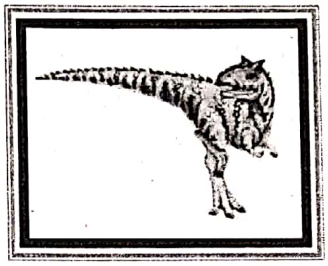
OPERATING SYSTEMS

ENG. ASMA ABEDELKARIM

DONE BY REEM MUIN

POWERUNIT

Chapter 1: Introduction



1

Chapter 1: Introduction

- What Operating Systems Do (1.1)
- Computer-System Organization (1.2)
- Computer-System Architecture (1.3)
- Operating-System Operations (1.4)
- Resource Management (1.5)
- Security and Protection (1.6)
- Virtualization (1.7)
- Kernel Data Structures (1.9)
- Computing Environments (1.10)
- Free/Libre and Open-Source Operating Systems (1.11)

2

Objectives

- Describe the general organization of a computer system and the role of interrupts
- Describe the components in a modern, multiprocessor computer system
- Illustrate the transition from user mode to kernel mode
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems

3

What Operating Systems Do?

4

* احنا ال user بنتعامل مع Applications هدول ال Apps ← running on hardware

What is an Operating System?

مش الأساس بس بدونها ما بنفع

عبارة عن Software

- 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

من ايش بتكون ال Computer System

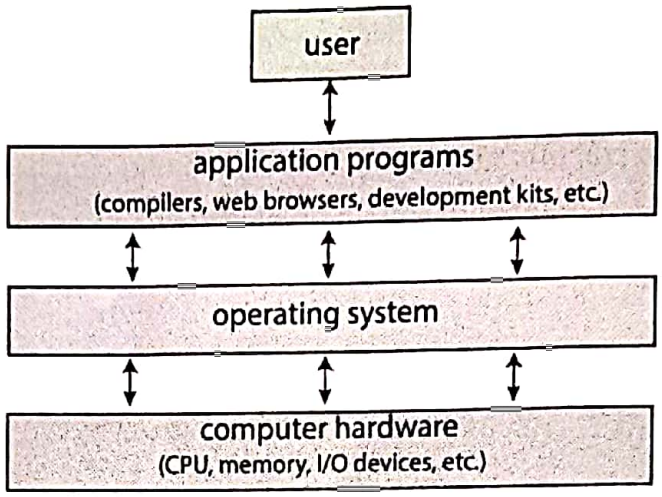
Computer System Structure

- 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

    البرامج الي بنشغل وبنعطي interface لل user انه يعمل شغلات متعددة على الجهاز.
  - Users
    - People, machines, other computers

    ما بعتبر جزء من المكونات بس بشكل عام بنحطو مع المكونات.

Abstract View of Components of Computer



* من أنواع الـ devices المتطلبات اللي بحتاجها من الـ OS تختلف حسب الجهاز لإنه الأهداف بتختلف.

What Operating Systems Do

- Depends on the point of view
- Users want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization** → مابهمني شو بحبس تحت
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
  - Operating system is a **resource allocator** and **control program** making efficient use of HW and managing execution of user programs
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Mobile devices like smartphones and tables are resource poor, optimized for usability and battery life
  - Mobile user interfaces such as touch screens, voice recognition
- Some computers have little or no user interface, such as embedded computers in devices and automobiles
  - Run primarily without user intervention

Defining Operating Systems

- Term OS covers many roles
  - Because of myriad designs and uses of OSes
  - Present in toasters through ships, spacecraft, game machines, TVs and industrial control systems
  - Born when fixed use computers for military became more general purpose and needed resource management and program control

منتي
ولد
Operating ال
System

9

Operating System Definition

- No universally accepted definition → ما إله تعريف بالزبط
- "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly الجزء من الـ os اللي بليما شغال، كإنه الـ core code للـ os (main code of os)
- "The one program running at all times on the computer" is the **kernel**, part of the operating system → ما بطفي إلا لو طفيت الجهاز.
- Everything else is either
  - A ***system program*** (ships with the operating system, but not part of the kernel), or → « جزء من الـ os بس ما بكون من أصل الـ kernal
  - An ***application program***, all programs not associated with the operating system → « الـ Applications اللي أنا بشغلها كـ user

مش جزء من الـ os

اشي بنفذ من الـ os بس مش جزء من الـ os. (مساندة وليست أساسية)

- Today's OSes for general purpose and mobile computing also include ***middleware*** – a set of software frameworks that provide additional services to application developers such as databases, multimedia, graphics

الهدف منها تقطيع خدمات إضافية كـ layer إضافية مش بالـ kernal.

10

Computer-System Organization

11

بتحكي مع الـ cpu لما بدنا نحكي مع الجهاز

* Controllers كل جهاز موجود جوا الـ Computer تبعي بكون الها اشي اسمه

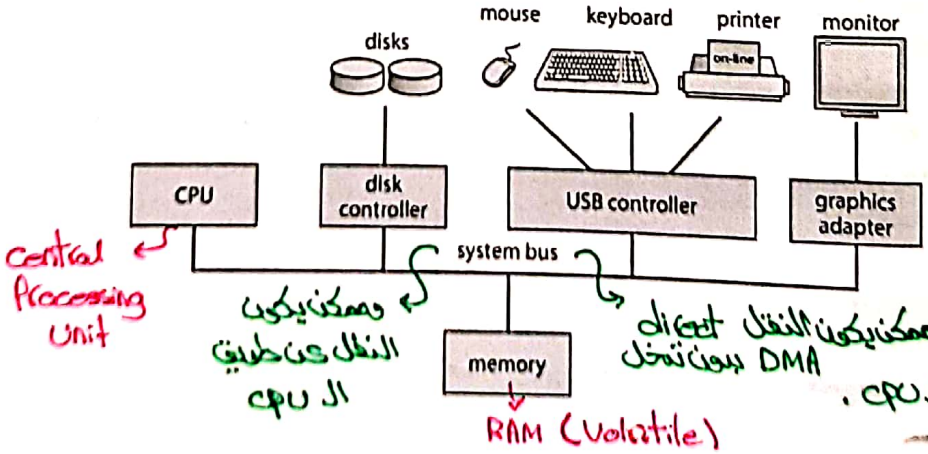
Computer System Organization

- 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

hardware / device driver → software

لازم يكون كشاني يقدر الـ Processor

يظل الـ controller

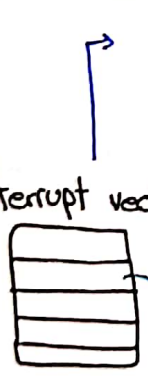


12

Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating system **device driver** to manage it
- 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**

13



Interrupts

14

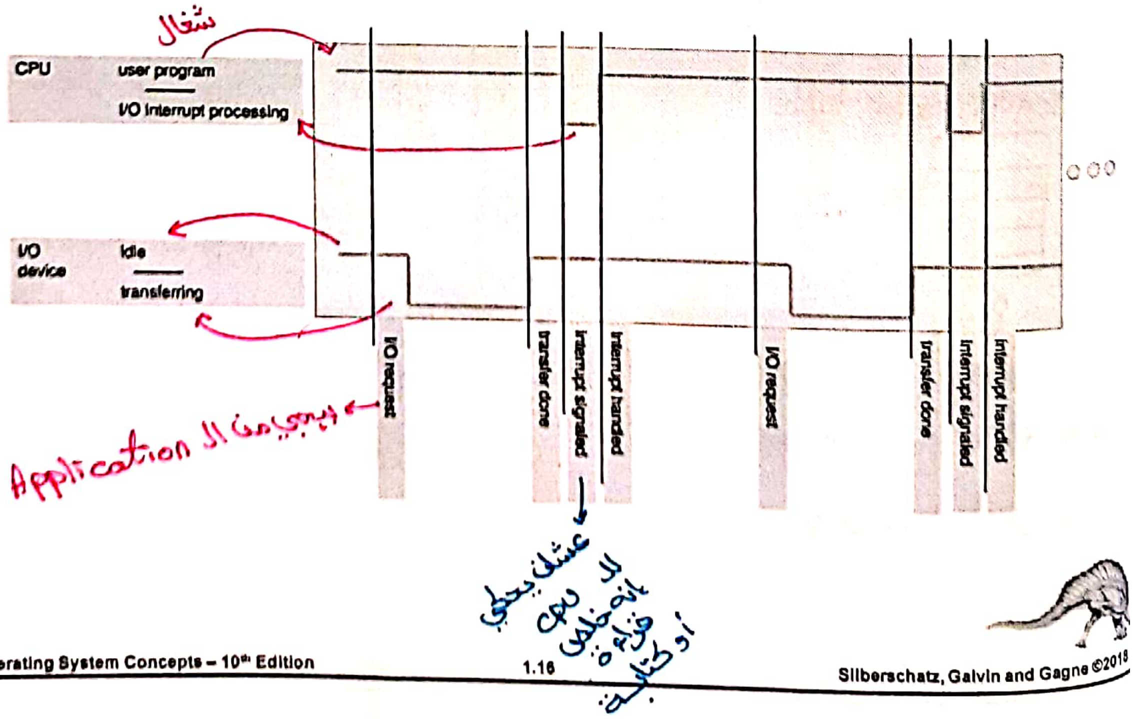
Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**
- **Interrupt Handling**
  - The operating system preserves the state of the CPU by storing the registers and the program counter → يخزن ال Next address to be executed
  - Determines which type of interrupt has occurred:
  - Separate segments of code determine what action should be taken for each type of interrupt

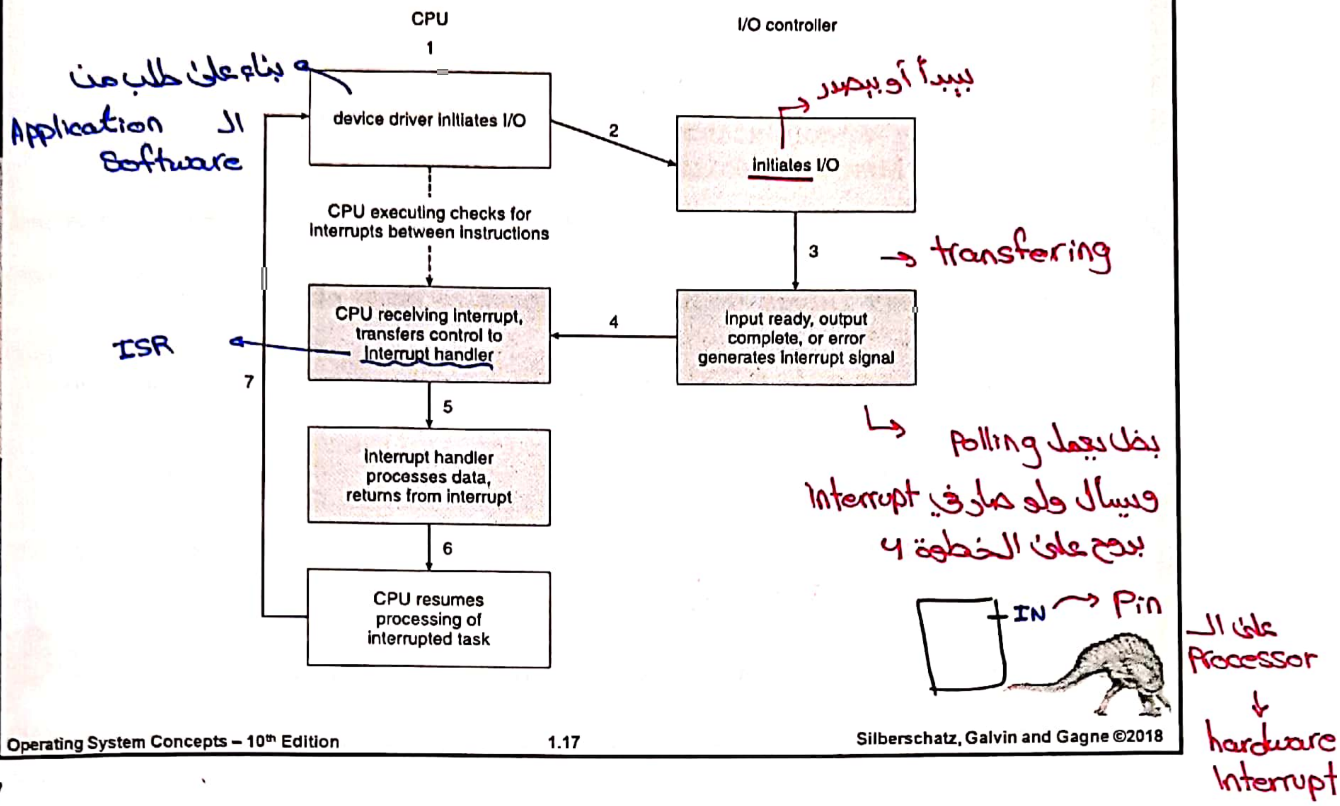
* مش كل أنواع ال I/O operation بتكون interrupt driven (رح نشوف هالحكي لقدام)

Interrupt Timeline

لـ بوضح كيف بتصير عملية التواصل بين ال Controller تبع ال device وبين ال OS عن طريق ال interrupt.



Interrupt-drive I/O Cycle



* أي برنامج بده ينفذ لازم يكون محمّل عال Main memory.

* أول برنامج نشغله يعني أول ما أفتح الجهاز وأول اشي بتحمل عال Main Memory هو ال Boot strap program.

* Boot strap program هو عبارة عن برنامج بكون حافظ وعارف وين مكان ال os بال harddisk وهو المسؤول بعمله loading على ال Main mem ليبلش يشتغل ال os.

Storage Structure

- Bootstrap program is loaded at power-up or reboot → لأنه ال access عليها أسرع.
  - Typically stored in ROM or EPROM, generally known as firmware.
  - Initializes all aspects of system.
  - Loads operating system kernal and starts execution.

مثال ثاني هو ال BIOS
والمثال الأول هو ال Boot Strap

Storage Structure

- **Main memory** – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile** → متطايرة
  - Typically **random-access memory** in the form of **Dynamic Random-access Memory (DRAM)**
- **Secondary storage** – extension of main memory that provides large **nonvolatile** storage capacity
  - **Hard Disk Drives** (HDD) – 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
  - **Non-volatile memory** (NVM) devices– faster than hard disks, nonvolatile → غير متطايرة
    - Various technologies
    - Becoming more popular as capacity and performance increases, price drops

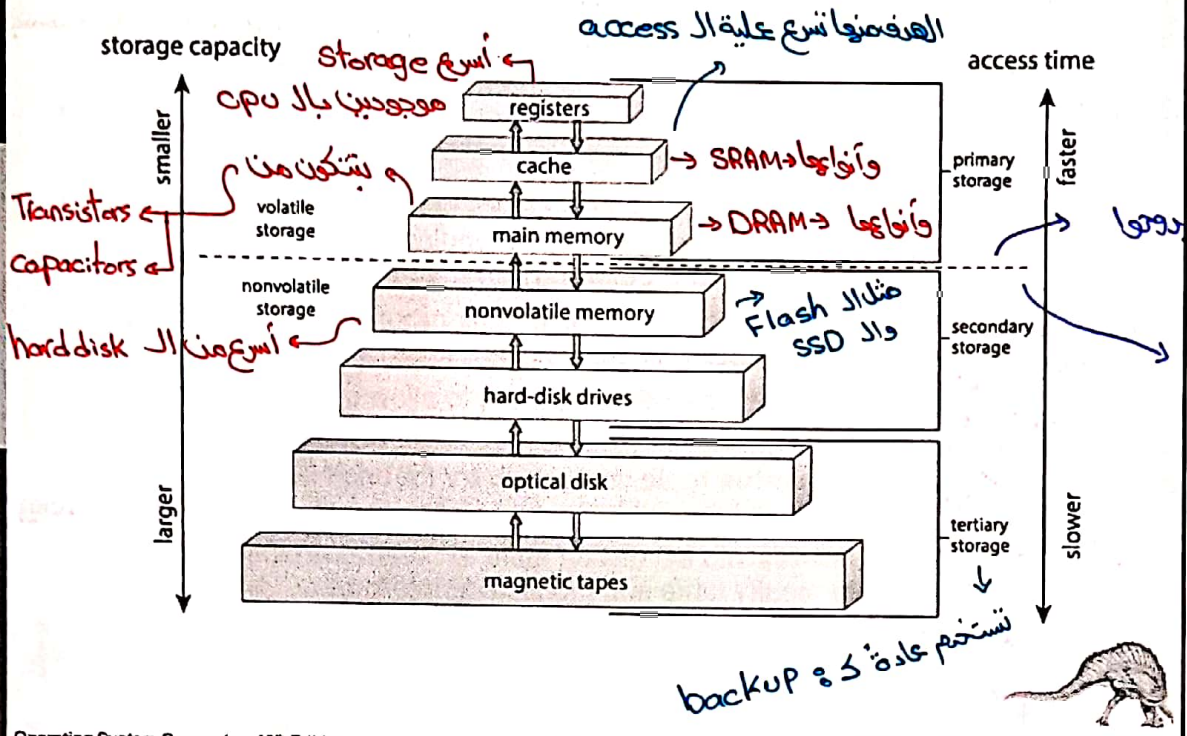
Storage Hierarchy

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

* كلما نزلنا بالهرم بختلف ال Speed
* Volatility وال cost وال
- **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

* مش ضروري ال computing device يكون فيه كل هدول *

Storage-Device Hierarchy



22

I/O Structure

23

I/O Structure

لما أنا كـ User أطلب I/O مابقدر أكمل تنفيذ الـ inst تاعوتي لحد ما يوصلني الرد

- Two methods for handling I/O م   بنسموه blocking و non-blocking م

Synch I/O ← ① After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing

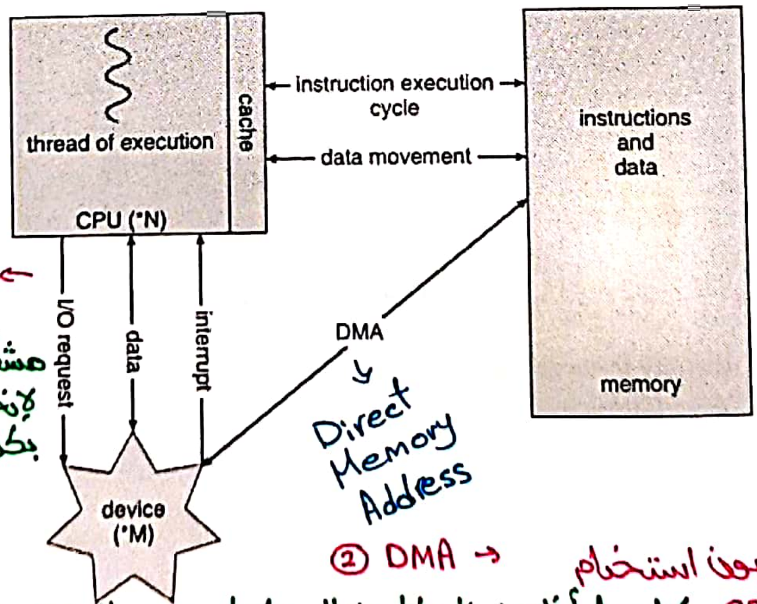
Asynch I/O ← ② After I/O starts, control returns to user program without waiting for I/O completion
  - **System call** – request to the OS to allow user to wait for I/O completion
  - **Device-status table** contains entry for each I/O device indicating its type, address, and state → Process بيحفظ بيانات كل
  - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt

بعد ما أعمل الـ I/O request ما بستنى بينما بيرجع على أنا بكمل الـ execution ولما يخلص بيرجعلي interrupt ← بقلي إنه خلص

24

Two types of Memory transfer between devices and Memory   الهدف منها نعرف انه في عنا 2

How a Modern Computer Works



① النقل عن طريق الـ cpu. مشكلته بياخذ وقت لإنه الـ bytes بكل نقلة بكون محدود.

DMA ↓ Direct Memory Address

② DMA → النقل بدون استخدام الـ cpu. كل ما أخلص block من الـ data بيعت للـ cpu إني خلصت عن طريق الـ interrupt مش كل byte.

