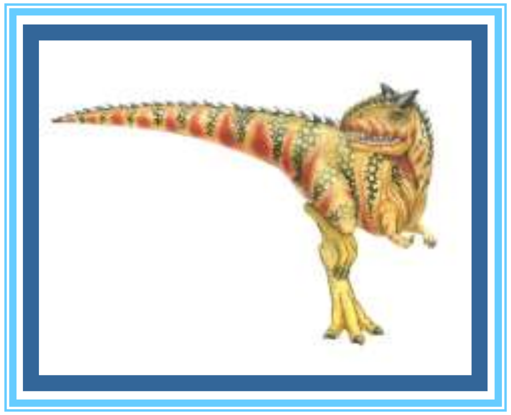
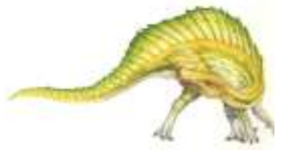
Chapter 1: Introduction



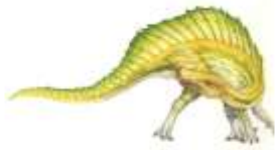
Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Process Management
- Memory Management
- Storage Management



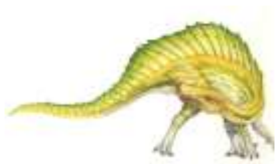
Objectives

- To describe the basic organization of computer systems
- To provide a grand tour of the major components of operating systems



What is an Operating System?

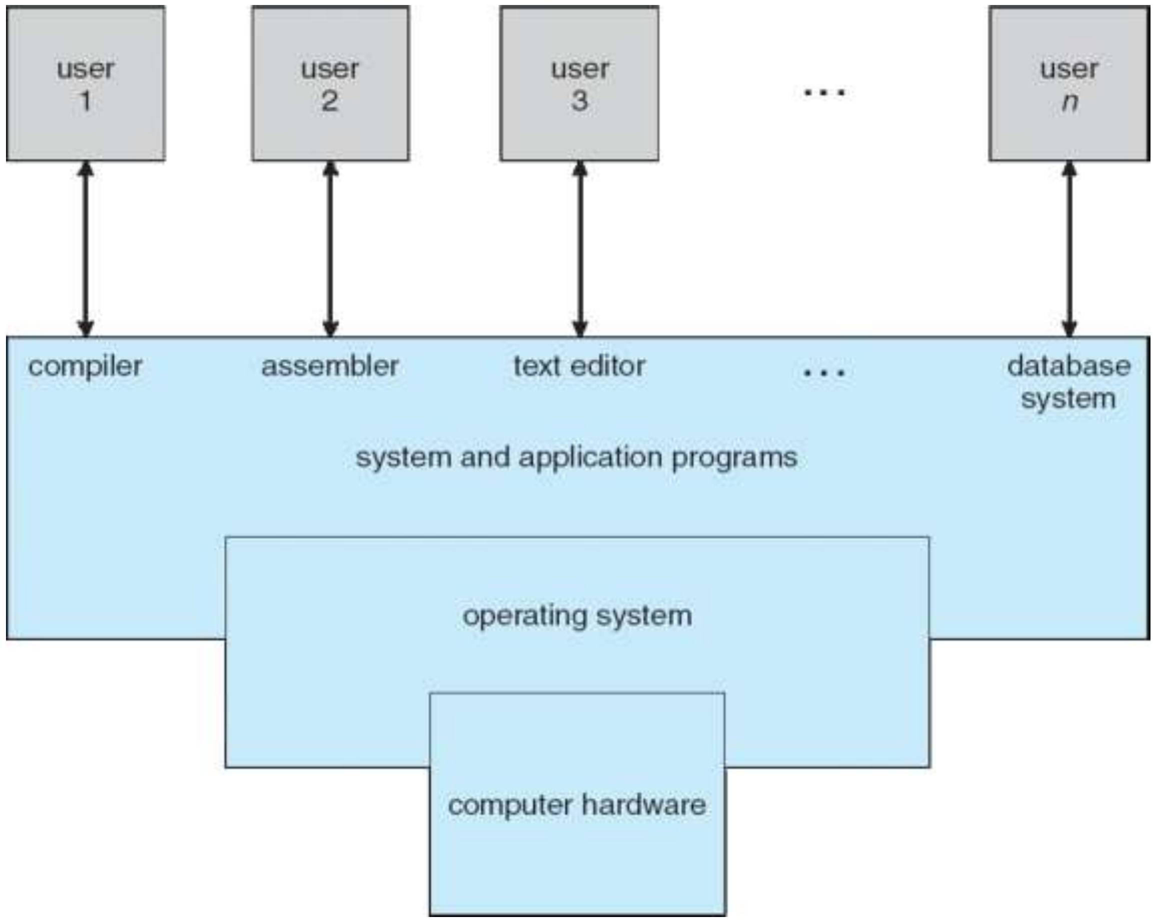
- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner



Computer System Structure

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

Four Components of a Computer System



Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Operating System Definition (Cont.)

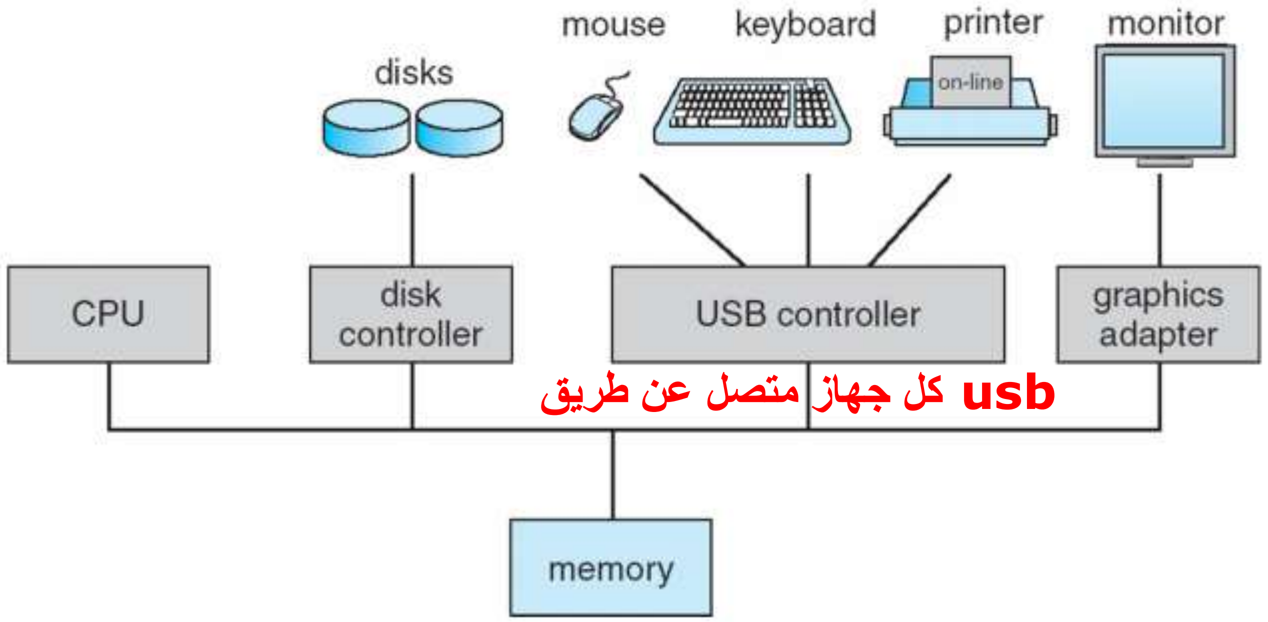
- No universally accepted definition
- "The one program running at all times on the computer" is the **kernel**.
- Everything else is either
  - a system program (ships with the operating system) , or
  - an application program.

Computer Startup

- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

Computer System Organization

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



في أنظمة التشغيل، كل جهاز مرتبط مثل الطابعة او الكيبورد يجب ان يكون لديه كونترولر لاجل ان يتعامل المعالج معه عن طريقه وليس بصورة مباشرة كمثال استقبال او ارسال بيانات للطابعة

Computer-System Operation

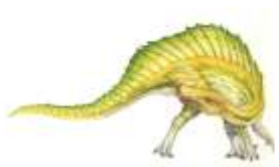
متزامن

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a specific device
- Each device controller has a local **buffer** **ذاكرة صغيرة محلية مؤقتة**
- 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

البفر بدونه لايمكن نقل البيانات بين الذاكرة والطابعة ايضا هنا تعامل... غير مباشر بين الاثنين

اعلام الجهاز المعالج بانه قد انهى العملية عن طريق التسبب ب مقاطعة

المقاطعة عبارة عن إشارة تظهر بشكل مفاجيء تتطلب اهتمامًا فوريًا من نظام التشغيل. تقوم بإعلام المعالج بأن هنالك عملية حرجة تحتاج إلى تنفيذ عاجل. في مثل هذه الحالة، تتم مقاطعة عملية العمل الحالية.



Common Functions of Interrupts

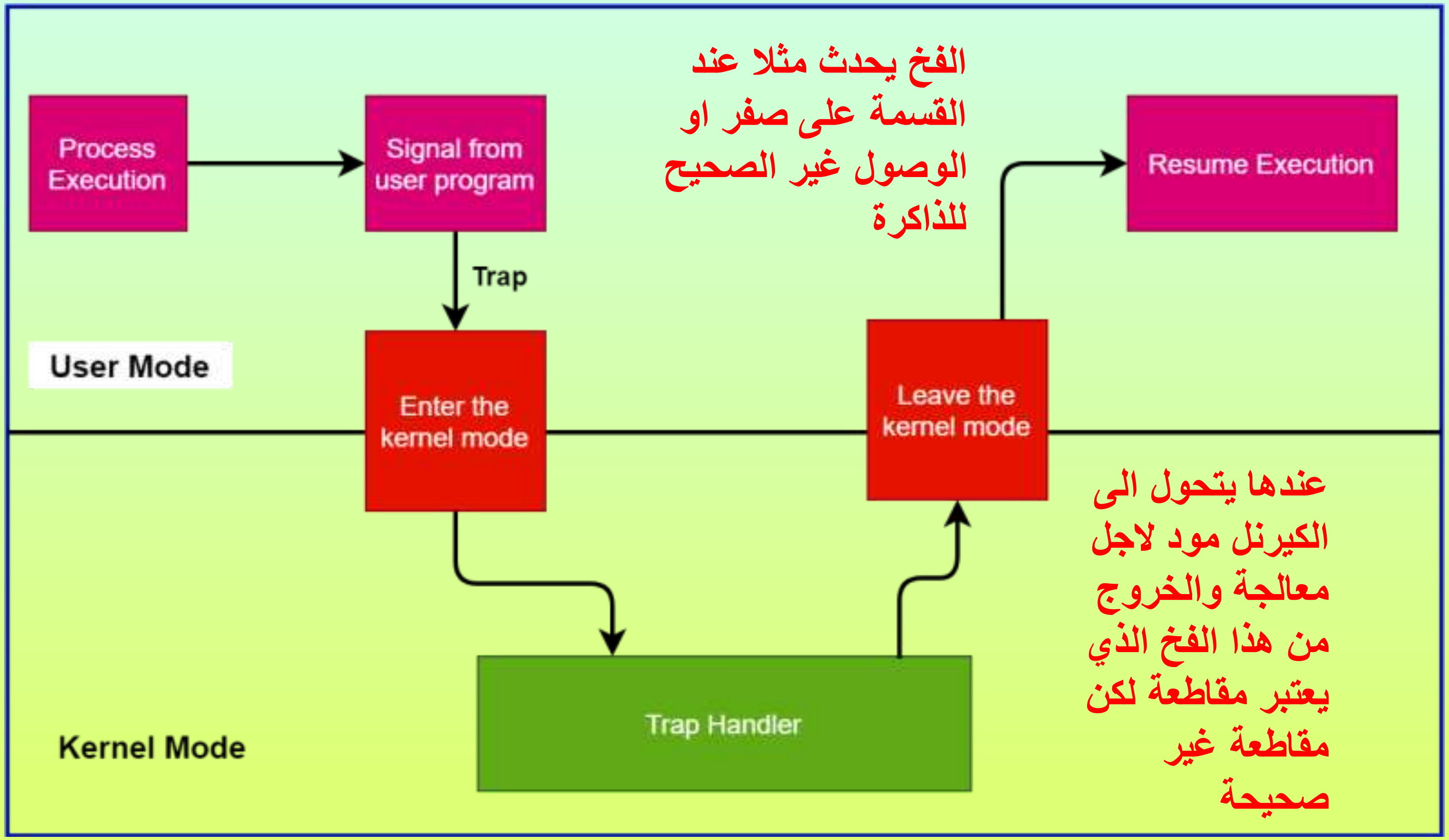
ناقل/موجه المقاطعة

- 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
- A **trap** or **exception** is a software-generated interrupt caused either by an **error or a user request**
- An operating system is **interrupt driven**

هو روتين برمجي يستدعيه الجهاز استجابة للمقاطعة. يقوم هذا الروتين بفحص المقاطعة وتحديد كيفية التعامل معها، ثم يقوم بتنفيذ المعالجة.

في أنظمة التشغيل، تعتمد التطبيقات على الأحداث من أجل تنفيذ الوظائف.لذلك يعتبر نظام التشغيل ايضا مقاد بالمقاطعات

Interrupt Handling



الفخ يحدث مثلا عند القسمة على صفر او الوصول غير الصحيح للذاكرة

عندها يتحول الى الكيرنل مود لاجل معالجة والخروج من هذا الفخ الذي يعتبر مقاطعة لكن مقاطعة غير صحيحة

Interrupt Handling

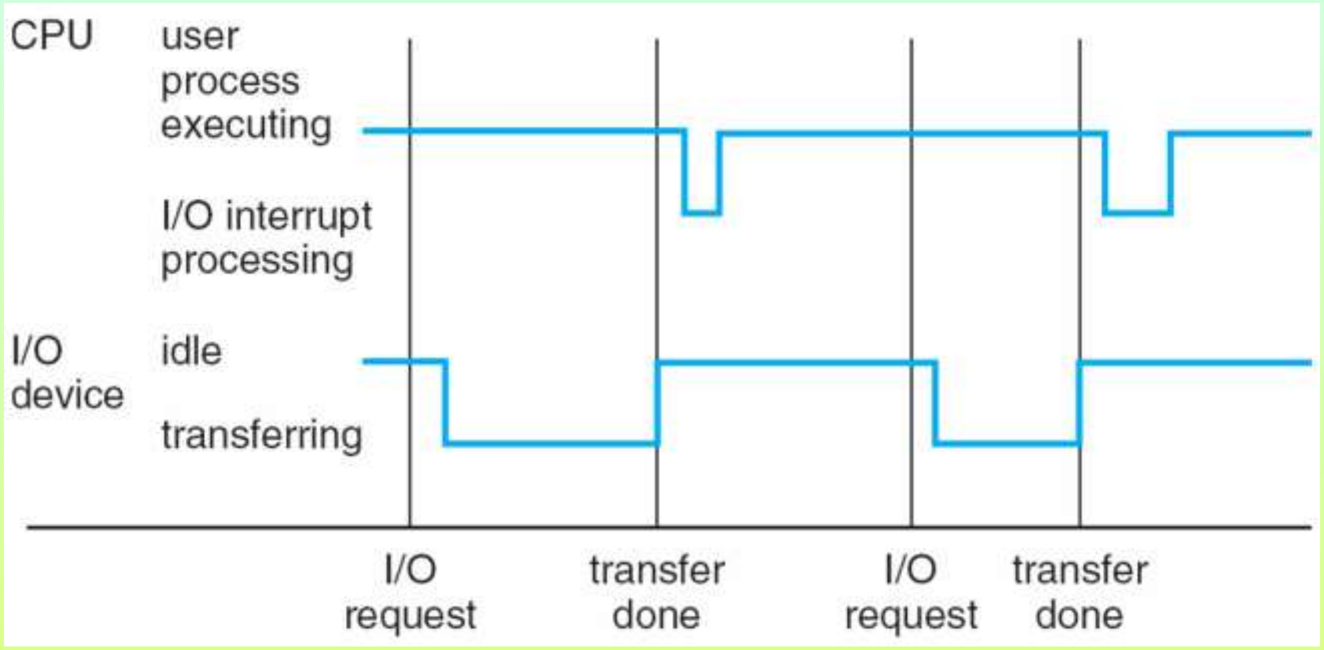
- The operating system preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - **polling**

    نوع معين من مقاطعة الإدخال/الإخراج التي ترسل رسالة إلى جزء الكمبيوتر الذي يضم واجهة الإدخال/الإخراج. تشير الرسالة إلى أن الجهاز جاهز للوصول إليه بدون جهاز تعريف

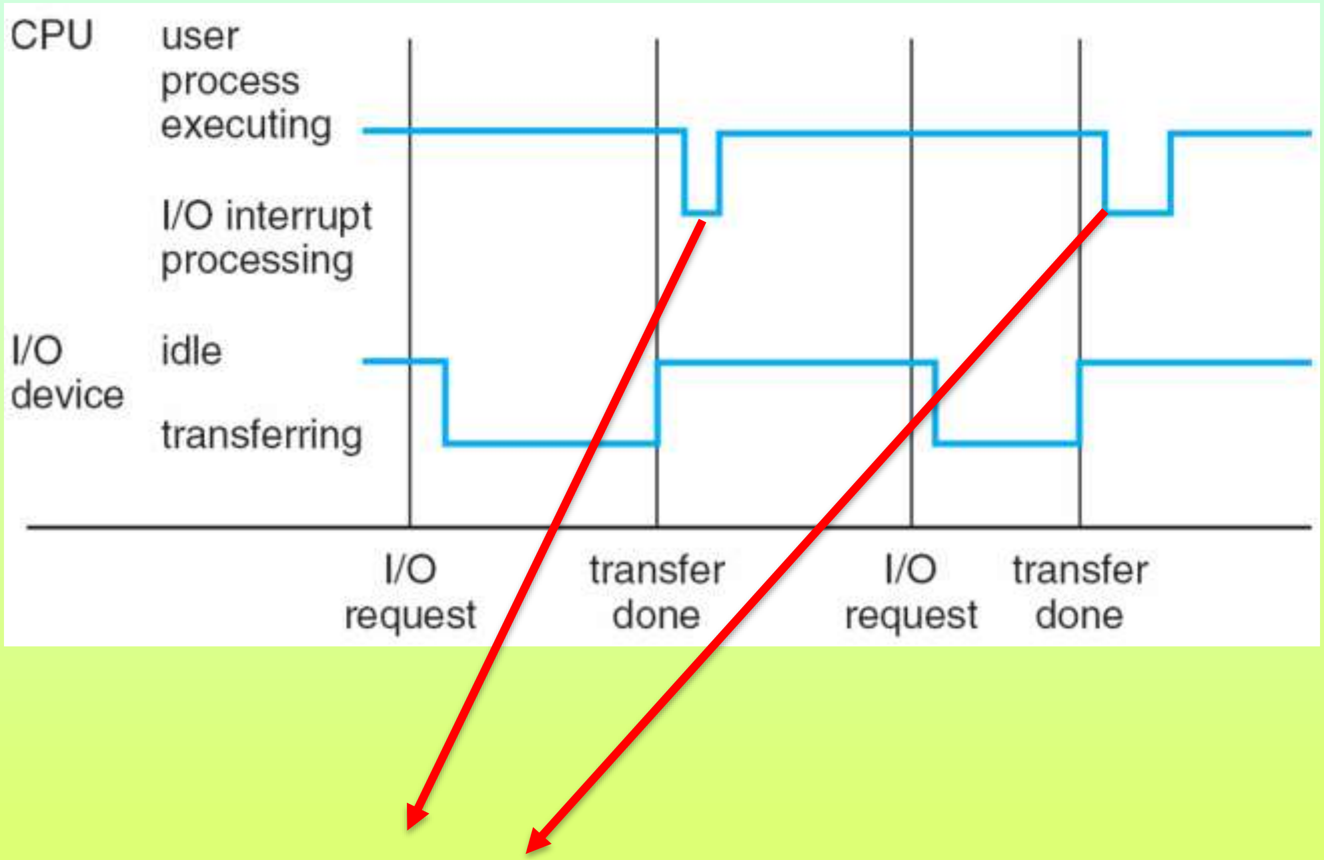
    المقاطعات الموجهة
  - **vectored** interrupt system

    في المقاطعات الموجهة، يقوم الجهاز الذي يطلب المقاطعة بتعريف نفسه مباشرة عن طريق إرسال رمز خاص إلى المعالج عبر الناقل. يتيح ذلك للمعالج التعرف على الجهاز الذي قام بإنشاء المقاطعة
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Timeline



Interrupt Timeline



Signal from i/o to the processor to inform finishing data transfer

I/O Structure

وفيه حالتين :

- After I/O starts, control returns to user program only upon I/O completion

1. انتظار المعالج اكمال 1/0 العودة الى برنامج المستخدم بعد اكمال عمليات 1/0 وذلك يتطلب ما يلي :

تعليمة انتظار تعطل المعالج لحين المقاطعة التالية

  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access) حلقة انتظار (التنافس للوصول الى الذاكرة)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing

في هذه الحالة يعالج طلب 1/0 واحد وبدون تزامن

2. بعد بدء عملية 1/0 التحكم يعود الى برنامج المستخدم دون انتظار 1/0 ، وذلك يتطلب ما يلي :

- After I/O starts, control returns to user program without waiting for I/O completion
  - **System call** – request to the OS 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
  - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt

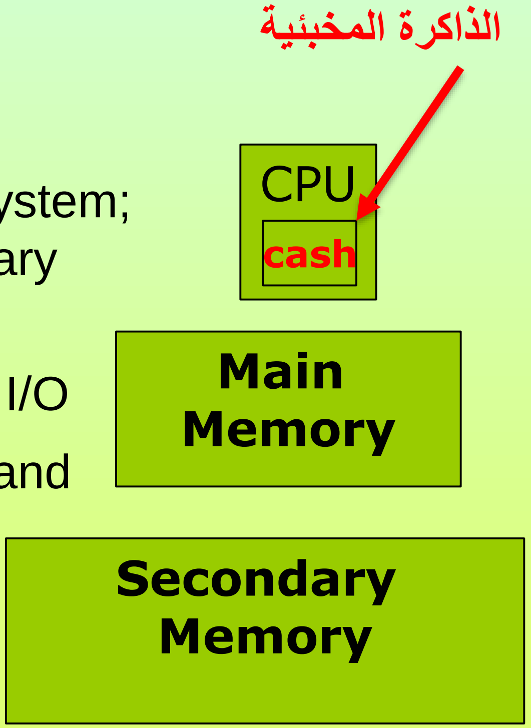
Storage Structure

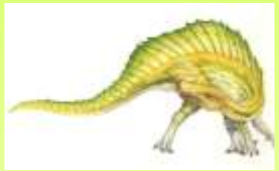
- Main memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Hard disks – rigid metal or glass platters covered with magnetic recording material
  - 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
- **Solid-state disks** – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular

Storage Hierarchy

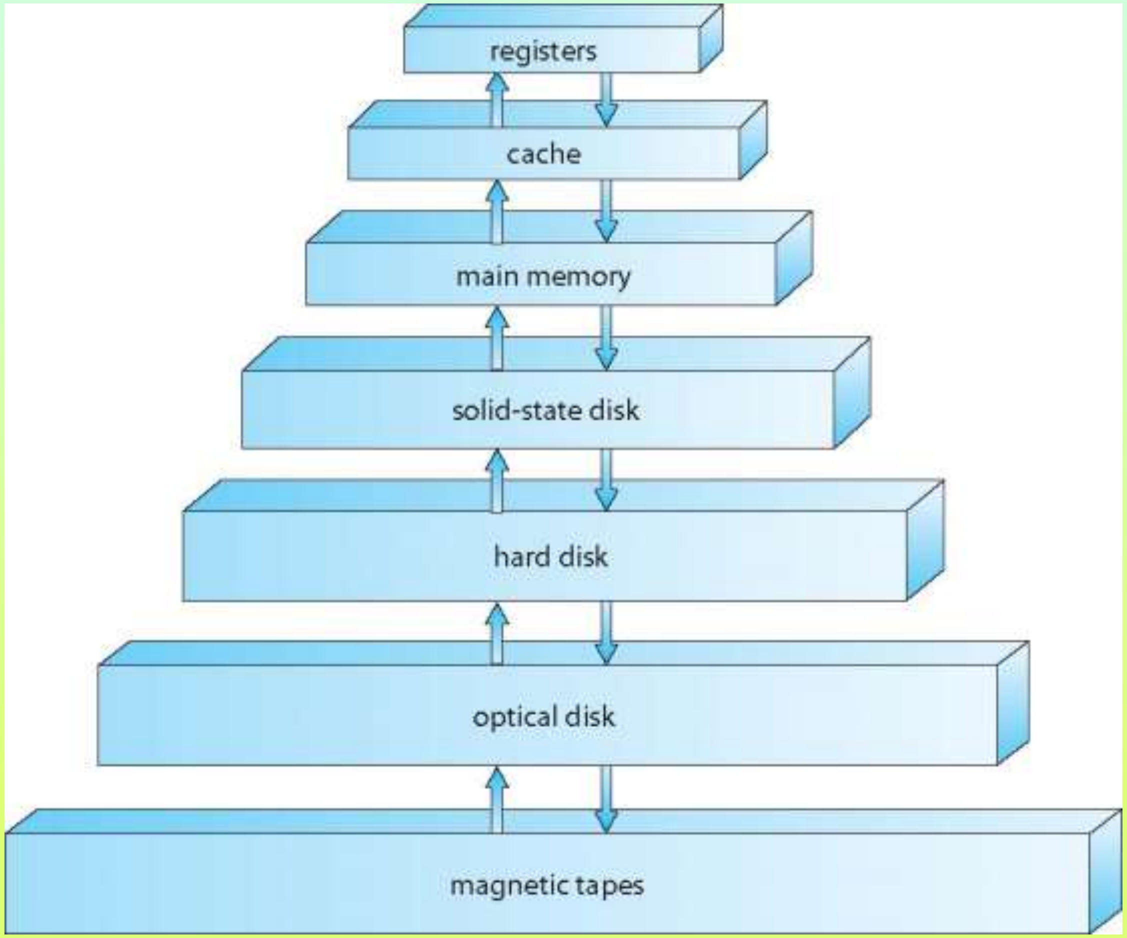
ينظم الخزن هرميا اعتمادا على :

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- **Device Driver** for each device controller to manage I/O
  - Provides **uniform interface** between controller and kernel





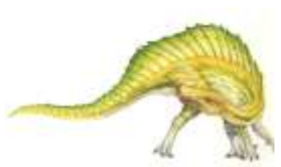
Storage-Device Hierarchy



Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- 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 smaller than **storage being cached**
  - Cache management important design problem
  - Cache size and replacement policy

الذاكرة المخبئية اقل حجما من الذاكرة التي نسخت منه

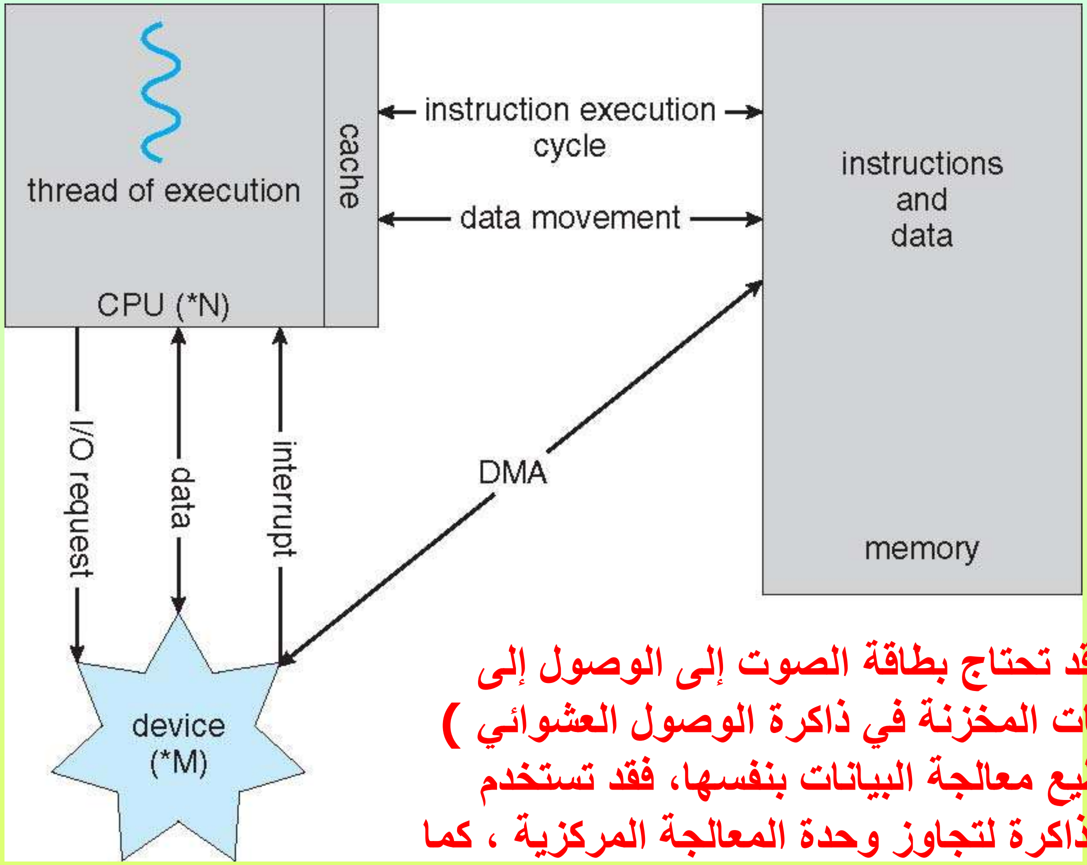


Direct Memory Access Structure

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

الوصول المباشر للذاكرة هو عملية نقل البيانات دون تدخل المعالج نفسه. غالبًا ما يتم استخدامه لنقل البيانات من/إلى أجهزة الإدخال/الإخراج

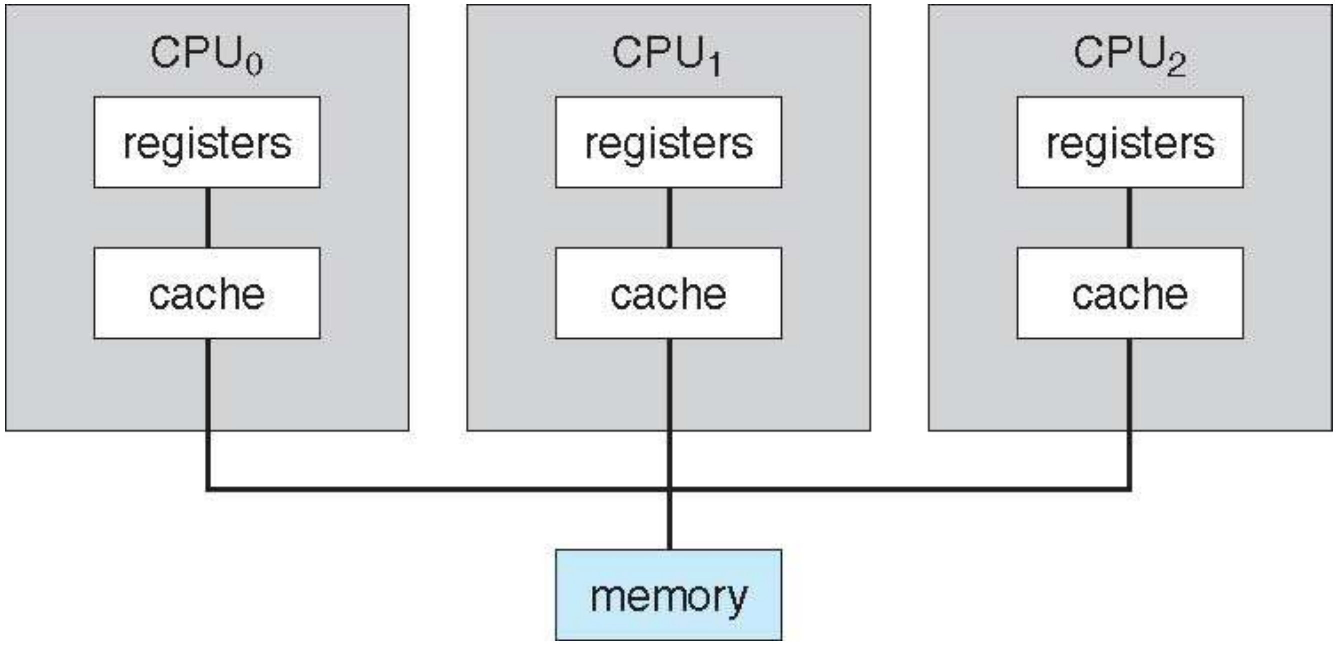
How a Modern Computer Works



على سبيل المثال، قد تحتاج بطاقة الصوت إلى الوصول إلى للكمبيوتر، (البيانات المخزنة في ذاكرة الوصول العشوائي) ولكن بما أنها تستطيع معالجة البيانات بنفسها، فقد تستخدم الوصول المباشر للذاكرة لتجاوز وحدة المعالجة المركزية ، كما يمكن لبطاقات الفيديو الوصول إلى ذاكرة النظام ومعالجة الرسومات دون الحاجة إلى وحدة المعالجة المركزية.

