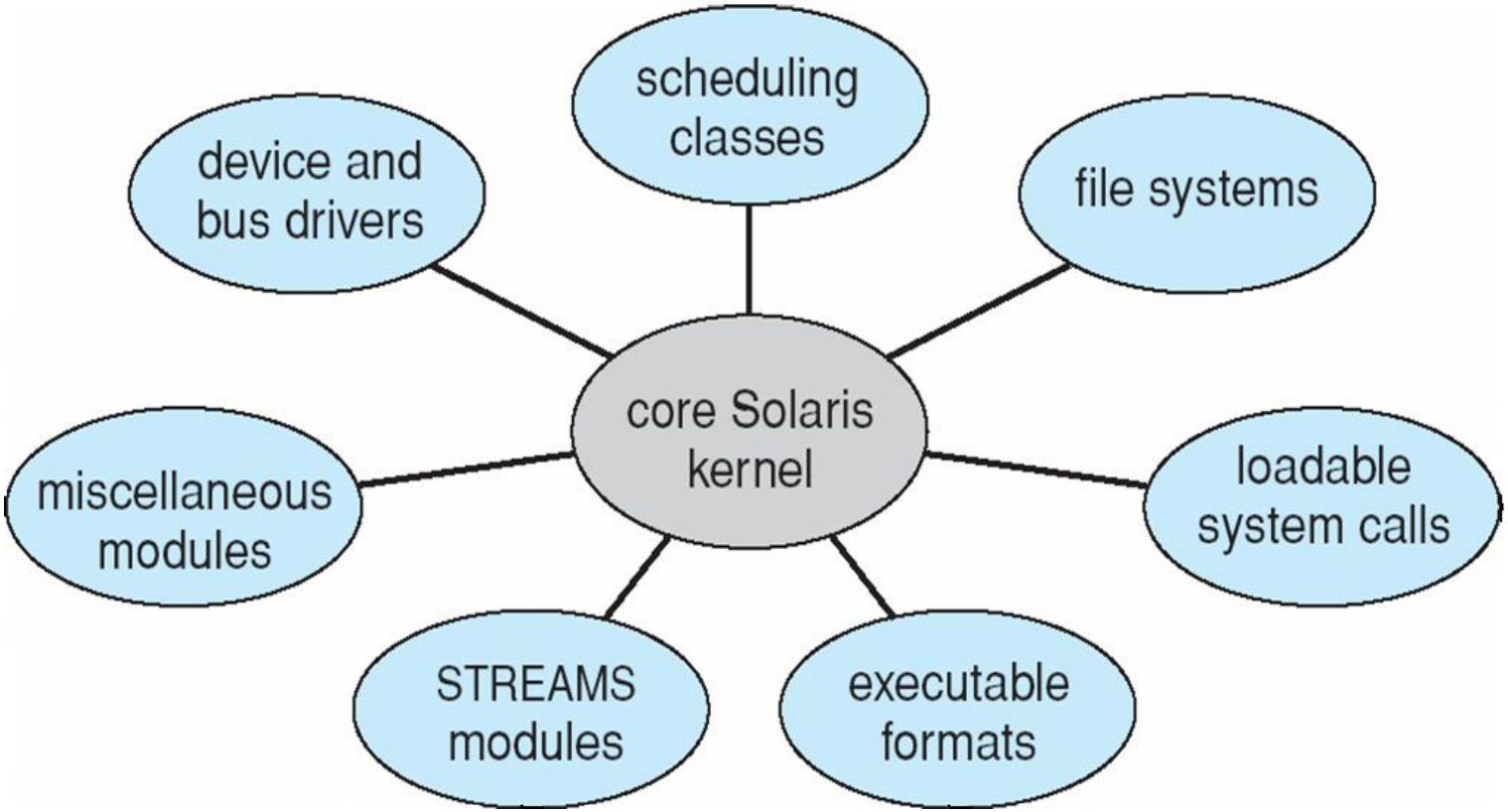
معظم أنظمة التشغيل الحديثة تستخدم Kernel Modules.
يتم تقسيم نظام التشغيل إلى وحدات مستقلة (Modules).
كل جزء من النظام يكون منفصل
يتواصل مع الأجزاء الأخرى عبر Interfaces معروفة
يمكن تحميله داخل الـ Kernel عند الحاجة
الخصائص
✔ يستخدم فكرة قريبة من Object-Oriented
✔ كل مكون أساسي مستقل
✔ يمكن إضافة أو إزالة Modules بسهولة