A von Neumann architecture

25

Direct Memory Access Structure

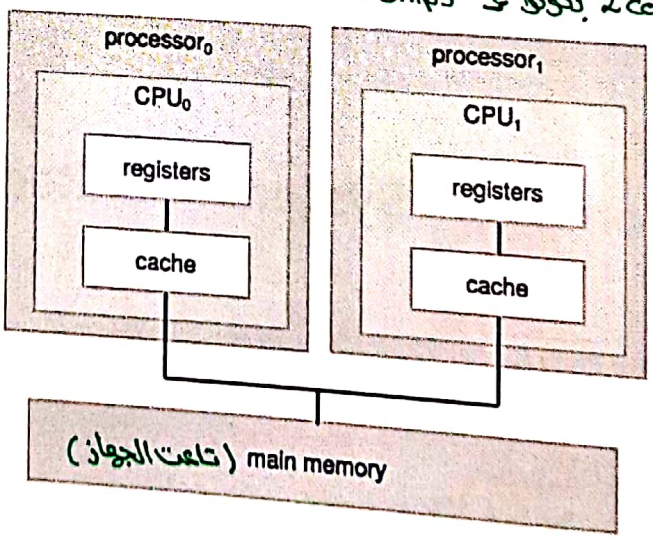
- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

Computer-System Architecture

Computer-System Architecture

- Most systems use a single general-purpose processor
  - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems, tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput** → Parallel systems
    2. **Economy of scale** →
    3. **Increased reliability** – graceful degradation or fault tolerance
  - Two types:
    1. **Asymmetric Multiprocessing** – each processor is assigned a specie task.
    2. **Symmetric Multiprocessing** – each processor performs all tasks

Symmetric Multiprocessing Architecture

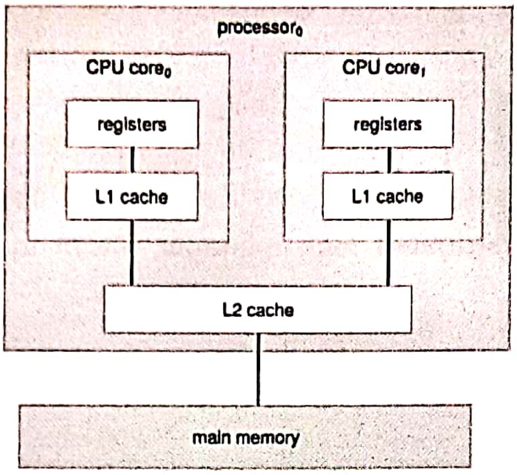


Dual-Core Design (Asymmetric)

Symmetric ←
Asymmetric ← يعني ممكن يكون داخل Several هيكل الجهاز سواء systems

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

هدول السطرين ما بحكوا عن الـ Dual-core



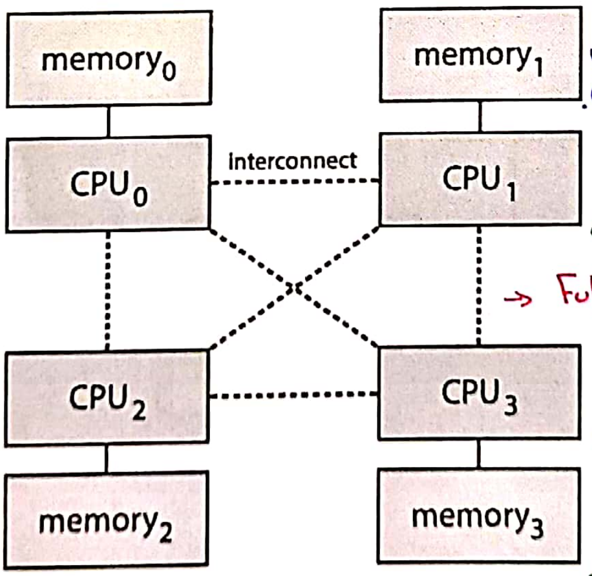
→ * مش ضروري الثنين يكونوا بنفذوا نفس أنواع الـ Task

* الـ 2 cores بكونوا بنفس الـ chip بس الـ chip بتكون multicore

هذا الـ design ما بكون بالـ Normal systems، موجود بـ Servers أو clusters ...

Non-Uniform Memory Access System → NUMA

← ممكن يكون devices منفصلين وممكن لو كان كثير advanced يكونوا بنفس الـ board.



← أكثر من cpu وكل cpu إلها main Mem خاص فيها (local Mem)

← كل ما يحتاج data من الـ local Mem تبعه بكون بينهم direct access فبكون أسرع شي.

→ Fully connected

← إذا احتاج الـ cpu data من Mem لـ cpu ثانية بتواصل مع هذا الـ Processor وبعطيه مسج إنه بدو منه data معينة، وهون طريق أطول ومش سريع.

- مش كل الـ Mem access، إلها نفس الـ delay.

مجموعة من ال devices بكونوا clustered together to serve users as cluster.

Clustered Systems

- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a **storage-area network (SAN)**
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot-standby mode ← بكون جزء شغال وجزء براقب التاني، إذا واحد منهم فشل بقوم واحد من المراقبين بداله.
    - **Symmetric clustering** has multiple nodes running applications, monitoring each other → كلها بتشتغل وبتراقب سوا وإذا حدا منا فشل واحد تاني بيكمل شغله
  - Some clusters are for **high-performance computing (HPC)**
    - Applications must be written to use **parallelization**
  - Some have **distributed lock manager (DLM)** to avoid conflicting operations

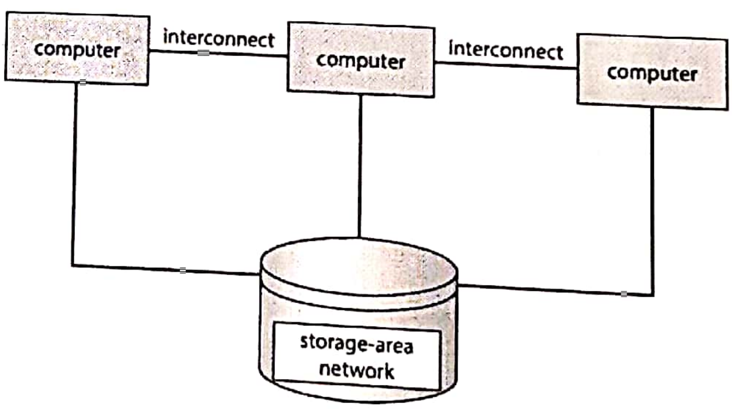
— بعض ال Clustered System بكون عليها CPUs و GPUs وانت بتستخدم remotely من جهازك، يعني من لابتوبك بتشبك على cluster وبتنفذ الإيشي بداله من لابتوبك.

— Clusters: تجمع من ال high-performance computing systems.

(multiple users على System ولكل user ال إمكانيات مختلفة)

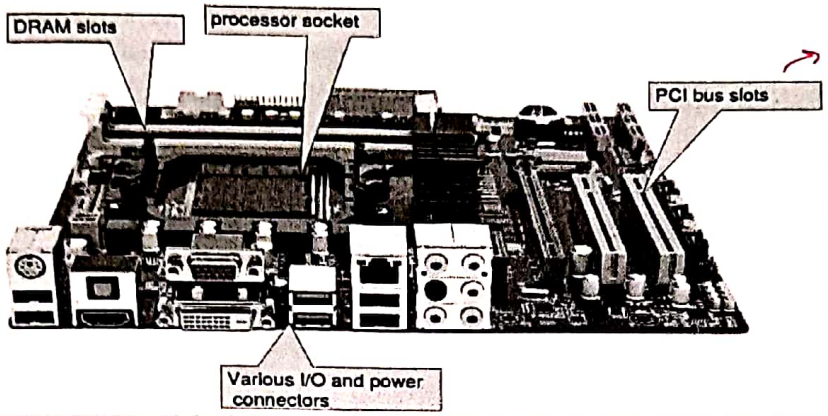
Clustered Systems

Clusters ←



PC Motherboard

Consider the desktop PC motherboard with a processor socket shown below:



system bus

This board is a fully-functioning computer, once its slots are populated. It consists of a processor socket containing a CPU, DRAM sockets, PCIe bus slots, and I/O connectors of various types. Even the lowest-cost general-purpose CPU contains multiple cores. Some motherboards contain multiple processor sockets. More advanced computers allow more than one system board, creating NUMA systems.

34

Operating-System Operations

35

Operating-System Operations

- Bootstrap program – simple code to initialize the system, load the kernel
  - It initializes all aspects of the system, from CPU registers to device controllers to memory contents
  - It must locate the operating-system kernel and load it into memory
- Starts **system daemons**: services provided outside of the kernel by system programs and are loaded into memory at boot time
  - On Linux, the first system program is "**systemd**," and it starts many other daemons
- Once this phase is complete, the system is fully booted, and the system waits for some event to occur
- Kernel **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 – **system call**
    - Other process problems include infinite loop, processes modifying each other or the operating system

Multiprogramming (Batch system)

- Single user cannot always keep CPU and I/O devices busy
- 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 job has to wait (for I/O for example), OS switches to another job
- In a multiprogrammed system, a program in execution is termed a **process**

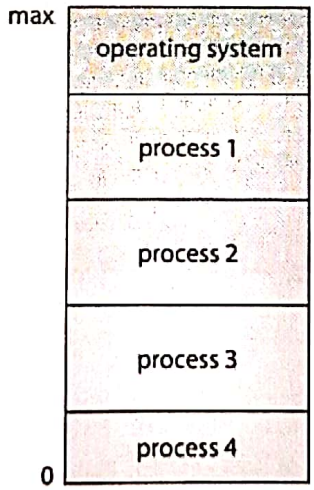
Multitasking (Timesharing)

- A logical extension of Batch systems– the 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 ← لو الـ code ما كفّا بالـ Mem

* من سرعة التنفيذ والتبديل بيتوهملك إنه كلهم شغالين بنفس الوقت.

* بتوهم إنه البرنامج كله نازل بالـ Mem بس هو ما بكون كله نازل.

Memory Layout for Multiprogrammed System



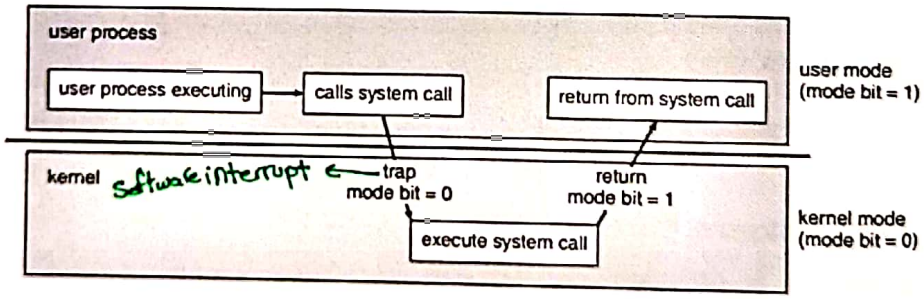
* الفايدة كمثال ما أخلي أي User يقدر يتعدى على شغل الـ kernal.
* لازم بمرحلة ما أقدر أفرق الـ process تكون تابعة لـ kernal و لا لـ User.

Dual-mode Operation

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode** ← Two types of processes → kernal, user
- **Mode bit** provided by hardware
  - Provides ability to distinguish when system is running user code or kernel code.
  - When a user is running ⇨ mode bit is "user" mode bit = 1
  - When kernel code is executing ⇨ mode bit is "kernel" mode bit = 0
- How do we guarantee that user does not explicitly set the mode bit to "kernel"? → . User الـ عليها access الـ أمثلة ما
  - System call changes mode to kernel, return from call resets it to user
- Some instructions designated as **privileged**, only executable in kernel mode

* أنا كـ User لو اختبت اشي من الـ System حطلبه من الـ OS.

Transition from User to Kernel Mode



الهدف إنه ؟
يتأكد إنه مافي Process معينة بتاخذ وقت أكثر من اللازم.

Timer

- We must ensure that the operating system maintains control over the CPU.
  - We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system.
- Timer to prevent infinite loop (or 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

Resource Management

Process Management

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

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

* الـ CPU سريعة البطء يدوبل لما تحتاج اشي من برّا (مثلاً لما تحتاج من الـ Mem)

Memory Management

- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory
- Memory management determines what is in memory and when → بتأكد شو الموجود بالـ Mem وشو لا.
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

بتعامل مع أي اشي داخل أجهزتنا كأنه files.

File-system Management

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

مثل الوينوز صار بيعمل one drive الك تلقائي backup

disk management

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a "long" period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Mounting and unmounting
  - Free-space management
  - Storage allocation
  - Disk scheduling
  - Partitioning
  - Protection

إنه الـ OS نفسه بعمل caching لبعض الشغلات اللي بنحتاجها.

Software وال hardware موجود بالـ

فكرتها إنك تخزن الأشياء اللي بنحتاجها هلأ أو بالمستقبل بأقرب مكان إلي

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

Characteristics of Various Types of Storage

أقرب اشي ← | ليكبر →

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 أسرع شي	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

يزيد من الشمال لليمين

يقل من الشمال لليمين

Movement between levels of storage hierarchy can be explicit or implicit

قديه بقدر أمشّي MB بالثانية

كل واحد للي بعده backed

50

* احنا كـ users ما بنشوف كل الاختلافات بين الـ I/O لإنه الـ OS هو اللي مسؤول يتعامل معهم باختلافاتهم عن طريق الـ drivers تبعتهم.

I/O Subsystem

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

51

Security and Protection

Protection and Security

ومين لأ.

وبخلي الsystem محمي بإنه من الأصل يكون عندي roles تحكم مين اليمسموحله يعمل اشي ← الوقاية

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS

ولو صار الattack كيف اتعامل معاه.

- **Security** – defense of the system against internal and external attacks ← لو صار كيف اتعامل معاه
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

تمييز أو تصنيف دم

- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file ← كل جروب إلهم نفس الصلاحيات
  - **Privilege escalation** allows user to change to effective ID with more rights

كيف بقدر أنا كادمن أزيد الـ Privilege لشيء معين

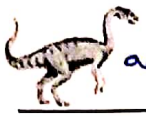
Privilege ← حصانة

Virtualization ← المحاكاة

* الفكرة منه هو أحاكي شيء معين بغير حقيقي.

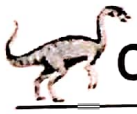
Virtualization

- Allows operating systems to run applications within other OSes
  - Vast and growing industry ← محاكاة
- **Emulation** used when source CPU type different from target type (i.e. PowerPC to Intel x86) → لازم يصير يترجم من الـ CPU هيك لـ CPU هاد.
  - Generally slowest method ← الـ CPU تاع IBM
  - When computer language not compiled to native code – **Interpretation** → ترجمة
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled → مثلاً أشتغل لينكس جوا ويندوز.
  - Consider VMware running WinXP guests, each running applications, all on native WinXP **host** OS
  - **VMM** (virtual machine Manager) provides virtualization services

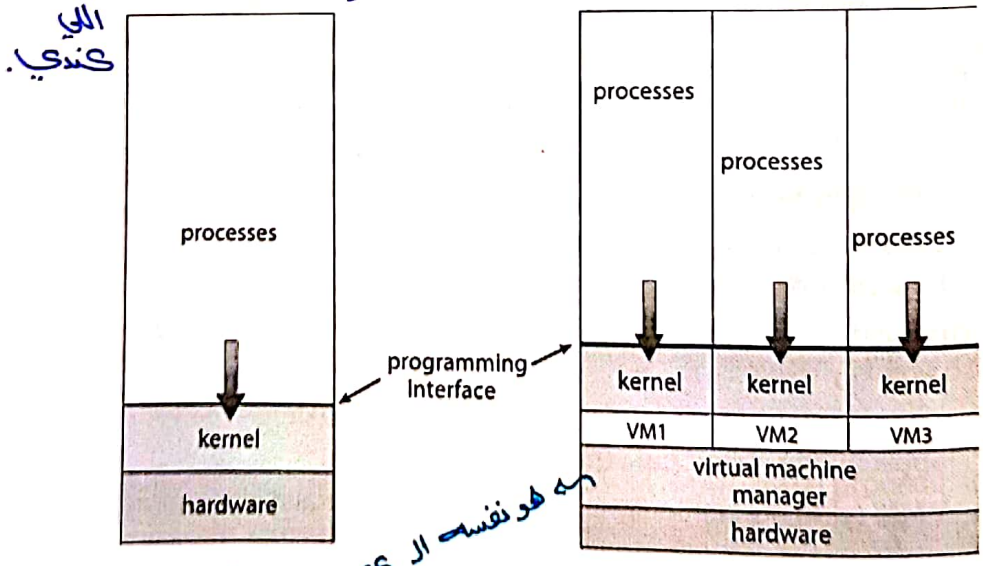


Virtualization (cont.)

- Use cases involve laptops and desktops running multiple OSes for exploration or compatibility
  - Apple laptop running Mac OS X host, Windows as a guest
  - Developing apps for multiple OSes without having multiple systems
  - Quality assurance testing applications without having multiple systems
  - Executing and managing compute environments within data centers
- VMM can run natively, in which case they are also the host
  - There is no general-purpose host then (VMware ESX and Citrix XenServer)



Computing Environments - Virtualization



Kernel Data Structure

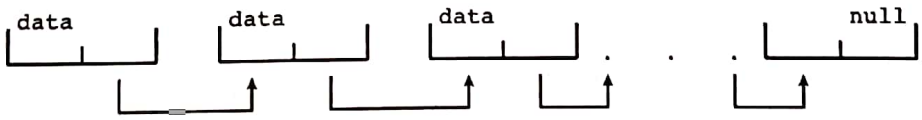
الهدف منه إنه نعرف إنه حتى الـ activities اللي بيعملوا الـ os
بتستعمل هاي الـ Data Structures.

last in first out : Stack ك لما تشتغل linked list *

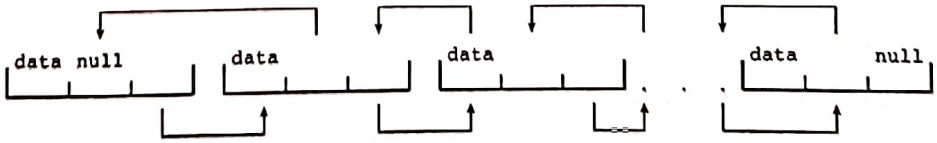
first in first out : Queue ك لما تشتغل linked list *

Kernel Data Structures

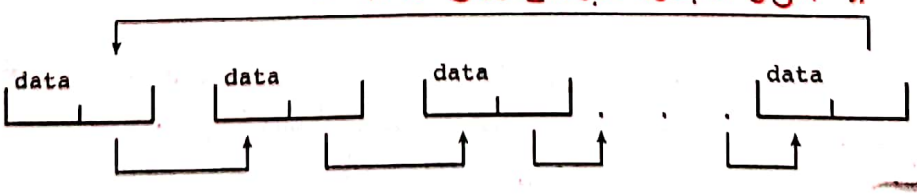
- Many similar to standard programming data structures
- ***Singly linked list*** → « كل item مربوطة بالأخرى »



- ***Doubly linked list*** → « كل واحد مربوط باللي قبله واللي بعده »

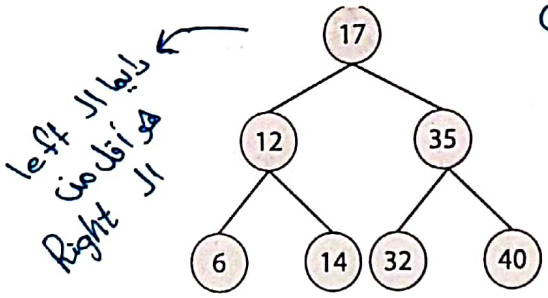


- ***Circular linked list*** → « آخر واحد بكون مربوط مع أول واحد »