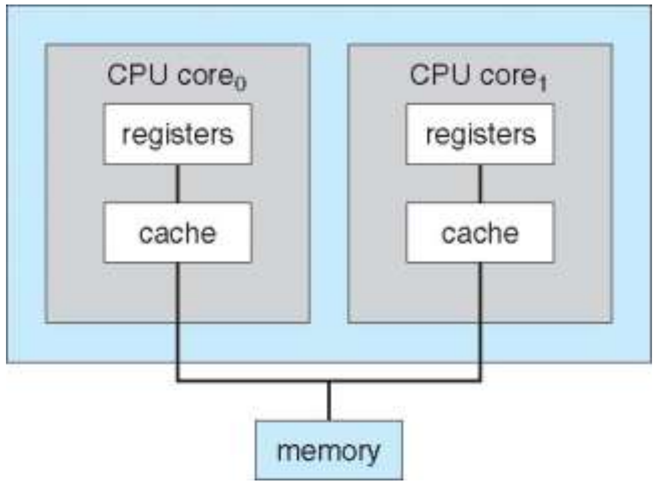
A von Neumann architecture

Symmetric Multiprocessing Architecture



A Dual-Core Design

- Multi-chip and **multicore**
- Systems containing all chips
  - Chassis containing multiple separate systems

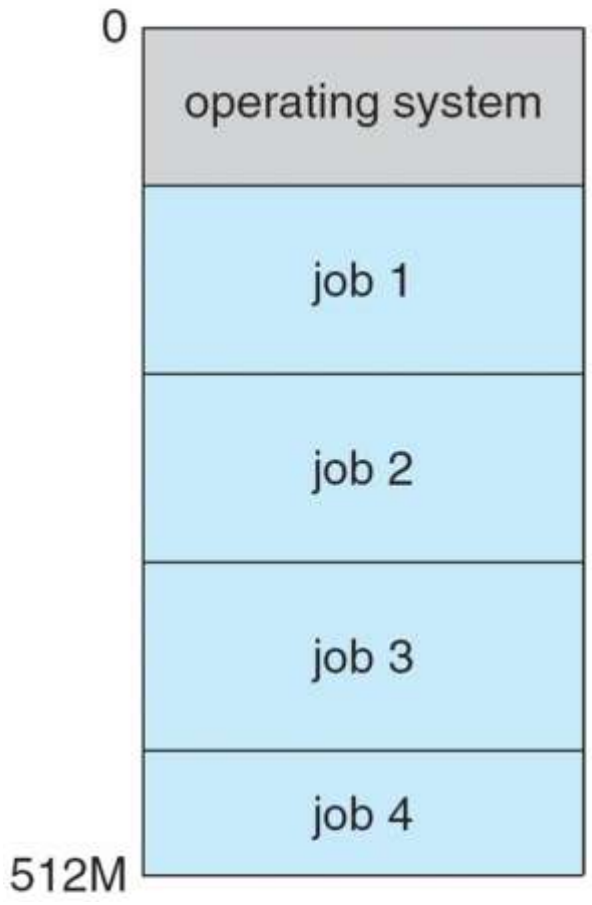


The more cores a CPU has, the more tasks it can perform simultaneously. One core can perform one task at a time while other cores handle other tasks the system assigns. This way, the overall performance is substantially improved when compared to old single-core CPUs.

Operating System Structure

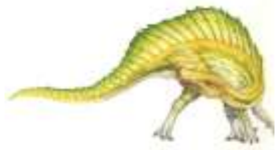
- **Multiprogramming** (**Batch system**) needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing** (**multitasking**) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 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 not completely in memory

Memory Layout for Multiprogrammed System



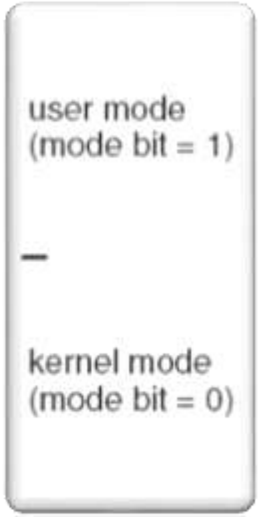
Operating-System Operations

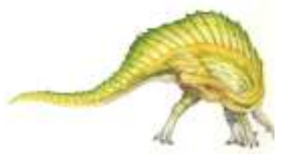
- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., division by zero)
    - Request for operating system service
    - Other process problems include infinite loop, processes modifying each other or the operating system



Operating-System Operations (cont.)

- **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
- Increasingly CPUs support multi-mode operations
  - i.e. **virtual machine manager** (**VMM**) mode for guest **VMs**

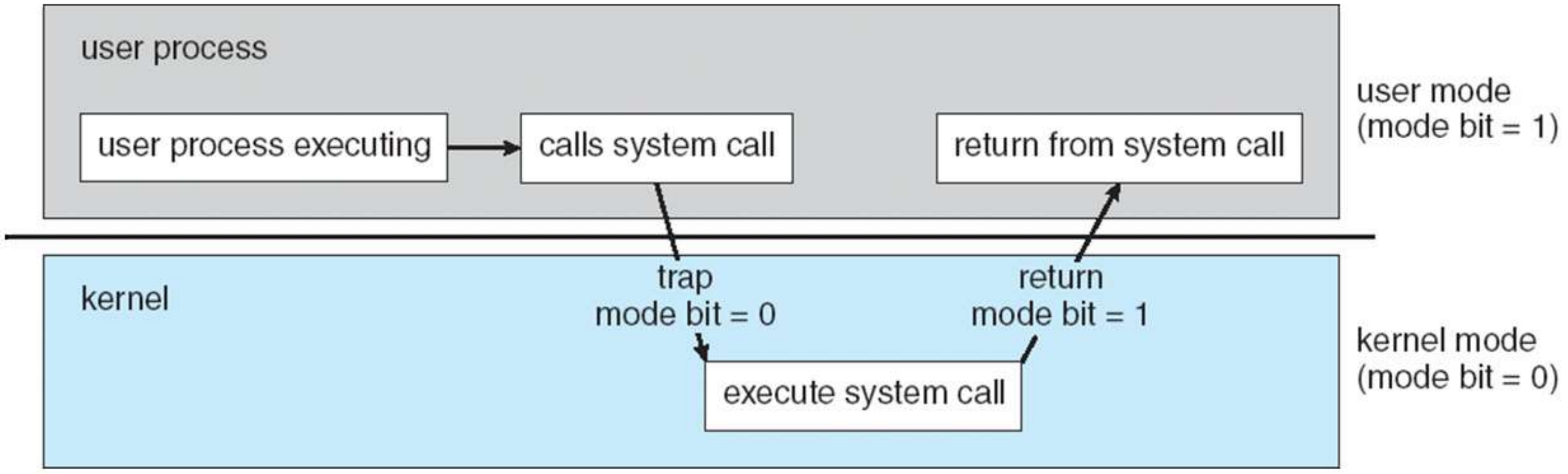




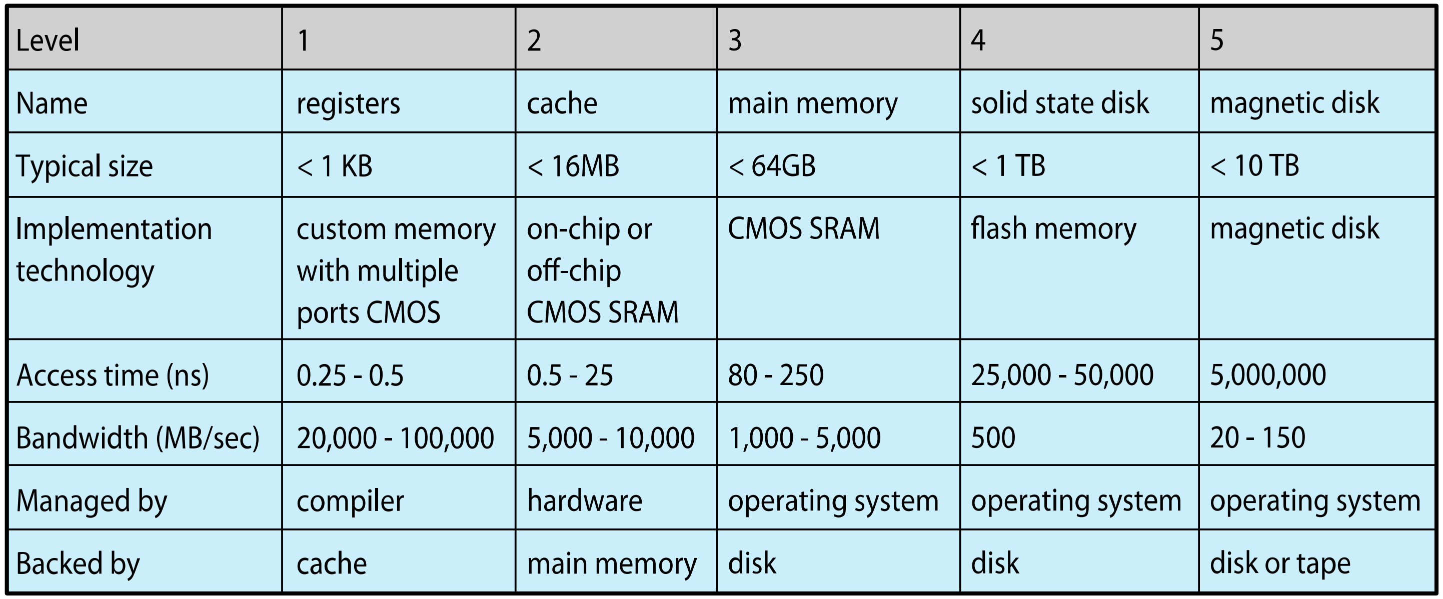
Transition from User to Kernel Mode

استنزاف المصادر

- Timer to prevent infinite loop / process hogging resources
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock.
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Performance of Various Levels of Storage

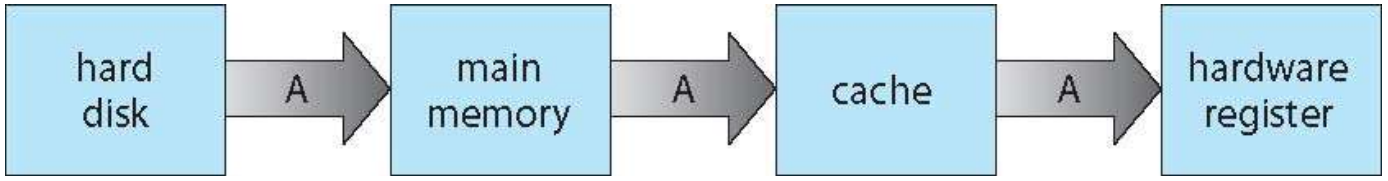


Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

Migration of data “A” from Disk to Register

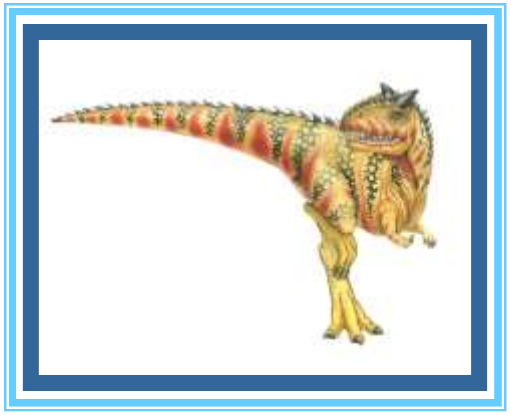
- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



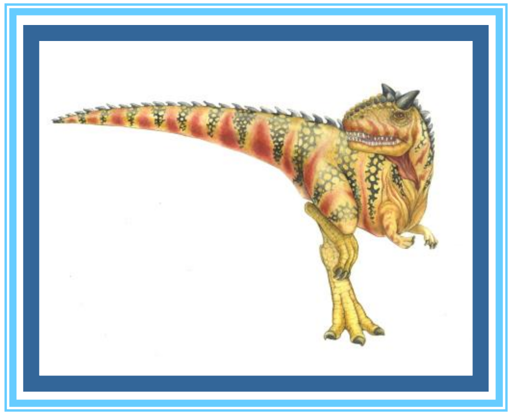
ترابط/تماسك الذاكرة او الذواكر المخبئية

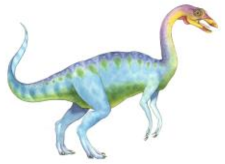
- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 17

End of Chapter 1



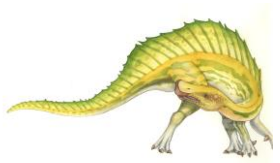
Chapter 2: Operating-System Structures





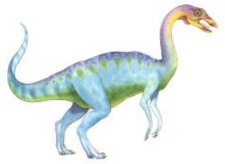
Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Operating System Debugging
- Operating System Generation
- System Boot



Objectives

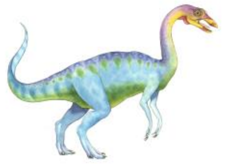
- Three views of an OS… each, respectively, focuses on
  - The services it provides [Ser]
  - The interface it makes available to users and programmers [Int]
  - Its components and interconnections [Com]
- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot



[Ser] Operating System Services (helpful to the user)

- 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**:
  - **User interface** - Almost all operating systems have a user interface (**UI**).
    - Varies between
      - **Command-Line Interface** (**CLI**): text commands and method for entering them
      - **Graphics User Interface** (**GUI**): window system
      - **Batch Interface** (**BI**): commands and directives are in makefiles
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device



[Ser] Operating System Services (helpful to the user)

- **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.

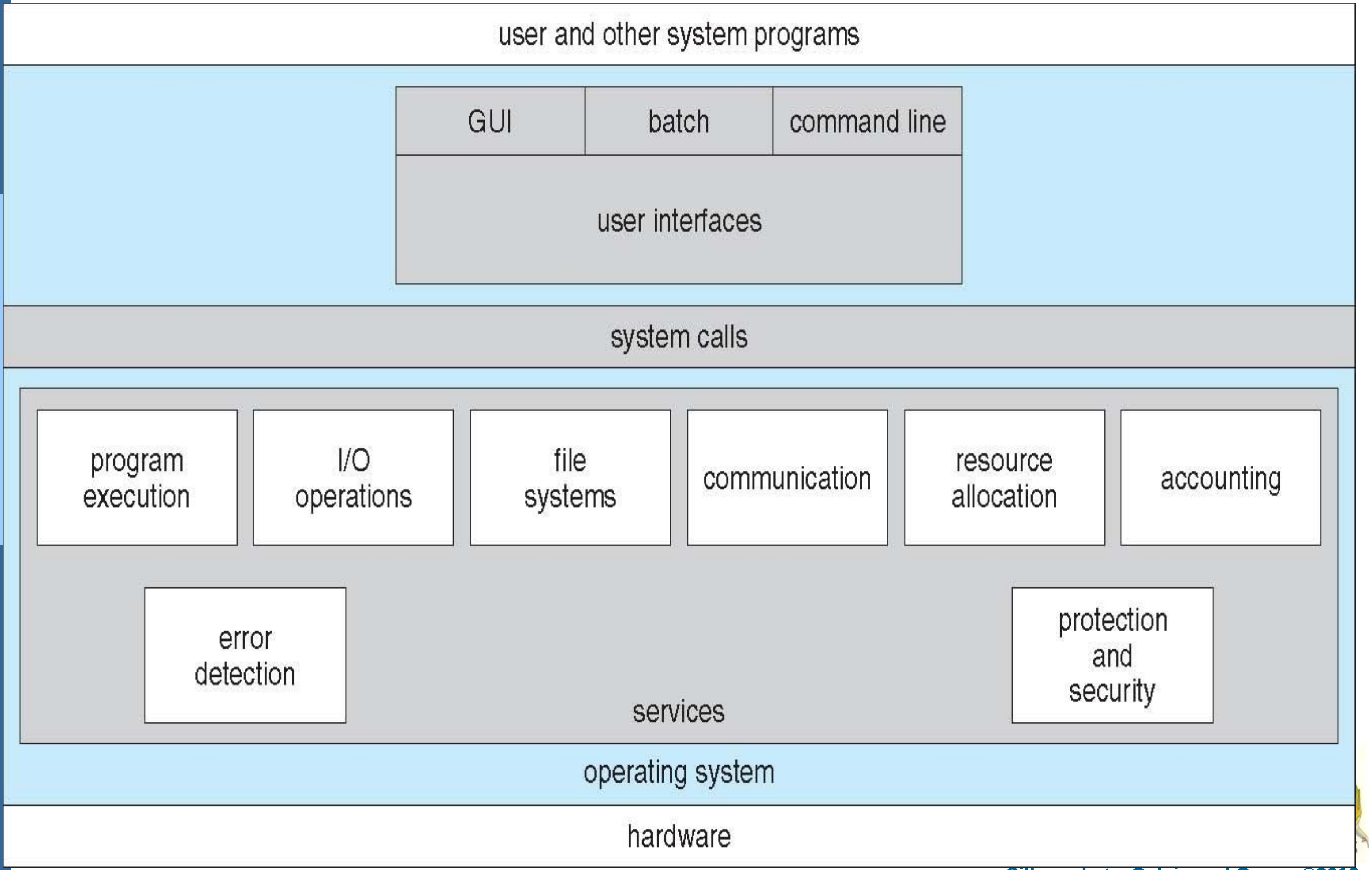
- **Communications** – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)

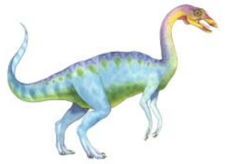
- **Error detection** – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
  - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

[Ser] Operating System Services (for efficient operation of the OS)

- Another set of OS functions exists for **ensuring the efficient operation** of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources -   CPU cycles, main memory, file storage, I/O devices.
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources
  - **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

[Ser] A View of Operating System Services



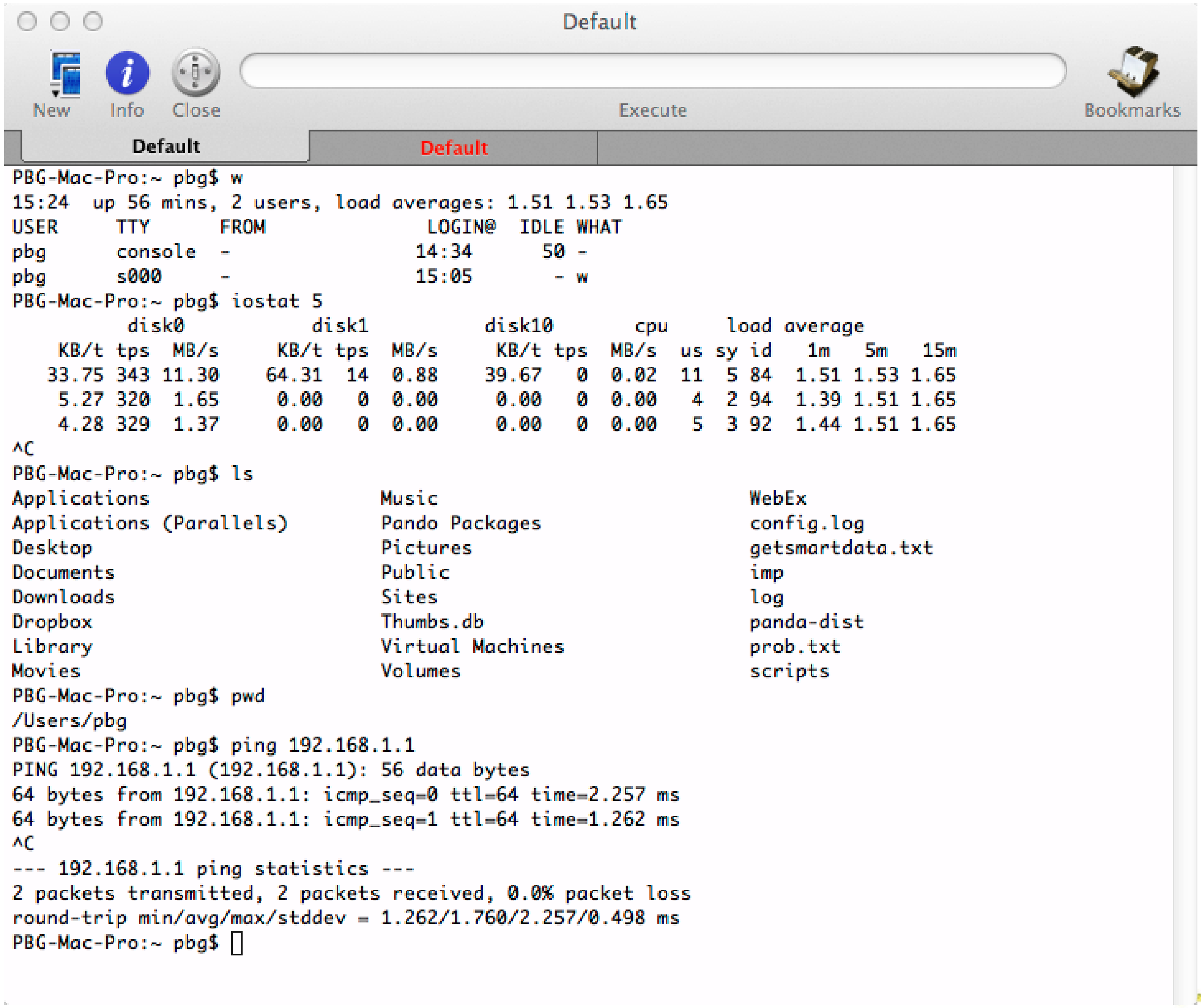


[Ser] Command-Line Interface - CLI

CLI or **command interpreter** allows direct command entry

- Sometimes implemented in kernel, sometimes by systems program (e.g., Windows XP, UNIX)
- Multiple CLIs in one system = **shells**
  - Bourne, C, Bash, or Korn shells,… etc in UNIX/Linux OS systems
- Function of CLI: fetches a command from user and executes it
  - Sometimes commands built into the CLI
    - Larger shells
  - Sometimes just names of system programs. Ex: *rm file.txt* in UNIX
    - Smaller shells. Adding new features doesn't require shell modification

[Ser] Bourne Shell Command Interpreter



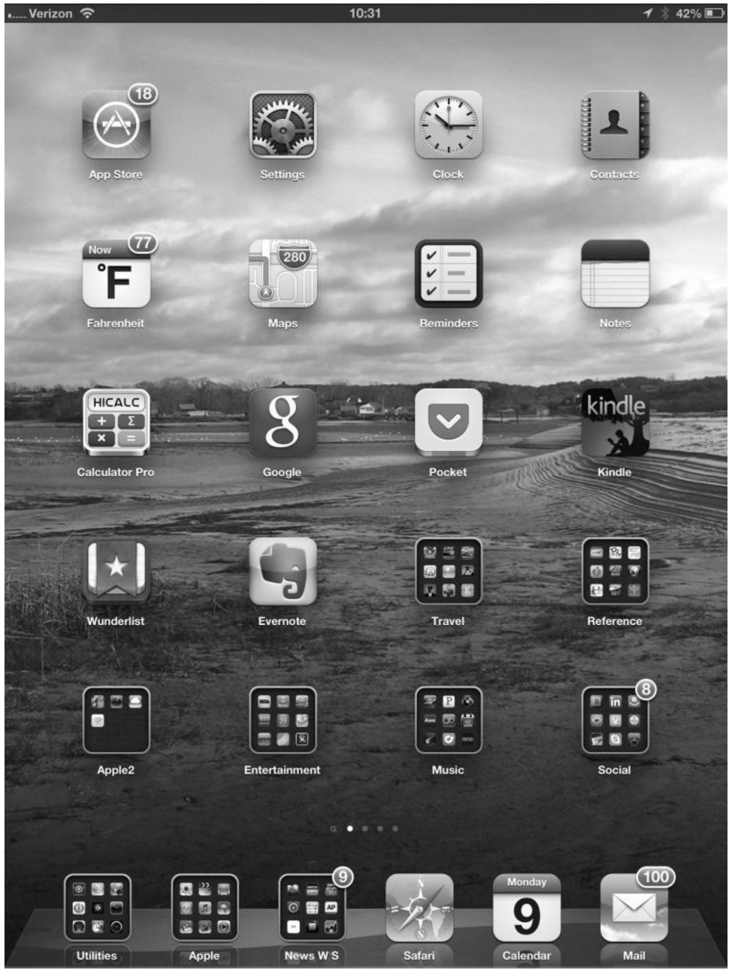
```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                14:34      50 -
pbg      s000     -                15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10        cpu     load average
    KB/t tps  MB/s     KB/t tps  MB/s     KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30    64.31  14  0.88    39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65     0.00   0  0.00     0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37     0.00   0  0.00     0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                Music                       WebEx
Applications (Parallels)    Pando Packages              config.log
Desktop                     Pictures                    getsmartdata.txt
Documents                   Public                      imp
Downloads                   Sites                       log
Dropbox                     Thumbs.db                   panda-dist
Library                     Virtual Machines            prob.txt
Movies                      Volumes                     scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$ 
```

[Ser] Graphical User Interface - GUI

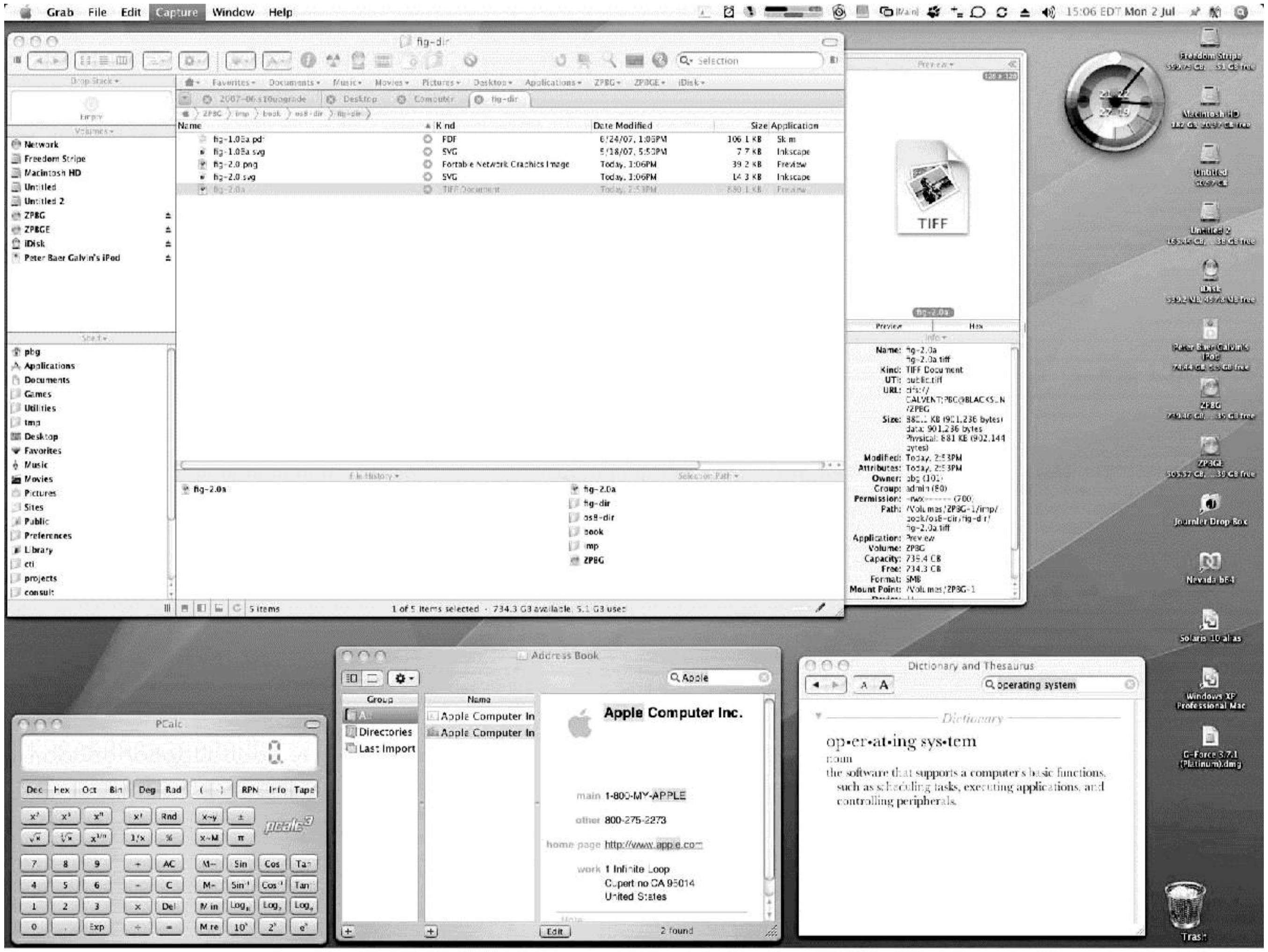
- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)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

[Ser] Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands.