بشكل عام : يشبه Layered Structure لكنه أكثر مرونة





Solaris Modular Approach







Virtual Machines

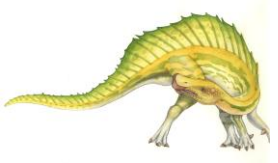
- A **virtual machine** takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system **host** creates the illusion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





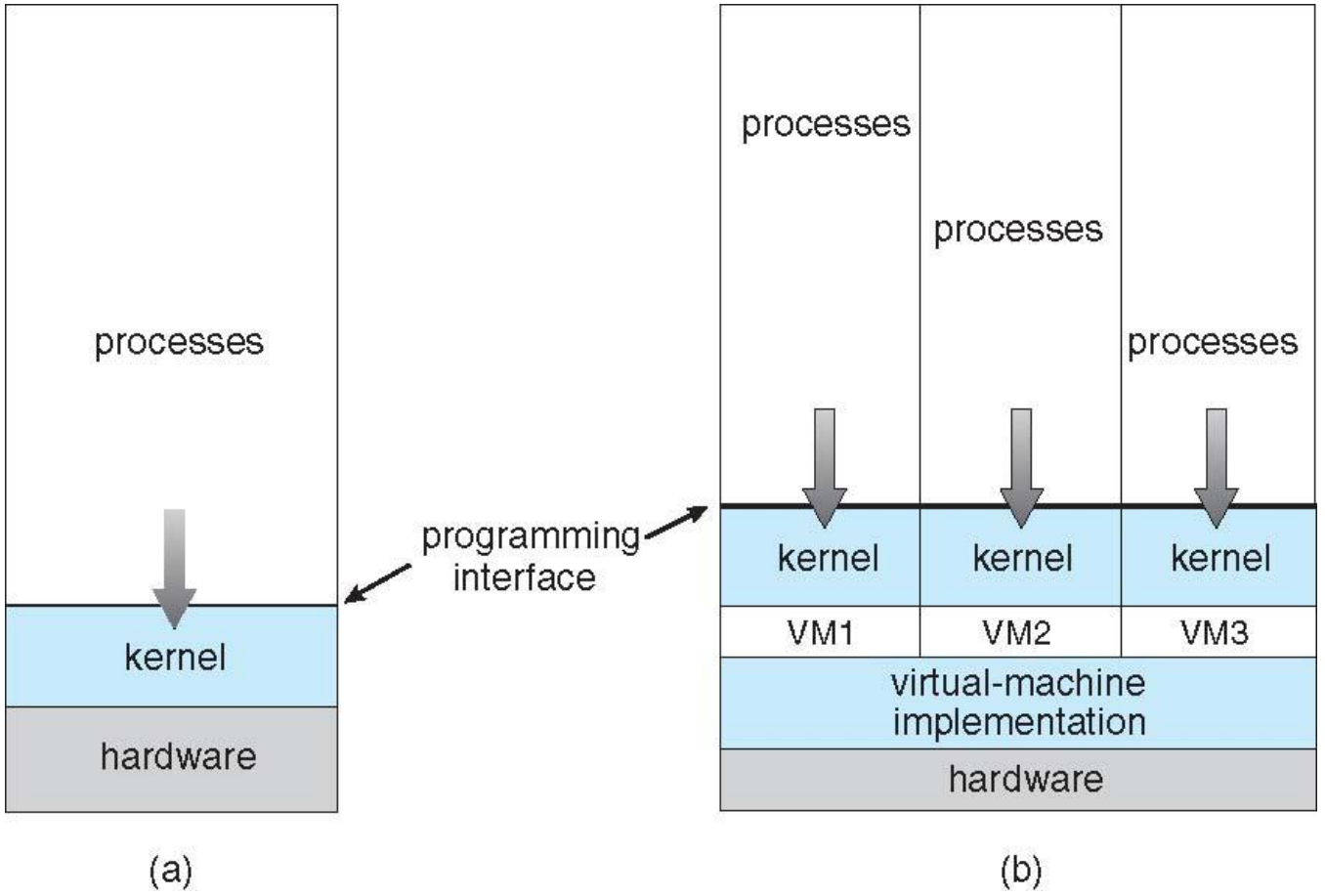
Virtual Machines History and Benefits

- First appeared commercially in IBM mainframes in 1972
- Fundamentally, multiple execution environments (different operating systems) can share the same hardware
- Protect from each other
- Some sharing of file can be permitted, controlled
- Commutate with each other, other physical systems via networking
- Useful for development, testing
- **Consolidation** (unite) of many low-resource use systems onto fewer busier systems
- “Open Virtual Machine Format”, standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms





Virtual Machines (Cont.)