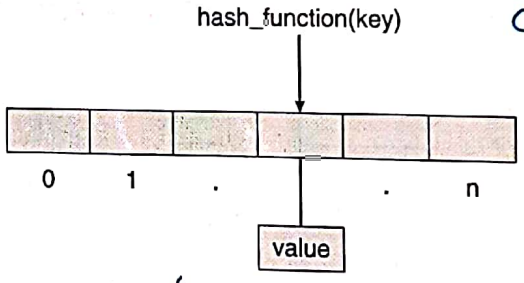
Kernel Data Structures

- **Binary search tree**
  left <= right
  - Search performance is *O(n)*
  - **Balanced binary search tree is** *O(lg n)*



Kernel Data Structures

- **Hash function** can create a **hash map**



- **Bitmap** – string of *n* binary digits representing the status of *n* items
- Linux data structures defined in ***include*** files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

Computer System Environments

Computing Environments

- Traditional
- Mobile
- Client Server
- Peer-to-Peer
- Cloud computing
- Real-time Embedded

Traditional

- Stand-alone general-purpose machines
- But blurred as most systems interconnect with others (i.e., the Internet)
- **Portals** provide web access to internal systems
- **Network computers (thin clients)** are like Web terminals
- Mobile computers interconnect via **wireless networks**
- Networking becoming ubiquitous – even home systems use **firewalls** to protect home computers from Internet attacks

4

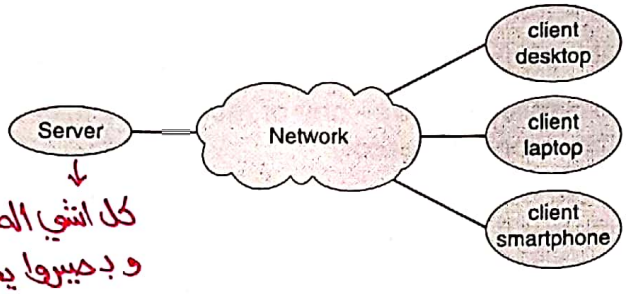
Mobile

- Handheld smartphones, tablets, etc.
- What is the functional difference between them and a "traditional" laptop?
- Extra feature – more OS features (GPS, gyroscope)
- Allows new types of apps like ***augmented reality***
- Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are **Apple iOS** and **Google Android**

5

Client Server

- Client-Server Computing
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server system** provides an interface to client to request services (i.e., database)
    - **File-server system** provides interface for clients to store and retrieve files



كل اشي اله بكون أعلى
و بحيروا يطلبوا منه
معينة Service

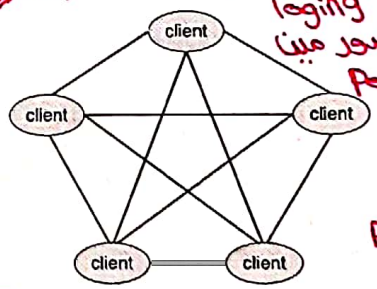
66

Peer-to-Peer (كلنا سواسية)

« Server ما في حد »

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via ***discovery protocol***
  - Examples include Napster and Gnutella, **Voice over IP (VoIP)** such as Skype

مشان هو اللي بقدم الخدمة، بعمل loging و بعرف مين Peers



فيه من ال Peers عنده الخدمة اللي بدو ياها فبصير التواصل بين ال Peers

بستخدم هاي الطريقة

بستخدم هاي الطريقة

من أول ما في حد بعمل loging عنده لما اطلب ، لما يكون بيعرف خدمة بدت للكل واللي عنده الخدمة بردلي و بيعطيني ياها .

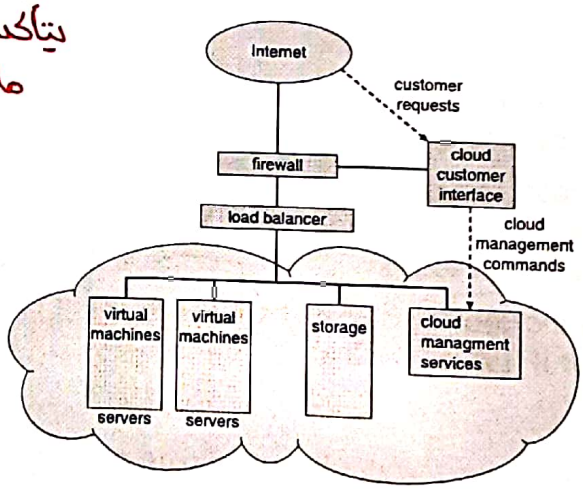
67

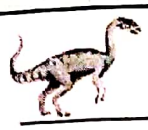
Cloud Computing

- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon EC2 has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage
- Many types
  - **Public cloud** – available via Internet to anyone willing to pay
  - **Private cloud** – run by a company for the company's own use
  - **Hybrid cloud** – includes both public and private cloud components
  - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
  - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
  - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)

Cloud Computing (cont.)

- Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
  - Internet connectivity requires security like firewalls
  - Load balancers spread traffic across multiple applications





Real-Time Embedded Systems

- Real-time embedded systems most prevalent form of computers
  - Vary considerable, special purpose, limited purpose OS, **real-time OS**
  - Use expanding
- Many other special computing environments as well
  - Some have OSes, some perform tasks without an OS
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met

بكفظ بستخدموا ASIC

Free and Open Source Operating Systems

* أول ما طلعوا الـ os كانوا الي open source و free الي بيه يساهم ويحسن بقدر يقدر.

Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source** and **proprietary**
- Counter to the **copy protection** and **Digital Rights Management** (**DRM**) movement ← أي حد بيعمل عالـ open source الـ الحرية تاخده وتبيعه حق
- Started by **Free Software Foundation** (**FSF**), which has "copyleft" **GNU Public License** (**GPL**) ← منظمة بتقولي انه الـ software مايكونش مسكر ويكون open source
  - Free software and open-source software are two different ideas championed by different groups of people
    - **https://www.gnu.org/philosophy/open-source-misses-the-point.en.html**
- Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**), and many more ← open source
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - http://www.virtualbox.com)
  - Use to run guest operating systems for exploration

* لما حد في شركات وتنافس حاربت الـ os أغلبها closed وهذا مالها علاقة بأنه مدفوع أو لأ.

* مثال الـ os مدفوع و closed ← الويندوز.
* الـ open/closed Source ما إله علاقة بجل هو مدفوع أو لأ.

Self Reading ←

The Study of Operating Systems

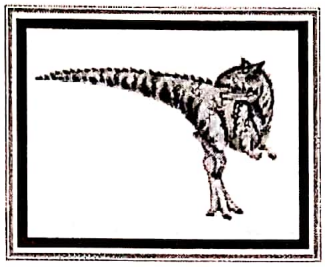
There has never been a more interesting time to study operating systems, and it has never been easier. The open-source movement has overtaken operating systems, causing many of them to be made available in both source and binary (executable) format. The list of operating systems available in both formats includes Linux, BUSD UNIX, Solaris, and part of macOS. The availability of source code allows us to study operating systems from the inside out. Questions that we could once answer only by looking at documentation or the behavior of an operating system we can now answer by examining the code itself.

Operating systems that are no longer commercially viable have been open-sourced as well, enabling us to study how systems operated in a time of fewer CPU, memory, and storage resources. An extensive but incomplete list of open-source operating-system projects is available from https://curlie.org/Computers/Software/Operating_Systems/Open_Source/

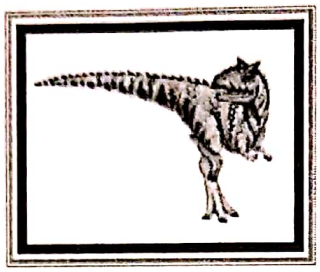
In addition, the rise of virtualization as a mainstream (and frequently free) computer function makes it possible to run many operating systems on top of one core system. For example, VMware (http://www.vmware.com) provides a free "player" for Windows on which hundreds of free "virtual appliances" can run. Virtualbox (http://www.virtualbox.com) provides a free, open-source virtual machine manager on many operating systems. Using such tools, students can try out hundreds of operating systems without dedicated hardware.

The advent of open-source operating systems has also made it easier to make the move from student to operating-system developer. With some knowledge, some effort, and an Internet connection, a student can even create a new operating-system distribution. Just a few years ago, it was difficult or impossible to get access to source code. Now, such access is limited only by how much interest, time, and disk space a student has.

End of Chapter 1



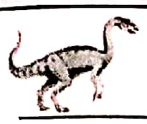
Chapter 2: Operating-System Services



1

Outline

- Operating-System Services (2.1)
- User and Operating-System Interface (2.2)
- System Calls (2.3)
- System Services (2.4)
- Linkers and Loaders (2.5)
- Why Applications are Operating System Specific (2.6)
- Operating-System Design and Implementation (2.7)
- Operating-System Structure (2.8)
- Building and Booting an Operating-System (2.9)
- Operating-System Debugging (2.10)



Objectives

- Identify services provided by an operating system
- Illustrate how system calls are used to provide operating system services
- Compare and contrast monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems
- Illustrate the process for booting an operating system
- Apply tools for monitoring operating system performance
- Design and implement kernel modules for interacting with a Linux kernel

3

شو الخدمات اللي بقدملي ياها ال OS

Operating-System Services

4

Operating System Services

- 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** (CLI), **Graphics User Interface** (GUI), **touch-screen, Batch**
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

5

Operating System Services (Cont.)

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

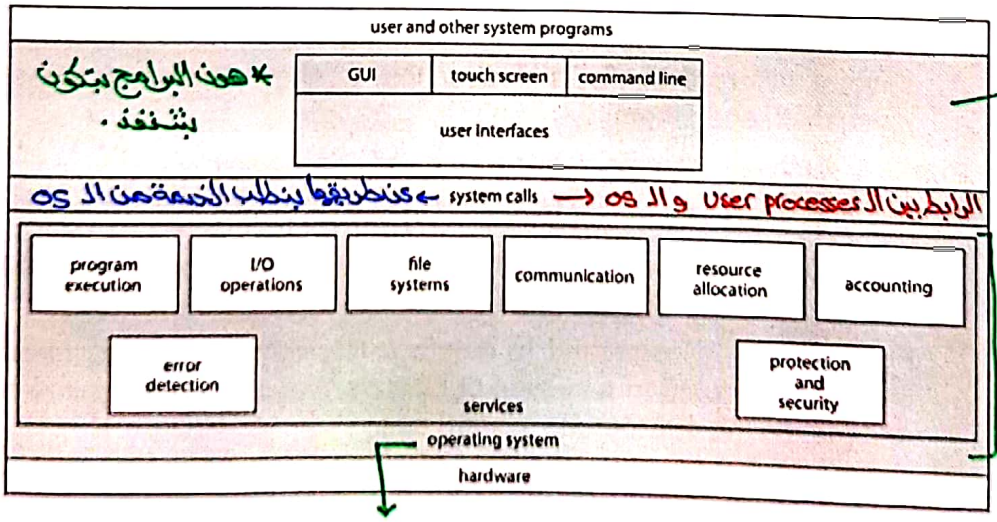
6

Operating System Services (Cont.)

- 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.
  - **Logging** - 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

7

A View of Operating System Services



8

User and Operating-System Interface

9

* الـ GUI أفخم من الـ CLI ولكن مع ذلك الـ CLI مازال موجود ويستخدم.
وعبارة عن برنامج يتيح الك التواصل مع الـ os وطلب الأشياء اللي بدك ياها على شكل commands.

① Command Line interpreter

- CLI allows direct command entry
- Sometimes implemented in kernel, sometimes by systems program يكون الـ os
- Sometimes multiple flavors implemented – **shells** → Command line الـ بتعطيك خدمة
  - For example, on UNIX and Linux systems, a user may choose among several different shells, including the C shell, Bourne-Again shell, Korn shell, and others. بالتوصل طريقة لكل وحدة منهم اسمها Shell.
- Primarily fetches a command from user and executes it
- These commands can be implemented in two general ways:

① The command interpreter itself contains the code to execute the command. → يكون الـ Code لتنفيذ الـ Command جوا الـ interpreter.
  - For example, a command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call.

② Implementing most commands through system programs.
  - In this case, the command interpreter does not understand the command in any way; it merely uses the command to identify a file to be loaded into memory and executed.

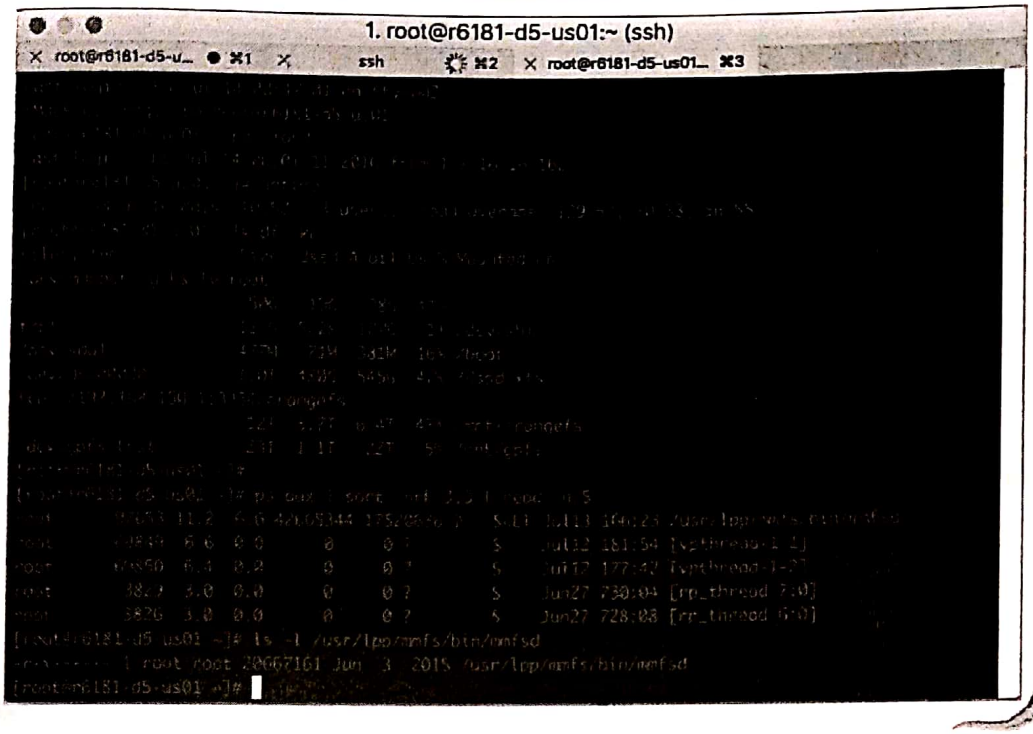
«bash» → main shell of Linux

حجم البرنامج بجيب أكبر وهيش بالسرعة بيقدر أتغيير commands جديدة. ← flexible ← هيش

لأنه الـ CLI بكون أصغر وفيه بس basics الـ commands وكل commands بدك تغييرها بتعملها load وبتغيرها. ← هيستخدم أكثر

10

Bourne Shell Command Interpreter



11

② 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)

التعامل عن طريق icons أكثر من الـ Commands.

12

handheld devices

③ 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

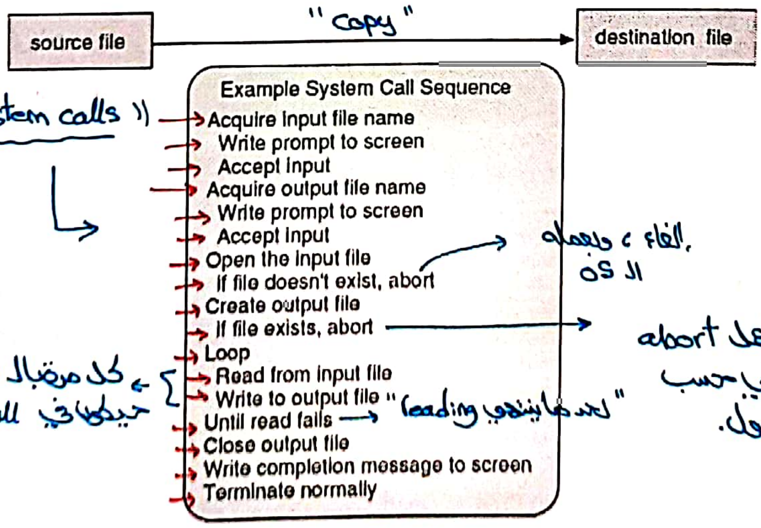
System Calls

System Calls

- **System calls** provide an interface to the services made available by an operating system
  - These calls are generally available as functions written in C and C++
  - Certain low-level tasks (for example, tasks where hardware must be accessed directly) may have to be written using assembly-language instructions
- Example: writing a simple program to read data from one file and copy them to another file.
  - UNIX cp command copies the input file in.txt to the output file out.txt.
    - cp in.txt out.txt
  - In an interactive system, program asks the user for the names., this approach will require a sequence of system calls, first to write a prompting message on the screen and then to read from the keyboard the characters that define the two files.

Example of System Calls

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



Application Programming Interface

- Frequently, systems execute thousands of system calls per second.
- Most programmers never see this level of detail,
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- A programmer accesses an API via a library of code provided by the operating system.
  - In the case of UNIX and Linux for programs written in the C language, the library is called **libc**.
- Note that the system-call names used throughout this text are generic
  - Each operating system has its own name for each system call