[Ser] The Mac OS X GUI



Name	Kind	Date Modified	Size	Application
fig-1.08a.pdf	PDF	6/24/07, 1:06PM	106.1 KB	Skim
fig-1.08a.svg	SVG	5/18/07, 5:53PM	7.7 KB	Inkscape
fig-2.0.png	Portable Network Graphics Image	Today, 1:06PM	39.2 KB	Preview
fig-2.0.svg	SVG	Today, 1:06PM	14.3 KB	Inkscape
fig-2.0a	TIFF Document	Today, 2:53PM	880.1 KB	Preview

5 items 1 of 5 items selected, 734.3 GB available, 5.1 GB used

Name: fig-2.0a
Kind: TIFF Document
Size: 880.1 KB (901,236 bytes)
Modified: Today, 2:53PM
Owner: pbg (101)
Group: admin (80)
Application: Preview
Volume: ZPBG
Capacity: 739.4 GB
Free: 734.3 GB
Format: SMB

Apple Computer Inc.
main 1-800-MY-APPLE
other 800-275-2273
home page http://www.apple.com
work 1 Infinite Loop
Cupertino CA 95014
United States
2 found

Dictionary
op·er·at·ing sys·tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

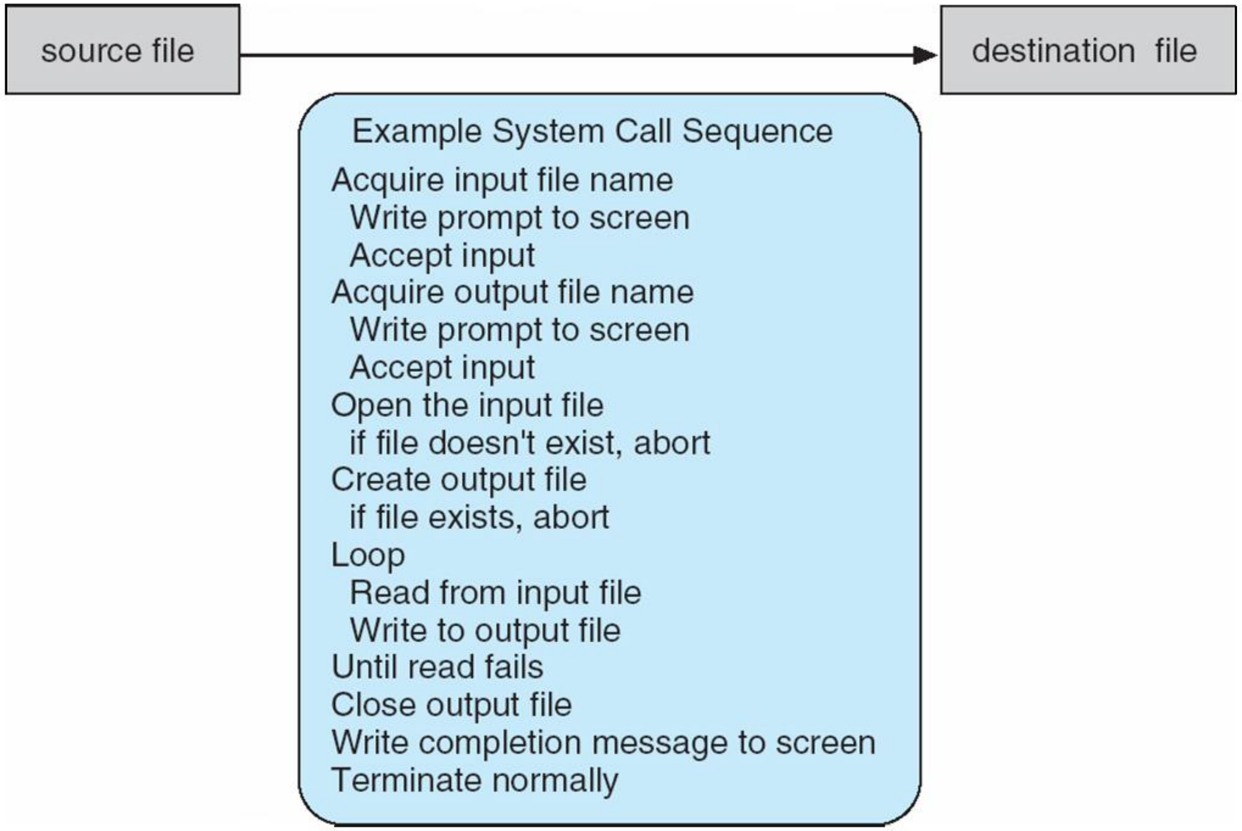
[Int] System Calls

- Provide programming interface to the services made available by the OS
- Typically written in a high-level language (C or C++)
  - With hardware-level tasks written in assembly language
- Mostly accessed by programs via a high-level **Application Programming Interface** (**API**) rather than direct system call use
  - API specifies set of functions available to the programmer
    - Example :functions ***ReadFile()*** or ***CreateProcess()*** in WIN32 API
    - Functions invoke the actual system calls on behalf of the programmer
      - Function ***CreateProcess()*** invokes system call ***NTCreateProcess()***
      - Why not invoke actual system call (instead of using API) ? Because:
        - » Program **portability**: compile/run on systems supporting same API
- Three most common APIs are
  - Win32 API for Windows
  - POSIX API for UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)

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

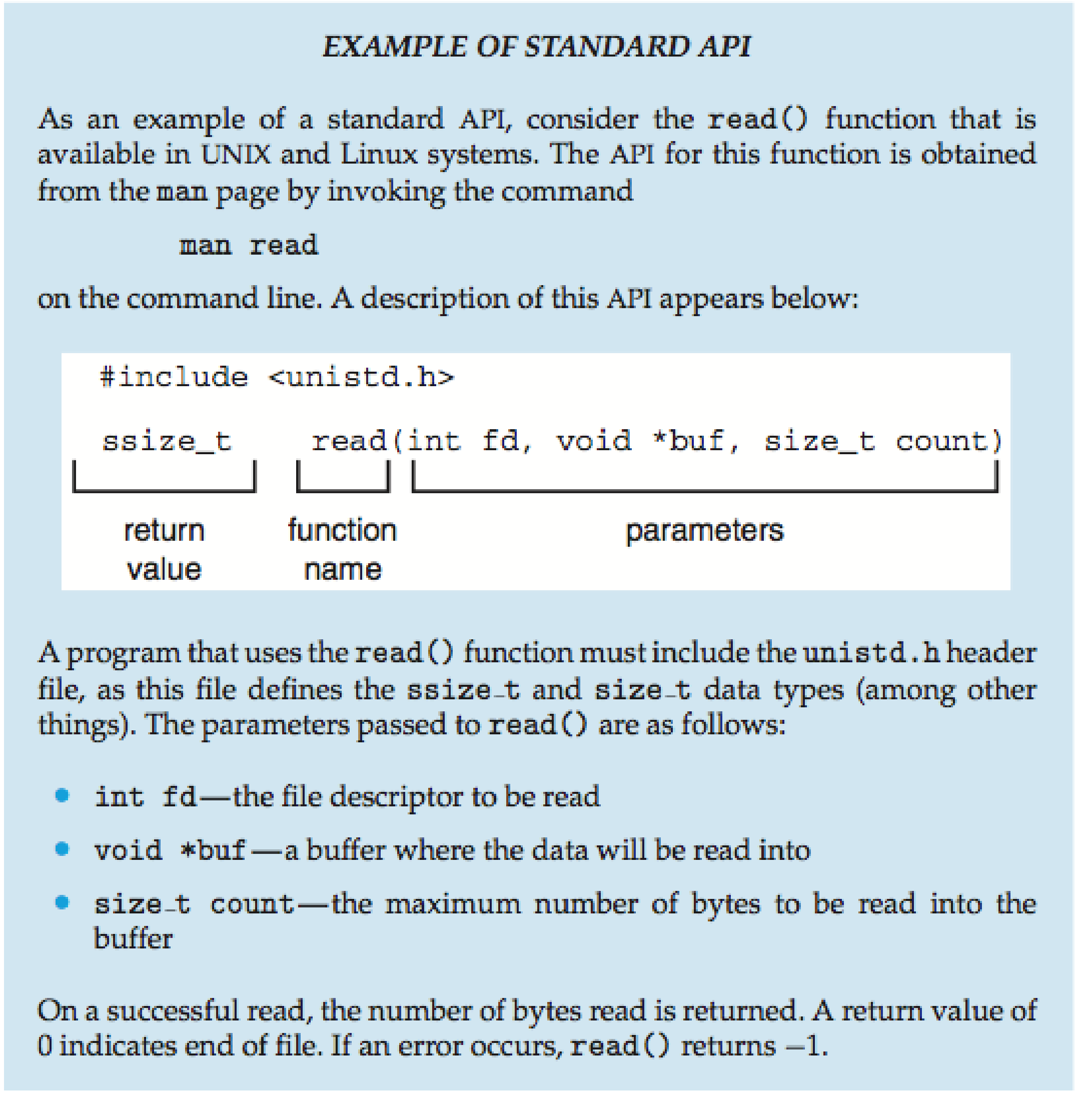
[Int] Example of System Calls

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



```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

[Int] Example of Standard API

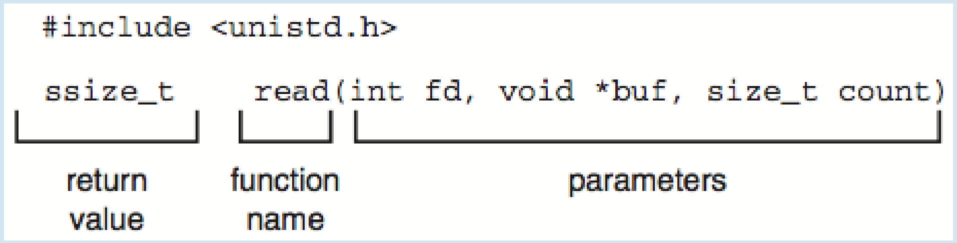


EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:



```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

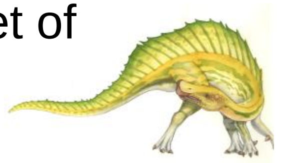
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

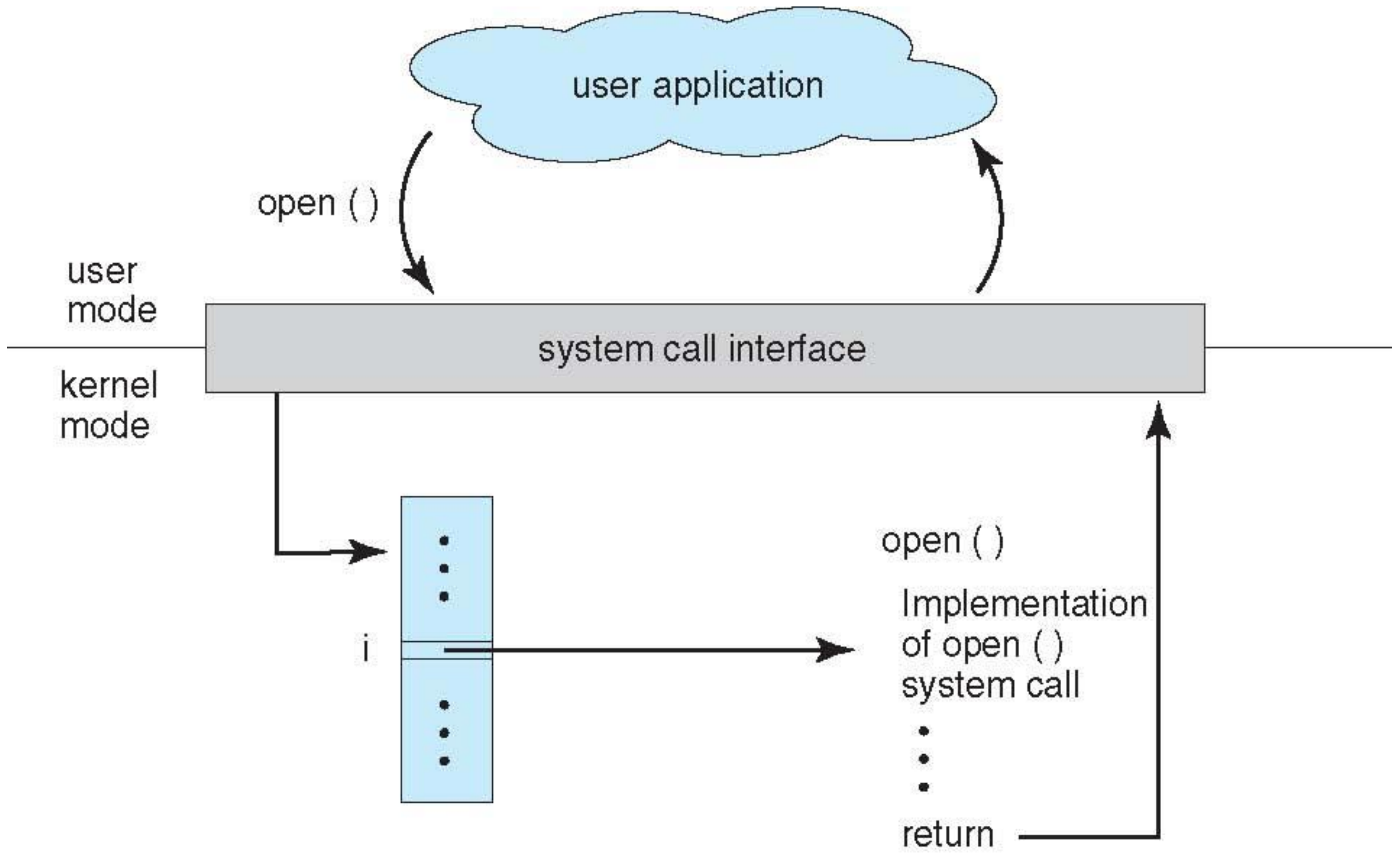
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

[Int] System Call Implementation

- Typically, a number is associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the 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
    - System-call interface is managed by run-time support library (set of functions built into libraries included with compiler)



[Int] API – System Call – OS Relationship

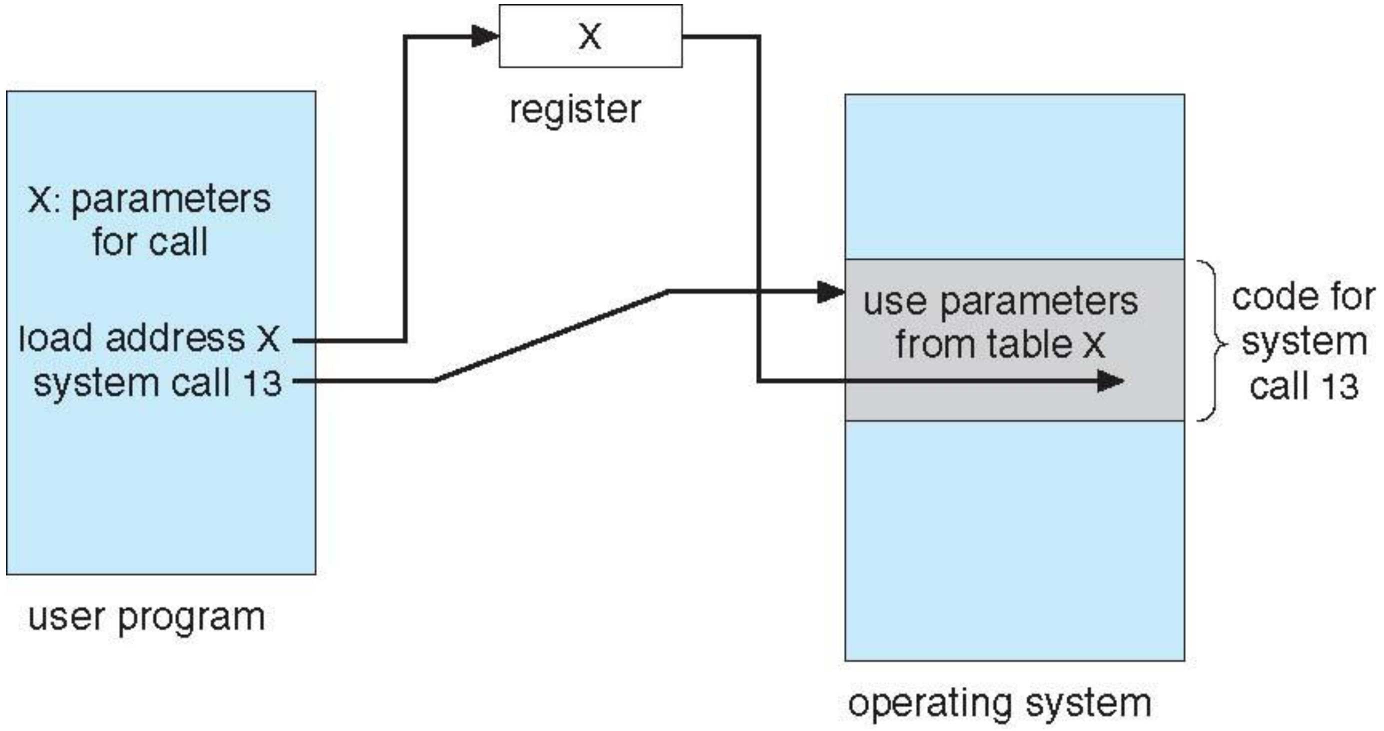


[Int] System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and system call
- Three general methods used to pass parameters to the OS

  (discussed in 03-60-266)
  - 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 runtime **stack** by the program and **popped** off the stack by the operating system
  - Block and stack methods are preferred… do not limit the number or length of parameters being passed

[Int] Parameter Passing via Table

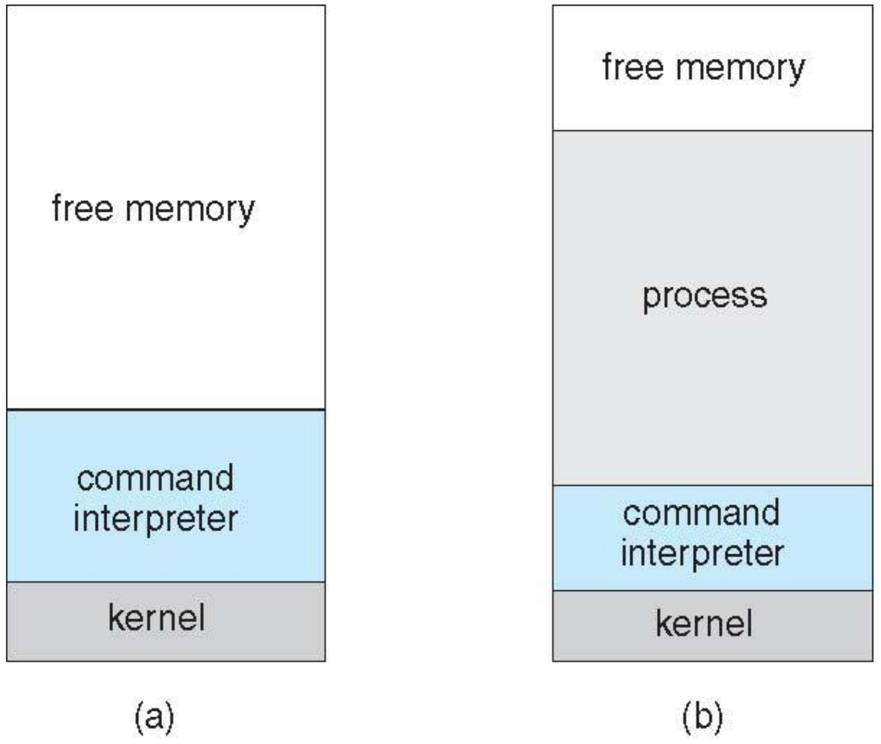


[Int] Types of System Calls (Process Control)

- Process control
    - Program should be able to start, control, and end or abort its execution
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

[Int] Process Control Example: MS-DOS

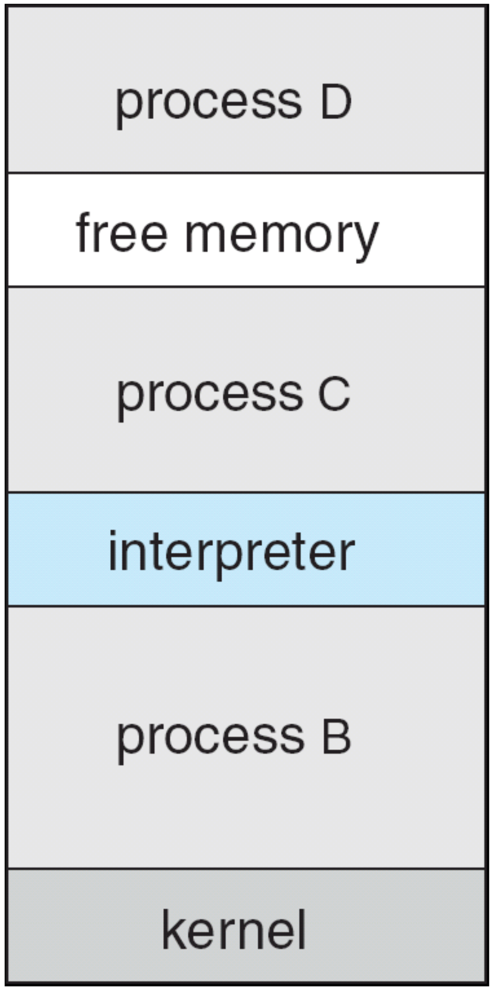
- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded

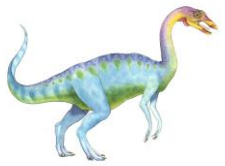


At system startup   running a program

[Int] Process Control Example: FreeBSD

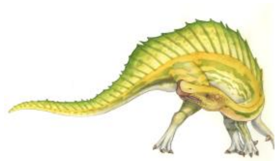
- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process
  - Executes exec() to load program into process
  - Shell waits for process to terminate or continues with user commands
- Process exits with:
  - code = 0 – no error
  - code > 0 – error code

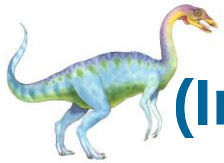




[Int] Types of System Calls (File Management and Device Management)

- File (and directory) management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - Process needs several resources to execute
  - Resources = Devices = memory, disk drives, files, … etc, controlled by OS
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

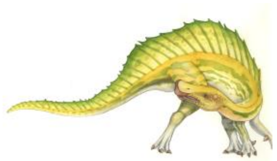




[Int] Types of System Calls (Information Maintenance and Communications)

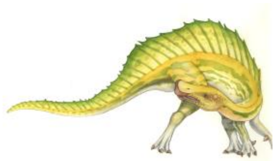
- Information maintenance
    - All kinds of statistics and data that can be requested
        - Nb of users, size of free memory, OS version number, disk space, etc
    - get time or date, set time or date
    - get system data, set system data
    - get and set process, file, or device attributes

- Communications
    - create, delete communication connection
    - send, receive messages if **message passing model** to **host name** or **process name**
        - From **client** to **server**
    - **Shared-memory model** create and gain access to memory regions
    - transfer status information
    - attach and detach remote devices

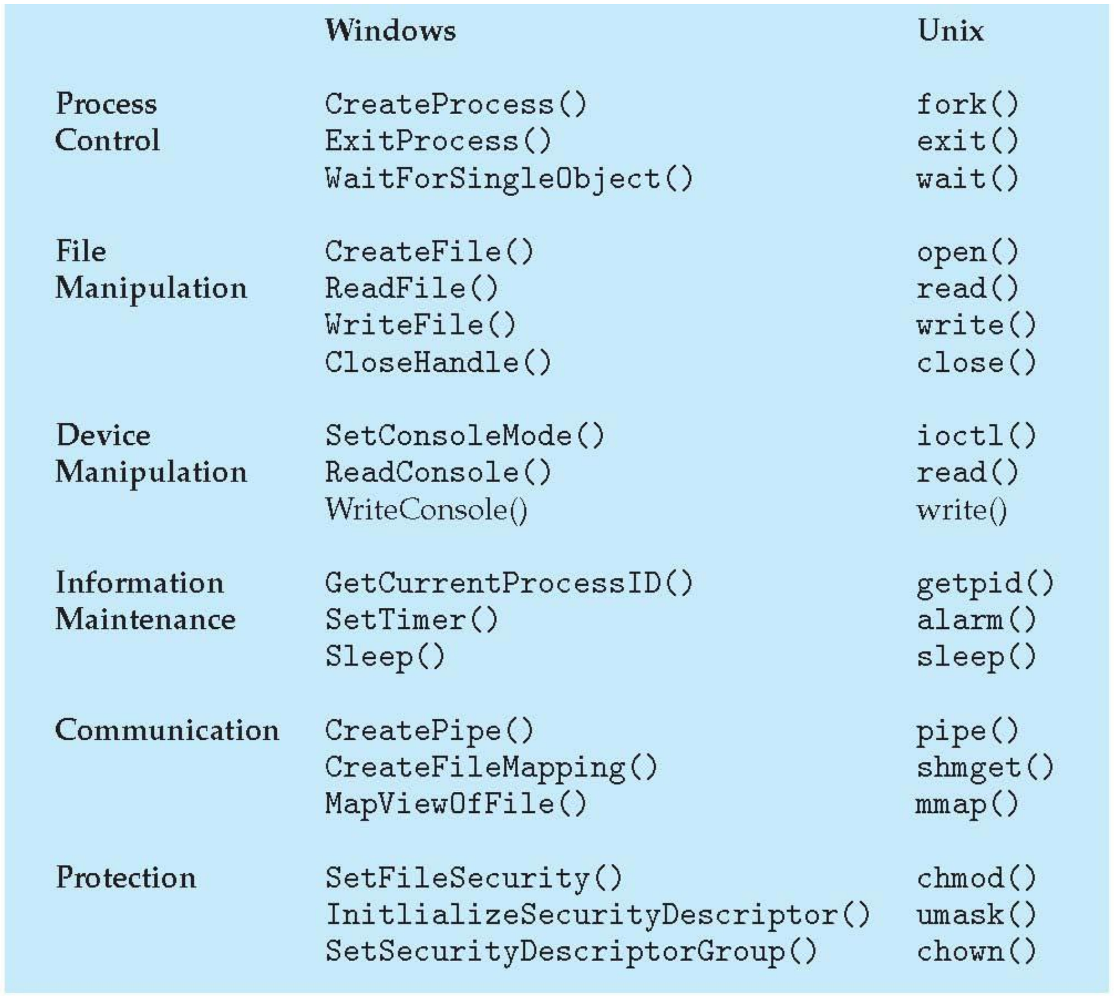


[Int] Types of System Calls (Protection)

- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access



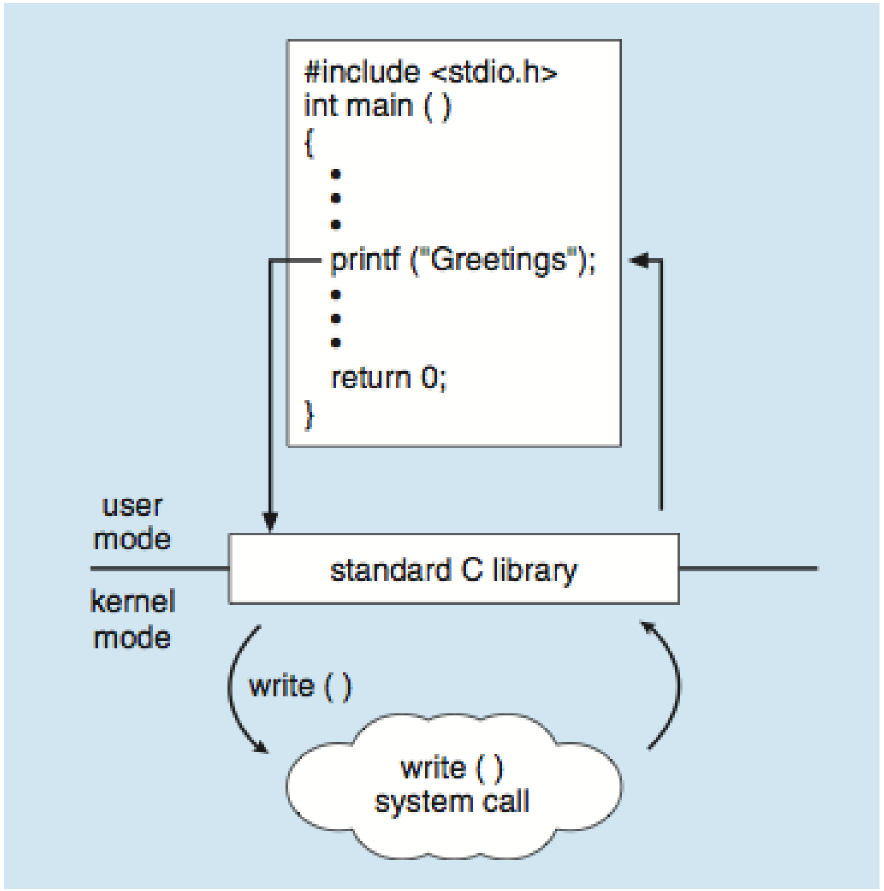
[Int] Examples of Windows and Unix System Calls



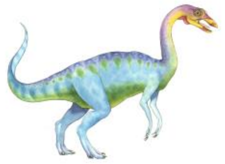
	Windows	Unix
Process Control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File Manipulation	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device Manipulation	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`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

[Int] Standard C Library Example

- C program invoking printf() library call, which calls write() system call
  - The standard C library provides portion of system-call interface for many versions of UNIX and Linux