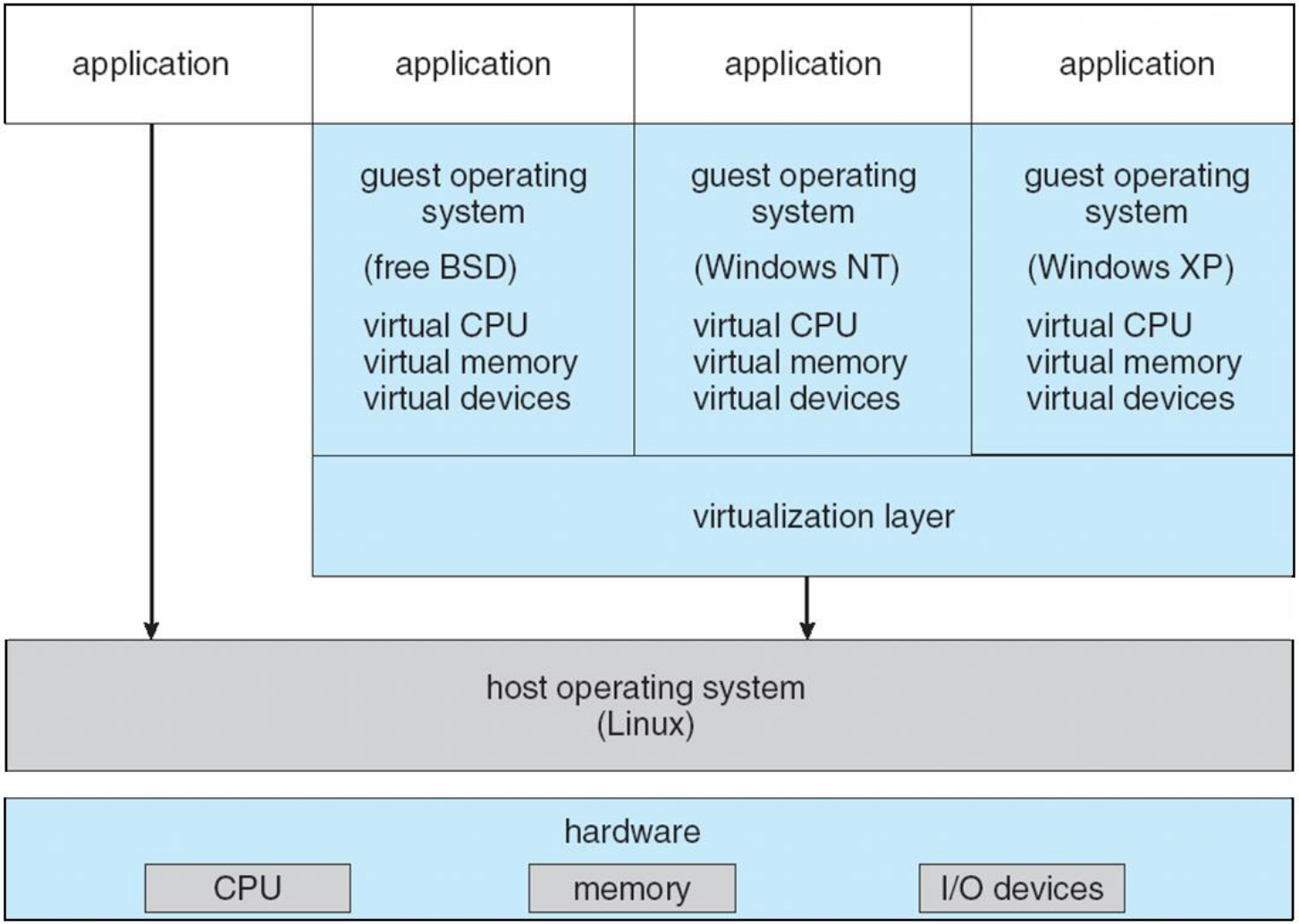


(a) Nonvirtual machine (b) virtual machine





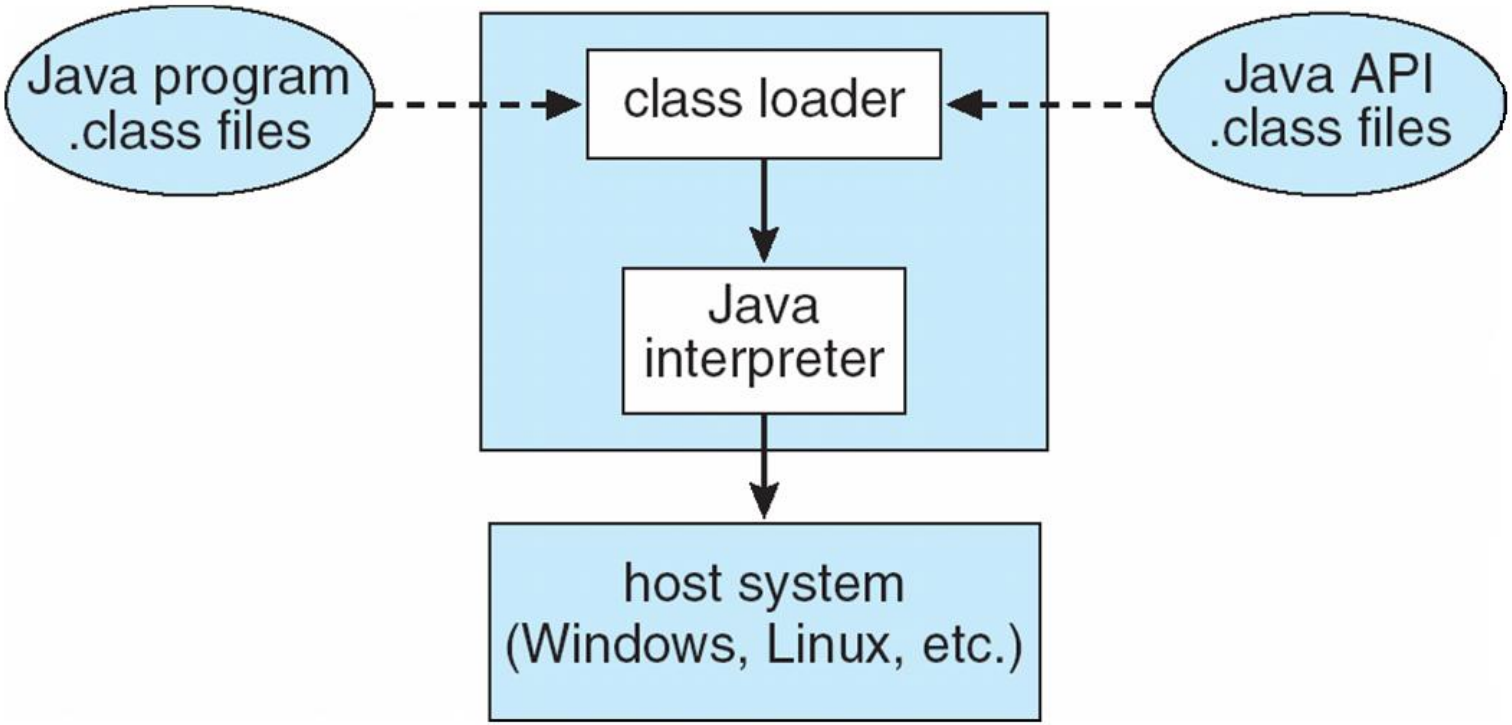
VMware Architecture

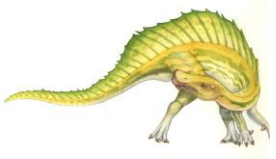






The Java Virtual Machine

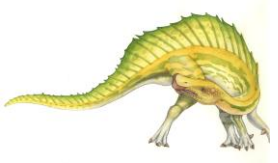






Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OSes generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
- Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
  - **Probes** fire when code is executed, capturing state data and sending it to consumers of those probes



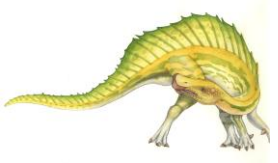


Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- SYSGEN program obtains information concerning the specific configuration of the hardware system



- *Booting* – starting a computer by loading the kernel
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution





System Boot

- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
  - When power initialized on system, execution starts at a fixed memory location
    - Firmware used to hold initial boot code