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)
```

return value | function name | parameters

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 into which the data will be read
- `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.

System Call Implementation

* ال API لحالها غير كافية *

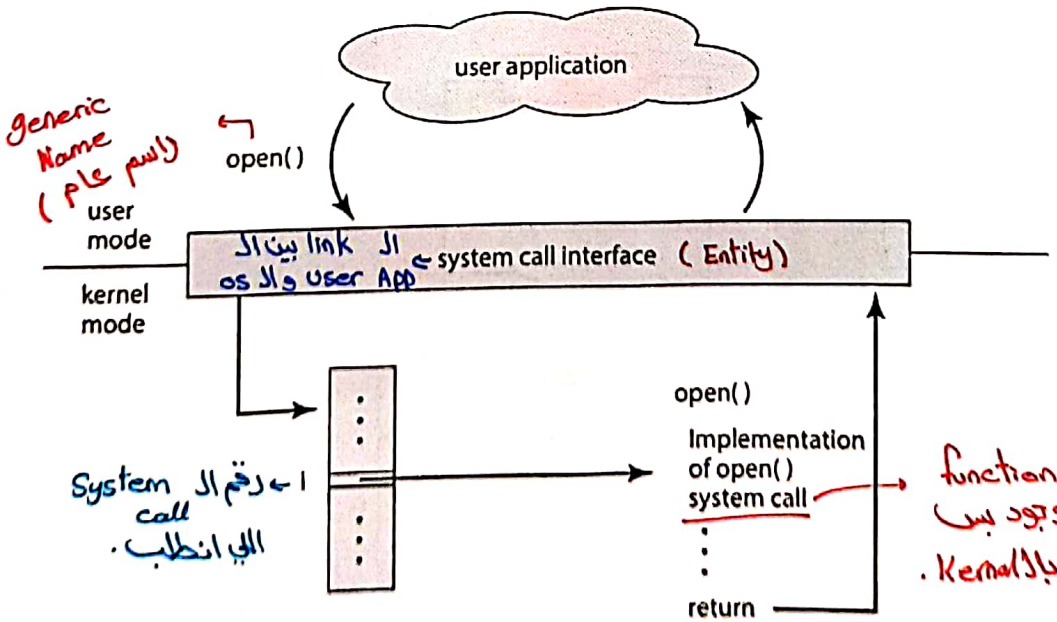
- Another important factor in handling system calls is the **run-time environment (RTE)**, the full suite of software needed to execute applications written in a given programming language, including its compilers or interpreters as well as other software, such as libraries and loaders
  - The RTE provides a **System-call interface**
- Typically, a number is associated with each system call
- **System-call interface** serves as the link to system calls made available by the operating system
  - Maintains a table indexed according to these numbers
  - 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 and are managed by the RTE

مجموعة كل البرمجيات اللي انت بتحتاجها لتنفذ تطبيقاتك .

هو المسؤول بشتغل كـ link بين البرنامج وبين ال OS

أنا كـ user أوقف التفاصيل ماني داعي

API – System Call – OS Relationship



الطريقة اللي بنعمل فيها Passing لل Parameters اللي ممكن يشتغل عليها ال System Call.

System Call Parameter Passing

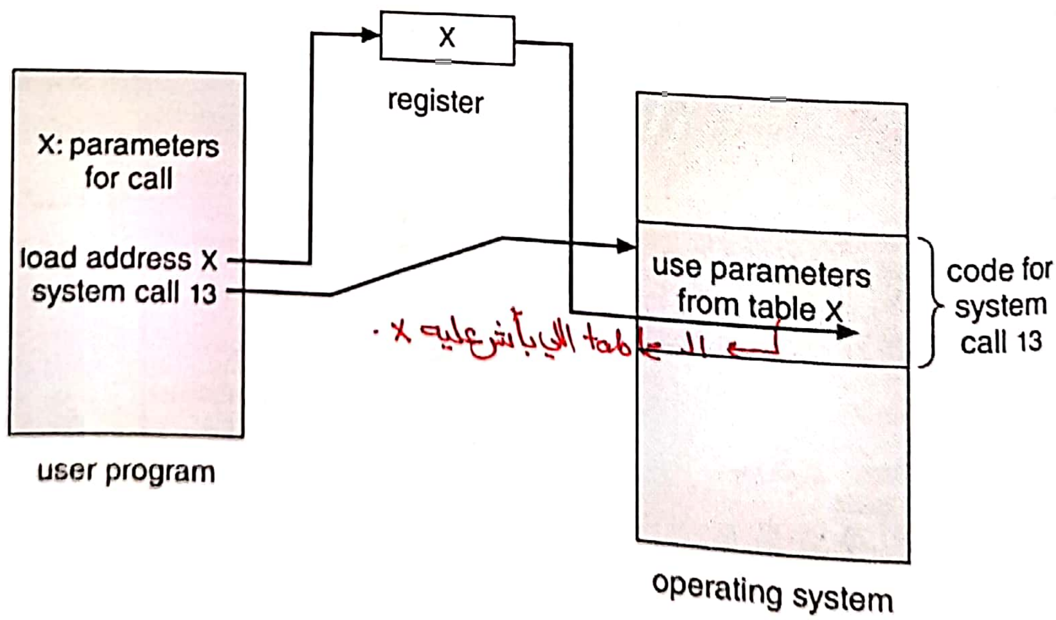
- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers → لو حجم الداتا اشي صغير : (مش flexible)
    - 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 (لو حجم الداتا كبير)
    - This approach taken by Linux and Solaris
  - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

« بيعطونا flexibility لأحجام كبيرة إنه نبعتها »

« عن طريق ال Memory »

Parameter Passing via Table

« الداتا كبيرة هون »



أنواع الـ functions اللي ممكن تتايوا من الـ OS

Types of System Calls

1. **Process control**
   - 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 → كإني بفضي جزء من الـ RAM
   - **Debugger** for determining **bugs, single step** execution
   - **Locks** for managing access to shared data between processes

« تستخدم عادة لما يكون برنامج شغال وبده يشتغل أكتر من Process بنفس الوقت »

Types of System Calls (Cont.)

2. **File management**
   - create file, delete file
   - open, close file
   - read, write, reposition
   - get and set file attributes
3. **Device management**
   - request device, release device
   - read, write, reposition
   - get device attributes, set device attributes
   - logically attach or detach devices

Types of System Calls (Cont.)

④ ▪ Information maintenance
- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

⑤ ▪ 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

Types of System Calls (Cont.)

⑥ ▪ Protection
- Control access to resources
- Get and set permissions
- Allow and deny user access

لأنه الأساسي اللي استخدمناها قبل كانت أساسي عامة .

Examples of Windows and Unix System Calls

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

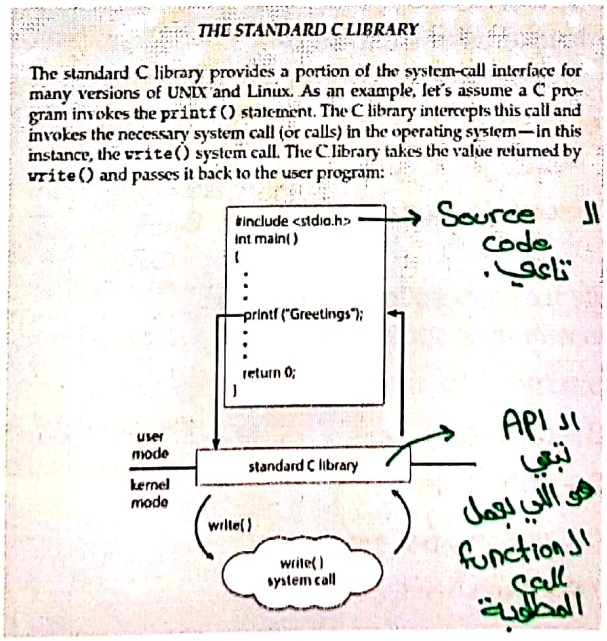
27

Standard C Library Example

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

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:



3

(control something)

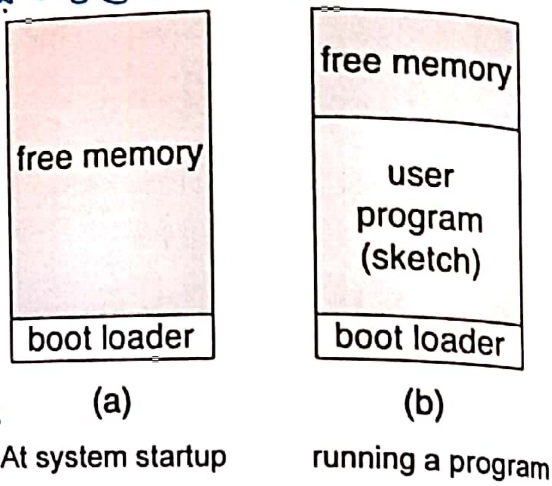
microcontroller ← يستخدم بال embedded systems للتحكم بشيء ما

Example: Arduino

"يعني أنا بنزل برنامج واحد بس" ← "Single task system"

- Single-tasking
- No operating system
- Programs (sketch) loaded via USB into flash memory
- Single memory space
- Boot loader loads program (مش OS)
- Program exit -> shell reloaded

وظيفته إنه خلي الـ microcontroller لما يصير running و يبلش تنفيذ من مكان ما نزل الـ sketch اللي إنت نزلته



* طريقة استخدامه و بتكتب الـ code تبعه على جهاز معين بتحمله load على الـ Memory تبع الـ Arduino.

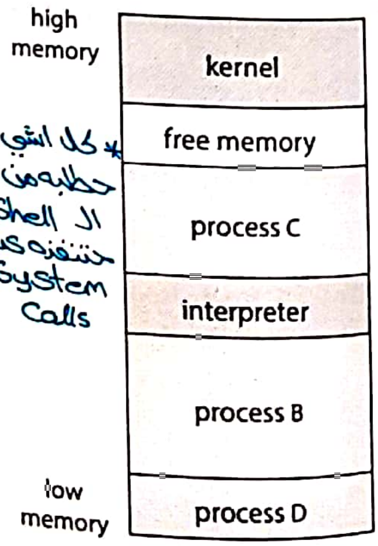
يستخدم لما يكون في البرامج اللي فيها أكثر من user.

Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process → البرنامج اللي طلبته
  - 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

ويكون مصمم داخل الـ System إنه شو الـ error codes الموجودة.

* كل الشي ← طلبته من الـ Shell تنفذه بشكل System Calls



System Services

System Services

جزء من ال OS بس مش جزء من ال Kernal.

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a file
  - 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

System Services (Cont.)

- 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

لـ ميات ممكن انت تعدل عليها.

*GUI → Sys programs يعتبر من أجزاء ال

System Services (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another' s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

جزء من ال OS بيعمل load للبرنامج تاعك

System Services (Cont.)

- **Background Services** → هدوم بيعملوا Settings معينة.
  - 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**

* بعضهم بيشتغلوا at sys startup وبعضهم منلما تقوم الـsys boot لحد ما تتوقيه حسب شو الهدف منها.

- **Application programs** → بعضها بيجي جزء من الـ os يلما بقدر اعتها من الـ kernal os
  - Don't pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Linkers and Loaders

Linkers and Loaders

- Source code compiled into object files designed to be loaded into any physical memory location – **relocatable object file**
- **Linker** combines these into single binary **executable** file
  - Also brings in libraries
- Program resides on secondary storage as binary executable
- Must be brought into memory by **loader** to be executed
  - **Relocation** assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general purpose systems don't link libraries into executables
- Rather, **dynamically linked libraries** (in Windows, **DLLs**) are loaded as needed, shared by all that use the same version of that same library (loaded once)

37

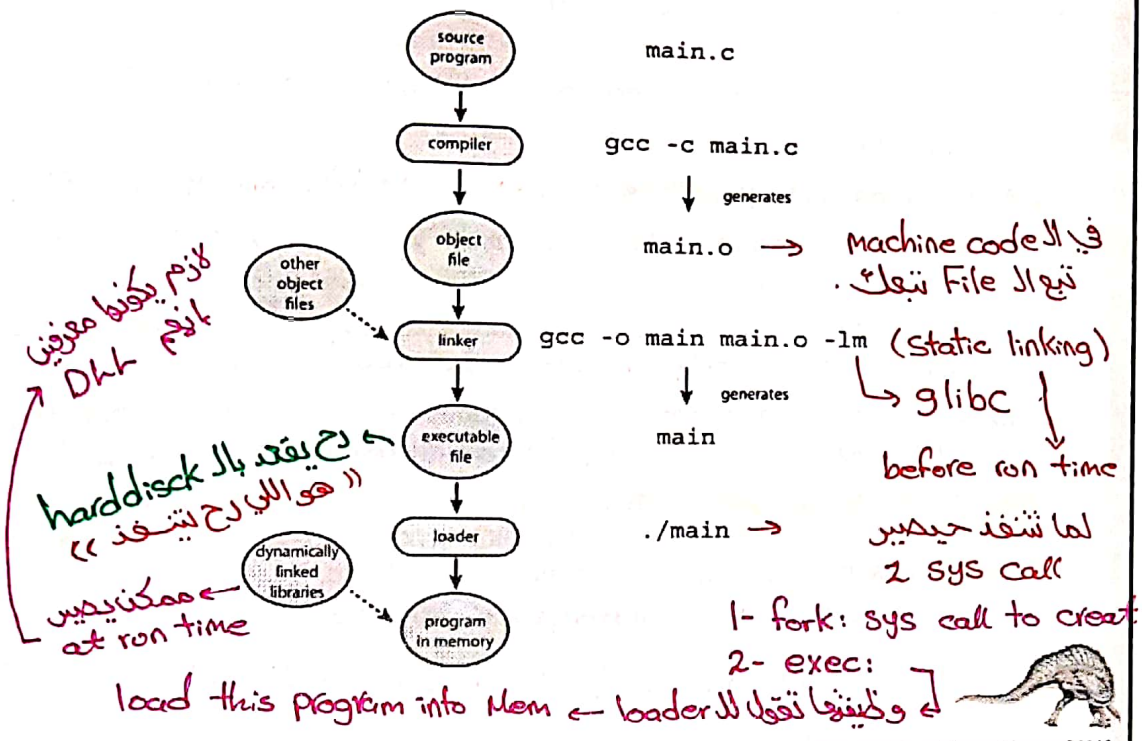
Linkers and Loaders (Cont.)

- Object, executable files have standard formats, so operating system knows how to load and start them
  - Include the compiled machine code
  - Include symbol table containing metadata about functions and variables that are referenced in the program.
- For UNIX and Linux systems, this standard format is known as **ELF** (for **Executable and Linkable Format**).
  - There are separate ELF formats for relocatable and executable files.
  - One piece of information in the ELF file for executable files is the program's entry point, which contains the address of the first instruction to be executed when the program runs.
- Windows systems use the **Portable Executable (PE)** format, and macOS uses the **Mach-O** format.

38

The Role of the Linker and Loader

Example of Simple running code



Operating System Concepts – 10th Edition 2.39 Silberschatz, Galvin and Gagne ©2018

39

active Entity لحتى يتحول لـ Passive Entity هو Programm الـ *
. Process بنعمله

Why Applications are Operating System Specific

Operating System Concepts – 10th Edition 2.40 Silberschatz, Galvin and Gagne ©2018

40

Why Applications are Operating System Specific

- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
  - Own file formats, etc.
- Apps can be multi-operating system → كيف أخلي الـ app يشتغل عأكثر من OS
  - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems → "Source code" الـ run بالأصل من الـ
  - App written in language that includes a VM containing the running app (like Java) لـ بقدر أكتبه byte code وبنفذه
  - Use standard language (like C), compile separately on each operating system to run on each → لما تعمل run خلص مابتقدر تعمله run ع OS ثاني
- **Application Binary Interface (ABI)** is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.
  - مشكلتها انه مافي Standard لكل الـ OS .
  - (code to run on OS) الواجهة اللي عن طريقها بكتب

Operating-System Design and Implementation

وحناخذ الفروقات الأساسية فقط.

Design and Implementation

↳ مافي حل واحد لكل المواضيع.

- Design and Implementation of OS is 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 → من وجهة نظر الـ user
  - **System goals** – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient → من وجهة نظر الـ System
- Specifying and designing an OS is highly creative task of **software engineering** → مش اشي سهل.

Policy and Mechanism

- **Policy: What** needs to be done? → الـ Rules تابعت الـ os تنبعي حسب الاحتياجات.
  - Example: Interrupt after every 100 seconds
- **Mechanism: How** to do something? → كيفابك تعمل وشو لازم تعمل
  - Example: timer
- Important principle: separate policy from mechanism
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.
  - Example: change 100 to 200

* إنا الـ Policy مربوط بالـ Mechanism لو احتجت تعديل رح تضطر تعدل حتى هالـ Policy بينما لوكان مفصول قدر الإمكان واحتجت تعديل حتعدل ببساطة بالـ Mechanism. مثال الـ Microkernel حشرحه بالمشغل.

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 Structure

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex – UNIX
  - Layered – an abstraction → like mac
  - Microkernel – Mach

* أغلبية الـ os بكونوا hybrid يعني مخلوط أنواع وطرق مختلفة .

Monolithic Structure – Original UNIX

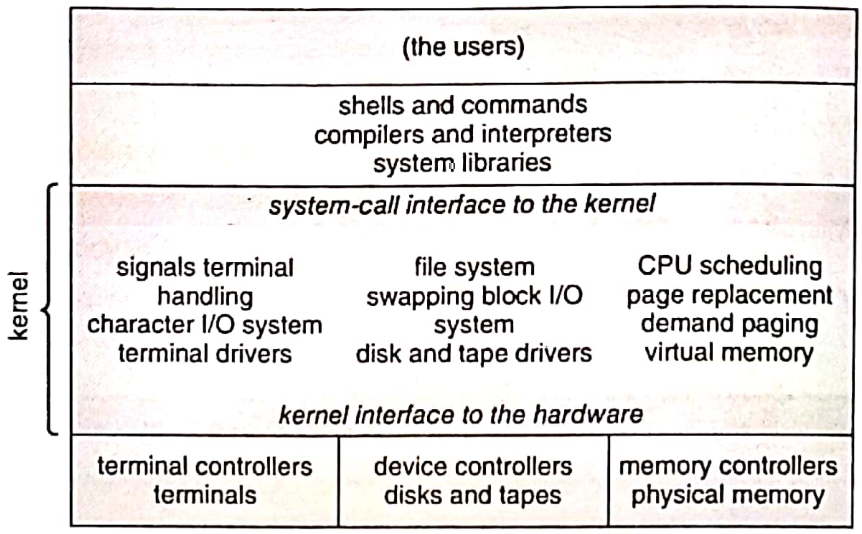
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

* كل الـ kernel بكون موجود بمكان واحد مش مقسم لأجزاء (وحدة layer) .

بيعطي سرعة بس معقد جدًا لإنه كله بنفس المكان .

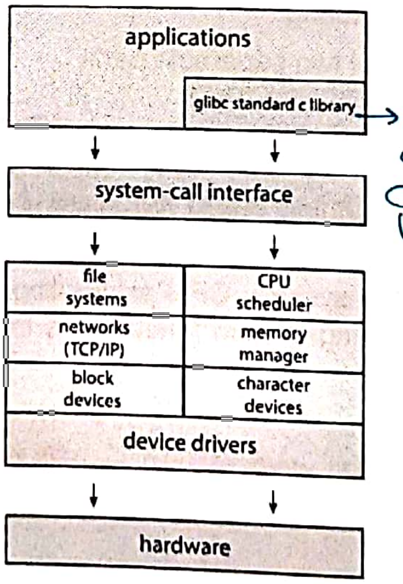
Traditional UNIX System Structure

Beyond simple but not fully layered



Linux System Structure

Monolithic plus modular design

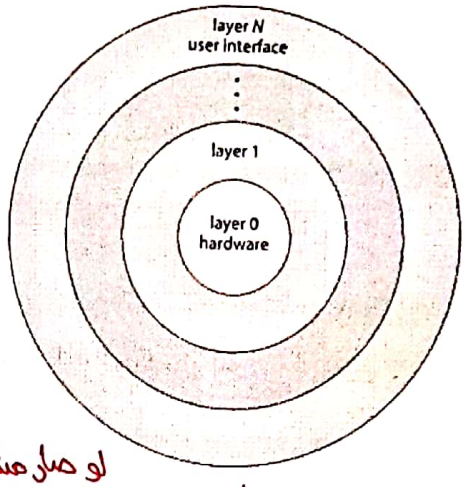


access للCernal عن طريقها

الفرق إنه ال Linux ← يستخدم ال modules

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 → لو صار مشكلة بـ layer الباقي ما بتأثر



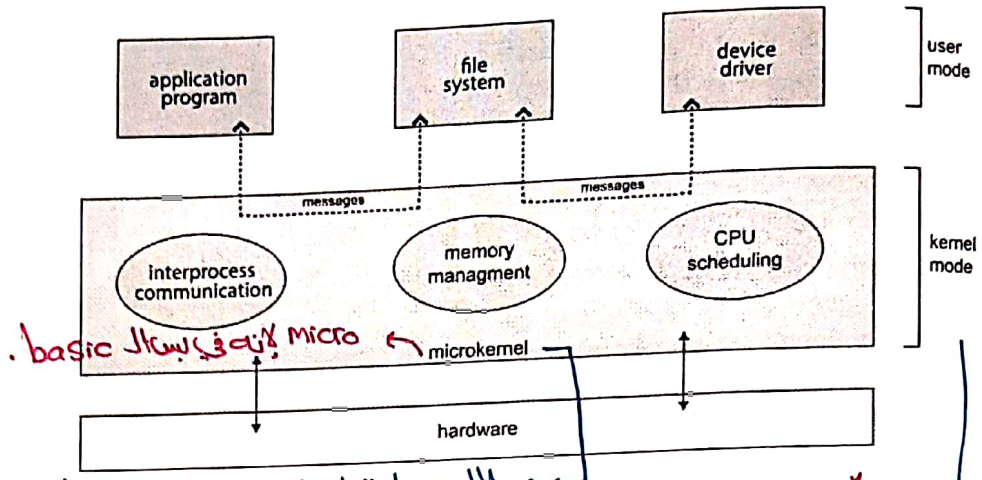
كل الـ layer بتكون متصلة بما قبلها وبعدها مباشرة فقط
مثل الـ network في الـ TCP/IP stack

* ما في OS بستخدموها بشكل أساسي، الـ Mac OS بستخدموها
بطريقة إنه في communication بين الـ layers

Microkernels

- Moves as much from the kernel into user space