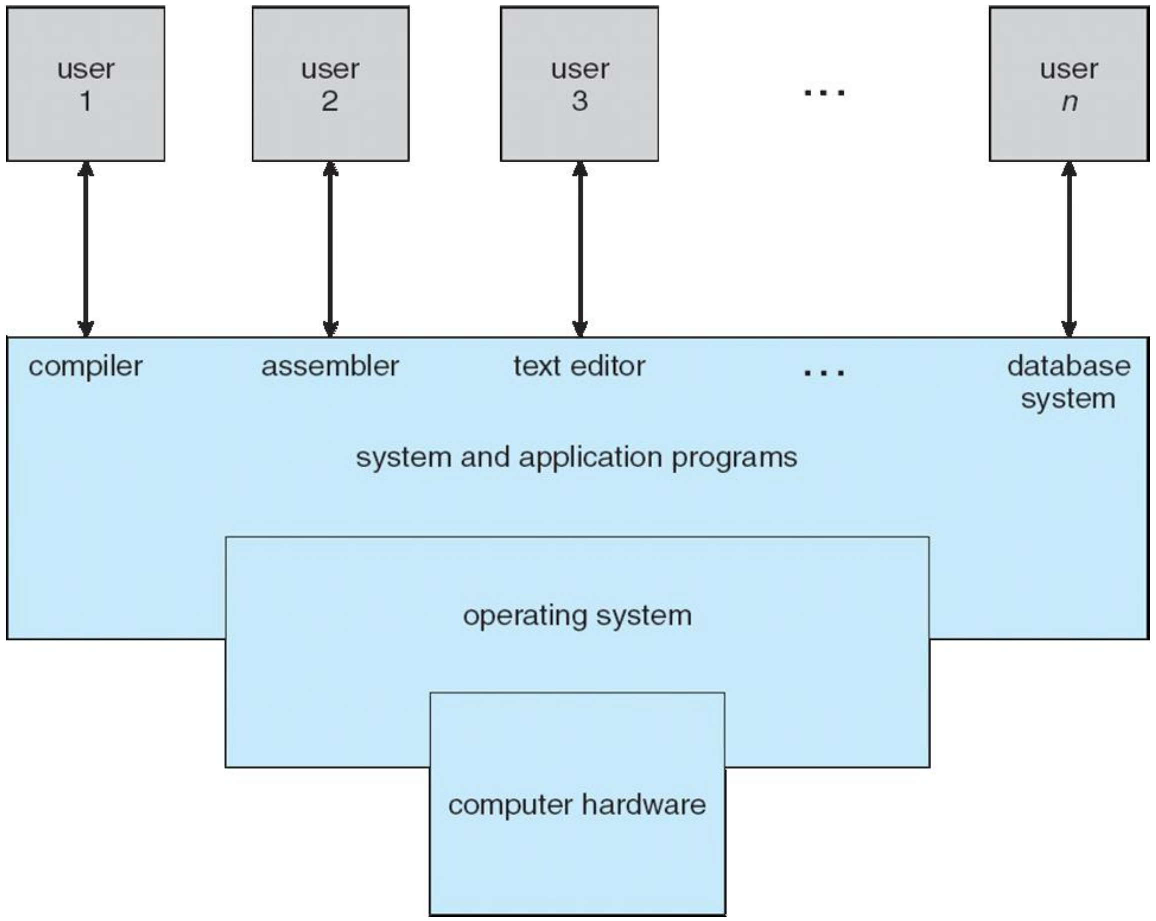
```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```



[Int] System Programs

- System programs (or, system utilities) provide a convenient environment for program development and execution.
  - Some are user interfaces to system calls, others are very complex
    - E.g., browsers, formatters, assemblers, debuggers, defragmenters… etc
- They can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

[Int] System Programs



[Int] System Programs

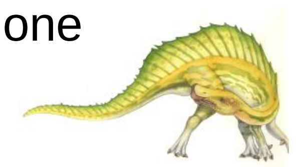
- Provide a convenient environment for program development and execution
    - Some of them are simply user interfaces to system calls; others are considerably more complex

- **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

[Int] System Programs

- **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



[Int] System Programs

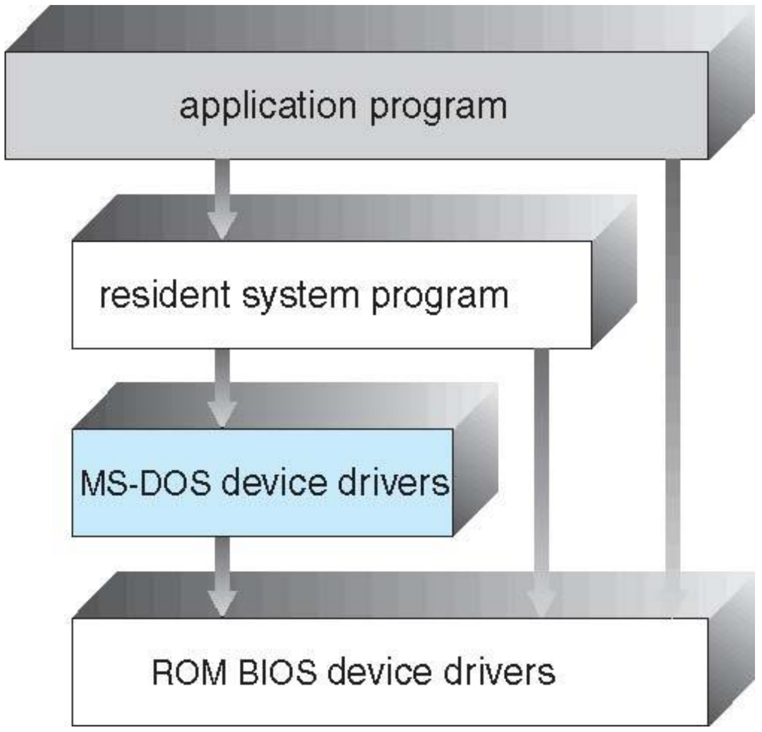
- **Background Services**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services**, **subsystems**, **daemons**
- **Application programs**
  - Don’t pertain to system
  - Run by users
  - Not typically considered part of OS; but user think they are
  - Launched by command line, mouse click, finger poke

[Com] Operating System Structure

- General-purpose OS is very large program
    - Hence, OS must be engineered intelligently for easy use and modification
  - OS design: partition into modules and define interconnections
- Various ways to structure ones
  - Simple structure – MS-DOS: Monolithic, small kernel, not well separated modules, no protection, limited by Intel 8088 hardware
  - More complex -- original UNIX: Monolithic, large kernel, two-layered UNIX (separates kernel and system programs), initially limited by hardware
  - Layered – an abstraction: Modular OS, freedom to change/add modules
  - Microkernel – Mach: Modularized the expanded but large UNIX, keeps only essential component as system-level or user-level programs, smaller kernel, and easy to extend

[Com] Simple Structure -- MS-DOS

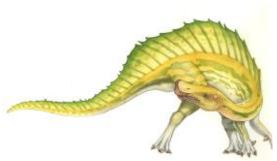
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated
    - E.g., App's can write directly to the display and disk drives
    - Intel 8088 processor had no **dual mode**
      - Vulnerable
      - No protection



[Com] Non Simple Structure -- UNIX

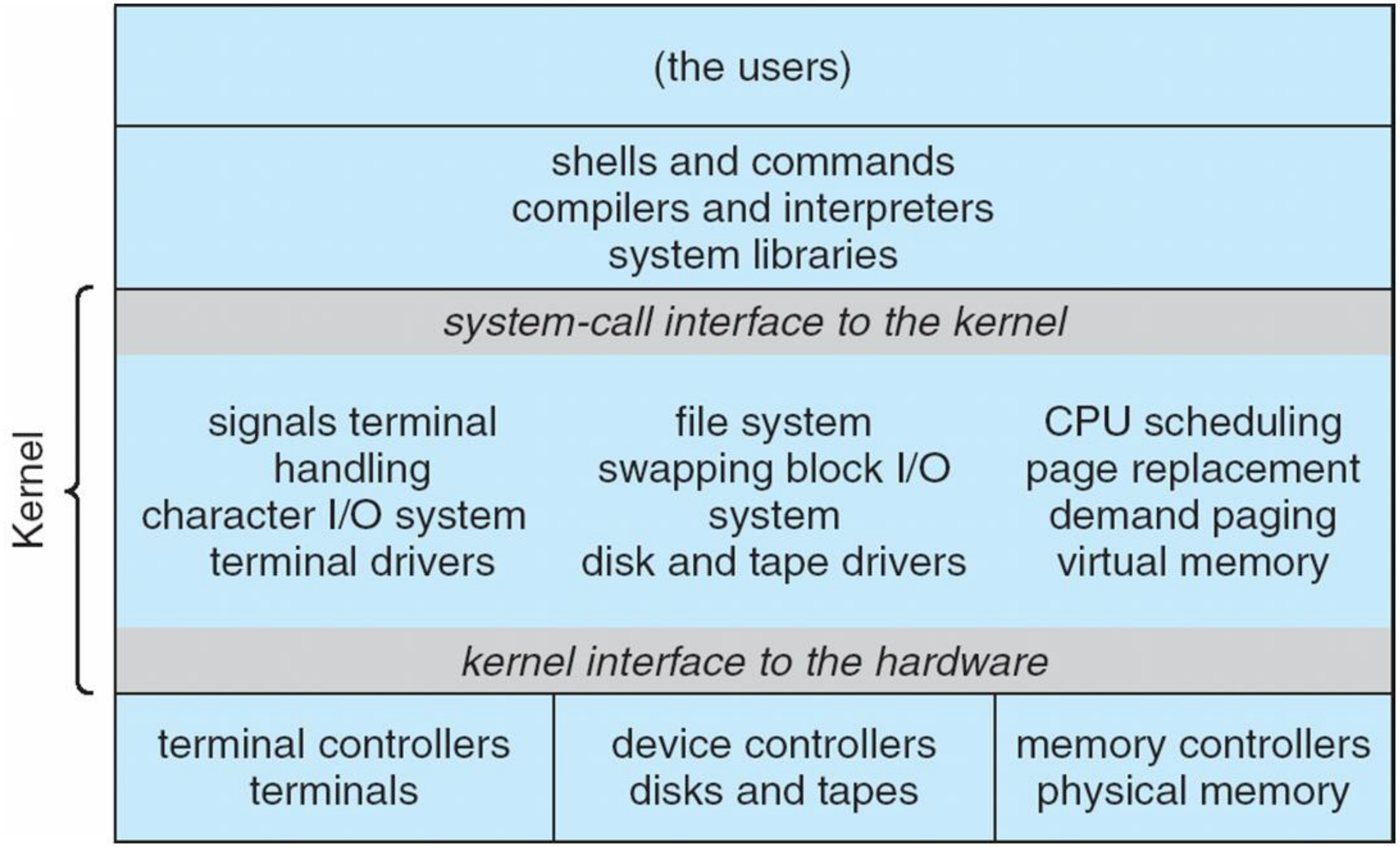
UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

- 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
    - Very large kernel



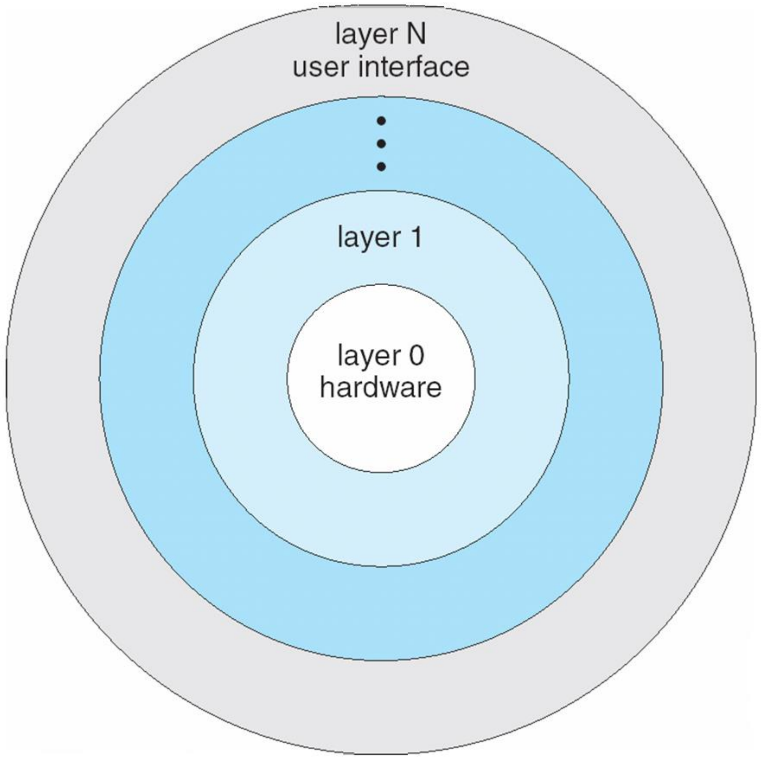
[Com] Traditional UNIX System Structure

Beyond simple but not fully layered



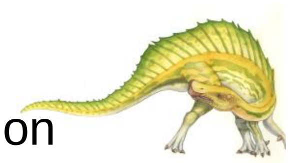
[Com] Layered Approach

- 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, layers are selected such that each uses functions (operations) and services of only lower-level layers
- Advantage: each layer is
  - Abstraction: *data + operations on data*
  - Simple to construct
  - Easy to debug and verify
- Problems:
  - defining the various layers,
  - less efficient than non-layered OS

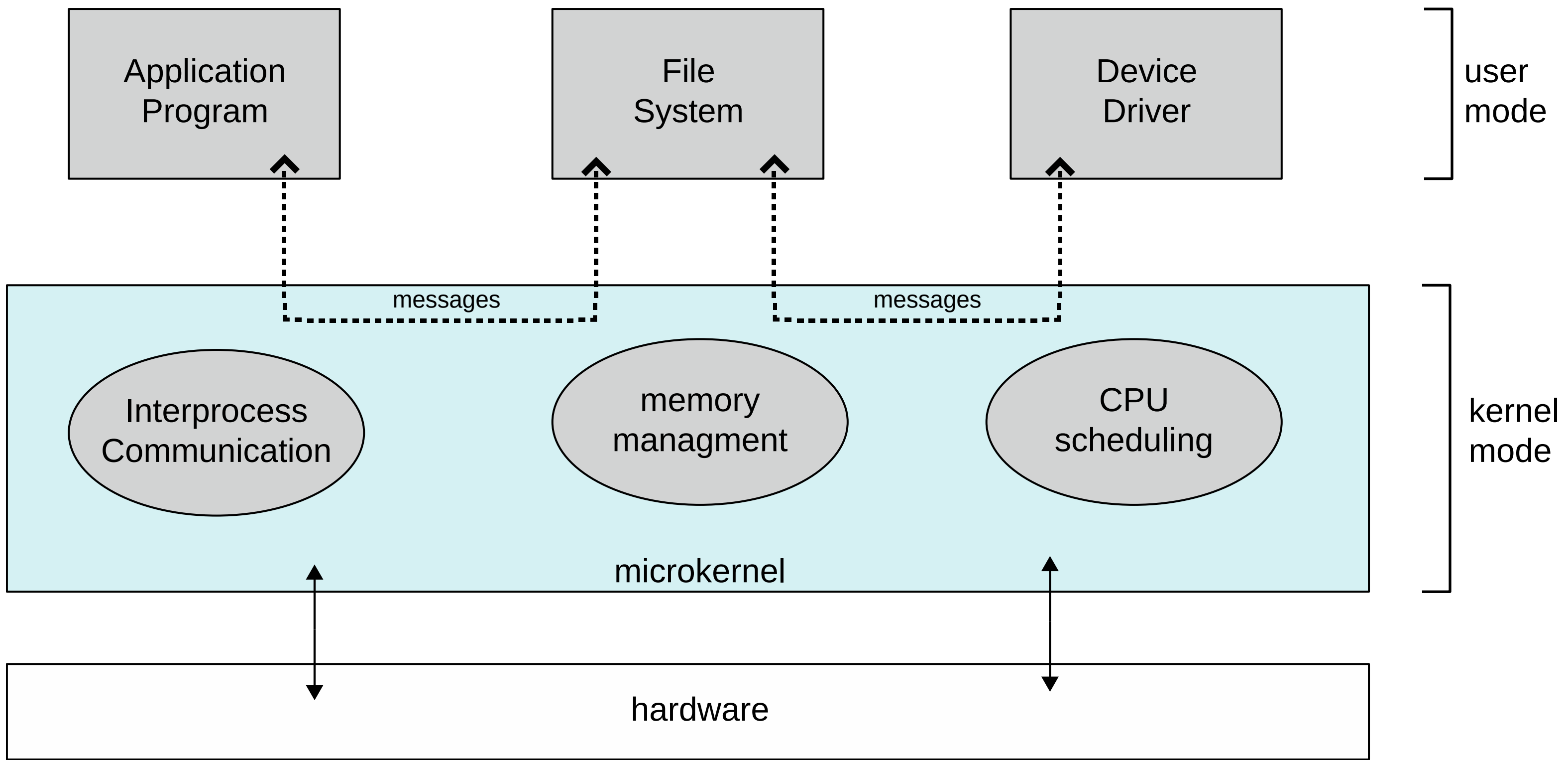


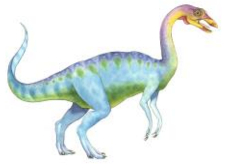
[Com] Microkernel System Structure

- Moves as much from the kernel into user space, hence, a small kernel
  - Kernel provides : process and memory management, and inter-process comm
  - All non-essential components are either user or system programs
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
  - Function of microkernel: communication between client program and services
- 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:
  - Performance overhead of user space to kernel space communication



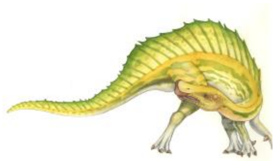
[Com] Microkernel System Structure



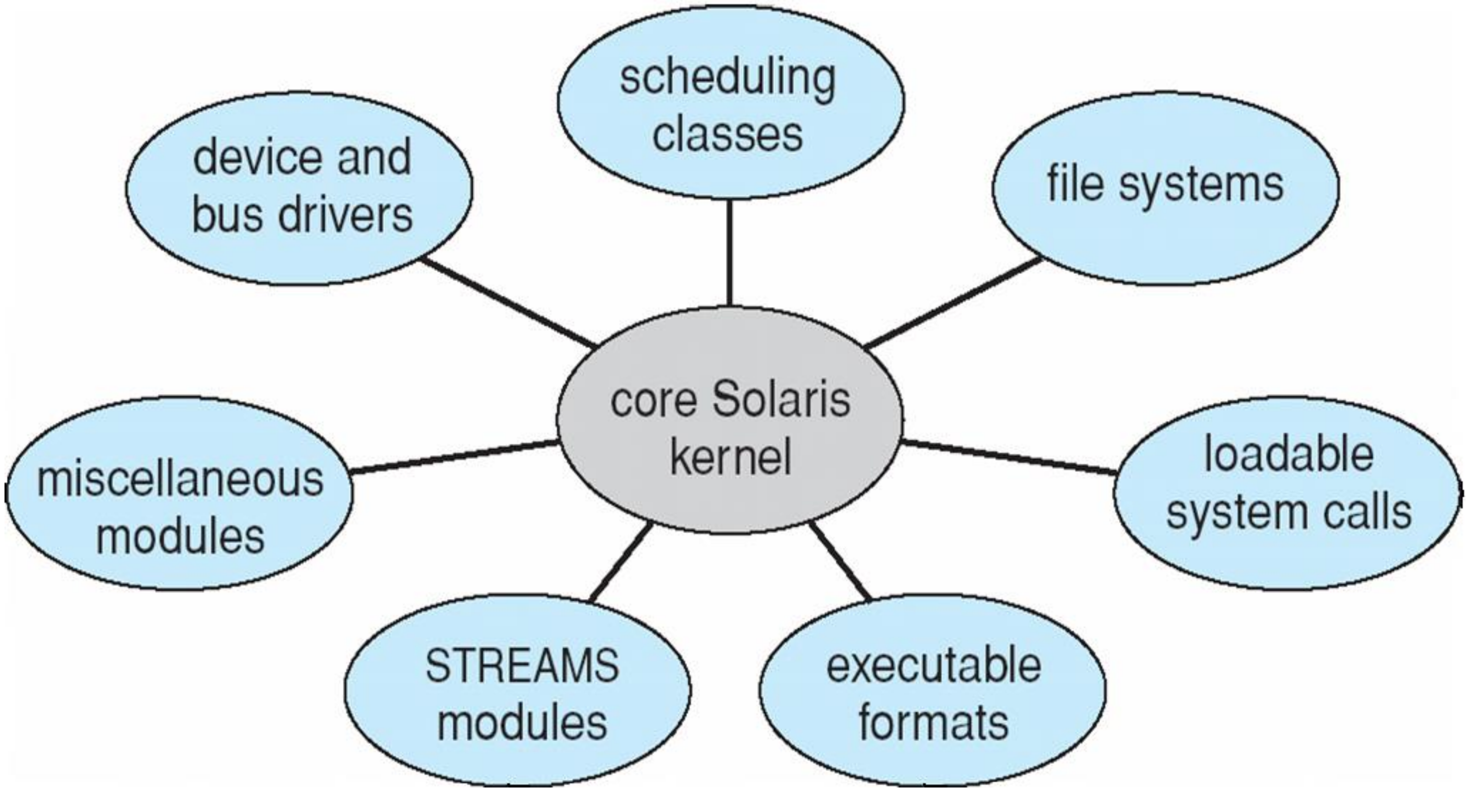


[Com] Modules

- Many modern operating systems implement **loadable kernel modules**
    - Kernel provides core services
    - Similar to microkernel
  - 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 flexibility
  - Each kernel module is abstract but can call any other module
  - Linux, Solaris, … etc

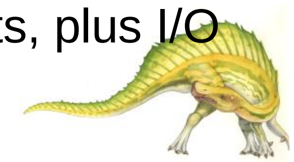


[Com] Solaris Modular Approach

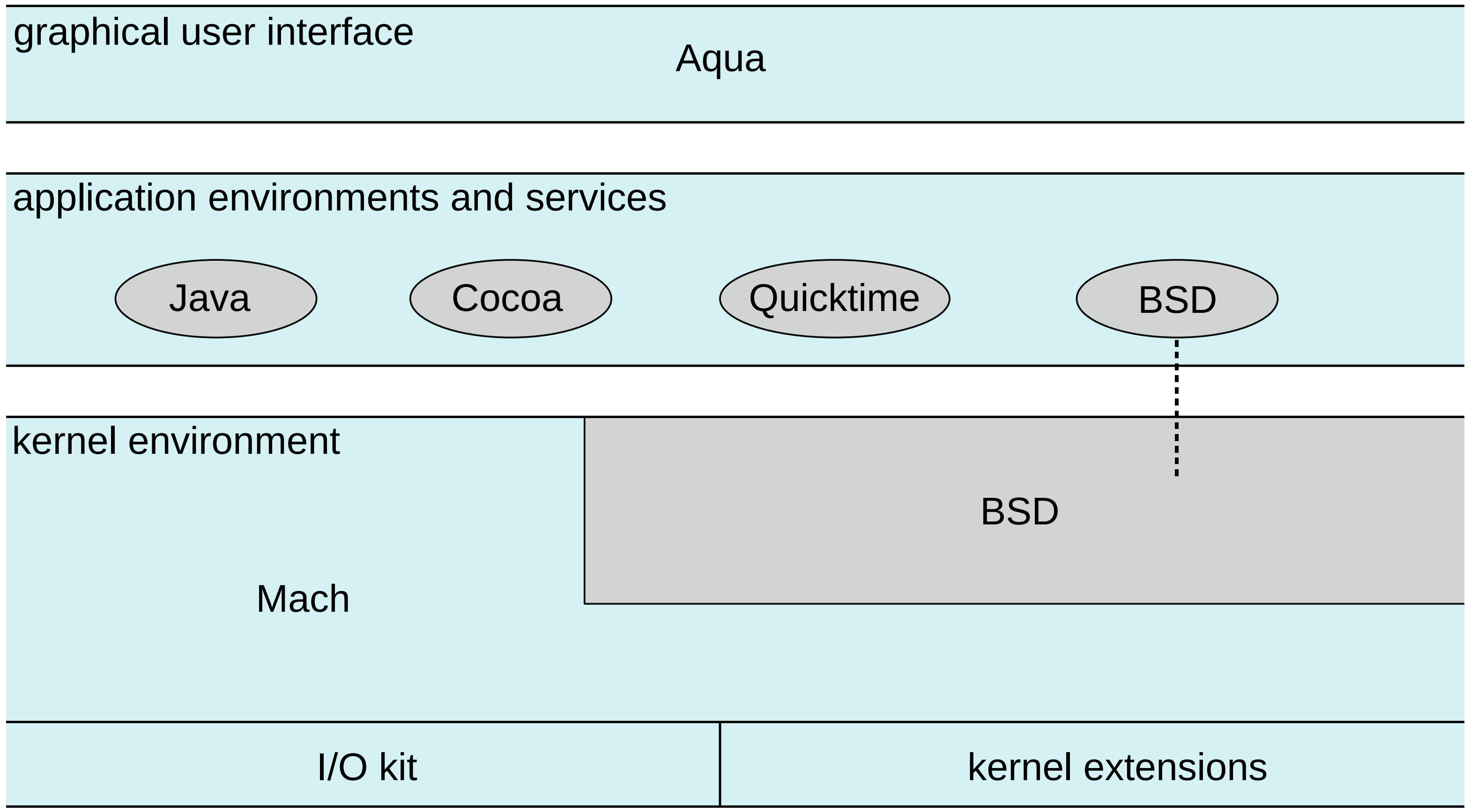


[Com] Hybrid Systems

- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels are monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Mac OS X is hybrid, layered with **Aqua** UI plus **Cocoa** API
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

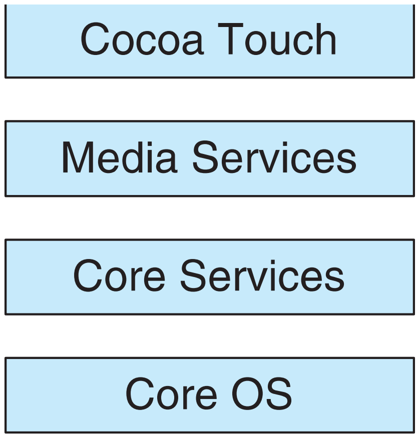


[Com] Mac OS X Structure



iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



Android

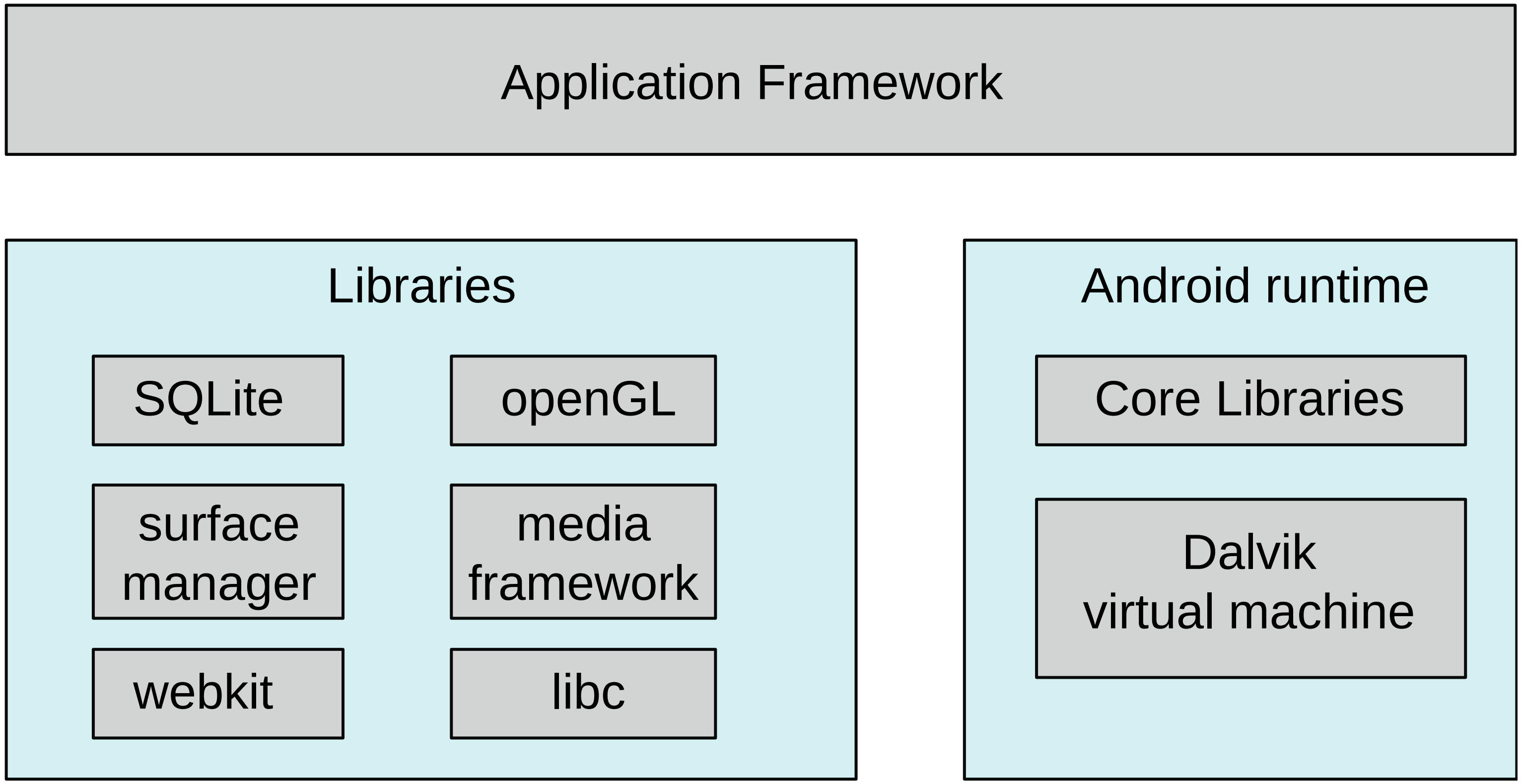
- Developed by Open Handset Alliance (mostly Google)
  - Open Source

- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management

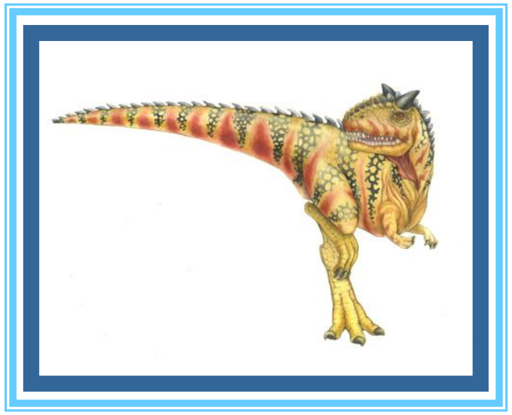
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM

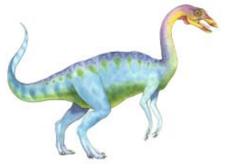
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



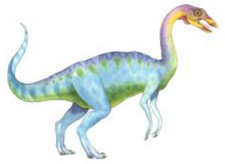
End of Chapter 2





Operating System Design Goals

- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start the design 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



Operating System Mechanisms and Policies

- Important principle to separate

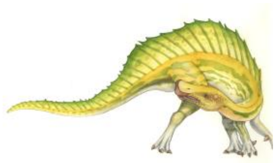
  Policy: ***What*** will be done?

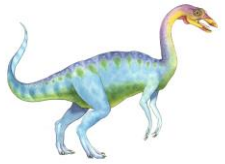
  Mechanism: ***How*** to do it?

- 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 (example – timer)

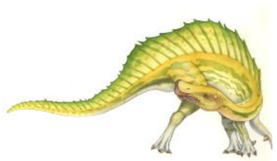
- Specifying and designing an OS is highly creative task of **software engineering**

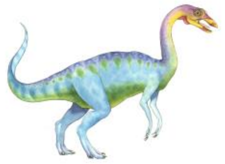




Operating System Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower
- **Emulation** can allow an OS to run on non-native hardware





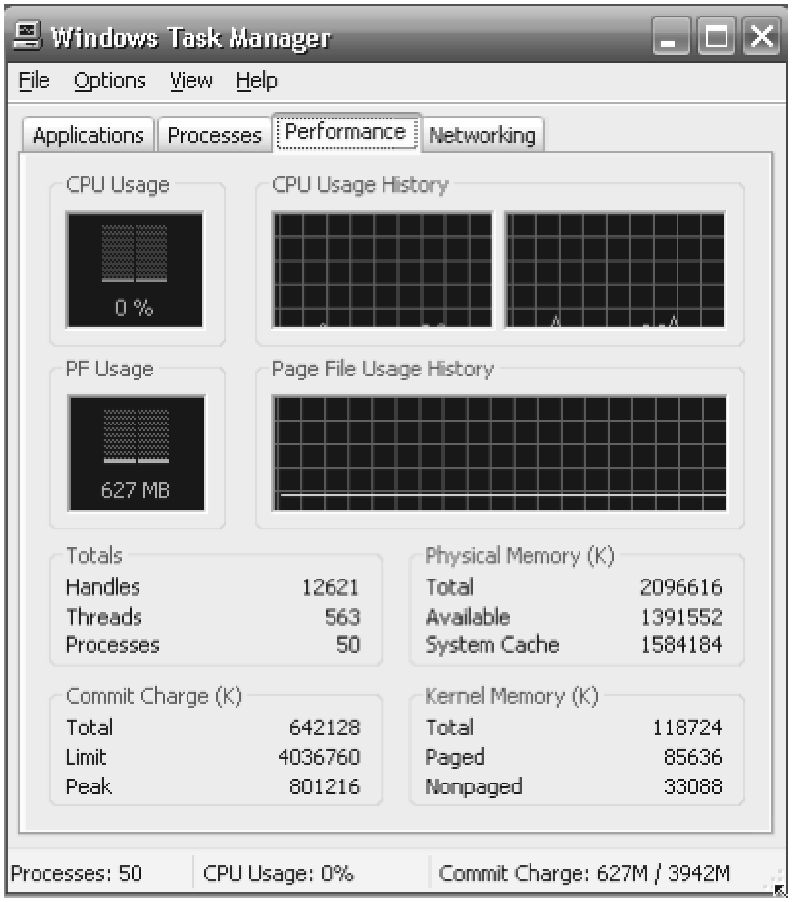
Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OS 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
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

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."

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, “top” program or Windows Task Manager



Totals	
Handles	12621
Threads	563
Processes	50

Physical Memory (K)	
Total	2096616
Available	1391552
System Cache	1584184

Commit Charge (K)	
Total	642128
Limit	4036760
Peak	801216

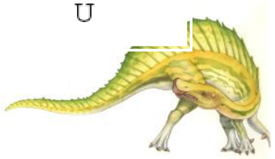
Kernel Memory (K)	
Total	118724
Paged	85636
Nonpaged	33088

Processes: 50 CPU Usage: 0% Commit Charge: 627M / 3942M

DTrace

- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
- **Probes** fire when code is executed within a **provider**, capturing state data and sending it to **consumers** of those probes
- Example of following XEventsQueued system call move from libc library to kernel and back

```
# ./all.d ‘pgrep xclock‘ XEventsQueued
dtrace: script ’./all.d’ matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                   U
  0   -> _XEventsQueued                U
  0     -> _X11TransBytesReadable      U
  0     <- _X11TransBytesReadable      U
  0     -> _X11TransSocketBytesReadable U
  0     <- _X11TransSocketBytesreadable U
  0     -> ioctl                       U
  0       -> ioctl                     K
  0         -> getf                    K
  0           -> set_active_fd         K
  0           <- set_active_fd         K
  0         <- getf                    K
  0         -> get_udatamodel          K
  0         <- get_udatamodel          K
...
  0         -> releasef                K
  0           -> clear_active_fd       K
  0           <- clear_active_fd       K
  0           -> cv_broadcast          K
  0           <- cv_broadcast          K
  0         <- releasef                K
  0       <- ioctl                     K
  0     <- ioctl                       U
  0   <- _XEventsQueued                U
  0 <- XEventsQueued                   U
```



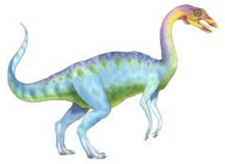
Dtrace (Cont.)

- DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds

```
sched:::on-cpu
uid == 101
{
   self->ts = timestamp;
}

sched:::off-cpu
self->ts
{
   @time[execname] = sum(timestamp - self->ts);
   self->ts = 0;
}
```

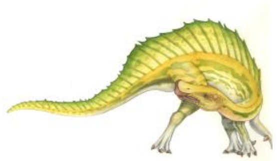
```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
    gnome-settings-d                    142354
    gnome-vfs-daemon                    158243
    dsdm                                189804
    wnck-applet                         200030
    gnome-panel                         277864
    clock-applet                        374916
    mapping-daemon                      385475
    xscreensaver                        514177
    metacity                            539281
    Xorg                               2579646
    gnome-terminal                     5007269
    mixer_applet2                      7388447
    java                              10769137
```

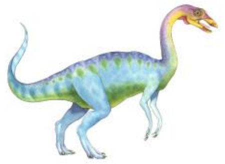
Figure 2.21 Output of the D code.



Operating System Generation

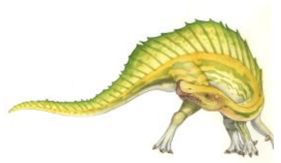
- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- **SYSGEN** program obtains information concerning the specific configuration of the hardware system
  - Used to build system-specific compiled kernel or system-tuned
  - Can general more efficient code than one general kernel



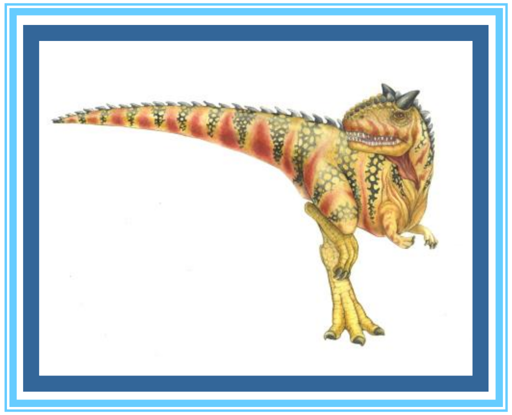


System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**



Chapter 3: Processes
العمليات



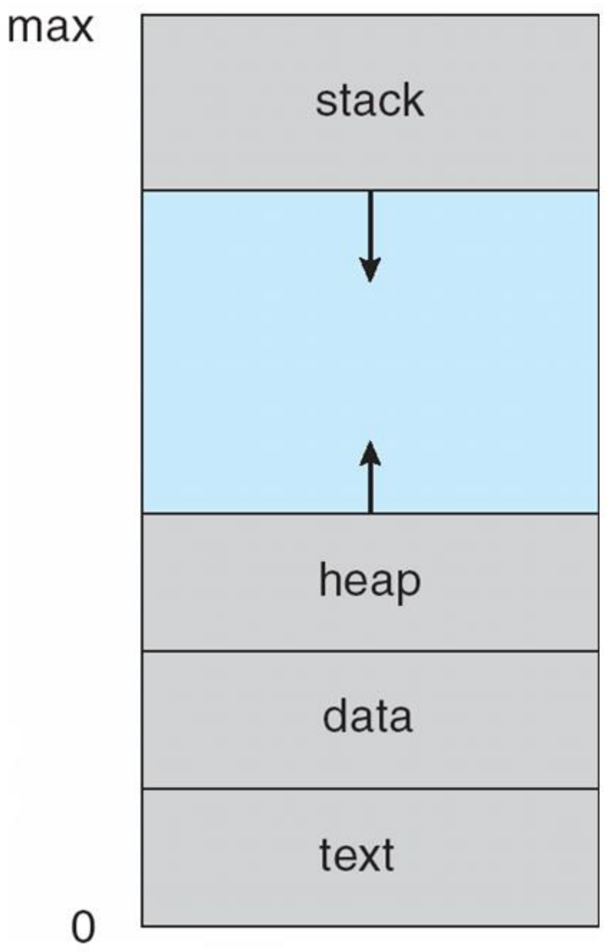
Process Concept مفهوم العملية

- An operating system executes a variety of programs:
- ينفذ نظام التشغيل برامج متنوعة
  - Batch system – **jobs** اما برامج نظام دفعية – تسمى الوظائف
  - Time-shared systems – **user programs** or **tasks** او برامج مشاركة الزمن – مثل برامج المتسخدم او المهام
- Textbook uses the terms ***job*** and ***process*** almost interchangeably الكتب تطلق عليها مصطلح الوظائف او العمليات
- **Process** – a program in execution; process execution must progress in sequential fashion
- العملية – هي برنامج في حالة تنفيذ و تنفيذ العملية يجب ان يكون بصورة متسلسلة
- Multiple parts تتكون العملية من عدة اجزاء و هي:
  - The program code, also called **text section** شفرات او ايعازات البرنامج وتدعى مقطع النص
    - Current activity including **program counter**, processor registers النشاطات الحالية و تشمل عداد البرنامج و مسجلات المعالج
  - **Stack** containing temporary data المكدس يحتوي على البيانات المؤقتة
    - Function parameters, return addresses, local variables مثل معلمات الوظائف ,عنوان العودة ,متغيرات محلية
  - **Data section** containing global variables مقطع البيانات يحتوي مغيرات عامة
    - **Heap** containing memory dynamically allocated during run time الكومة هو ذاكرة تغير بالموقع والحجم اثناء تنفيذ البرنامج

Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - البرمامج هو كيان خامل مخزون بالقرص, بينما العملية هي كيان فعال
  - Program becomes process when executable file loaded into memory
  - البرنامج يتحول الى عملية عندما يتم تحميل الملف التنفيذي الى الذاكرة الرئيسية
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- يبدا تنفيذ البرنامج عند النقر علية بالمواس في واجهة رسوم المستخدم او ادخال اسم الملف في نظام سطر الأوامر.
- One program can be several processes يمكن أن يكون برنامج واحد وله عدة عمليات
  - Consider multiple users executing the same program مثلا عدة مستخدمين ينفذون نفس البرنامج

Process in Memory رسم العملية في الذاكرة



المكدس

ذاكرة
الكومة

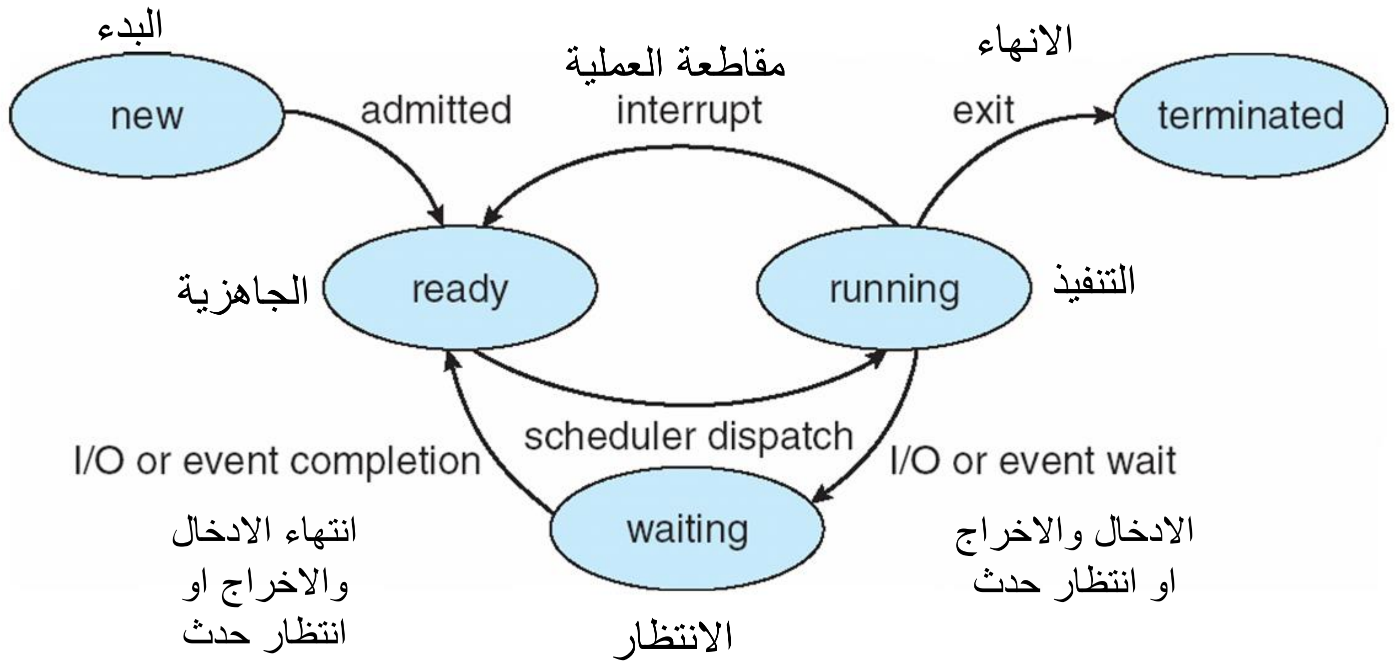
مقطع
البيانات

مقطع
النص-
التعليمات

Process State حالات العملية

- As a process executes, it changes **state** تمر العملية اثناء تنفيذها بعدة حالات وهي:
  - **new**: The process is being created الجديد: وفيها يتم انشاء العملية
  - **running**: Instructions are being executed
  - التنفيذ: وفيها يتم تنفيذ ايعازات العملية
  - **waiting**: The process is waiting for some event to occur
  - الانتظار:العملية تنتظر حدث معين
  - **ready**: The process is waiting to be assigned to a processor
  - الجاهزية: العملية جاهزة وتنتظر اسنادها الى المعالج
  - **terminated**: The process has finished execution
  - الانهاء: العملية انتهت من التنفيذ

Diagram of Process State

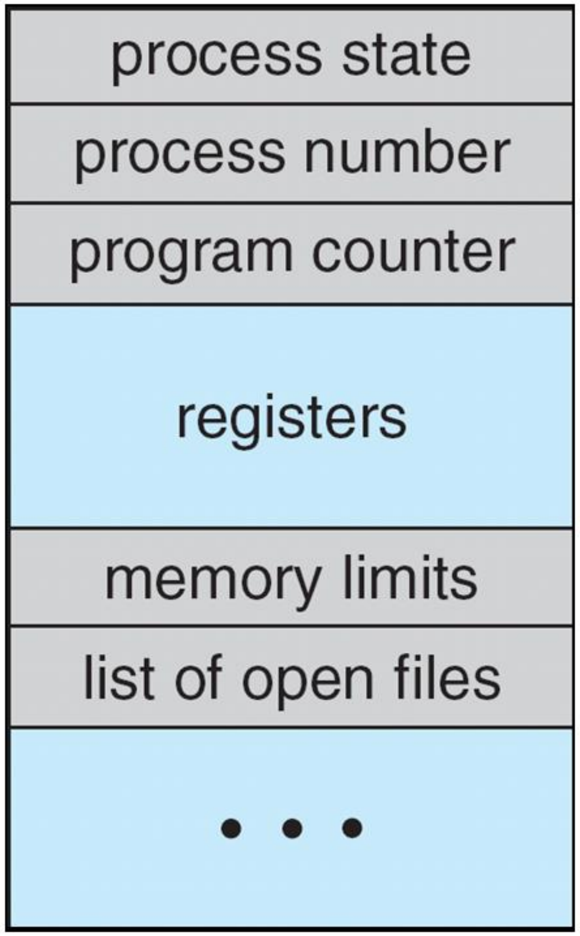


Process Control Block (PCB) مقطع تحكم العملية

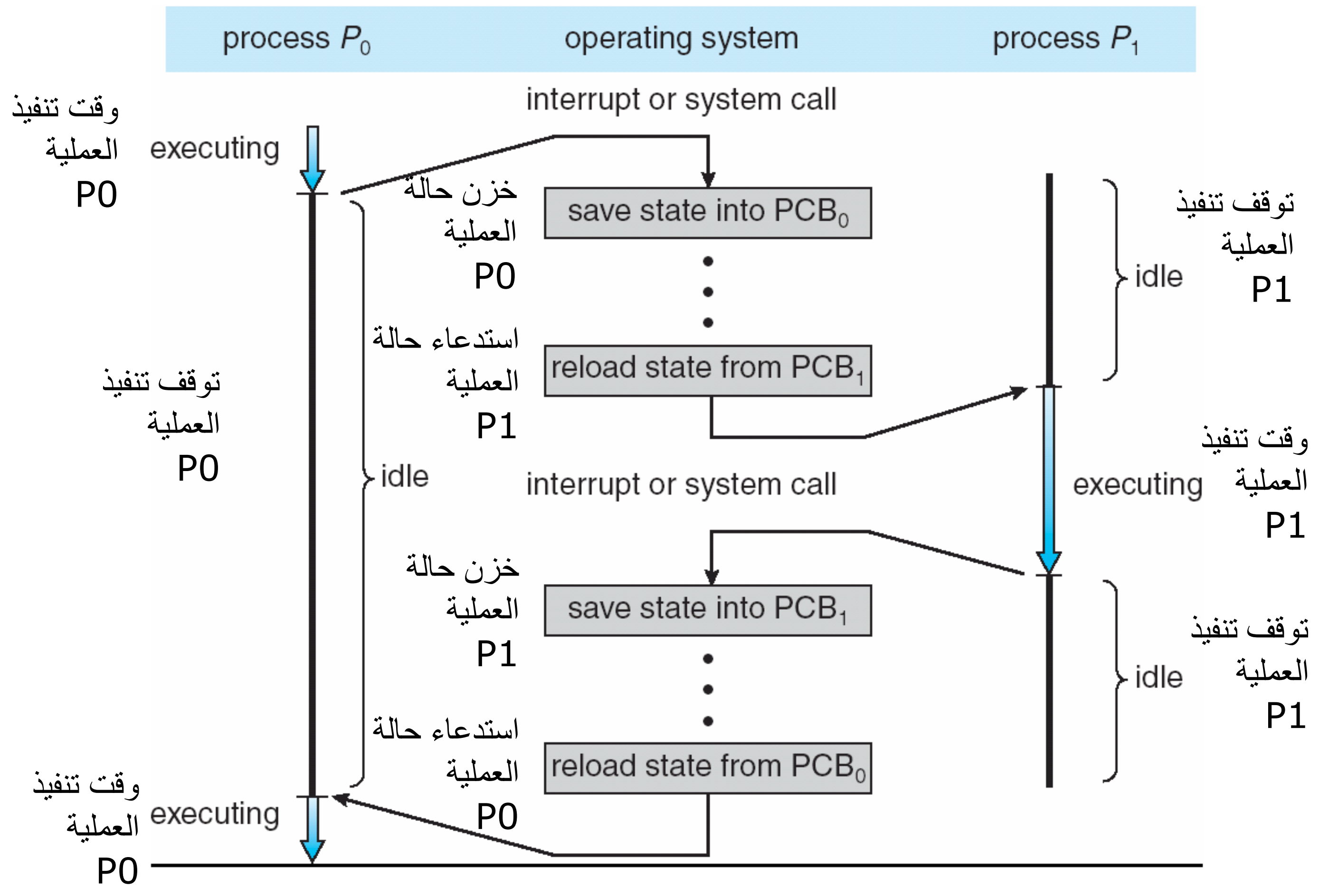
Information associated with each process هو المعلومات المرتبطة بكل عملية

(also called **task control block**) كذلك يدعى كتلة تحكم المهمة يتكون من

- Process state – running, waiting, etc حالة العملية
- Program counter – location of instruction to next execute عداد البرنامج- يحتوي عنوان التعليمة الاحقة في التنفيذ
- CPU registers – contents of all process-centric registers مسجلات وحدة المعالجة المركزية - محتويات جميع المسجلات التي تركز على العملية
- CPU scheduling information- priorities, scheduling queue pointers معلومات جدولة وحدة المعالجة المركزية - الأولويات، جدولة مؤشرات طابور الانتظار
- Memory-management information – memory allocated to the process معلومات إدارة الذاكرة - الذاكرة المخصصة للعملية
- Accounting information – CPU used, clock time elapsed since start, time limits معلومات الحوسبة - وحدة المعالجة المركزية المستخدمة، الوقت المنقضي منذ البداية، الحدود الزمنية
- I/O status information – I/O devices allocated to process, list of open files - I/O معلومات حالة أجهزة المخصصة للعملية وقائمة الملفات المفتوحة



CPU Switch From Process to Process رسم تحويل المعالج من عملية الى اخرى



Context Switch تبادل السياق

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- عندما يتحول المعالج من عملية لاخرى يجب ان يحفظ حالة العملية القديمة عن طريق حزن مقطع تحكم العملية القديمة وتحميل مقطع تحكم العملية الجديدة
- Context of a process represented in the PCB
  سياق العملية مخزون في مقطع تحكم العملية
- Context-switch time is overhead; the system does no useful work while switching
- وقت تبديل السياق هو عبأ; لا يقوم النظام بأي عمل مفيد أثناء التبديل

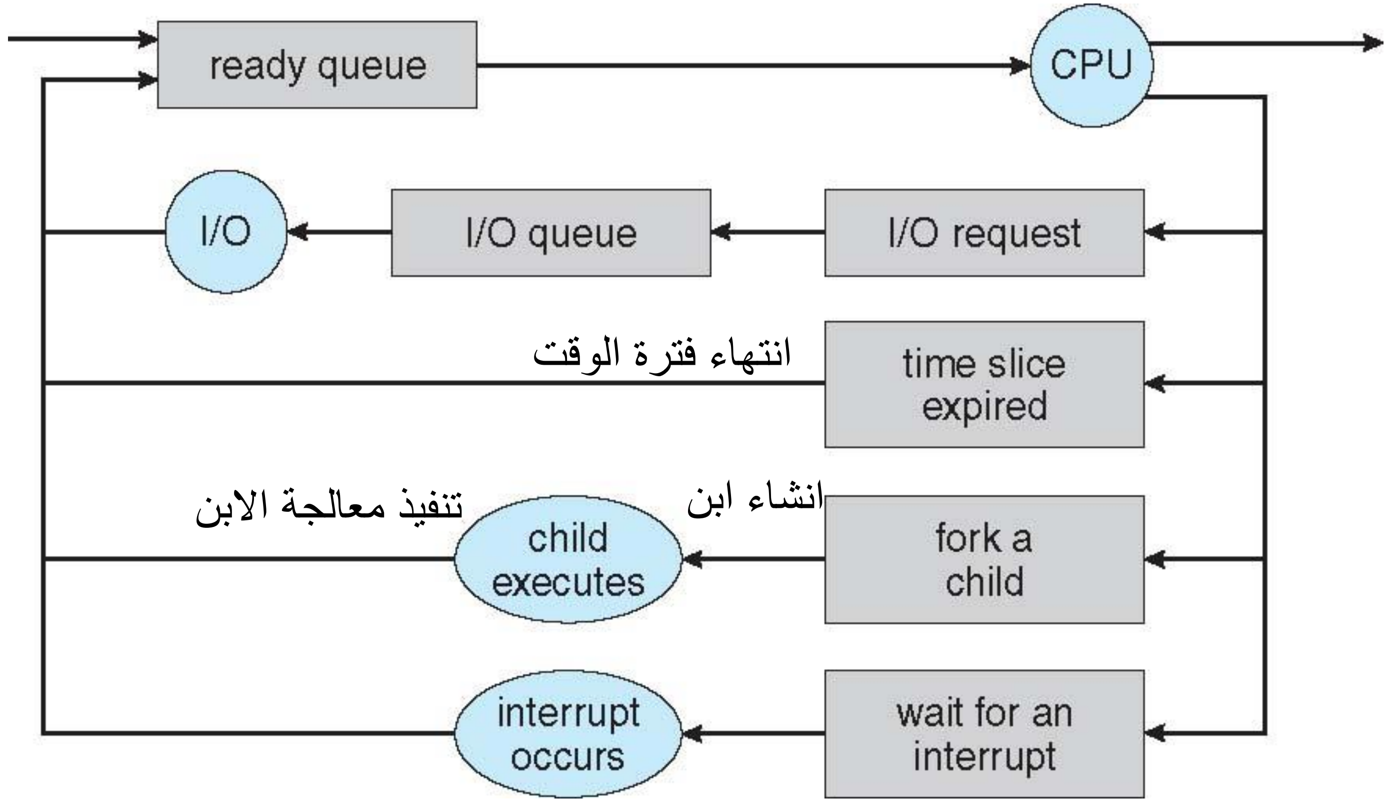
Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing للاستخدام الاعظم لوحدة المعالجة المركزية، يجب تبديل العمليات بسرعة على وحدة المعالجة المركزية لتقاسم الوقت
- **Process scheduler selects among available processes for next execution on CPU** جدولة العمليات يحدد من بين العمليات المتاحة للتنفيذ التالي على وحدة المعالجة المركزية
  - **Maintains scheduling queues of processes** تقسم طوابير جداول المعالجة الى
    - **Job queue – set of all processes in the system** 1- طابور الوظائف: تعيين جميع العمليات الى النظام
    - **Ready queue – set of all processes residing in main memory, ready and waiting to execute** - 2- طابور الجاهزية - مجموعة من جميع العمليات الموجودة في الذاكرة الرئيسية، جاهزة وتنتظر التنفيذ
    - **Device queues – set of processes waiting for an I/O device**
    - **3- طابور الأجهزة - مجموعة العمليات التي تنتظر اجهزة I/O**

Processes migrate among the various queues العمليات تتنقل بين مختلف الطوابر اثناء تنفيذها

Representation of Process Scheduling تمثيل جدولة العمليات

- **Queuing diagram** represents queues, resources, flows
  - مخطط الطوابير يمثل الطوابير, المصادر, المسارات



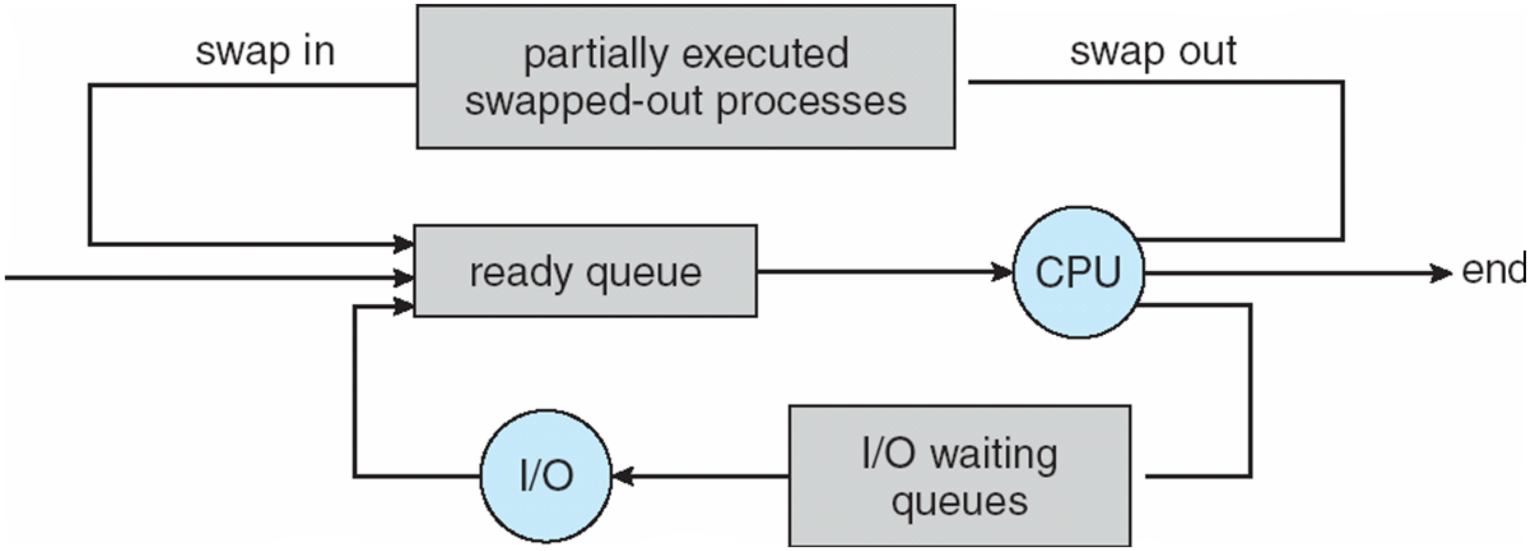
Schedulers انواع المجدولات

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU 1- جدولة قصيرة الامد (مجدولات المعالج)- تحديد العملية التي يجب تنفيذها بعد ذلك وتخصيص وحدة المعالجة المركزية
  - Sometimes the only scheduler in a system في بعض الأحيان المجدول الوحيد في النظام
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
  - يتم استدعاء مجدول قصير الامد بشكل متكرر (ميلي ثانية) (يجب أن يكون سريعًا)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue 2-جدولة طويلة الأمد (أو جدولة الوظائف) - يحدد العمليات التي يجب إحضارها إلى طابور الجاهزية
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow) يتم استدعاء المجدول طويل الامد بشكل غير متكرر (ثانية ودقائق) (قد يكون بطيئاً)
  - The long-term scheduler controls the **degree of multiprogramming**
  - يتحكم المجدول طويل الامد في درجة البرمجة المتعددة
- Processes can be described as either: يمكن وصف العمليات إما:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts العملية المقيدة للادخال و الاخراج- هي التي تقضي اغلب وقتها في عمل الادخال و الاخراج بدل الحسابات , اشغال قصير للمعالج