- **Mach** is an example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach and BSD unix
- Communication takes place between user modules using **message passing**
- **Benefits:**
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- **Detriments:**
  - Performance overhead of user space to kernel space communication

Microkernel System Structure



basic بإنه في بسيط micro ←

هو اللي بعمل تواصل مع اللي فوق عن طريق message passing ال

بس بخلي فيه الشغلات الأساسية والرئيسية والباقي بطلعه لل user mode

وهنا بخلي ال kernal أبسط وأصغر ومعمي بسبب كون خسرنا السرعة.

53

Modules ← linux بيستخدموا

- Many modern operating systems implement **loadable kernel modules (LKMs)**
  - 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 flexiblity**
  - Linux, Solaris, etc.

54

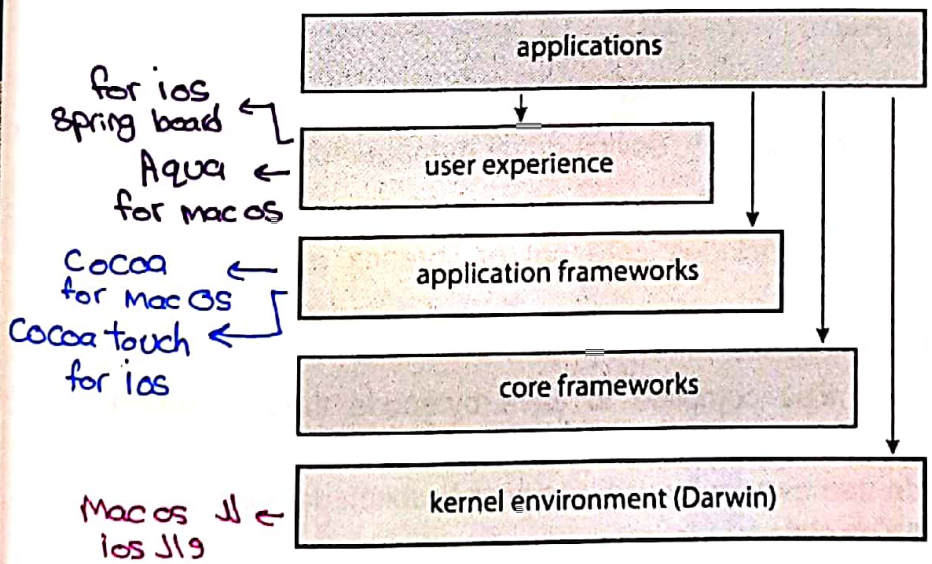
Hybrid Systems

- Most modern operating systems are not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, but:
    - It retains some behavior typical of microkernel systems, including providing support for separate subsystems (known as operating-system **personalities**) that run as user-mode processes
    - It provides support for dynamically loadable kernel modules
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

55

for laptops for phones

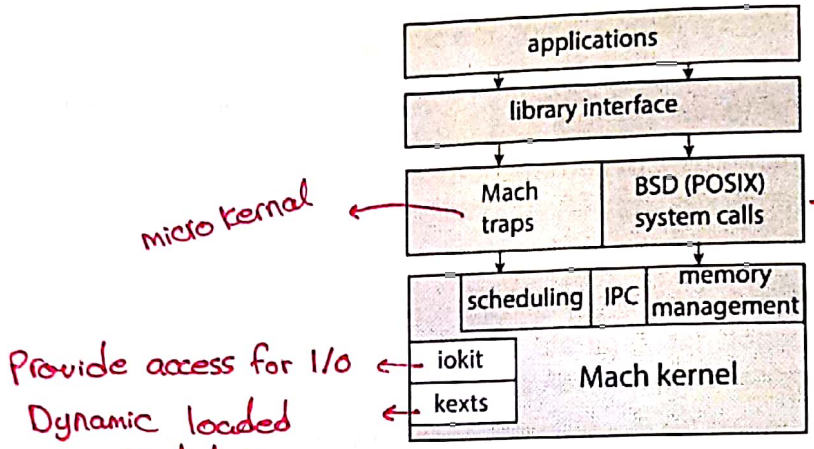
macOS and iOS Structure



زي الـ layers ← بس مشكل layer متعلق بالي تحته وفوقه بس لا في تواصل بين الـ layers

6

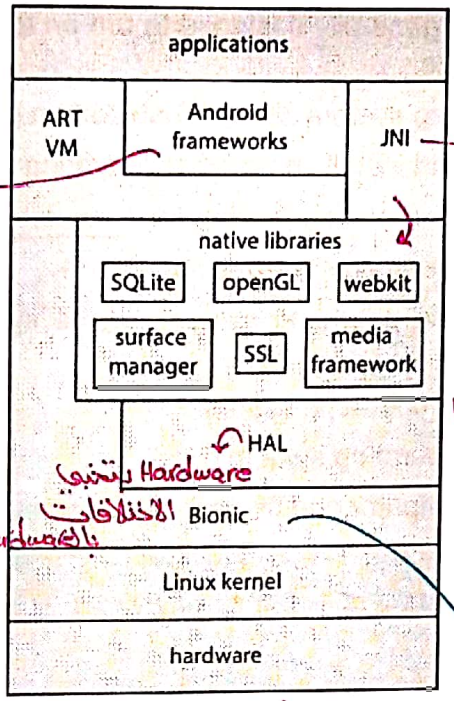
Darwin



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



to run and write your app بنكتب الـ code جوا الـ ART VM

Portable بكون

* وجود الـ VM بيطأ.

طريقة ثانية للكتابة الكود تاعك ملزم يكون Portable لإنه الـ native libraries بتختلف من جهاز للثاني.

access أعمق و أوسع

objective C library بتوصلني الـ kernal

الأختلافات بالـ hardware بتتبع Hardware

« JNI غالبًا الأفضل »

60

بالعادة مابنعمل هيك لإنه الـ os بكون نازل مع الجهاز إلا لو أنا بدي استخدم os ثاني.

Building and Booting an Operating-System

61

Building and Booting an Operating-System

- Operating systems generally designed to run on a class of systems with variety of peripherals
- Commonly, operating system already installed on purchased computer
  - But can build and install some other operating systems
  - If generating an operating system from scratch → ما بستخدمه هون.
    - Write the operating system source code
    - Configure the operating system for the system on which it will run
    - Compile the operating system
    - Install the operating system
    - Boot the computer and its new operating system

System Boot

- When power initialized on system, execution starts at a fixed memory location
- Operating system must be made available to hardware so hardware can start it → كيف يكون الـ Boot عارف وين الـ address اللي بيلش منه الكود
- The process of starting a computer by loading the kernel is known as **booting** the system
  - Small piece of code – **bootstrap loader**, **BIOS**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it ← جزء من الـ BIOS
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
  - Modern systems replace BIOS with **Unified Extensible Firmware Interface (UEFI)** → متطور أكثر و عادة بتكون one step و أسرع و أحسن
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**
- Boot loaders frequently allow various boot states, such as single user mode

برنامج بتعمل فحص لـ I/O devices انه أمورها تمام.

Bootstrap عبارة عن loader

Operating-System Debugging

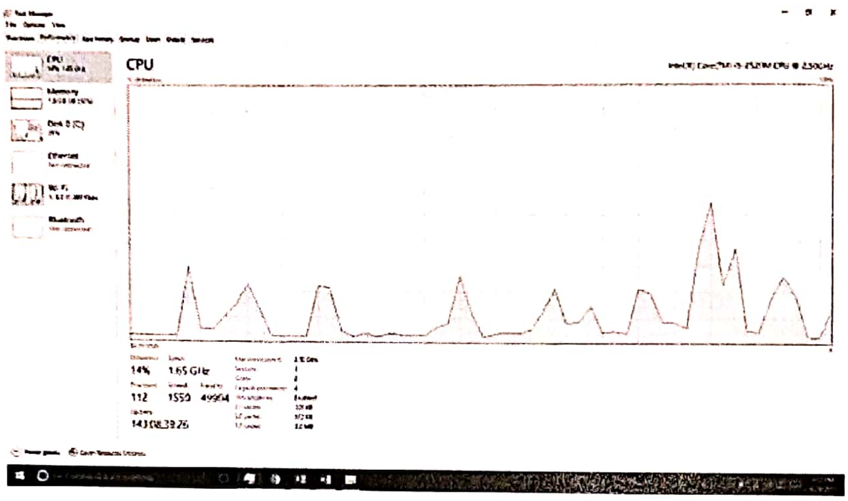
Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- Also **performance tuning**
- 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
  1. Sometimes using ***trace listings*** of activities, recorded for analysis
  2. **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



* في ال Counters كمان قبل ال Tracing .

Tracing

- Collects data for a specific event, such as steps involved in a system call invocation
- Tools include (بيعطيك ال sys call الي عملها البرنامج بالترتيب.)
  - strace – trace system calls invoked by a process
  - gdb – source-level debugger
  - perf – collection of Linux performance tools
  - tcpdump – collects network packets (زي ال wireshark)

(strace, gdb: debugging — عمستوى ال Process)

(perf, tcpdump: عمستوى ال System كامل.)

BCC

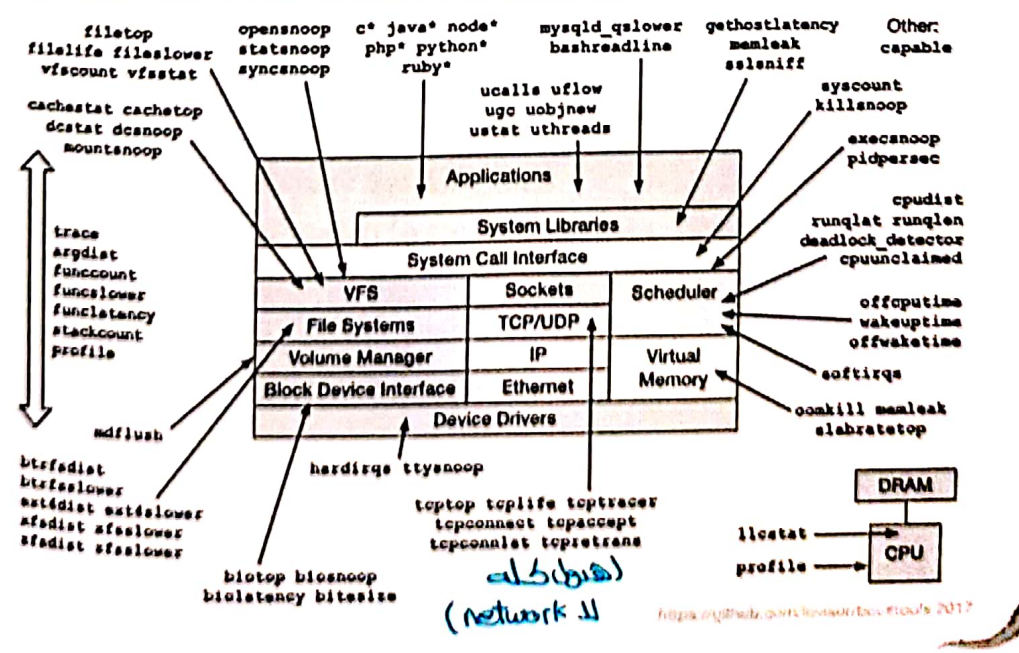
- Debugging interactions between user-level and kernel code nearly impossible without toolset that understands both and an instrument their actions
- BCC (BPF Compiler Collection) is a rich toolkit providing tracing features for Linux
  - See also the original DTrace
- For example, disksnoop.py traces disk I/O activity →

TIME(s)	T	BYTES	LAT(ms)
1946.29186700	R	8	0.27
1946.33965000	R	8	0.26
1948.34585000	W	8192	0.96
1950.43251000	R	4096	0.56
1951.74121000	R	4096	0.35

- Many other tools (next slide)

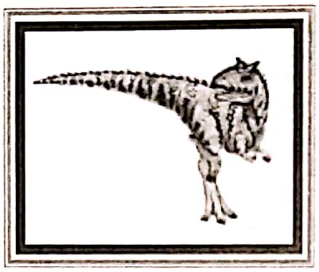
68

Linux bcc/BPF Tracing Tools

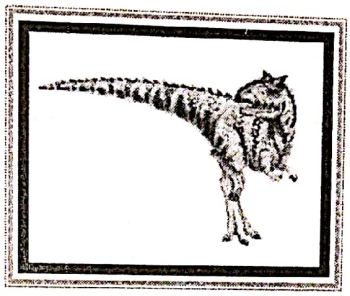


69

End of Chapter 2



Chapter 3: Processes



Outline

- Process Concept (3.1)
- Process Scheduling (3.2)
- Operations on Processes (3.3)
- Inter-process Communication (3.4)
- IPC in Shared-Memory Systems (3.5)
- IPC in Message-Passing Systems (3.6)

Objectives

- Identify the separate components of a process and illustrate how they are represented and scheduled in an operating system.
- Describe how processes are created and terminated in an operating system, including developing programs using the appropriate system calls that perform these operations.
- Describe and contrast interprocess communication using shared memory and message passing.
- Design programs that uses pipes and POSIX shared memory to perform interprocess communication.
- Describe client-server communication using sockets and remote procedure calls.
- Design kernel modules that interact with the Linux operating system.

Process Concept

Process Concept

- An operating system executes a variety of programs that run as a process.
- **Process** – a program in execution; process execution must progress in sequential fashion. No parallel execution of instructions of a single process
  ↳ thread طالما ماخذ
- The status of the current activity of a process is represented by:
  - The value of the program counter and,
  - The contents of the processor's registers
- The memory layout of a process is typically divided into multiple sections:
  - The program code, also called **text section**
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

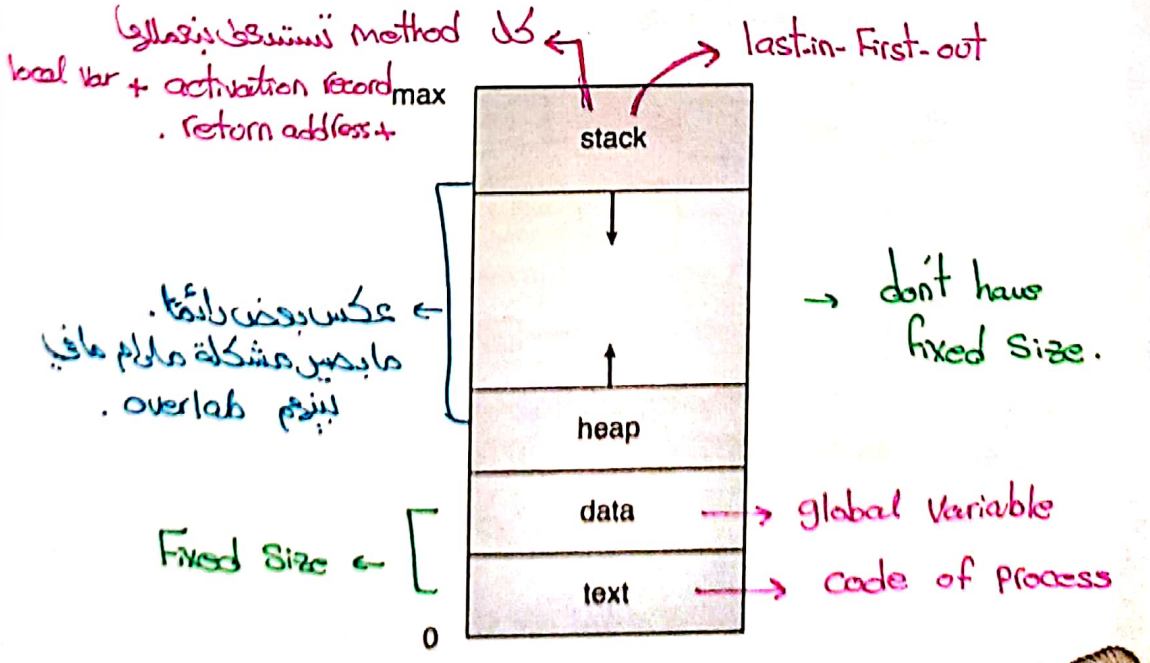
*process ≠ executing entity of the program.

*أي Process كيشتغل تنفذ لازم تكون موجودة عالـ Mem.

→ like array and objects

5 *program ≠ pasive entity saved on harddisk.

Process in Memory



6

Process Concept (Cont.)

- Program is a **passive** entity stored on disk (**executable file**).
- A process is an **active** entity; with a program counter specifying the next instruction to execute and a set of associated resources
- Program becomes process when an executable file is loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc.
- One program can be several processes → like web browser
  - Consider multiple users executing the same program
- A process can itself be an execution environment for other code
  - Example: Java programming environment JVM

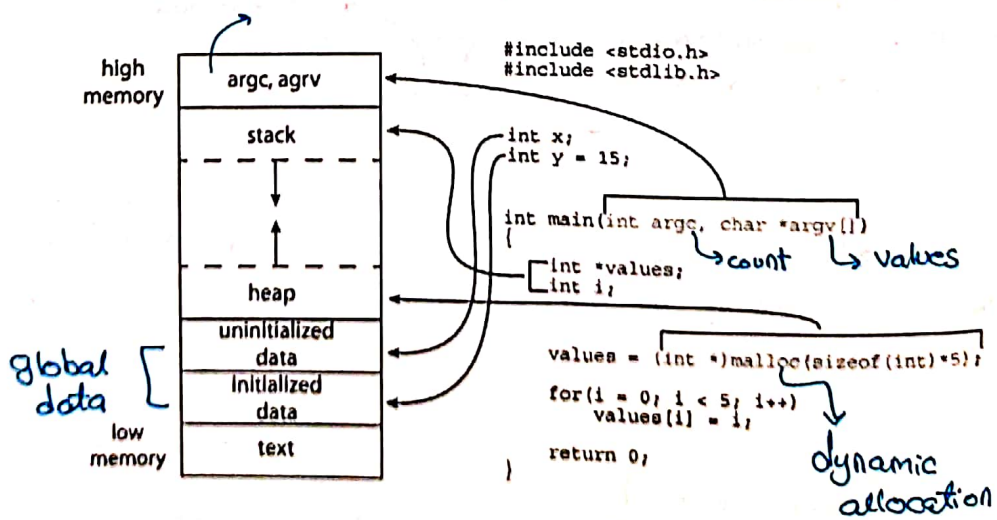
user واحد مستقل لأكثر من instance.

وظيفتها تنسك كل الـ byte code وتحوله machine code ليفهمه الـ Computing device

7

Memory Layout of a C Program

الـ OS مكانها هنا



Size الـ global var حجم بعطيني .fixed size لأنه

8

Process State

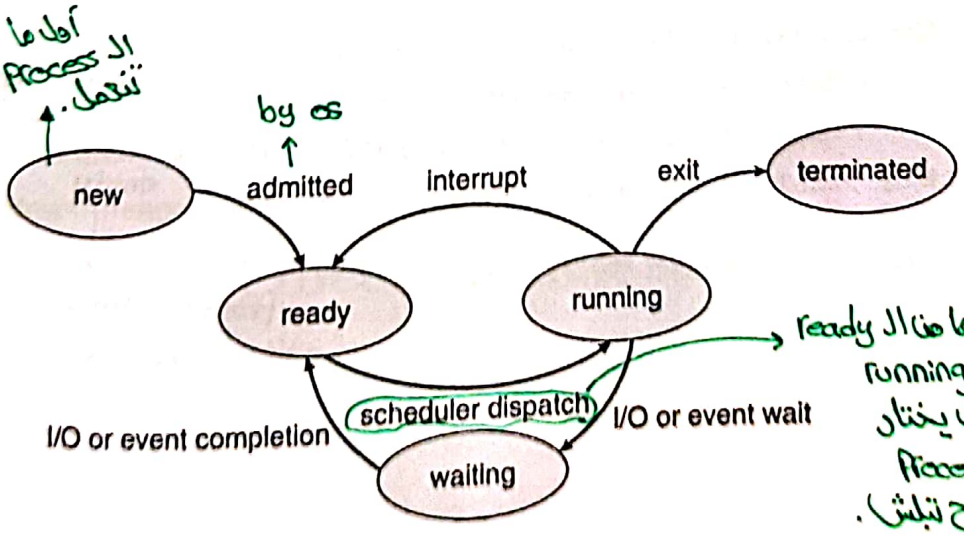
- As a process executes, it changes state → Process حالة هاي ال

1. **New**: The process is being created → Process هاي ال انعملت.
2. **Running**: Instructions are being executed → run هاي ال Process هلا بتهيل وشغالة.
3. **Waiting**: The process is waiting for some event to occur → event هاي ال Process بتنتظر
4. **Ready**: The process is waiting to be assigned to a processor → I/O operation / child process
5. **Terminated**: The process has finished execution → هاي ال Process انتوت.

ال Process جاهز ينفذ أول ما يجي دوره.

9

Diagram of Process State



ready بياخنها من ال running ل ومسؤول يختار ال Process اللي رح تنفذ.

10

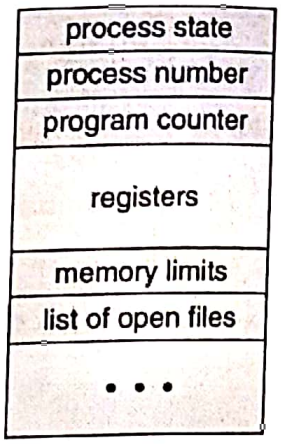
يخزن في الـ Mem لكل Process في مجموعة من المعلومات بتمكن الـ OS من انه يوقف شغل Process الـ أو لا بعد ينتقل بين States Process الـ

Process Control Block (PCB)

← اسم علم

Information associated with each process(also called **task control block**)

1. Process state – running, waiting, etc.
2. Program counter – location of instruction to next execute
3. CPU registers – contents of all process-centric registers
4. CPU scheduling information- priorities, scheduling queue pointers
5. Memory-management information – memory allocated to the process ← Process عادة بكون فيها بداية ونهاية الـ ← الـ address البداية والنهاية
6. Accounting information – CPU used, clock time elapsed since start, time limits
7. I/O status information – I/O devices allocated to process, list of open files



* when program loaded in memory it becomes a Process.

← وهاي الـ Process بحيين الها States مختلفة والها كل المعلومات المخزنة بالـ PCB

11

Threads

- So far, process has a single thread of execution
- Consider having multiple program counters per process
  - Multiple locations can execute at once
    - Multiple threads of control -> **threads** ← حيكون لكل Thread الـ PC خاص فيها ← multiple executing entity
- Must then have storage for thread details, multiple program counters in PCB ← أصغر من الـ Process
- Will be explored in detail in Chapter 4

running on the same time. بكونوا

12

Process Representation in Linux

PCB ←

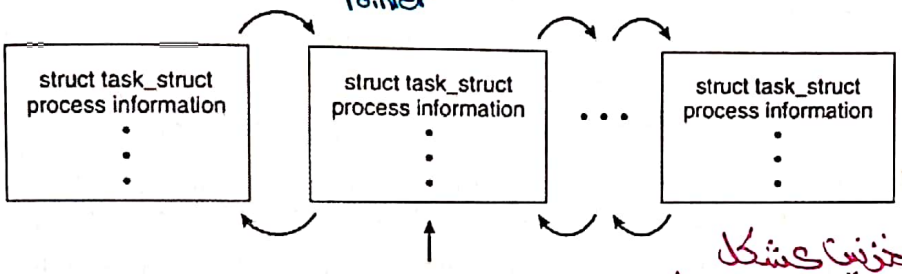
Represented by the C structure `task_struct` → Linux library معرفة بال

فيها كل المعلومات اللي بنحتاج نعرفها عن ال Process.