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts العملية المقيدة للمعالج -هي التي تقضي اغلب وقتها في الحسابات, اشغال طويل للمعالج

Addition of Medium Term Scheduling إضافة جدولة متوسطة الأمد

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease يمكن إضافة جدولة متوسطة الأجل إذا كانت درجة البرمجة المتعددة تحتاج إلى تقليل
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**

المبادلة:إزالة العملية من الذاكرة، تخزن على القرص، وتجلب مرة أخرى من القرص لمتابعة التنفيذ



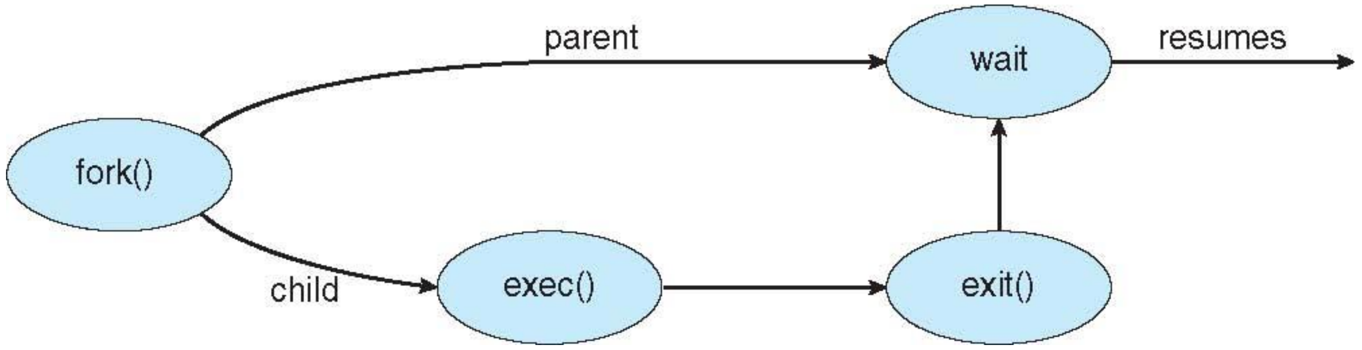
Process Creation إنشاء العملية

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
  - **عملية الأباء تنشأ عمليات ابناء، والتي، بدورها تنشأ عمليات أخرى، وتشكيل شجرة من العمليات**
- Generally, process identified and managed via a **process identifier** (**pid**) بشكل عام، يتم تحديد العملية وإدارتها عبر معرّف العملية
- Resource sharing options خيارات مشاركة الموارد
  - Parent and children share all resources الاباء و الاولاد يشتركون في جميع الموارد
  - Children share subset of parent's resources الاباء الاولاد يشتركون في بعض موارد
  - Parent and child share no resources الاباء و الاولاد لا يشتركون في الموارد
- Execution options خيارات التنفيذ
  - Parent and children execute concurrently الاباء و الاولاد تنفذ في وقت واحد
  - Parent waits until children terminate الاباء تنتظر حتى انتهاء الاولاد

Process Creation (Cont.)

- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
  - تنفذ العملية العبارة الأخيرة ثم يطلب من نظام التشغيل لحذفه باستخدام استدعاء نظام الخروج() .
    - Returns status data from child to parent (via **wait()**) ارجاع البيانات من الابناء الى الاباء من خلال الانتظار wait()
  - Process' resources are deallocated by operating system

تحرر موارد العملية بواسطة نظام التشغيل

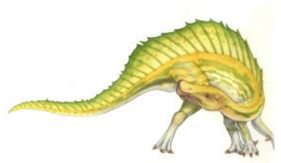


Process Termination انهاء العملية

- Parent may terminate the execution of children processes using the **`abort()`** system call. Some reasons for doing so: قد ينهي الأباء تنفيذ العمليات الابناء باستخدام استدعاء النظام إحباط() . بعض الأسباب للقيام بذلك :
  - Child has exceeded allocated resources تجاوز الابناء المورد المخصص لها
  - Task assigned to child is no longer required المهمة المعينة إلى الابناء لم تعد مطلوبة
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates يتم إنهاء الأباء ولا تسمح أنظمة التشغيل لابناء بالمتابعة إذا تم إنهاء الأباء

Cooperating Processes العمليات المتعاونة

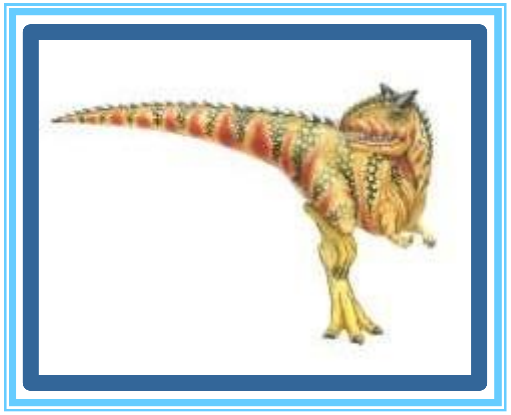
- ***Independent*** process cannot affect or be affected by the execution of another process العملية مستقلة: لا يمكن أن تؤثر أو تتأثر بتنفيذ عملية أخرى
- ***Cooperating*** process can affect or be affected by the execution of another process العملية المتعاونة : مكن أن تؤثر أو تتأثر بتنفيذ عملية أخرى
- Advantages of process cooperation مزايا التعاون في العمليات
  - Information sharing 1- تبادل المعلومات
  - Computation speed-up 2-زيادة سرعة الحساب
  - Modularity 3- النمطيه
  - Convenience 4- الملائمة



Producer-Consumer Problem مشكلة المنتج والمستهلك

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - نموذج للعمليات المتعاونة ، تنتج عملية المنتج المعلومات التي تستهلكها عملية المستهلك
  - **unbounded-buffer** places no practical limit on the size of the buffer
    - لايضع المخزن المؤقت غير المقيد أي حد عملي على حجم المخزن
  - **bounded-buffer** assumes that there is a fixed buffer size
    - يفترض المخزن المؤقت المحدد وجود حجم خزن مؤقت ثابت

Chapter 4: Threads & Concurrency التزامن

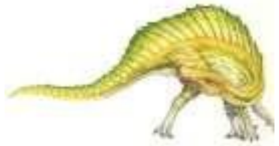


Chapter 4: Threads

- Overview
- Multicore Programming

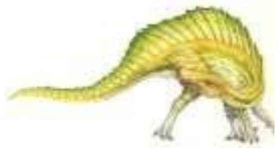
  البرمجة متعددة الانوية المحتوية على اكثر من معالج مثل كور اي 3 ..الخ

- Multithreading Models
- Thread Libraries
- Implicit Threading تجزئة ضمنية للعمليات
- Threading Issues المشاكل التي تواجهنا اثناء عملية التجزئة
- Operating System Examples الامثلة على النظم التي تستخدم التجزئة



Objectives

- Identify the basic components of a thread, and contrast threads and processes تعريف الوحدات الاساسية لعملية التجزئة
- Describe the benefits and challenges of designng multithreaded applications وصف الفوائد والتحديات لتصميم تطبيقات متعددة التجزئة
- Illustrate different approaches to implicit threading including thread pools, fork-join, and Grand Central Dispatch
- Describe how the Windows and Linux operating systems represent threads كيفية تمثيل التجزئة في الانظمة
- Design multithreaded applications using the Pthreads, Java, and Windows threading APIs كيفية تصميم التجزئة



Motivation

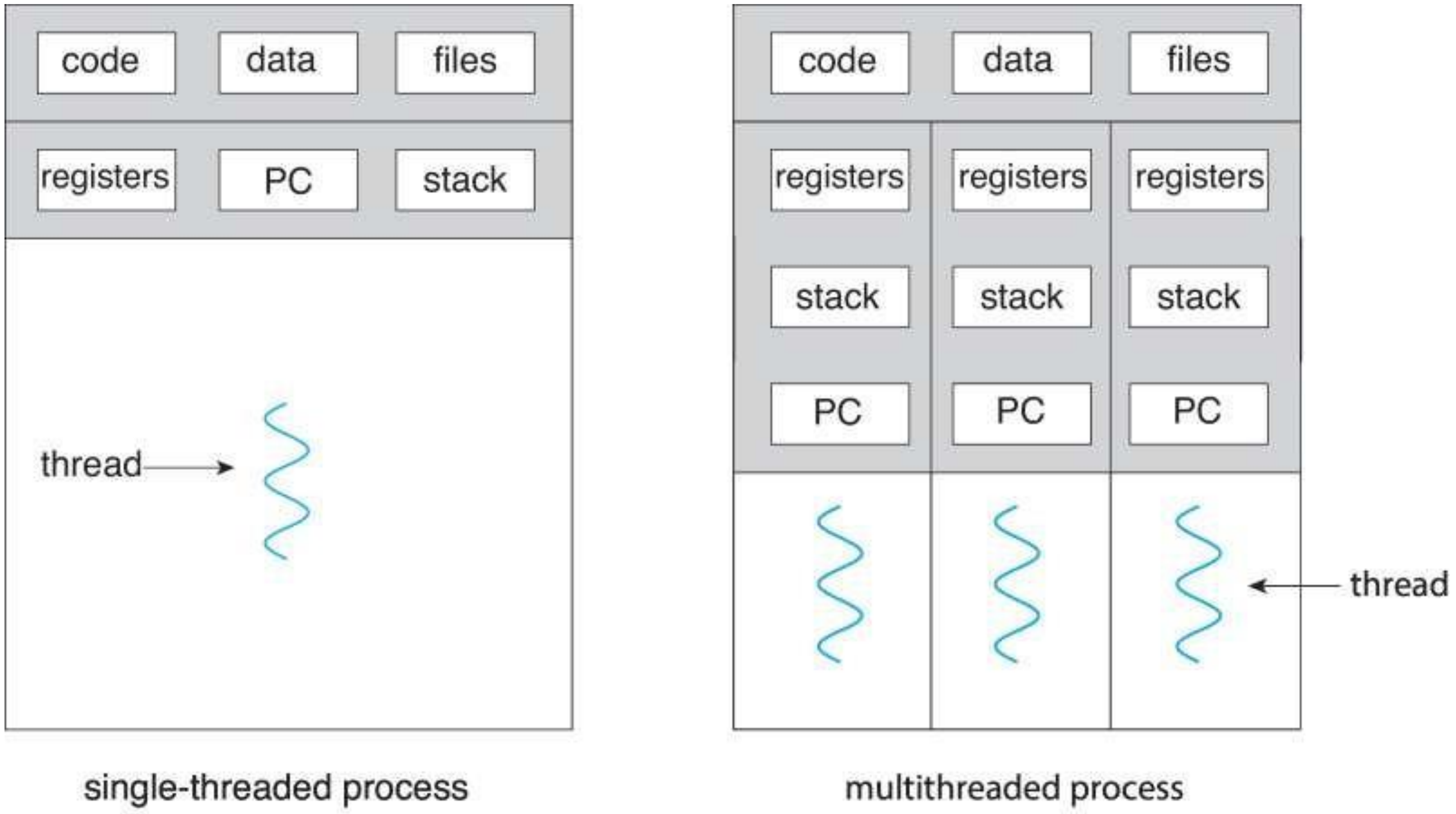
- Most modern applications are multithreaded كي يسهل التعامل معها معالجة واكتشاف احطاء
- Threads run within application الثريد الواحد قد يكون دالة او جزء من برنامج اكبر
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request

  كمثال اليوتيوب نفسه ملتبل ثريد .. واحدة نها لمعالجة الصوت والاخرى للصورة والاخرى للصال .. وهكذا
- **Process** creation is **heavy-weight** while

  thread creation is **light-weight**

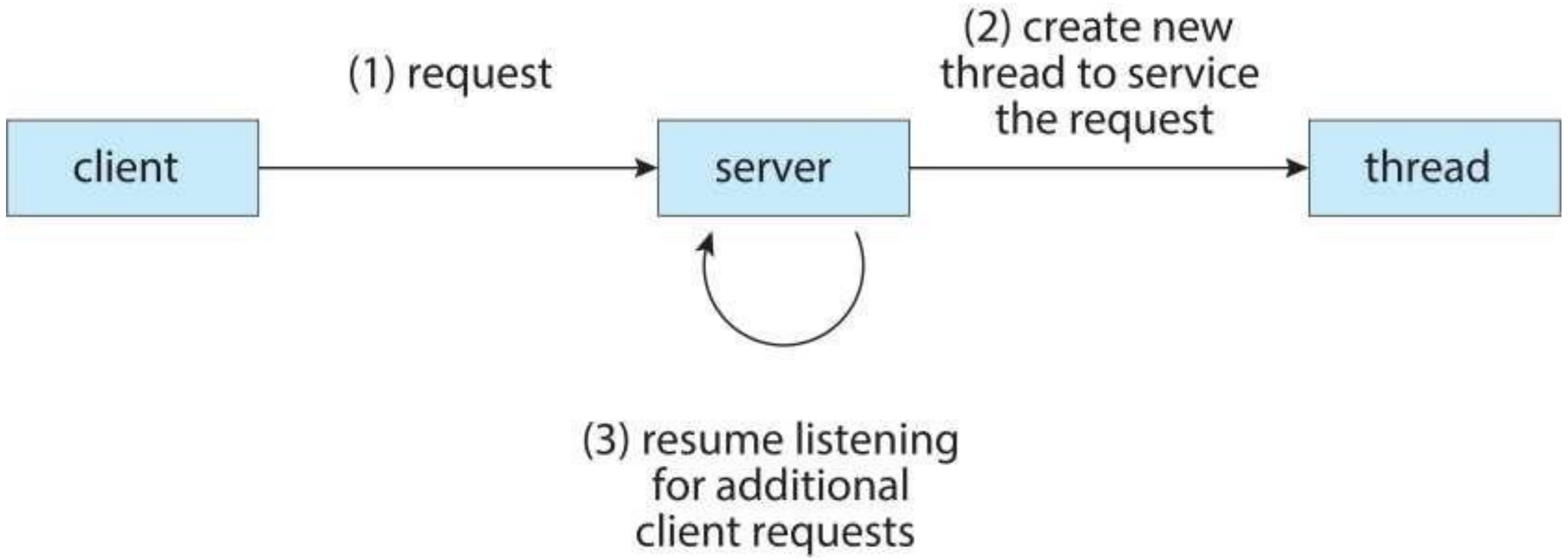
  اكيد دراسة كيفية عمل القريد لاجل ان نعرف كيف نستخدمه لانه لا ياخذ مصادر كثيرة مثل المعالجة .. فالافضل انشاء ثريد متعددة لانجاز المعالجة افضل من انشاء معالجة
- Can simplify code, increase efficiency
- Kernels are generally multithreaded الانظمة الحديثة هي متعددة الثريد اصلا فيجب فهمها

Single and Multithreaded Processes



Multithreaded Server Architecture

معمارية خادم متعدد المسار / متعدد التجزئة


(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

على فرض انه **server** هو عبارة عن **CPU** والـ**Clinent** هو المستخدم الذي يطلب عملية ما لانجازها ، فان الطلب الواحد يؤدي الى انشاء مسار/مسارات للعملية الواحدة لاجل تنفيذها ، مع الاستمرار بالاصغاء الى **Clinent** عله يطلب عمليات اخرى

Benefits

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces

زيادة استجابة البرامج بحيث لا يتوقف كل البرنامج عن التنفيذ في حال توقف جزء منه لسبب ما

- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing

مشاركة مصادر الحاسبة من قبل اكثر من ثريد/مسار من عمليات مختلفة او عملية واحدة لنفس المصدر بحيث لا يكون المصدر حكرا على انجاز شيء واحد فقط

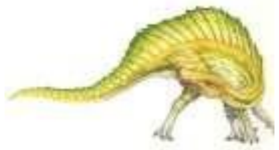
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching

رخص انشاء الثريد/المسار فمن الارخص (من ناحية مصادر الحاسبة) هو انشاء الثريد واقل عبئا واسرع من انشاء بروسيس او عملية

- **Scalability –** process can take advantage of multicore Architectures

قابلية التوسع في استخدام النظام ، بحيث يمكن استخدام معمارية متعدد الانوية بشكل امثل

- Reference**:** https://www.tutorialspoint.com/operating_system/os_multi_threading.htm



Multicore Programming

- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  يحتوي الملتي كور على عدة معالجات داخلها وهذا يضع عبيء وتحديات منها :
  - **Dividing activities** تجزئة النشاطات للاستفادة القصوى من العالجات المتعددة
  - **Balance**

    موازنة استخدام كل المعالجات في الملتي كور بالتساوي لكي لايقع العبيء على عدد منها دون الاخريات
  - **Data splitting** تجزئة البيانات بين المعالجات المختلفة وليس فقط العمليات
  - **Data dependency** معرفة البيانات المعتمدة على غيرها كي يتم تصميم الثريد بطريقة صحيحة
  - **Testing and debugging** اختبار البيانات وتصحيح الاخطاء في الملتي كور والملتي ثريد

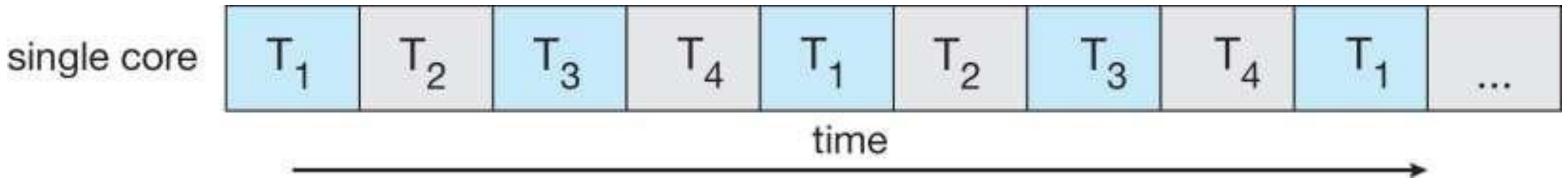
- ***Parallelism*** implies a system can perform more than one task simultaneously

  التوازي يعني أن النظام يمكنه أداء أكثر من مهمة في وقت واحد
- ***Concurrency*** supports more than one task making progress

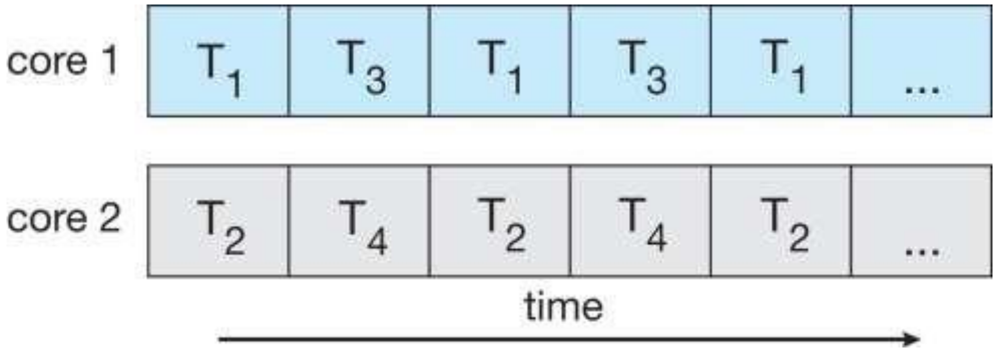
  يدعم التزامن أكثر من مهمة واحدة لتحقيق التقدم
  - Single processor / core, scheduler providing concurrency

Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



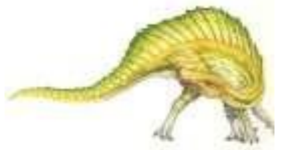
Multicore Programming

- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each

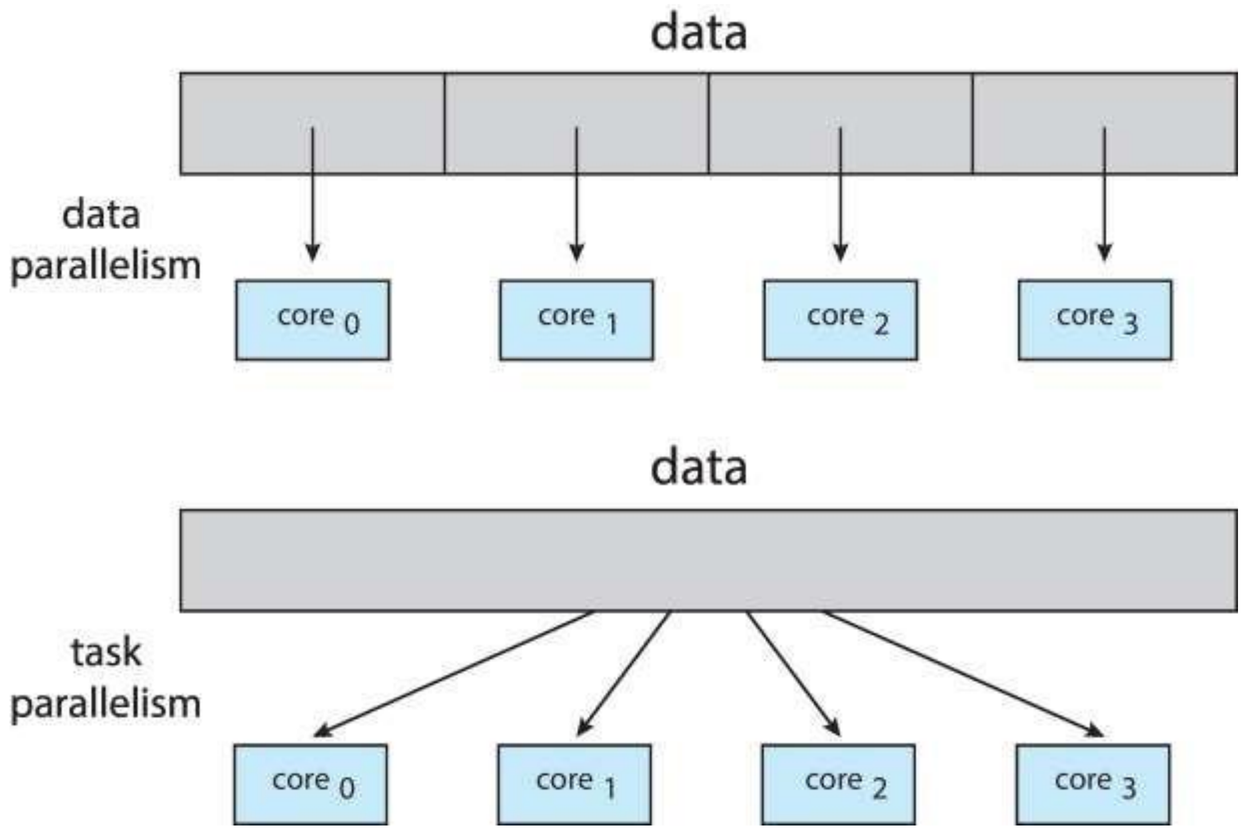
    توازي البيانات على الكور مثل ضرب مصفوفة 1000 في 1000
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation

    توازي البرنامج الى عدة مسارات على الكور

- Reference: https://www.tutorialspoint.com/data-parallelism-vs-task-parallelism

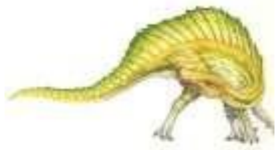


Data and Task Parallelism

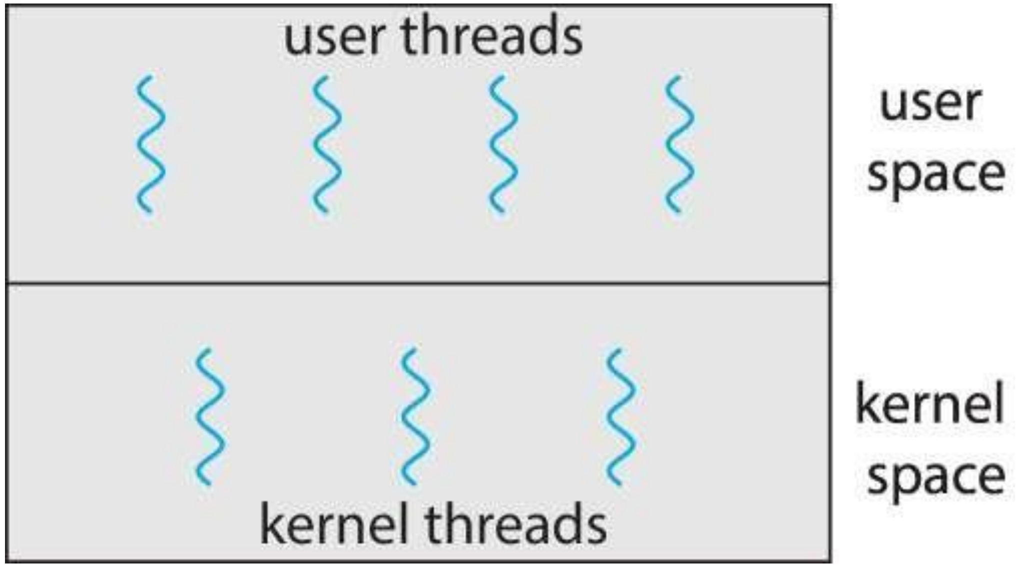


User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Linux
  - Mac OS X
  - iOS
  - Android



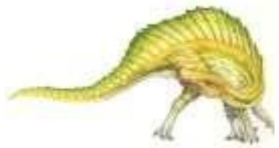
User and Kernel Threads



Reference:
https://www.tutorialspoint.com/operating_system/os_multi_threading.htm

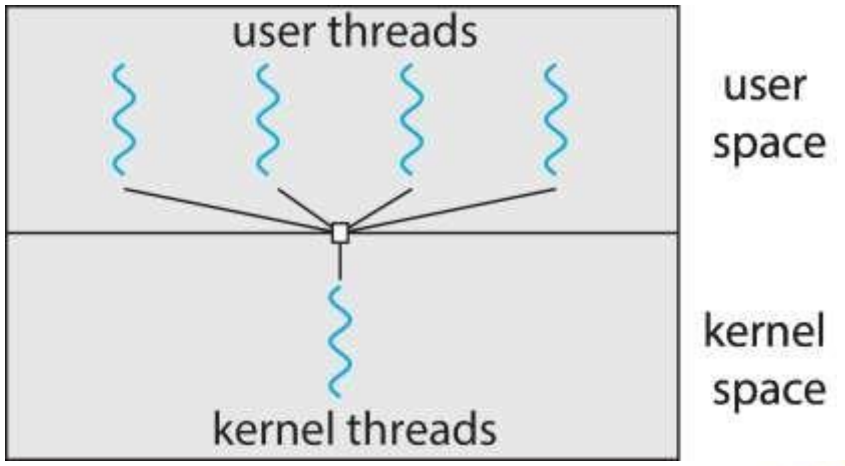
Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many



Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

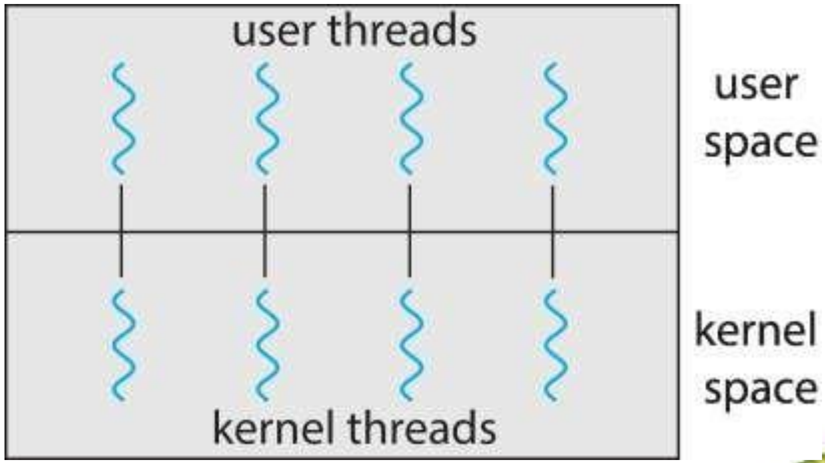


One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead

عدد المسارات وتزامنها مقيد بقدرة النظام لتحمل المسارات وليس الى ما لانهاية

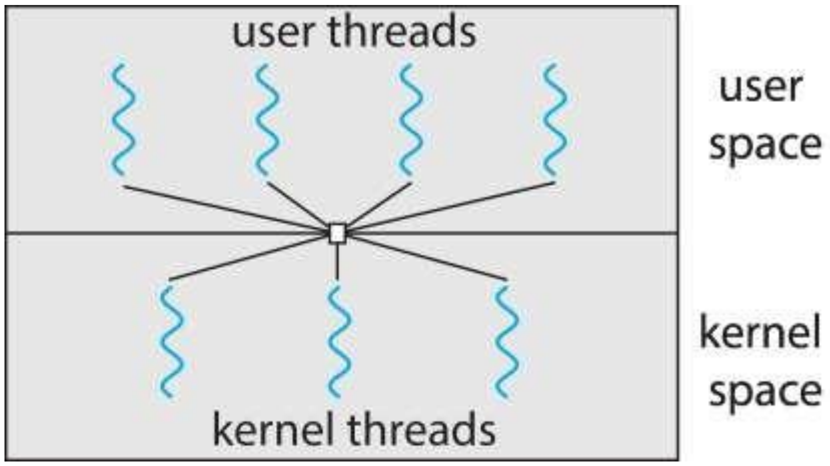
- Examples
  - Windows
  - Linux



Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Windows with the *ThreadFiber* package
- Otherwise not very common

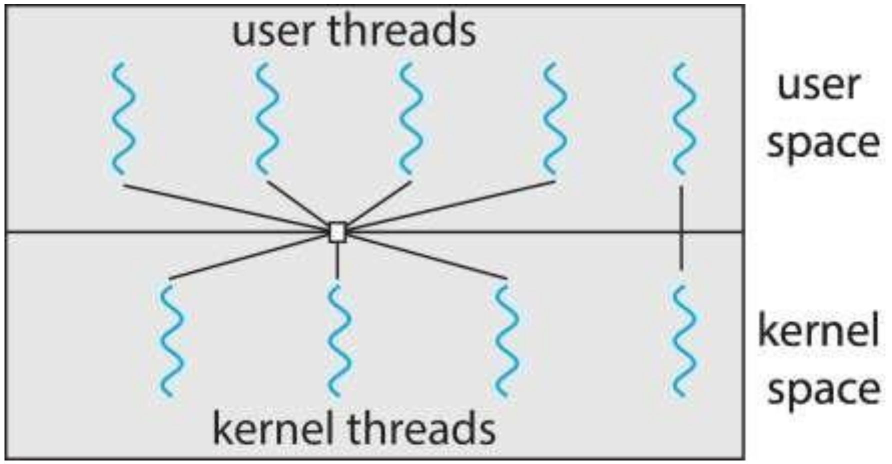
لاحظ ليس شرطا ان يكون عدد الثريد في الكيرنل هو نفسه عدد الثريد في اليوزر سبيس



Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread

يحتوي على مستويين اما متعدد: متعدد او واحد الى واحد
اي انه يسمح بكلاهما حسب الحاجة



Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Linux & Mac OS X)

Pthreads Example

```
#include <pthread.h>
#include <stdio.h>

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  /* set the default attributes of the thread */
  pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid, &attr, runner, argv[1]);
  /* wait for the thread to exit */
  pthread_join(tid,NULL);

  printf("sum = %d\n",sum);
}
```

Pthreads Example (cont)

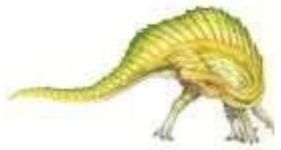
```
/* The thread will execute in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

Pthreads Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

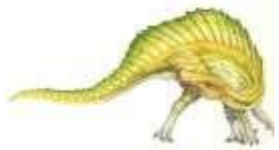
Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations



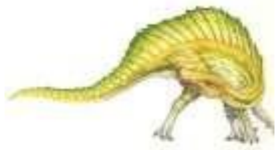
Semantics of fork() and exec()

- Does **`fork()`** duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork
- **`exec()`** usually works as normal – replace the running process including all threads



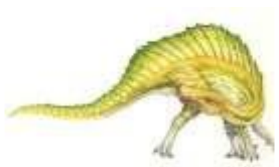
Signal Handling

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred.
- A **signal handler** is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled by one of two signal handlers:
     1. default
     2. user-defined
- Every signal has **default handler** that kernel runs when handling signal
  - **User-defined signal handler** can override default
  - For single-threaded, signal delivered to process

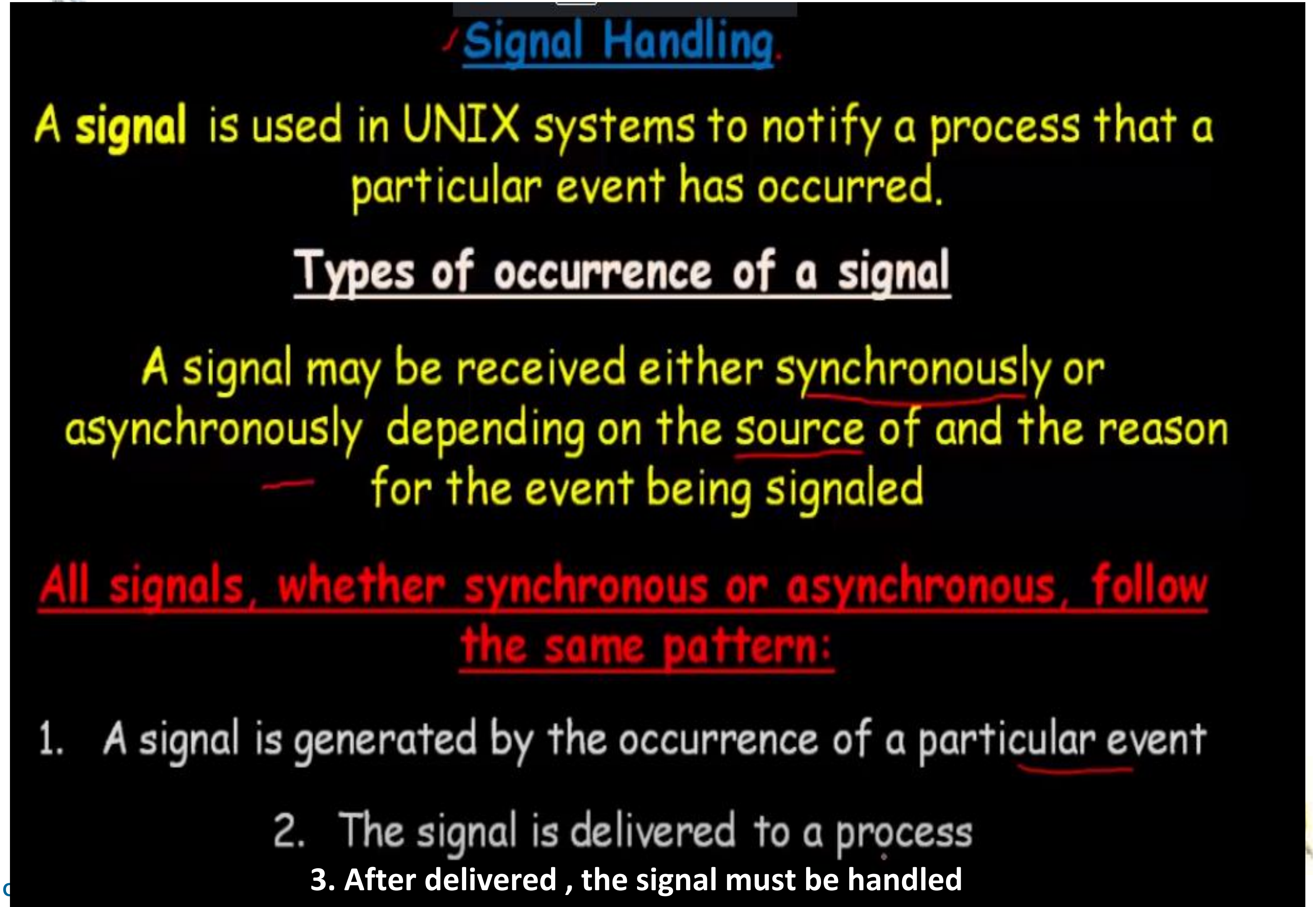


Signal Handling (Cont.)

- Where should a signal be delivered for multi-threaded?
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process



Signal Handling (Cont.)



Signal Handling

A **signal** is used in UNIX systems to notify a process that a particular event has occurred.

Types of occurrence of a signal

A signal may be received either synchronously or asynchronously depending on the source of and the reason for the event being signaled

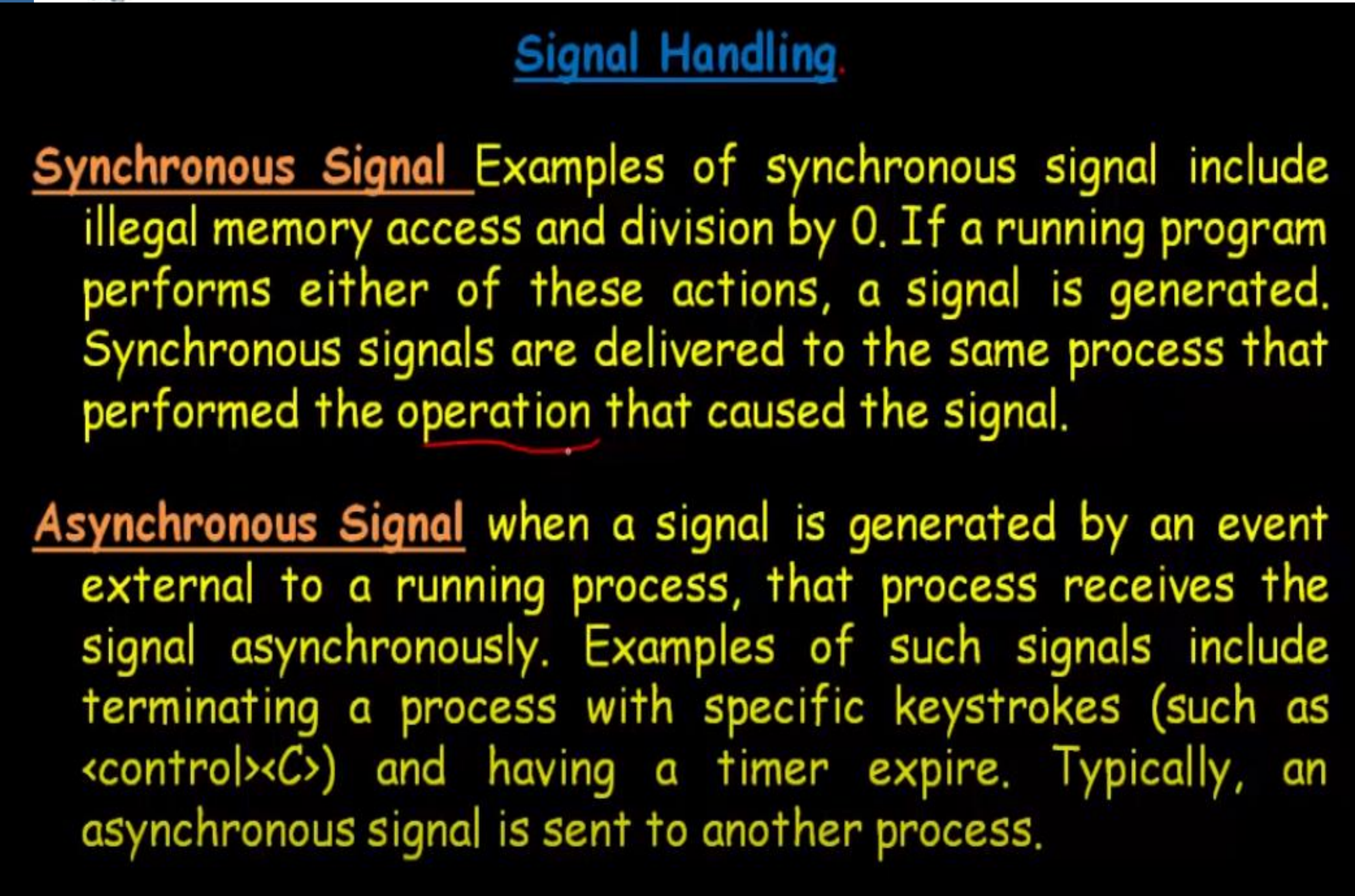
All signals, whether synchronous or asynchronous, follow the same pattern:

1. A signal is generated by the occurrence of a particular event

2. The signal is delivered to a process

3. After delivered , the signal must be handled

Signal Handling (Cont.)

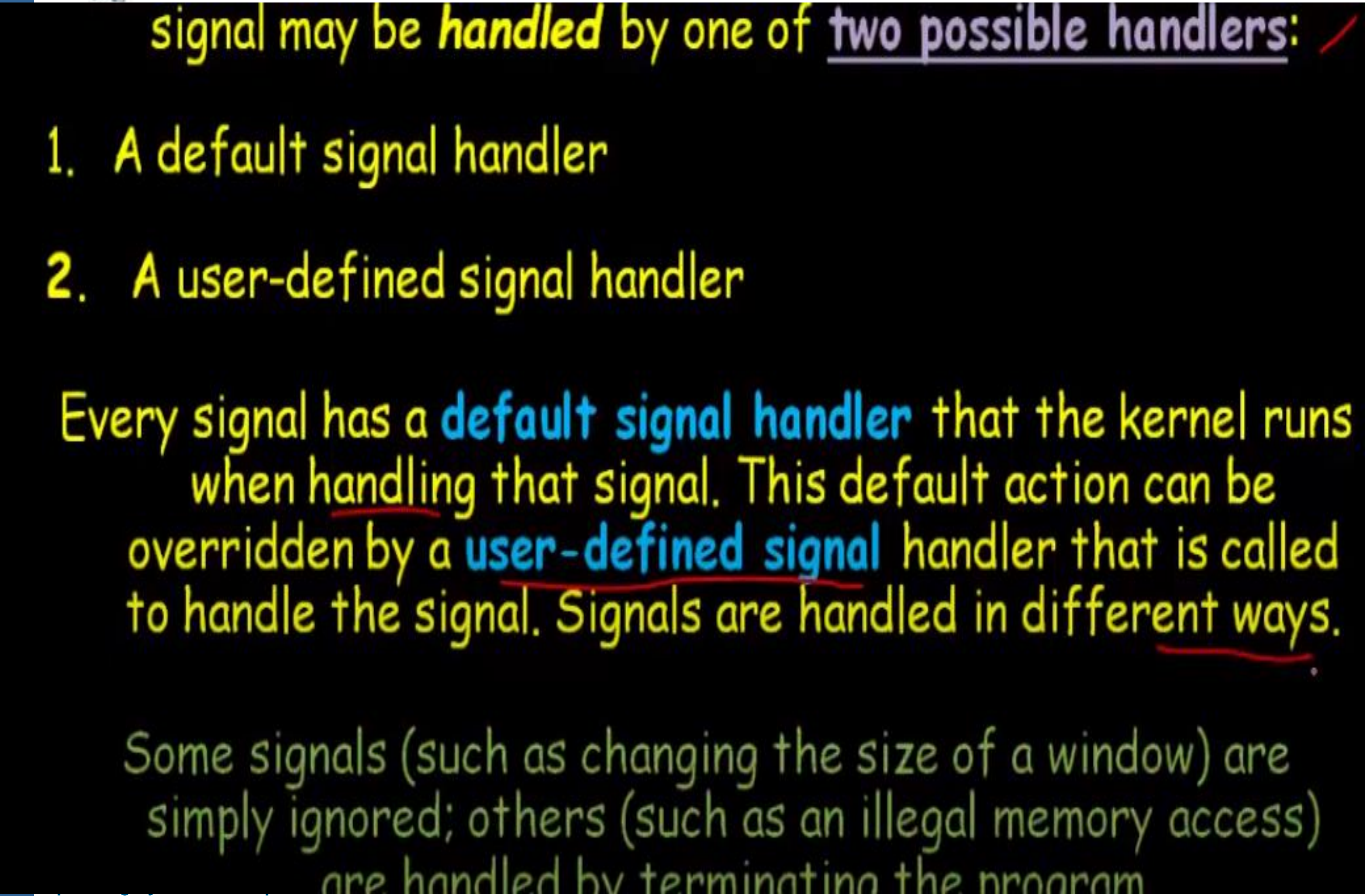


Signal Handling.

Synchronous Signal Examples of synchronous signal include illegal memory access and division by 0. If a running program performs either of these actions, a signal is generated. Synchronous signals are delivered to the same process that performed the operation that caused the signal.

Asynchronous Signal when a signal is generated by an event external to a running process, that process receives the signal asynchronously. Examples of such signals include terminating a process with specific keystrokes (such as <control><C>) and having a timer expire. Typically, an asynchronous signal is sent to another process.

Signal Handling (Cont.)



signal may be ***handled*** by one of two possible handlers:

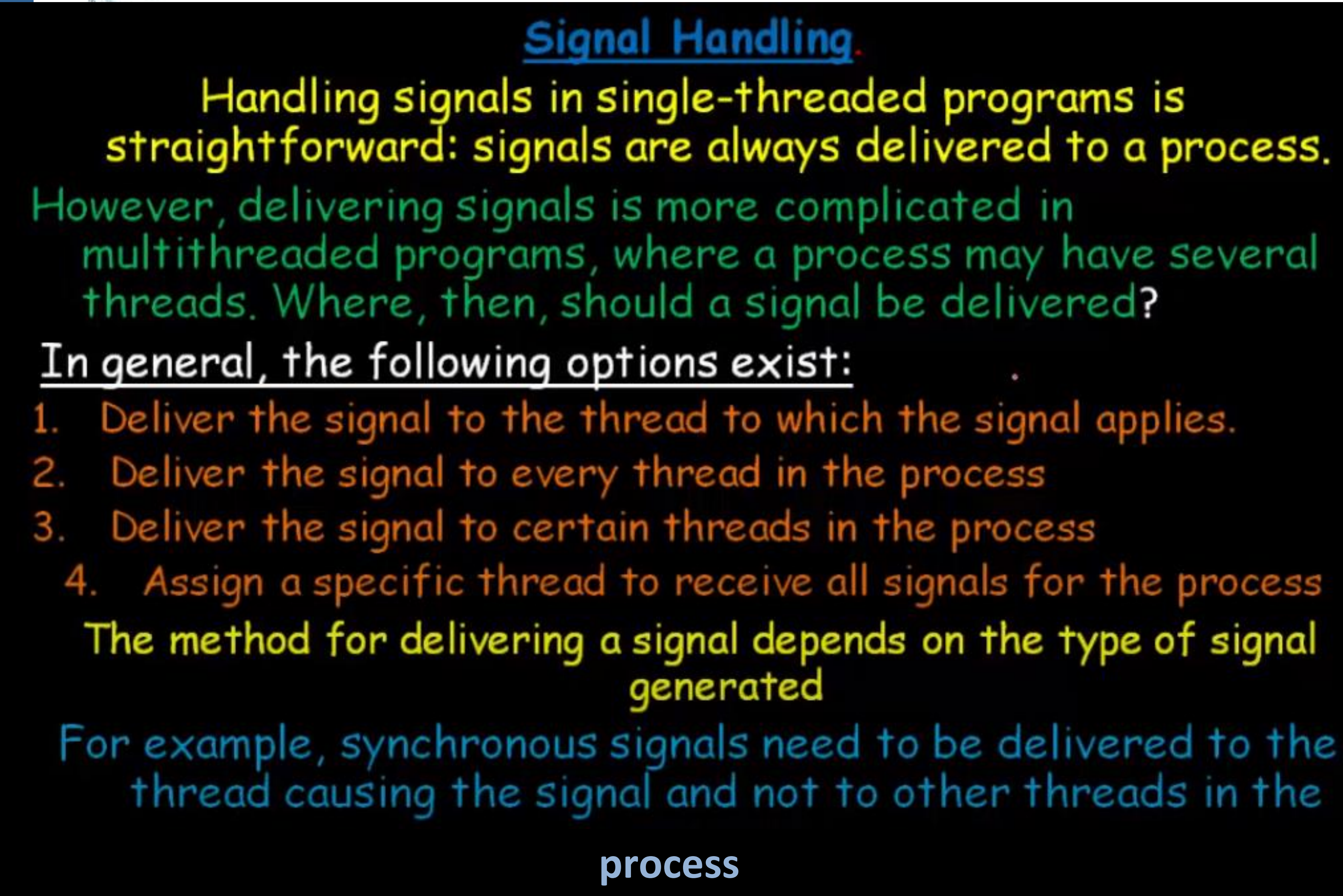
1. A default signal handler

2. A user-defined signal handler

Every signal has a default signal handler that the kernel runs when handling that signal. This default action can be overridden by a user-defined signal handler that is called to handle the signal. Signals are handled in different ways.

Some signals (such as changing the size of a window) are simply ignored; others (such as an illegal memory access) are handled by terminating the program

Signal Handling (Cont.)



Signal Handling.

Handling signals in single-threaded programs is straightforward: signals are always delivered to a process.

However, delivering signals is more complicated in multithreaded programs, where a process may have several threads. Where, then, should a signal be delivered?

In general, the following options exist:

1. Deliver the signal to the thread to which the signal applies.
2. Deliver the signal to every thread in the process
3. Deliver the signal to certain threads in the process
4. Assign a specific thread to receive all signals for the process

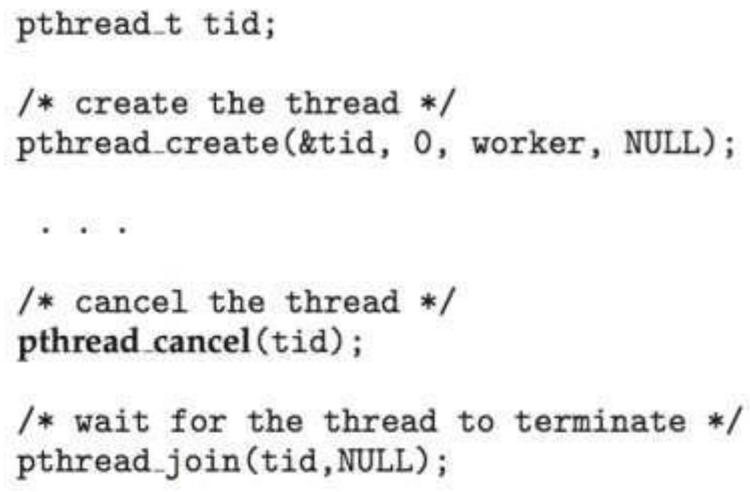
The method for delivering a signal depends on the type of signal generated

For example, synchronous signals need to be delivered to the thread causing the signal and not to other threads in the

process

Thread Cancellation

- Terminating a thread before it has finished
- Thread to be canceled is **target thread**
- Two general approaches:
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled
- Pthread code to create and cancel a thread:



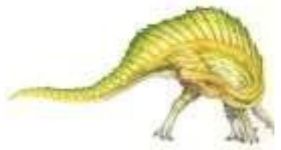
```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

 . . .

/* cancel the thread */
pthread_cancel(tid);

/* wait for the thread to terminate */
pthread_join(tid,NULL);
```

Thread Cancellation (Cont.)

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- If thread has cancellation disabled, cancellation remains pending until thread enables it
- Default type is deferred
  - Cancellation only occurs when thread reaches **cancellation point**
    - I.e. `pthread_testcancel()`
    - Then **cleanup handler** is invoked
- On Linux systems, thread cancellation is handled through signals



Thread Cancellation in Java

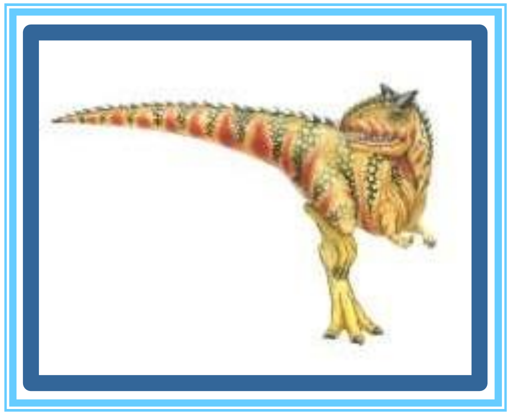
- Deferred cancellation uses the **`interrupt()`** method, which sets the interrupted status of a thread.

```
Thread worker;

 . . .

/* set the interruption status of the thread */
worker.interrupt()
```

- A thread can then check to see if it has been interrupted:

```
while (!Thread.currentThread().isInterrupted()) {
    . . .
}
```

End of Chapter 4



Chapter 4: Threads

الفصل الرابع: المسارات

- A *thread*: is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, (and a thread ID.)

- المسار: هو وحدة أساسية من استخدام وحدة المعالجة المركزية، تتكون من عداد برنامج ومكدس ومجموعة من السجلات و (ومعرف المسار).

- Traditional processes have a single thread of control – There is one program counter, and one sequence of instructions that can be carried out at any given time.

- العمليات التقليدية (عمليات ذات المسار) الواحد لديها مسار واحد من السيطرة – هناك عداد برنامج واحد، وتسلسل واحد من التعليمات التي يمكن تنفيذها في أي وقت من الأوقات

- Multi-threaded applications: have multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files. As shown in Figure 4.1.

- التطبيقات المتعددة المسارات : هي التطبيقات التي تمتلك اكثر من مسار ظمن العملية الواحدة, حيث لكل مسار (عداد برنامج, و مكدس, و مسجلات) خاص بة, لاكنها تتشارك الشفرات (التعليمات) المشتركة, و هياكل معينة مثل الملفات المفتوحة. كما موضح في شكل 4-1.

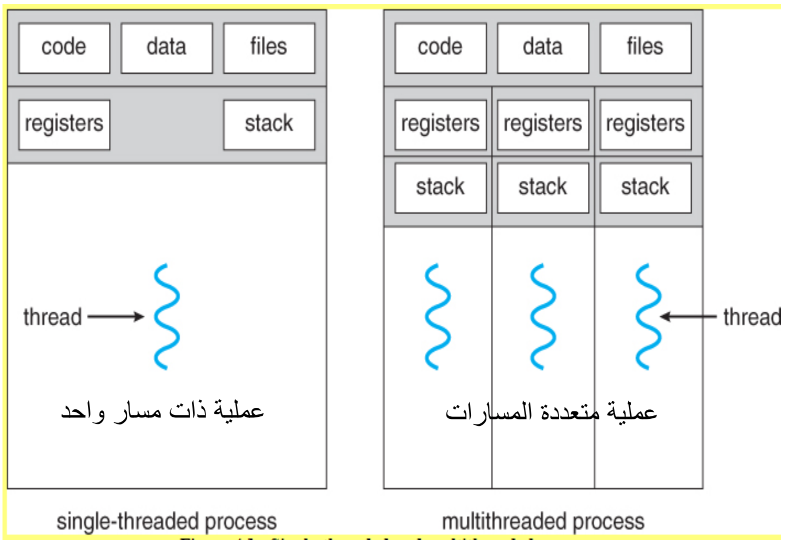


Figure 4.1 - Single-threaded and multithreaded processes

- **Benefits** **فوائد المسارات المتعددة**
- 1- Responsiveness - One thread may provide rapid response while other threads are blocked or slowed down doing intensive calculations.
- الاستجابة - قد يوفر مسار من المسارت المتعددة استجابة سريعة بينما بقية المسارت تكون محظورة أو بطيئة نتيجة إجراء عمليات حسابية مكثفة.
- 2- Resource sharing - By default threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.
- مشاركة الموارد - تشترك المسارت المتعددة بشكل افتراضي في التعليمات البرمجية المشتركة والبيانات والموارد الأخرى، مما يسمح بتنفيذ مهام متعددة في وقت واحد في مساحة عنوان واحدة.
- 3- Economy - Creating and managing threads (and context switches between them) is much faster than performing the same tasks for processes.
- - الاقتصاد - إنشاء وإدارة المسارات المتعددة (وتبادل السياق بينهما) هو أسرع بكثير من أداء نفس المهام في العمليات ذات المسار الواحد.
- Scalability, i.e. Utilization of multiprocessor architectures - A single threaded process can only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be split amongst available processors.
- قابلية التوسيع: العملية ذات المسار الواحد يمكن ان تنفذ فقط على معالج واحد بغض النظر عن عدد المعالجات الموجودة, بينما تنفيذ التطبيقات ذات المسارات المتعددة يمكن ان يقسم على عدد المعالجات المتوفرة.

Multithreading Models: نماذج المسارات المتعددة

1 . Many-To-One Model   1-نموذج متعدد إلى واحد

Many user-level threads are all mapped onto a single kernel thread.

يتكون من مسارات متعددة في حيز المستخدم تحول الى مسار واحد في حيز النواة كما في الشكل:

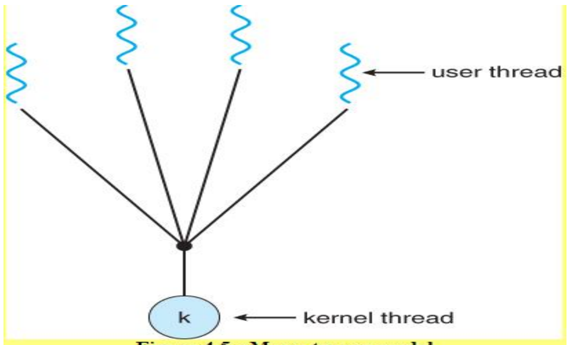


Figure 4.5 - Many-to-one model

2 . One-To-One Model: 2- نموذج واحد إلى واحد

The one-to-one model creates a separate kernel thread to handle each user thread.

يولد مسار منفصل في حيز النواة للتعامل مع كل مسار في حيز المستخدم كما في الشكل:

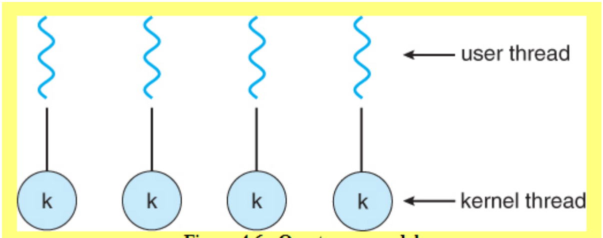


Figure 4.6 - One-to-one model

3. Many-To-Many Model   3- نموذج متعدد إلى متعدد

The many-to-many model multiplexes any number of user threads onto an equal or number of kernel threads, combining the best features of the one-to-one and many-to-one models.

عدد من المسارات المترابطة في حيز المستخدم التي عددها يساوي أو أكبر من عدد من المسارات المترابطة في حيز النواة ، يجمع بين أفضل ميزات نموذج الواحد إلى واحد ونموذج المتعددة إلى واحد.

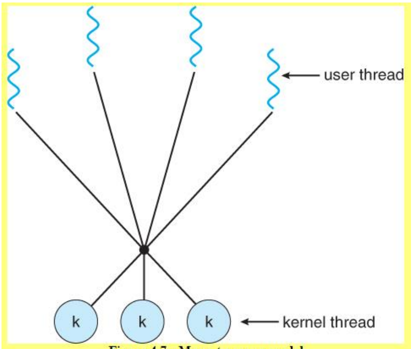


Figure 4.7 - Many-to-many model

4. Two-level model   4- نموذج من مستويين

One popular variation of the many-to-many model is the two-tier model, which allows either many-to-many or one-to-one operation.

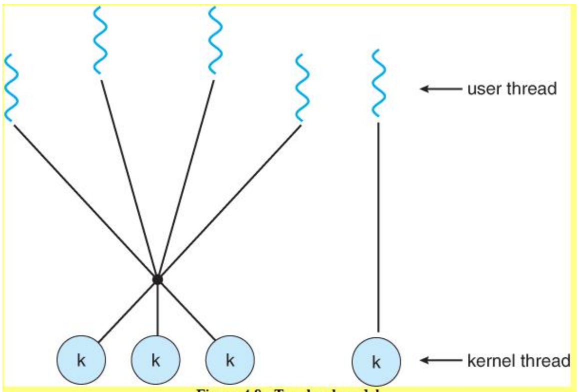
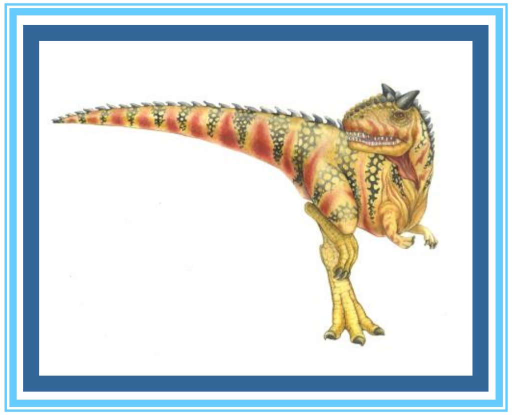


Figure 4.8 - Two-level model

احد النماذج الشائعة يتكون من نموذجين متعدد الى متعدد و واحد الى واحد.