```
pid t_pid;                    /* process identifier */
long state;                   /* state of the process */
unsigned int time_slice       /* scheduling information */
struct task_struct *parent;   /* this process's parent */
struct list_head children;    /* this process's children */
struct files_struct *files;   /* list of open files */
struct mm_struct *mm;         /* address space of this
process */
```

Pointer

Pointer



current
(currently executing proccess)

Pointer على ال kernal ← بتشير على ال Process اللي بتنفذ حاليا.

مخزنين كشكل double linked list

Process Scheduling

Process Scheduling

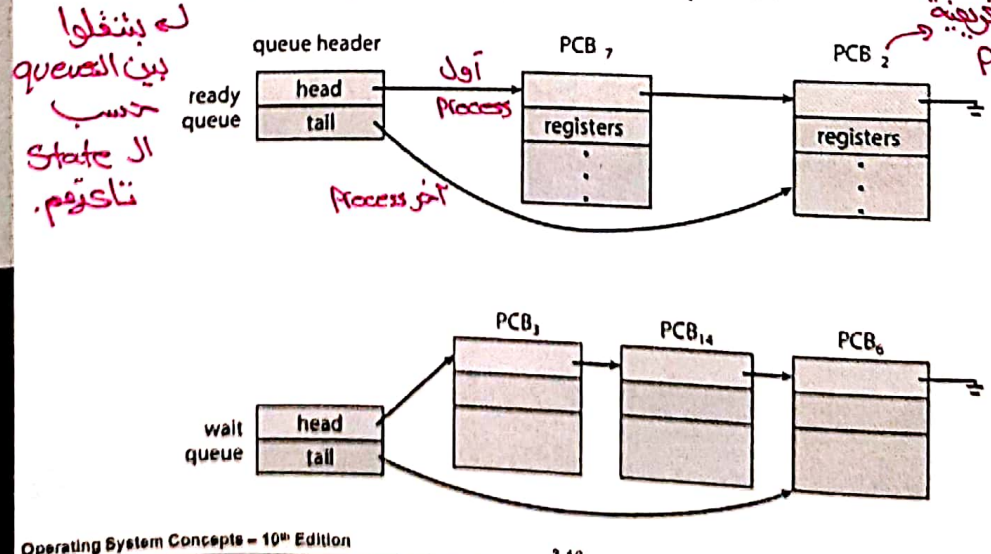
وظيفته

- **Process scheduler** selects among available processes for next execution on CPU core
- Goal -- Maximize CPU use, quickly switch processes onto CPU core
- Each CPU core can run one process at a time
  - For a system with a single CPU core, there will never be more than one process running at a time
  - A multicore system can run multiple processes at one time
  - If there are more processes than cores, excess processes will have to wait until a core is free and can be rescheduled
- The number of processes currently in memory is known as the **degree of multiprogramming**
- In general, most processes can be described as either:
  - I/O bound, which spends more of its time doing I/O than it spends doing computations, or
  - CPU bound which, in contrast, generates I/O requests infrequently, using more of its time doing computations

15

Scheduling Queues

- Maintains **scheduling queues** of processes
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Wait queues** – set of processes waiting for an event (i.e., I/O)
  - Processes migrate among the various queues



16

Scheduling Queues (Cont.)

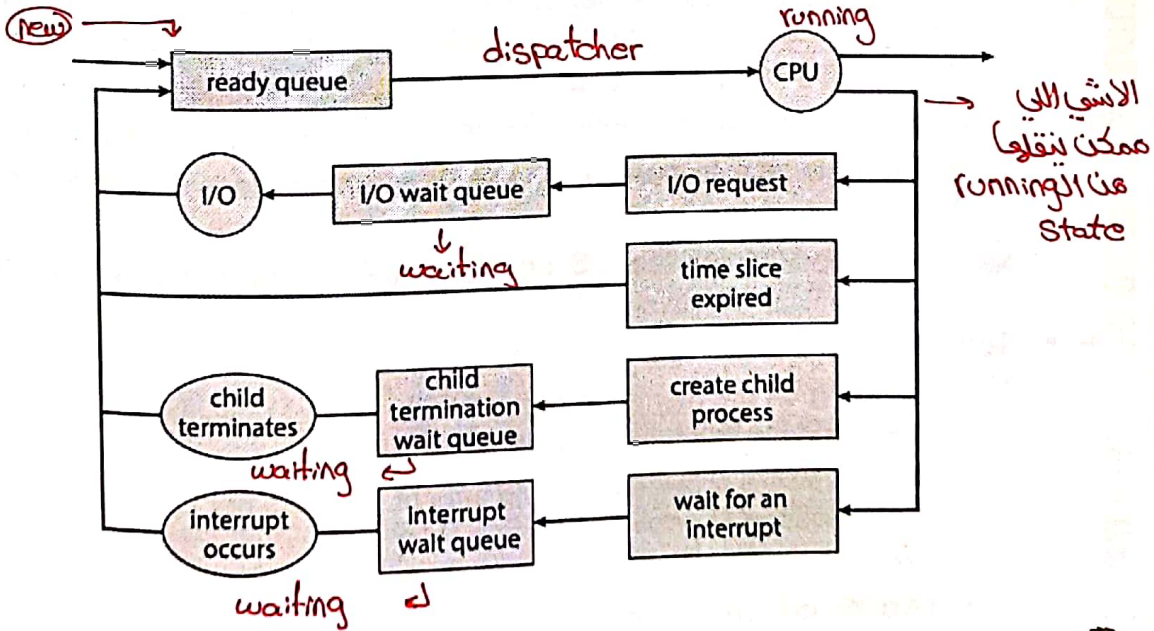
- A new process is initially put in the ready queue
- It waits there until it is selected for execution, or dispatched
- Once the process is allocated a CPU core and is executing, one of several events could occur:
  1. The process could issue an I/O request and then be placed in an I/O wait queue
  2. The process could create a new child process and then be placed in a wait queue while it awaits the child's termination
  3. The process could be removed forcibly from the core, as a result of an interrupt or having its time slice expire, and be put back in the ready queue

بالقوة

خلص الوقت اللي مسموحلك.

.7

Representation of Process Scheduling



الاشي اللي ممكن ينقلوا من الrunning State

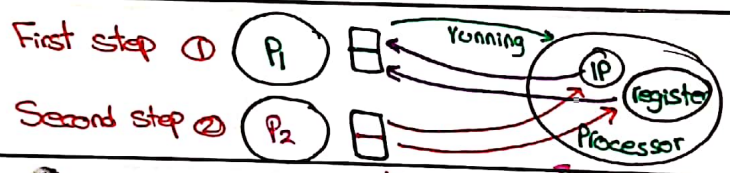
18

the process is active but is not running; it's not currently running on the CPU but is active because it's loaded into memory.

CPU Scheduling

- A process migrates among the ready queue and various wait queues throughout its lifetime.
- The role of the **CPU scheduler** is to select from among the processes that are in the ready queue and allocate a CPU core to one of them.
- The CPU scheduler executes at least once every 100 milliseconds, although typically much more frequently
- Some operating systems have an intermediate form of scheduling, known as **swapping**
  - Key idea is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming
  - Later, the process can be reintroduced into memory, and its execution can be continued where it left off
- Swapping is typically only necessary when memory has been overcommitted and must be freed up

19



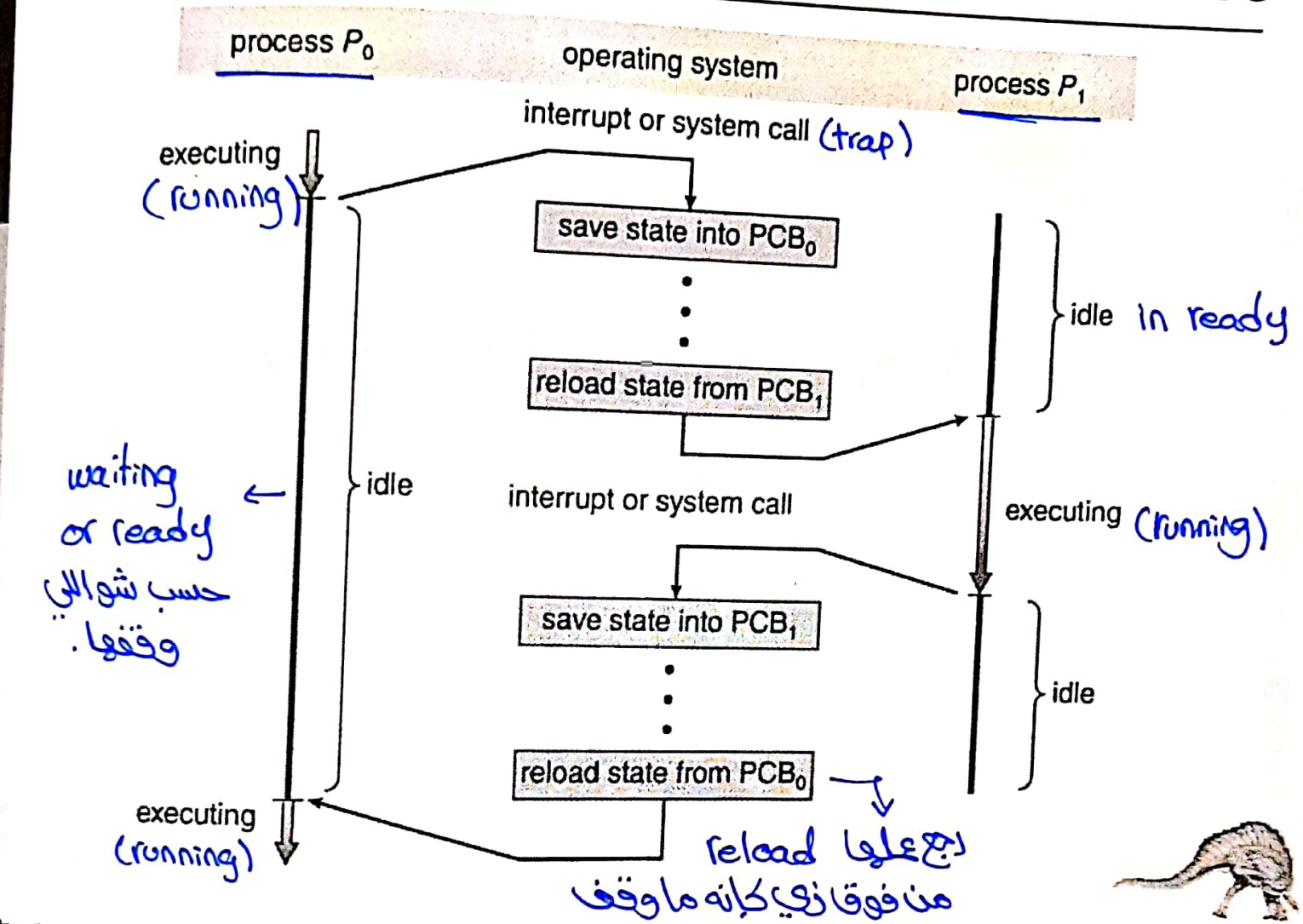
Context Switch

- 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**
- **Context** of a process represented in the PCB
- Context-switch time is pure overhead; the system does no useful work while switching
  - The more complex the OS and the PCB → the longer the context switch
- Time varies from machine to machine
  - Depending on the memory speed, the number of registers that must be copied, and the existence of special instructions
  - Some hardware provides multiple sets of registers per CPU → multiple contexts loaded at once

State or register for more than one processor.

20

CPU Switch From Process to Process



يكون كتير أكثر من Process شغالة بنفس الوقت

Multitasking in Mobile Systems

- Some mobile systems (e.g., early versions of iOS) allow only one process to run, others suspended
- Due to screen real estate, user interface limits iOS provides for a
  - Single **foreground** process- controlled via user interface → Process شغالة ومبينة عالشاشة
  - Multiple **background** processes– in memory, running, but not on the display, and with limits → Processes مش ظاهرة عالشاشة
    - Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback (محدودة جدا)
  - As hardware for mobile devices began to offer larger memory capacities, multiple processing cores, and greater battery life, subsequent versions of iOS began to support richer functionality for multitasking with fewer restrictions.
- Android runs foreground and background, with fewer limits → more flixeble than ios
  - Background process uses a **service** to perform tasks
  - Service can keep running even if background process is suspended
  - Service has no user interface, small memory use

مش سهل الـ multitasking في الـ mobile لأن الـ Computers resources → very limited.

Operations on Processes

3

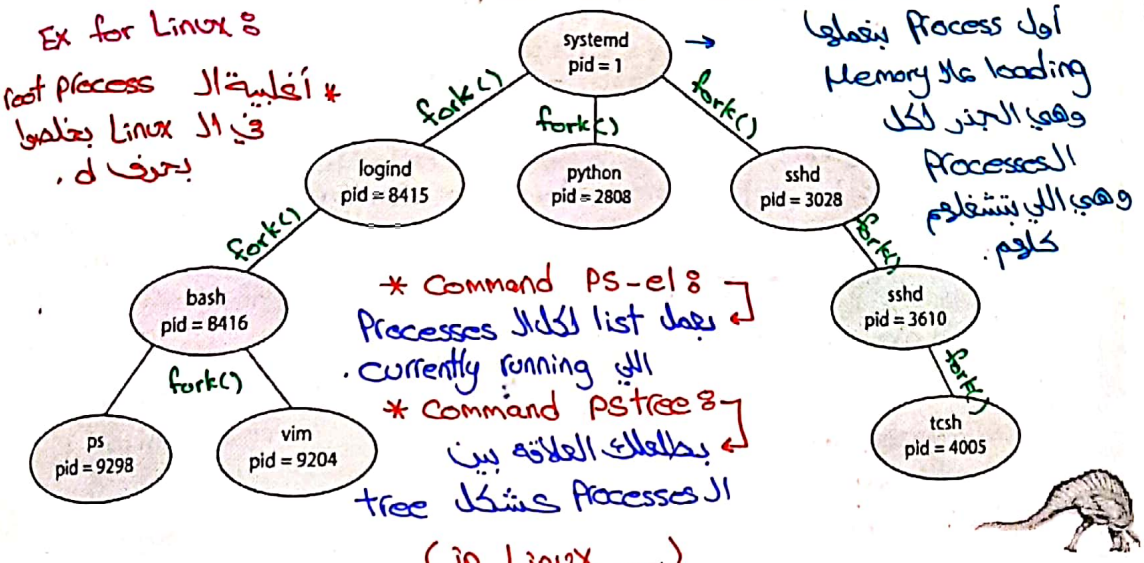
Operations on Processes

- System must provide mechanisms for:
  - Process creation
  - Process termination

4

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)



Process Creation (Cont.)

- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Address space
  - Child duplicate of parent
  - Child has a program loaded into it

* ال Parent بقدر يتحكم بال Child لأنه بعرف ال id تاعه.

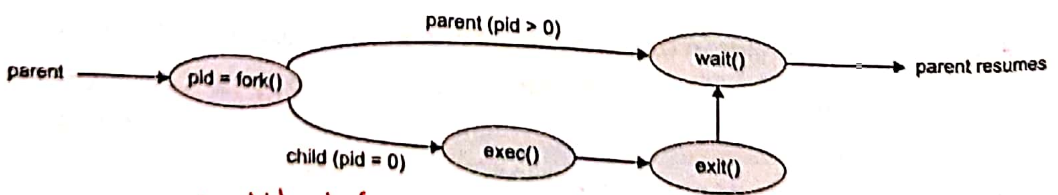
Process Creation – UNIX

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

   /* fork a child process */
   pid = fork();

   if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
   }
   else if (pid == 0) { /* child process */
      execlp("/bin/ls","ls",NULL);
   }
   else { /* parent process */
      /* parent will wait for the child to complete */
      wait(NULL);
      printf("Child Complete");
   }

   return 0;
}
```

- pid_t pid; ← Process id
- fork() → System call to creat a child
- 2 processe بيصير شغالين بعد fork() → Parent, Child
- fork بترجعله ال id تبع ال Child.
- fork بترجعله صفر. ← ما عنده أي Child
- return 1; ↳ error stream
- execlp → list of directory
- بصير ينفذ code ال child
- execlp → System call to load prog into Mem
- wait(NULL) → I will not get any Status from child.
- بيخلي ال Parent يروح على wait queue بينتظر
- بس ما ال Child تخلص بروح على ready وبينتظر دوره وبس يجي دوره بصير بال Running ويطبع "Child complete"

Process Creation – UNIX (Cont.)

- **fork()** system call creates new process
- **exec()** system call used after a **fork()** to replace the process' memory space with a new program
- Parent process calls **wait()** waiting for the child to terminate



← مش أنا ناديته تنفذ أول ما خلص ال Child تنفيذ.

Process Creation – Windows

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

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

   /* allocate memory */
   ZeroMemory(&si, sizeof(si));
   si.cb = sizeof(si);
   ZeroMemory(&pi, sizeof(pi));

   /* create child process */
   if (!CreateProcess(NULL, /* use command line */
    "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
    NULL, /* don't inherit process handle */
    NULL, /* don't inherit thread handle */
    FALSE, /* disable handle inheritance */
    0, /* no creation flags */
    NULL, /* use parent's environment block */
    NULL, /* use parent's existing directory */
    &si,
    &pi))
   {
     fprintf(stderr, "Create Process Failed");
     return -1;
   }

   /* parent will wait for the child to complete */
   WaitForSingleObject(pi.hProcess, INFINITE);
   printf("Child Complete");

   /* close handles */
   CloseHandle(pi.hProcess);
   CloseHandle(pi.hThread);
}
```


Process Creation – Windows (Cont.)

- Processes are created in the Windows API using the `CreateProcess()` function.
  - Similar to `fork()` in that a parent creates a new child process
  - However,
    - `fork()` has the child process inheriting the address space of its parent, `CreateProcess()` requires loading a specified program into the address space of the child process at process creation
    - `fork()` is passed no parameters, `CreateProcess()` expects no fewer than ten parameters
  - The first two parameters passed to `CreateProcess()` are the application name and command-line parameters
  - If the application name is NULL (as it is in this example), the command-line parameter specifies the application to load
  - `WaitForSingleObject()`, which is passed a handle of the child process—pi.hProcess—, waits for this process to complete

Process Termination

- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
  - Returns status data from child to parent (via **wait()**)
  - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the **abort()** system call (**TerminateProcess()** in Windows). Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting, and the operating system does not allow a child to continue if its parent terminates

Process Termination (Cont.)

- Some operating systems do not allow child to exist if its parent has terminated. If a process terminates, then all its children must also be terminated
  - **Cascading termination.** All children, grandchildren, etc., are terminated
  - The termination is initiated by the operating system
- In Linux and UNIX systems, we can terminate a process by directly using the **exit()** system call, providing an exit status as a parameter

```
exit(1);
```

- The C run-time library will include a call to exit() by default if it's not called directly
- The parent process may wait for termination of a child process by using the **wait()** system call. The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If no parent waiting (did not invoke **wait()**) process is a **zombie**
- If parent terminated without invoking **wait()**, process is an **orphan**

Android Process Importance Hierarchy

- Because of resource constraints such as limited memory, mobile operating systems often have to terminate processes to reclaim system resources such as memory.
- Rather than terminating an arbitrary process, Android has identified an **importance hierarchy** of processes → يقسم الـ Process لتصنيفات حسب أهميتها
- From **most** to **least** important: → من الأكثر للأقل أهمية بالترتيب
  - Foreground process → بالواجهة بتظهر
  - Visible process → background بس بالـ foreground بيستخدموا الـ
  - Service process → File لـ مثلا كنت متشغله قبل ولسا بعمل تحميل لـ app
  - Background process → مش ظاهرة للـ user ولا بتشوفها
  - Empty process → ما لهم يعمل اشي ماخذ مكان عالفاضي
- Android will begin terminating processes that are least important.
  - ↳ إنهاء

* ممكن بعض الـ Process تصنف نوعين مع بعض مثلا تكون Visible and Service

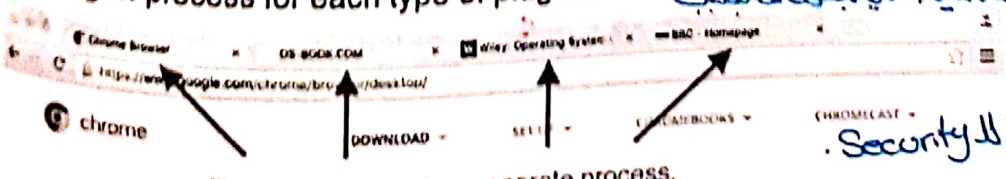
it is running a process that is not relating to the foreground process but it is aparent to the user

أنا كـ user بكون بشايفها.

↳ بتحبس أهميتها أعلى هلأنها تكون Visible بس.

Multi-process Architecture – Chrome Browser

- Many web browsers ran as single process (some still do) → زمان لا واحد ضربهم كلهم بضربوا
  - If one web site causes trouble, entire browser can hang or crash
- Google Chrome Browser is multi-process with 3 different types of processes: → هلأ صار هيك
  - **Browser** process manages user interface, disk and network I/O → (الأم) (الأوك)
  - **Renderer** process renders web pages, deals with HTML, Javascript. A new renderer created for each website opened → بفتح render all tab كل
    - Runs in **sandbox** restricting disk and network I/O, minimizing effect of security exploits
  - **Plug-in** process for each type of plug-in → Flash driver زي والأشياء اللي بتكون بحسب



منيحة الـ Security.

Each tab represents a separate process.

Inter-process Communication

Processes ← بهم يحكوا مع بعض.

35

Interprocess Communication

← هو بحاجة يحكوا مع بعض. بحاجة إنوم يحكوا مع بعض

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing → مشاركة المعلومات
  - Computation speedup → أقسم task لأكثر من Process للسرعة
  - Modularity → أقسم الـ task لـ modules.
  - Convenience → كشان الكود يكون أسهل للقراءة
- Cooperating processes need **interprocess communication (IPC)**
- Two models of IPC
  - **Shared memory** → (race condition) السيئة تاعونا . ← بنحير نتسابق مين حيعدل ويكتب
    - A region of memory that is shared by the cooperating processes is established →
  - **Message passing** (ميزتها انها أسرع)
    - Communication takes place by means of messages exchanged between the cooperating processes ← بحبوها لما يكون تحكي بهم يحكوا مع بعضهم من Machine مختلفة. Mechanisims أسهل

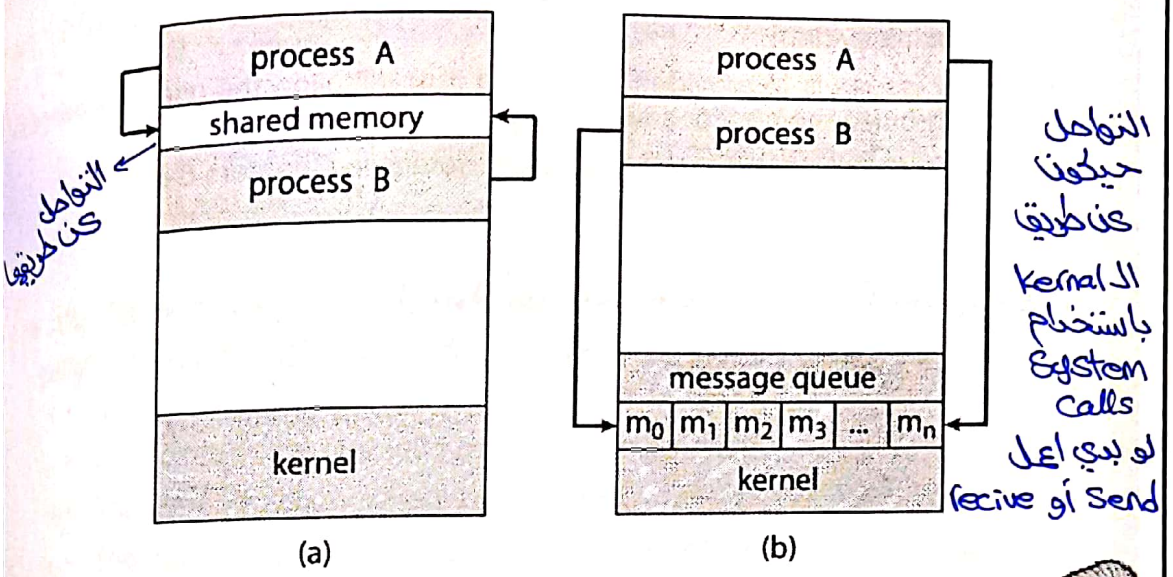
36

← ماعنا مساحة مشتركة بنحكي عن طريق الـ Msg بس مشكلتها إنها أبطأ.

Communications Models → device الاجهزة

(a) Shared memory.

(b) Message passing.



(a) (b)

التواصل كن طريقها

التواصل حيكون عن طريق kernal الـ باستخدام System calls لو بدي اعمل recive أو send

IPC in Shared-Memory Systems

IPC – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- Synchronization is discussed in great details in Chapters 6 & 7.

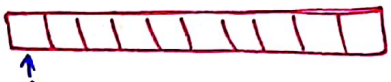
Producer-Consumer Problem

- Paradigm for cooperating processes:
  - ***Producer*** process produces information that is consumed by a ***consumer*** process
- One solution to the producer–consumer problem uses shared memory
  - Have available a buffer of items that can be filled by the producer and emptied by the consumer
- Two variations:
  - **unbounded-buffer** places no practical limit on the size of the buffer:
    - Producer never waits
    - Consumer waits if there is no buffer to consume
  - **bounded-buffer** assumes that there is a fixed buffer size
    - Producer must wait if all buffers are full
    - Consumer waits if there is no buffer to consume

.Consumer و Producer شايفينه التنين الـ buffer / in/out

Bounded-Buffer – Shared-Memory Solution

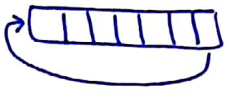
```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```



شبه الـ class بالجافا مش مهم التفاصيل.

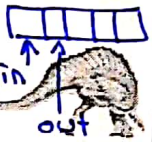
بالبداية in و out حيأشروا كنفس المكان يعني فاضي الـ buffer

bounded array (Circular Array)



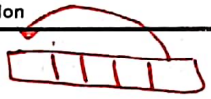
- The shared buffer is implemented as a circular array with two logical pointers: ***in*** and ***out***
  - ***in*** points to the next free position in the buffer → بالبداية بكون 0
  - ***out*** points to the first full position in the buffer.
  - The buffer is empty when `in == out`
  - The buffer is full when `((in + 1) % BUFFER SIZE) == out`
- Solution is correct, but can only use `BUFFER_SIZE-1` elements

بكون الـ buffer مليان لما الـ in كان قبل الـ out بوحدة



عملتها كشان

41



أخليه Circular بتعالج حالة لو كان الـ in عاخر Pointer فالـ out حيكون أول الـ array لإنه Circular.

ما بقدر أستخدم كل الـ Size تاع الـ buffer لازم يضل في وحدة empty تباشر عليه in.

Producer Process – Shared Memory

local variable

```
item next_produced;

while (true) {
    /* produce an item in next produced */
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
}
```

infinite loop

معناها الـ buffer full ما بقدر أحط لحد ما الـ Consumed يفضي و يحرك الـ out ويصير عندي مساحة أحط.

* الي بوقف ال producer إنه يكون ال buffer (full) والي بوقف ال consumer يكون ال buffer (empty)

Consumer Process – Shared Memory

```
item next_consumed;

while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */
}
```

*بحالة ال Full دايمًا في element مش معبى ، مابقدر أمشي ال in خطوة ويصير بساوي ال out .

*هاي الحالة ما فيها Race conditions لأنه ال Race ما بصير بال local variables بصير بال Shared var .

ال buffer وال in وال out هم ال Shared ،

- in=out بس لما يكون ال buffer empty ← فما في overwrite عال buffer
- ال Producer بعمل update لل in وال Consumer لل out فما في بينهم (Race conditions)

What about Filling all the Buffers?

- Suppose that we wanted to provide a solution to the consumer-producer problem that fills **all** the buffers.
- We can do so by having an integer **counter** that keeps track of the number of full buffers. (counter → Shared variable)
- Initially, **counter** is set to 0.
- The integer **counter** is incremented by the producer after it produces a new buffer.
- The integer **counter** is decremented by the consumer after it consumes a buffer.

Producer

```
while (true) {
        /* produce an item in next produced */

        while (counter == BUFFER_SIZE)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
        counter++;
}
```

→ حالة الـ Full

صرنا نعتمد عالcounter لتحديد الbuffer لو فاضي أو مليان مش عال in وال out.

Consumer

```
while (true) {
        while (counter == 0)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        counter--;
        /* consume the item in next consumed */
}
```

→ Empty

→ لحد ما يصير في Item جوا الbuffer

* هيك حلينا مشكلة إنه فيه element بضل فاضي بحالة الFull بس صار فيه Race condition لإنه الثنين بعدلوا عالcounter

Race Condition

- `counter++` could be implemented as

load ← `register1 = counter`
increment ← `register1 = register1 + 1`
Store ← `counter = register1`

- `counter--` could be implemented as

load ← `register2 = counter`
decrement ← `register2 = register2 - 1`
Store ← `counter = register2`

- Consider this execution interleaving with "count = 5" initially:

S0: producer execute `register1 = counter` {register1 = 5}
S1: producer execute `register1 = register1 + 1` {register1 = 6}
S2: consumer execute `register2 = counter` {register2 = 5}
S3: consumer execute `register2 = register2 - 1` {register2 = 4}
S4: producer execute `counter = register1` {counter = 6 }
S5: consumer execute `counter = register2` {counter = 4}

الثاني قرأ ال counter قبل ما تتعدل
وهيك صار incorrect لإنه كان لازم تتعدل 5 .

Race Condition (Cont.)

- Question – why was there no race condition in the first solution (where at most N – 1) buffers can be filled? because they are not modify the same variables.
- More in Chapter 6.

IPC – Message Passing

- Processes communicate with each other without resorting to shared variables
- Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space
  - It is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network
- IPC facility provides two operations:
  - `send`(*message*)
  - `receive`(*message*)
- The *message* size is either fixed or variable
  - Fixed-sized messages: straightforward system-level implementation, the task of programming is more difficult
  - Variable-sized messages: complex system-level implementation, the programming task is simpler.

Message Passing (Cont.)

- If processes *P* and *Q* wish to communicate, they need to:
  - Establish a ***communication link*** between them
  - Exchange messages via send/receive
- Implementation issues:
  - How are links established?
  - Can a link be associated with more than two processes?
  - How many links can there be between every pair of communicating processes?
  - What is the capacity of a link?
  - Is the size of a message that the link can accommodate fixed or variable?
  - Is a link unidirectional or bi-directional?

Implementation of Communication Link

- Physical:
  - Shared memory
  - Hardware bus
  - Network
- Logical:
  - Direct or indirect
  - Synchronous or asynchronous
  - Automatic or explicit buffering

مكان بتقعد فيه الـ msg وواحد ببعت وواحد بستقبل من هالمكان.

device نفس الـ

ممكن يكونوا 2 different devices

هاد اللي بهمنا

51

* الـ direct مشكلته إنه لو Process عملت عالـ id تاعها فلازم نعدل كل شي عالجديد.

الهم علاقة بالـ naming ، هلأ لما بدي أبعت msg بدي أحدد اسم المستقبل والعكس ولا لأ.

Direct Communication

- Processes must name each other explicitly:
  - ***Symmetry*** in addressing; both the sender process and the receiver process must name the other to communicate
    - **send** (*P, message*) – send a message to process P
    - **receive**(*Q, message*) – receive a message from process Q
  - ***Asymmetry*** in addressing; only the sender names the recipient; the recipient is not required to name the sender
    - **send**(P, message) – send a message to process P
    - **receive**(id, message)—Receive a message from any process. The variable id is set to the name of the process with which communication has taken place.
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

2 process بين link كل بكون بس

بنيكم 2 process كل 1 link

52

واحد ببعت وواحد بستقبل

الاثنين ببعتوا وبستقبلوا

Indirect Communication

→ like Ports

- Messages are directed and received from mailboxes (also referred to as ports)
  - A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

كتير mailbox كن طريقه بنقدر الـ Processes تبعت أو تستقبل
لو كان كتير الـ لا تبع الـ mailbox .

53

→ by default → owner بكون هو الـ

owner بكون هو الـ by default ←

* الـ mailbox ممكن تكون Created من قبل الـ OS أو من قبل الـ User program .

Indirect Communication (Cont.)

- Operations
  - Create a new mailbox (port)
  - Send and receive messages through mailbox
  - Delete a mailbox

] owner بعمله الـ

- Primitives are defined as:
  - **send**(*A, message*) – send a message to mailbox A
  - **receive**(*A, message*) – receive a message from mailbox A

* بالـ indirect أنا لما ابعت الـ msg ما بكون محددة
مين اللي حيستقبلها.

54

Indirect Communication (Cont.)

"worst case"

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

من ناحية الاستقبال فممنوع ال msg يستقبله اثنين لازم بس واحد.

أخلي ال mailbox بين 2 processes فقط.

يعني بالنهاية بنطبيه لواحد بس ، اثنين بسروا receive و system بخلي وحدة بس اللي تاخذ الرسالة

ما أسمح لـ 2 processes يعملوا receive بنفس الوقت وهذا صعب.

* بال indirect بنعرف مين اللي أرسل بس ما بنقدر نحدد المستقبل أو مين اللي يستقبل بالزبط.

Synchronization

Message passing may be either blocking or non-blocking

* **Blocking** is considered **synchronous** → ما بكمل إلا ليوصله انه كل شي وصل.
  - **Blocking send** -- the sender is blocked until the message is received
  - **Blocking receive** -- the receiver is blocked until a message is available
* **Non-blocking** is considered **asynchronous** → بكمل شغل ما بستنى
  - **Non-blocking send** -- the sender sends the message and continue
  - **Non-blocking receive** -- the receiver receives:
    - A valid message, or → لو في مسج اعملو receive
    - Null message → لو ماني اشي اعملو receive
- Different combinations possible
  - If both send and receive are blocking, we have a **rendezvous**

لـ بنستنى بعض احنا اثنين

حسب شو بتطلب ال sys تبعي

Producer-Consumer: Message Passing

- Producer

```
message next_produced;
while (true) {
  /* produce an item in next_produced */

  send(next_produced);
}
```

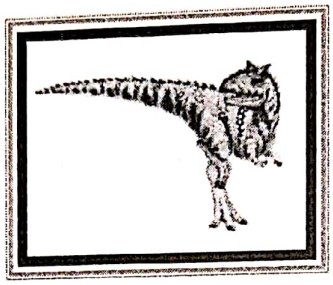
- Consumer

```
message next_consumed;
while (true) {
  receive(next_consumed)

  /* consume the item in next_consumed */
}
```

Buffering → وين الصبح بتقعد

- Queue of messages attached to the link.
- Implemented in one of three ways
  1. Zero capacity – no messages are queued on a link. → ما في buffer فما بقدر أبعت اشي إلا لما الـ receiver يوصله.
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full → يكون الحجم محدود.
  3. Unbounded capacity – infinite length
     Sender never waits → مابستنى ما في حجم محدود وما في control والبرنامج تاعها complex.

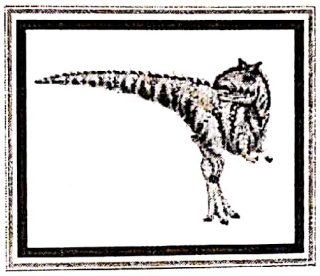
End of Chapter 3



Posix ح نركز فيها على الـ

Chapter 4: Threads & Concurrency

كلها بخطوط عالك ح
بخطوط الـ Parallism



Outline

- Overview (4.1)
- Multicore Programming (4.2)
- Multithreading Models (4.3)
- Thread Libraries (4.4)
- Implicit Threading (4.5)
- Threading Issues (4.6)
- Operating System Examples (4.7)

Objectives

- Identify the basic components of a thread, and contrast threads and processes
- Describe the benefits and challenges of designing multithreaded applications
- 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
- Design multithreaded applications using the Pthreads, Java, and Windows threading APIs

3

Overview

4

(multithread Software) * صار العبئ على الـ Programmer إنه يعمل الـ

Motivation

إنه صار الـ hardware تبعنا (multithreaded) (multicore)

- Most modern applications are multithreaded
- Threads run within application → your application is running as multiple threads running at the same time.
- Multiple tasks within the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request

لأننا نشفنا لما نعمل fork كيف بياخذ مساحة كبيرة فلذلك طلع الـ multithread لأنه بيشارك المساحة.