Chapter 5: CPU Scheduling



Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms

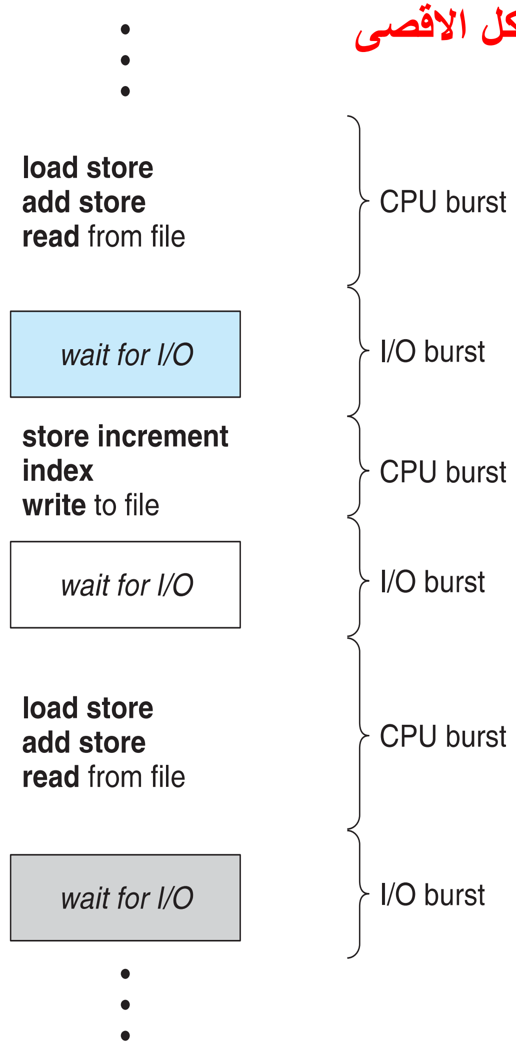
Basic Concepts

نحتاج الوصول الى كون
CPU Burst
اطول ما يمكن لزيادة استغلال
المعالج بالشكل الاقصى

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait

CPU burst: the amount of time the process uses the processor before it is no longer ready.

ببساطة المعالج ليس مشغولا دائما بالمعالجة حتى ولو
اجزاء من الثانية بسبب اجباره على انتظار مدخلات من
اجهزة الادخال التي هي عادة ابطأ من المعالج كما في مسالة
عنق الزجاجة لتحميل ملف من الهارد الى وحدة المعالجة
المركزية ولذلك .. نحتاج ان نستغل المعالج في وقت
الانتظار هذا ونقحم معالجة اخرى فيه بدلا من الانتظار



معايير اختيار خوارزميات الجدولة

Scheduling Criteria

الاستغلال

- **CPU utilization** – keep the CPU as busy as possible

كل ما كانت الخوارزمية تشغل العالج اكثر كلما كانت افضل

الإنتاجية

- **Throughput** – # of processes that complete their execution per time unit

كل ما كانت عدد العالجات المكتملة/ لكل وحدة زمنية اكثر ، كلما كانت الخوارزمية افضل

زمن الاستجابة

- **Turnaround time** – amount of time to execute a particular process

كل ما كان مقدار الوقت اللازم لتنفيذ عملية معينة اقل ، كلما كانت الخوارزمية افضل

زمن الانتظار

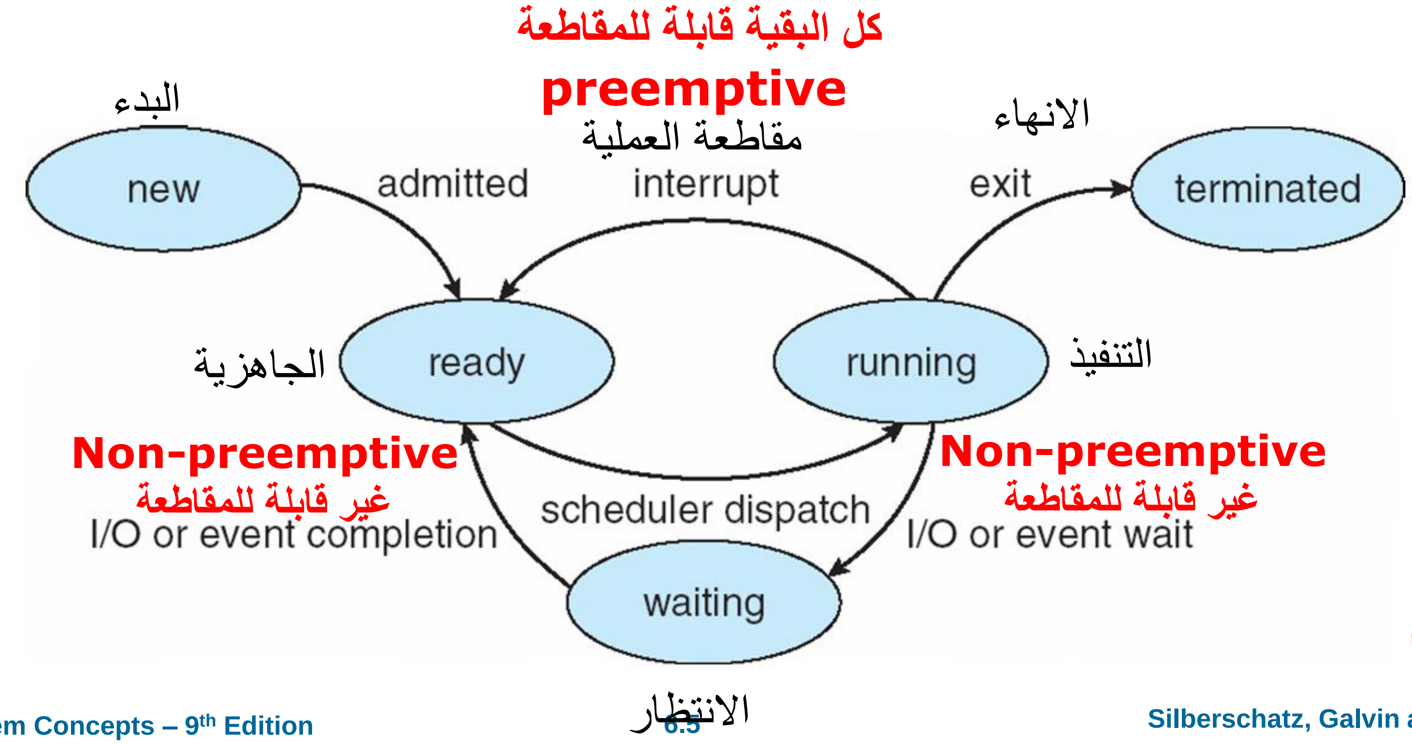
- **Waiting time** – amount of time a process has been waiting in the ready queue

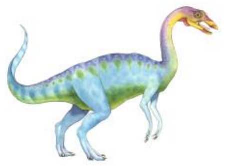
كل ما كان مقدار الوقت الذي كانت العملية تنتظره في قائمة الانتظار الجاهزة اقل، كلما كانت الخوارزمية افضل

Scheduling Algorithm Optimization Criteria

اهداف الجدولة بشكل عام

- Max CPU utilization اقصى استغلال للمعالج
- Max throughput اقصى انتاجية للمعالج
- Min turnaround time اقل زمن للاستجابة لتنفيذ المعالجة
- Min waiting time اقل زمن للانتظار في طابور الجاهزية
- Min response time اقل زمن للاستجابة

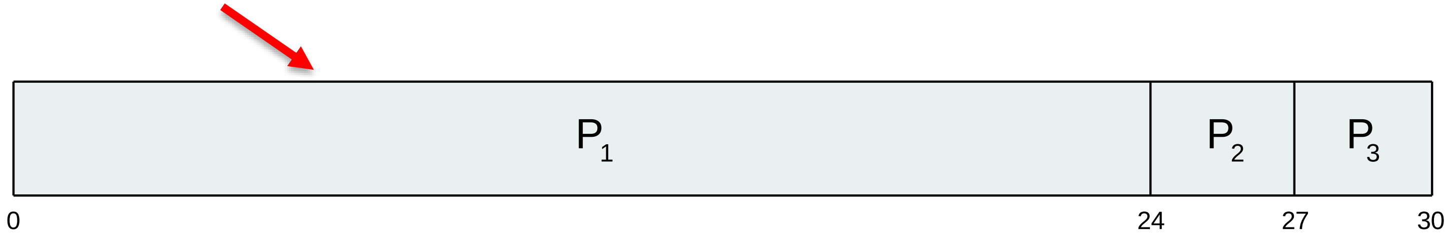




First- Come, First-Served (FCFS) Scheduling
خوارزمية : القادم اولا يخدم اولا

المعالجة Process	Burst Time	arrival time
P_1	24	0
P_2	3	0
P_3	3	0

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The **Gantt Chart** for the schedule is:



- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: (0 + 24 + 27)/3 = 17
- Turnaround time = Waiting time + Burst time

  For $P_1= 0+24=24$, $P_2=24+3=27$, and $P_3=27+3=30$

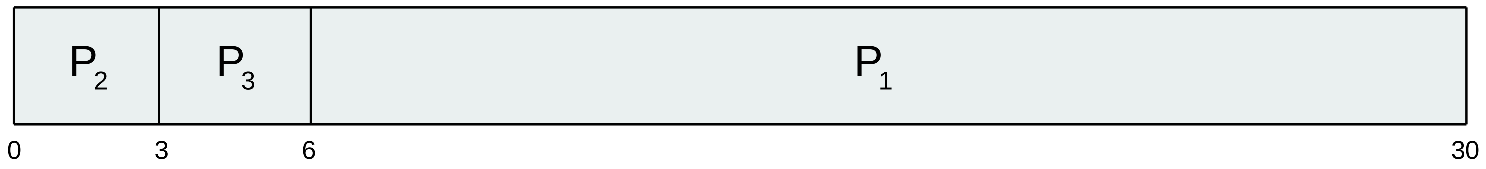
ملاحظة : زمن التنفيذ الكلي يجب ان يكون في مخطط كانت هو نفسه مجموع التنفيذ للمعالجات كلها

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- Turnaround time = Waiting time + Burst time

  For $P_1= 6+24=30$, $P_2=0+3=3$, and $P_3=3+3=6$

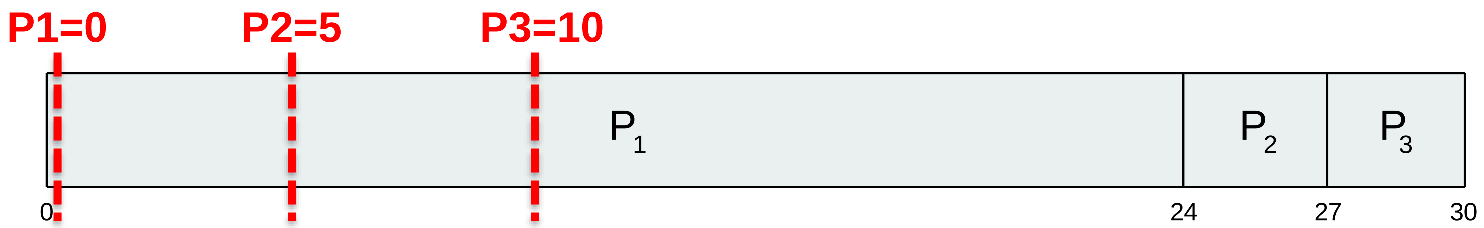
ملاحظة : عند البدأ بالعمليات
الاقل زمنا بالتنفيذ فان ذلك يوثر
على جودة تنفيذ كل العمليات
ومعدل الانتظار الكلي مقارنة
بالمثال السابق نفسه عندما بدأنا
بالمعالجات الاكثر زمنا للتنفيذ

ملاحظة : عند وصول العمليات الاكثر زمنا بالتنفيذ اولا للمعالج فان ذلك يؤدي الى زيادة معدل الانتظار الكلي
وعند وصول العمليات الاقل زمنا بالتنفيذ اولا للمعالج فان ذلك يؤدي الى تقليل معدل الانتظار الكلي

FCFS Scheduling (Cont.)

Process	Burst Time	Arrival Time
P_1	24	0
P_2	3	5
P_3	3	10

- The processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



- Waiting time for $P_1 = 0\text{-}\mathbf{0}=0$; $P_2 = 24\text{-}\mathbf{5}=19$; $P_3 = 27\text{-}\mathbf{10}=17$
- Average waiting time: (0 + 19 + 17)/3 = 12
- Turnaround time = Waiting time + Burst time

  For P_1= 0+24=24, P_2=19+3=22, and P_3=17+3=20

Shortest-Job-First (SJF) Scheduling

خوارزمية : المعالجات القصيرة الزمن اولا

(هذا ما نبحث عنه حسب الامثلة السابقة)

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

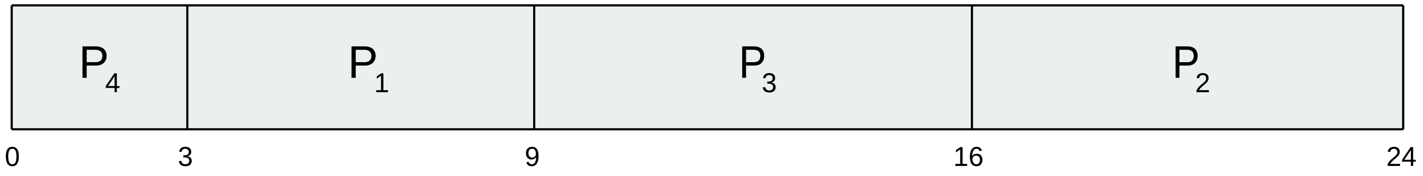
Example of SJF

	Process	Burst Time
الثانية	P_1	6
الرابعة	P_2	8
الثالثة	P_3	7
الاولى	P_4	3

على افتراض ان زمن الوصول لكل المعالجات الاربعة هو 0

- SJF scheduling chart

سيتم جدولة المعالجات على المعالج ابتداءا من العملية الاقصر فالاقصر وهكذا تترتب في مخطط كانت



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

Example1 of Non-Preemptive SJF

- Consider the following five processes each having its own unique burst time and arrival time.

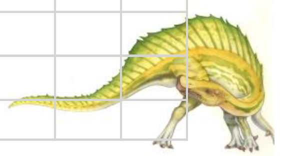
Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

Solution

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
p4	1	2	3																					
p3		1	2	3	4	5	6	7	8															
p1			1	2	3	4	5	6																
p5					1	2	3	4																
p2						1	2																	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	p4	p4																					
p4																								
p3		1	2	3	4	5	6	7	8															
p1			1	2	3	4	5	6																
p5					1	2	3	4			not come till now at=3													
p2						1	2				not come till now at=3													

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	p4	p4	p1																				
p4																								
p3		1	2	3	4	5	6	7	8															
p1			1	2	3	4	5	6																
p5					1	2	3	4																
p2						1	2				not come till now at=3													

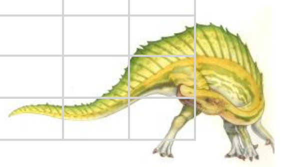


Solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4	p1	p1	p1	p1	p1	p1															
p4																									
p3			1	2	3	4	5	6	7	8															
p1				1	2	3	4	5	6																
p5						1	2	3	4																
p2							1	2																	

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4	p1	p1	p1	p1	p1	p1															
p4																									
p3			1	2	3	4	5	6	7	8															
p1																									
p5						1	2	3	4																
p2							1	2																	

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4	p1	p1	p1	p1	p1	p1															
p4																									
p3			1	2	3	4	5	6	7	8															
p1																									
p5						1	2	3	4																
p2							1	2																	

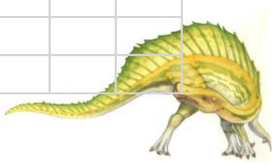


Solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4	p1	p1	p1	p1	p1	p1	p2	p2													
p4																									
p3			1	2	3	4	5	6	7	8															
p1																									
p5						1	2	3	4																
p2																									

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4	p1	p1	p1	p1	p1	p1	p2	p2													
p4																									
p3			1	2	3	4	5	6	7	8															
p1																									
p5						1	2	3	4																
p2																									

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4	p1	p1	p1	p1	p1	p1	p2	p2	p5	p5	p5	p5									
p4																									
p3			1	2	3	4	5	6	7	8															
p1																									
p5																									
p2																									



Solution

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	**p4**	**p4**	**p1**	**p1**	**p1**	**p1**	**p1**	**p1**	**p2**	**p2**	**p5**	**p5**	**p5**	**p5**									
p4																								
p3		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**															
p1																								
p5																								
p2																								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	**p4**	**p4**	**p1**	**p1**	**p1**	**p1**	**p1**	**p1**	**p2**	**p2**	**p5**	**p5**	**p5**	**p5**	**p3**	**p3**	**p3**	**p3**	**p3**	**p3**	**p3**	**p3**	
p4																								
p3																								
p1																								
p5																								
p2																								

Example2 of Non-Preemptive SJF

- Consider the following five processes each having its own unique burst time and arrival time.

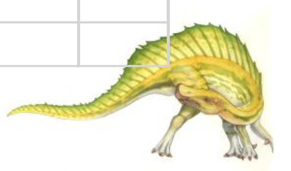
Process Queue	Burst time	Arrival time
P0	7	0
P1	4	2
P2	1	4
P3	4	5

Solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p0		1	2	3	4	5	6	7										
p1				1	2	3	4											
p2						1												
p3							1	2	3	4								

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		p0	p0	p0	p0	p0	p0	p0										
p0		1	2	3	4	5	6	7										
p1				1	2	3	4											
p2						1												
p3							1	2	3	4								

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		p0	p0	p0	p0	p0	p0	p0										
p0																		
p1				1	2	3	4											
p2						1												
p3							1	2	3	4								



Solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		p0	p0	p0	p0	p0	p0	p0	p2									
p0																		
p1				1	2	3	4											
p2																		
p3							1	2	3	4								

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		p0	p0	p0	p0	p0	p0	p0	p2									
p0																		
p1				1	2	3	4											
p2																		
p3							1	2	3	4								

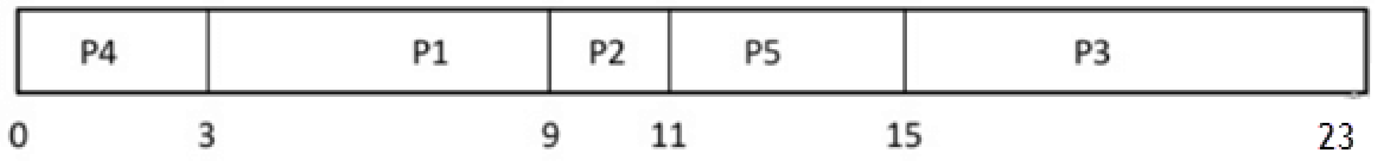
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		p0	p0	p0	p0	p0	p0	p0	p2	p1	p1	p1	p1					
p0																		
p1																		
p2																		
p3							1	2	3	4								

Solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		p0	**p0**	**p0**	**p0**	**p0**	**p0**	**p0**	**p2**	**p1**	**p1**	**p1**	**p1**	**p3**	**p3**	**p3**	**p3**	
p0																		
p1																		
p2																		
p3																		

Solution

- Gantt chat



- Let's calculate the average waiting time for above example.

- Wait time
- P4= 0-0=0
- P1= 3-2=1
- P2= 9-5=4
- P5= 11-4=7
- P3= 15-1=14
- Average Waiting Time= 0+1+4+7+14/5 = 26/5 = 5.2

Example of Preemptive SJF

- Preemptive SJF
- In Preemptive SJF Scheduling, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.
- Consider the following five process:

Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

Solution

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
p4		1	2	3																					
p3			1	2	3	4	5	6	7	8															
p1				1	2	3	4	5	6																
p5						1	2	3	4																
p2							1	2																	

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4																					
p4		1	2	3																					
p3			1	2	3	4	5	6	7	8															
p1				1	2	3	4	5	6																
p5						1	2	3	4																
p2							1	2																	

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		p4	p4	p4	p1																				
p4																									
p3			1	2	3	4	5	6	7	8															
p1				1	2	3	4	5	6																
p5						1	2	3	4																
p2							1	2																	

Solution

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	p4	p4	p1	p5																			
p4																								
p3		1	2	3	4	5	6	7	8															
p1				2	3	4	5	6																
p5					1	2	3	4																
p2						1	2																	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	p4	p4	p1	p5	p2	p2																	
p4																								
p3		1	2	3	4	5	6	7	8															
p1				2	3	4	5	6																
p5						2	3	4																
p2						1	2																	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	p4	p4	p1	p5	p2	p2	p5	p5	p5														
p4																								
p3		1	2	3	4	5	6	7	8															
p1				2	3	4	5	6																
p5						2	3	4																
p2																								

Solution

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	**p4**	**p4**	**p1**	**p5**	**p2**	**p2**	**p5**	**p5**	**p5**	**p1**	**p1**	**p1**	**p1**	**p1**									
p4																								
p3		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**															
p1				**2**	**3**	**4**	**5**	**6**																
p5																								
p2																								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	**p4**	**p4**	**p1**	**p5**	**p2**	**p2**	**p5**	**p5**	**p5**	**p1**	**p1**	**p1**	**p1**	**p1**									
p4																								
p3		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**															
p1																								
p5																								
p2																								

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	p4	**p4**	**p4**	**p1**	**p5**	**p2**	**p2**	**p5**	**p5**	**p5**	**p1**	**p1**	**p1**	**p1**	**p1**	**p3**	**p3**	**p3**	**p3**	**p3**	**p3**	**p3**	**p3**	
p4																								
p3		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**															
p1																								
p5																								
p2																								

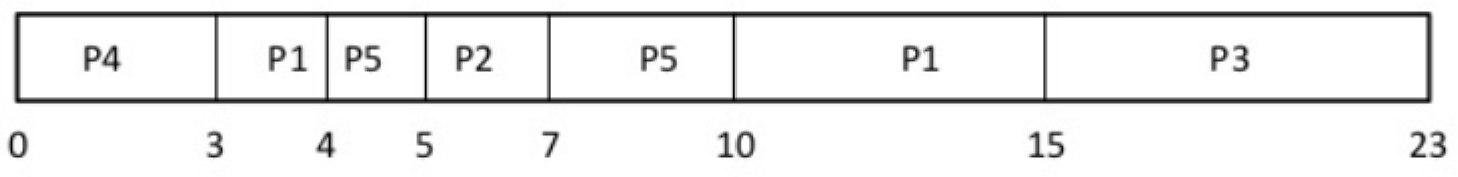
Example of Preemptive SJF

- Preemptive SJF
- In Preemptive SJF Scheduling, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.
- Consider the following five process:
- Process Queue / Burst time / Arrival time

Process Queue	Burst time	Arrival time
P1	**6**	**2**
P2	2	5
P3	8	1
P4	3	0
P5	**4**	**4**

Solution

- Gantt chat



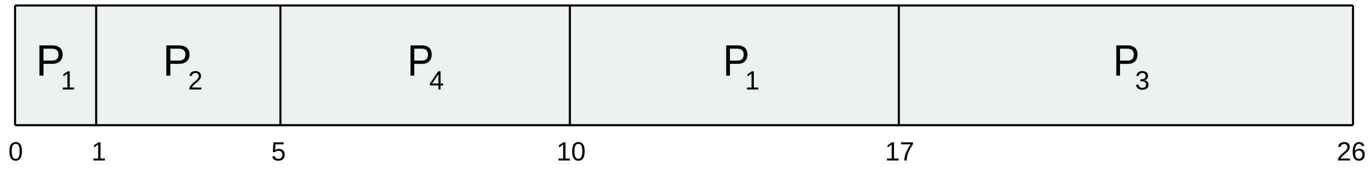
- Wait time
- P4= 0-0=0
- **P1= (3-2) + 6 =7**
- P2= 5-5 = 0
- **P5= 4-4+2 =2**
- P3= 15-1 = 14
- Average Waiting Time = 0+7+0+2+14/5 = 23/5 =4.6

Example of Preemptive Shortest-remaining-time-first SJF

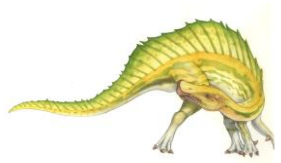
- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



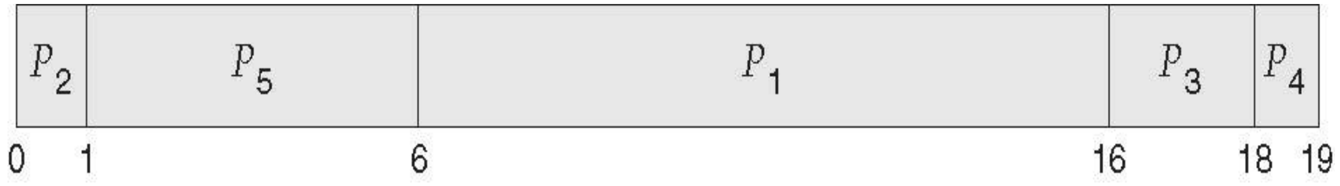
- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec



Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart

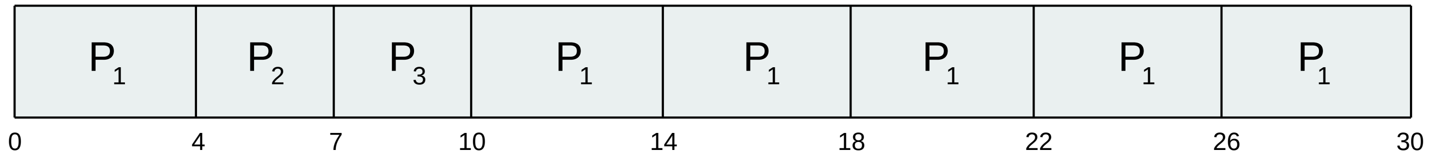


- Average waiting time = 8.2

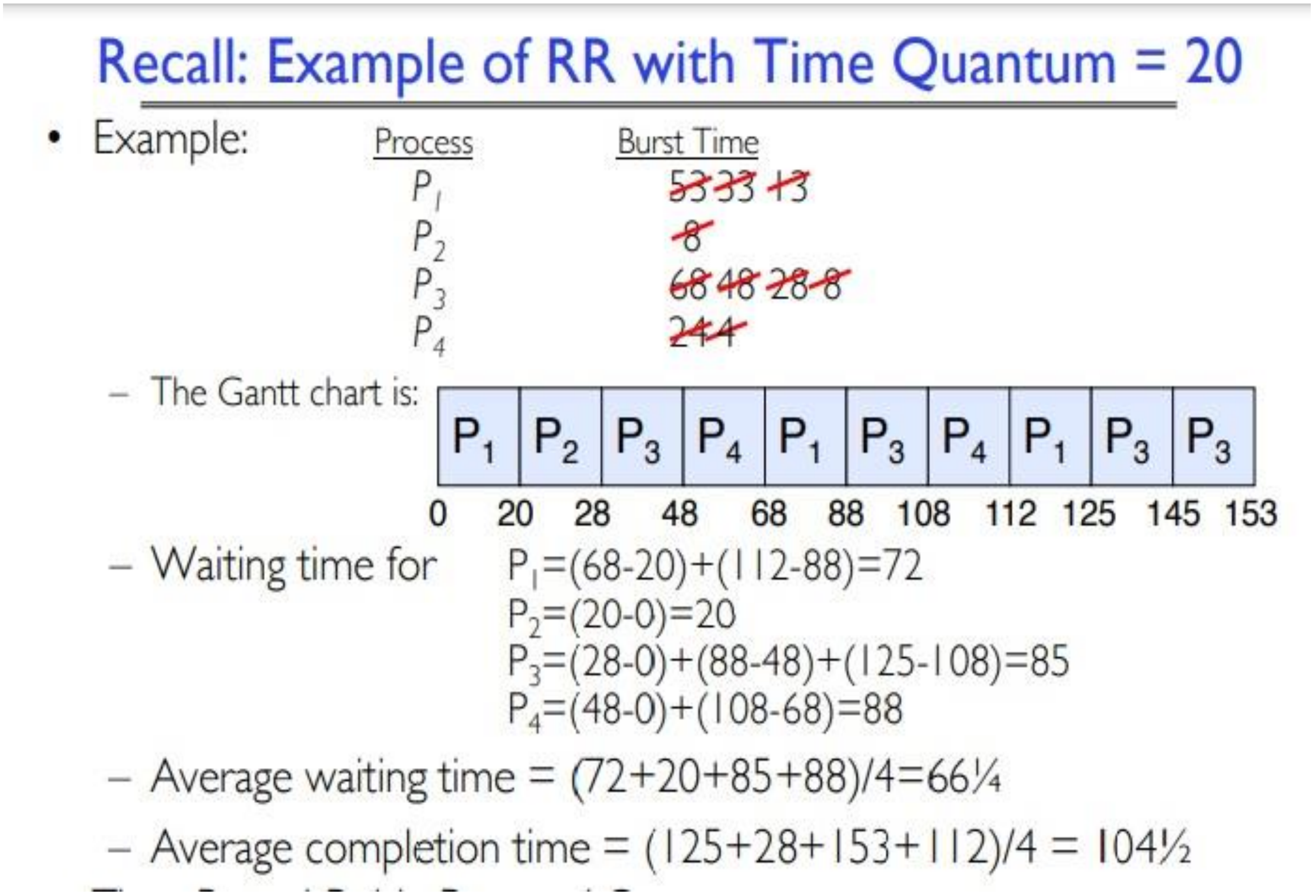
Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec



Recall: Example of RR with Time Quantum = 20

- Example:

Process	Burst Time
P_1	~~53~~ ~~33~~ ~~13~~
P_2	~~8~~
P_3	~~68~~ ~~48~~ ~~28~~ ~~8~~
P_4	~~24~~ ~~4~~

  – The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

  – Waiting time for $P_1=(68-20)+(112-88)=72$
  $P_2=(20-0)=20$
  $P_3=(28-0)+(88-48)+(125-108)=85$
  $P_4=(48-0)+(108-68)=88$

  – Average waiting time = (72+20+85+88)/4=66¼

  – Average completion time = (125+28+153+112)/4 = 104½