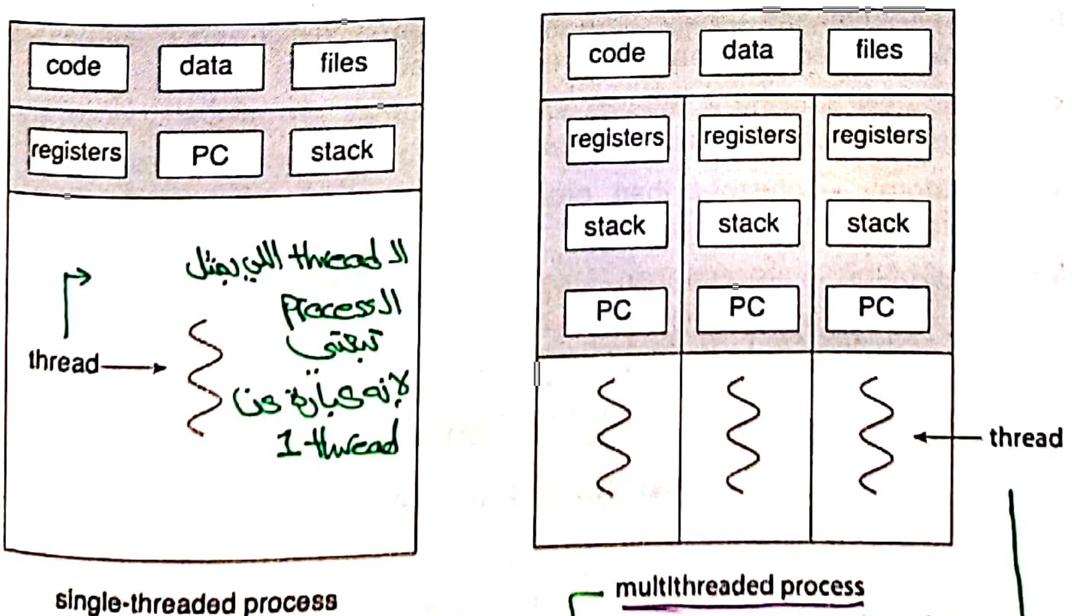
(هلأ صار الـ multithread أساسي ومهم مش إضافي أو اختياري)

- Process creation is heavy-weight while thread creation is light-weight
- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack. → كل thread بيكون اله كل اشي خاص فيه مثل ID و register ولكن بيشاركوا مع الـ other threads اللي بنفس الـ Process
  - It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

* الـ threads هي اللي بتكون running على الـ OS وممكن الـ Core يحوي أكثر من threads.

يعني الـ OS نفسه بيشتغل بحيث إنه الـ code تبعه ما بشتغل كـ Processes بالعكس لو عندي Process الها multiple task و in parallel بتعمل الـ threads.

Single and Multithreaded Processes



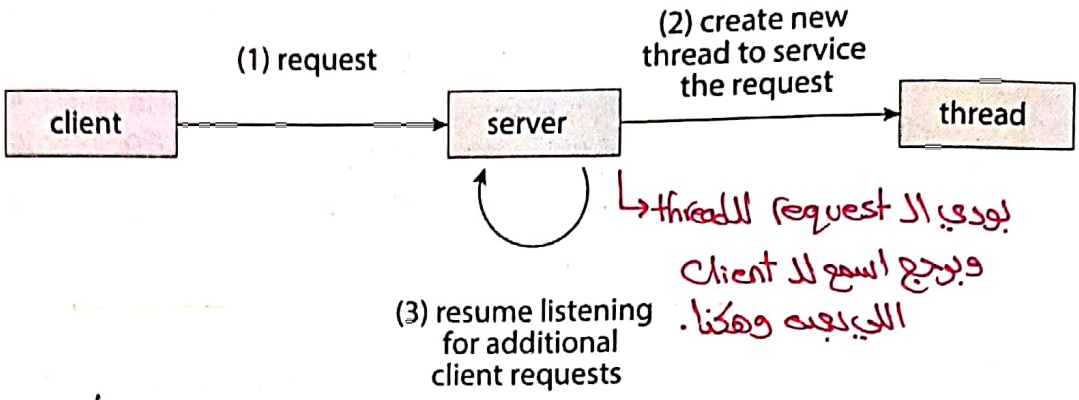
هاي كلها عبارة عن Process وحدة بس multithreaded.

كل thread بيكون الها stack و registers و PC خاص فيه

كل الـ thread بيشاركوا الـ code والـ data والـ files.

Multithreaded Server Architecture

(مثال حال multithreading)



↳ thread الـ request بودي الـ
وبرجع اسمع لـ Client
اللي بعده وهكذا.

الـ Server بستنى الـ Client يطلبوا منه أشياء
ويكون كل Client بعمل request بدله بـ Response
كليه.

* ما بنبط الـ Server يخلي الـ Client الباقي يستنوا
لحدما هو يخلص من اللي معاه لذلك بيتم استخدام
الـ multithread.

Benefits

يعني ما حضل أوقف واستنى ليخلص اللي قبلي ←

- **Responsiveness** – may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing** – threads share resources of process, easier than shared memory or message passing → لأنه بوفر مساحة مش كل thread حيحتاج مساحة لحاله.
- **Economy** – cheaper than process creation, thread switching lower overhead than context switching
- **Scalability** – process can take advantage of multicore architectures

↳ عدد الـ cores كلما كانوا اكثر كلما
الـ Server اشتغل اسرع
(multithreaded لما يكون)

(اما لو كان Single Process قد ما زيدت ما رح يسرّع الـ Server
يعني).

يعني ← الـ code والـ data ما بحتاج اعملهم duplicate لو كانوا نفس الاشي بالـ thread بيشاركوا بهن

Multicore Programming

9

* عشان صار عندي CPU's multicore وهيك صارت الحاجة لنستخدم الـ multithreads.

Multicore Programming

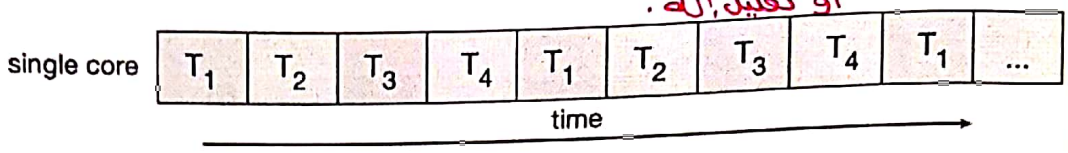
- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities** → كيف أقسم الـ code تبعي ليشتغلوا
  - **Balance** → كيف أعمل توازن
  - **Data splitting** → كيف أقسم الداتا صح
  - **Data dependency** → ما يكون بين الداتا اعتماد
  - **Testing and debugging** → صعب يكون عم بتختبر كتير اشي
- ***Parallelism*** implies a system can perform more than one task simultaneously → يكون عندي أكثر من "task running at the same time"
- ***Concurrency*** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

أكثر من activity شغالة أنا بشوفهم كلهم بنفس الوقت
بس بالأصل ما بشتغلوا سوا بس الـ Switching بينهم سريعة جدا
عشان هيك تبعيالك إنهم بيشتغلوا بنفس الوقت.

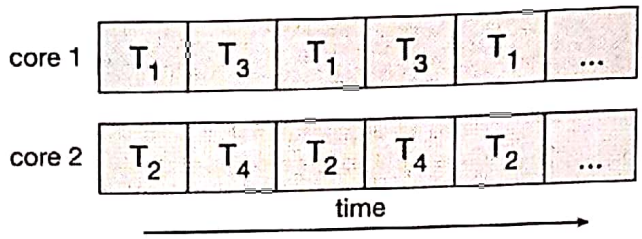
10

Concurrency vs. Parallelism

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



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



(T 8 means thread)

11

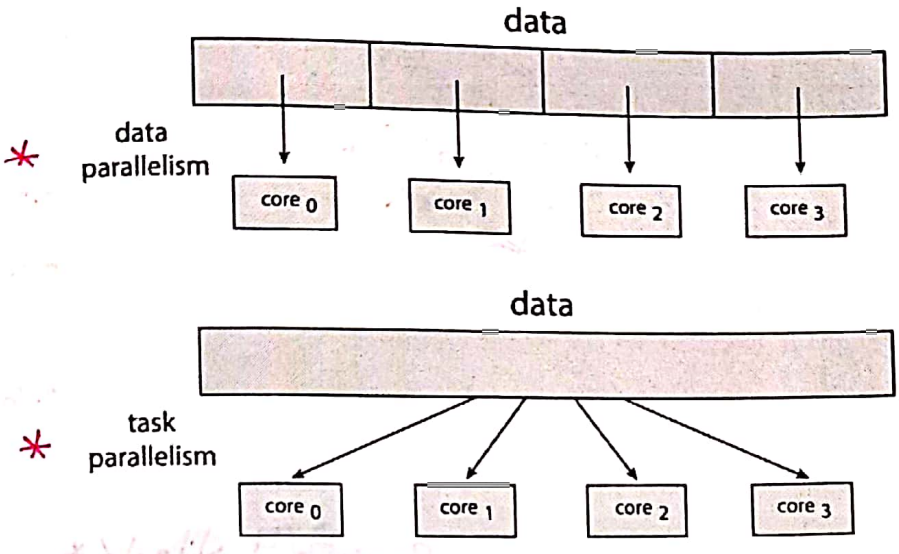
Multicore Programming

- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each (Same task)
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation

(different task with Same data or without Same data)

12

Data and Task Parallelism



13 * الـ ideal case انه يكون كل الـ code تاعك Parallism وهذا مش موجود بكون وقتها الـ S=0) وهذا مش صح لإنه مستحيل يكون هيك. (ideal case → speedup = N

Amdahl's Law → شو الـ boundary اللي بتحد الـ speedup تاعك؟

Parallel ← أكثر سرعة بقدر يكون فيها

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- *S* is serial portion → قديه الجزء من الـ code تاعك اللي ما فيه Parallism
- *N* processing cores → عدد الـ cores اللي موجودين عالجهاز

اللي بتتحكم بـ Amdahl's law

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

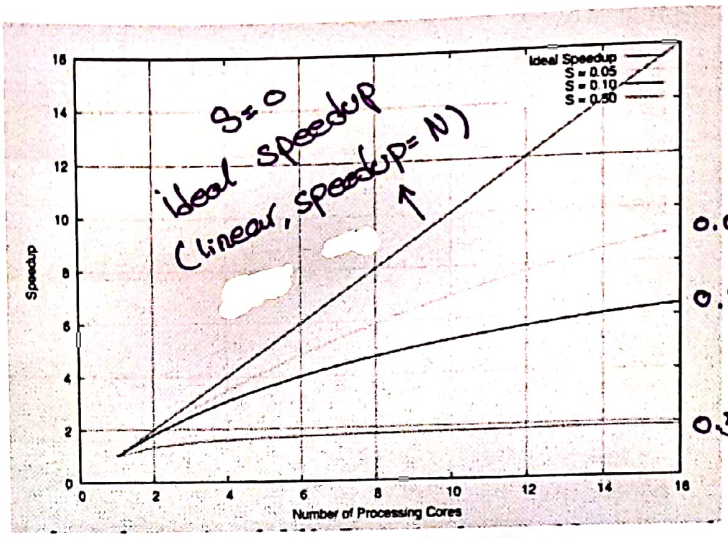
→ أكثر speedup بنقدر نوصله

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times ↳ S = 0.25 ↳ N = 2
- As *N* approaches infinity, speedup approaches 1 / *S*

Serial portion of an application has disproportionate effect on performance gained by adding additional cores

- But does the law take into account contemporary multicore systems? yes

Amdahl's Law



S=0
Ideal speedup
(Linear, speedup= N)

0.05 = S

0.10 = S

0.50 = S

* maximum speedup that you can obtained

15

Multithreading Models

16

* بال Multithreads بنقدر نعملها بال User و ال Kernel .

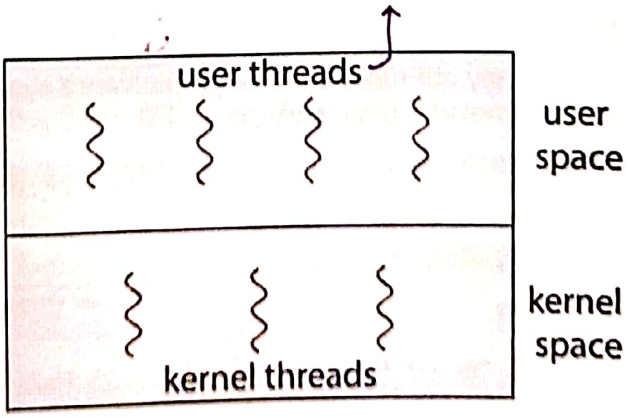
User Threads and Kernel Threads

↱ المكان اللي بتنفذ فيه ال application تبعتك

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads** → الموجودة اللي حنشرحها
  - Windows threads
  - Java threads → بال Np بناخنها
- **Kernel threads** - Supported by the Kernel → بتنفذ فيها ال system calls اللي بعملها البرنامج
- Examples – virtually all general -purpose operating systems, including:
  - ✓ Windows
  - ✓ Linux
  - ✓ Mac OS X
  - ✓ iOS } for handheld devices
  - ✓ Android

User and Kernel Threads

هل بقدر أنادي مثلاً أي حدا من ال Kernel وكيف ؟



Multithreading Models

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

19

kernal thread الـ threads بشاركوا نفس أكثر من ← مشكلتها

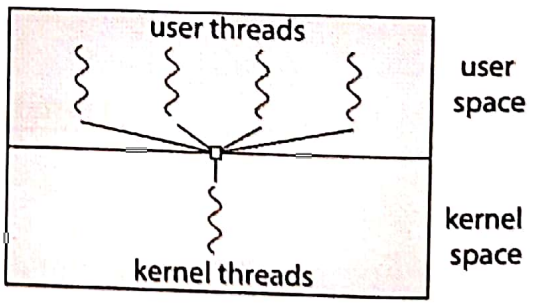
Many-to-One

User thread أكثر من بشاركوا نفس kernal thread الـ ومخصصين اله بس.

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block : أول مشكلة
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time : ثاني مشكلة
- Few systems currently use this model → مش كتير موجود حاليا
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**

← لو واحد عمل block للشي الباقي حينعملوم block .

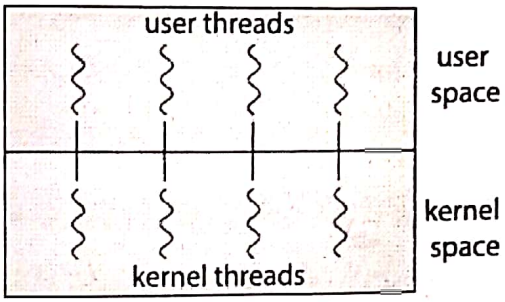
ماي OS كامل محسنه بهاي الطريقة ، هدول الأمثلة لأجزاء فقط .



20

One-to-One "موجود بكثرة"

- 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
- Examples
  - Windows
  - Linux



* أغلبية الـ OS بتتحكم بعاي المشكلة وكمان حل عندنا لما صارت multi cores فما صارت المشكلة كبيرة.

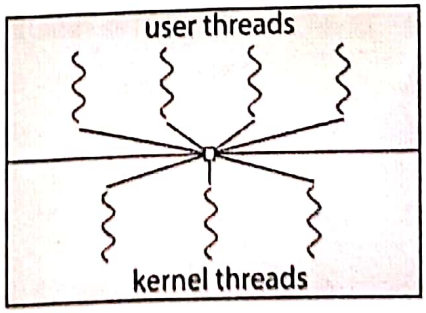
مشكلته لو حبيت عندك كثير user threads حتاج لكل واحد kernel thread خاص فيها.

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 → بكون عدد الـ kernel بالعادة أقل من الـ user
- Windows with the *ThreadFiber* package
- Otherwise not very common

لأنه هاي الطريقة بتخيف صعوبة عنا.

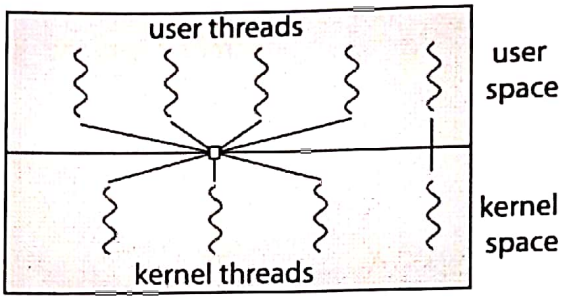
* ممكن يكون فيه مشكلة الـ blocking بس مش قد الـ Many-to-one



ما في mapping معين كل لكل

Two-level Model → مش كتير موجود برضه.

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



* برضه ما بيستخدم النظام كامل ممكن أجزاء من النظام تكون بتستخدم هاي الطريقة.

Thread Libraries

multithreading programs بمكتوبي أكتب

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  1. Library entirely in user space → function والـ code والـ data كلها موجودة بالـ user space ما بتعتمد على الـ kernal
     - Invoking a function in the library results in a local function call in user space and not a system call
  2. Kernel-level library supported by the OS → كلشي بكون بالـ kernal فكلشي عن طريق الـ system call
     - Code and data structures for the library exist in kernel space
     - Invoking a function in the API for the library typically results in a system call to the kernel
- Three main thread libraries are in use today: POSIX Pthreads, Windows, and Java
  - ↳ totally user level.
  - totally kernal level
  - . user level وفيها kernal level وفيها

Pthreads → Standard

↳ أكثر هيدا النوع Implementation

- 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*** → كل حد بعمل اللي بده ياه حسب اللي ماشيها
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Linux & Mac OS X)

" مثال لـ Linux "

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);
}
```

Parent thread

→ Id for thread

يعرف thread

نفس أعمل فيه كل الـ attributes تبعت الـ thread من Security وغيرها.

address

هون تاركينه عالـ default بس لازم نعرفها.

هون استنى الـ thread ليخلص، وممكن يكون ما بستنى و تنينهم بشتغلوا بس هون مش هيك

* بحدد الـ Function اللي بدي اياه يتنفذ لما الـ thread يشتغل

ما بقدر أطبع الـ sum غير لما يخلص الـ thread من خلال الـ join

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);
}
```

بحط فيه الـ code لما الـ thread ينفذ.

Pointer برجع لاشي أنا مش محددة نوعه.

Pointer لـ data شو ما كان نوعها.

حتجيلي الـ Size → بحول من String لـ integer

global Variable رح يشوفوا الـ thread والـ Parent نبعي اللي بلشنا.

* ليه ما بلش من 1 مش صفر؟ لأنه هو رح يعمل Sum مابده الصفر شو حيفرق معه

* ليه المساواة؟

زي الـ return

Ex 8 thread 8 → حيحولها لـ int والـ upper حيصير فيه 8 فح يجمع من 8 → 1 عشان هيك كنسل الصفر وحط مساواة.

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);
```

لو كان 10 threads مثلا وحدة.

هون لازم يعمل creation طبقًا وأقسم لكل thread شغلها.

لهيك ما حيطلع من الـ loop إلا لما يخلص كل الـ threads.

نفس المثال السابق بس هون بالـ Windows.

Windows Multithreaded C Program

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* The thread will execute in this function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 1; i <= Upper; i++)
      Sum += i;
   return 0;
}
```

int هنا بالـ windows

global

double word

like Void*

حيحس فيزيا العدد اللي بدي أحطه

لحد الرقم اللي بدي أحطه

Windows Multithreaded C Program (Cont.)

```
int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;

   Param = atoi(argv[1]);
   /* create the thread */
   ThreadHandle = CreateThread(
      NULL, /* default security attributes */
      0, /* default stack size */
      Summation, /* thread function */
      &Param, /* parameter to thread function */
      0, /* default creation flags */
      &ThreadId); /* returns the thread identifier */

   /* now wait for the thread to finish */
   WaitForSingleObject(ThreadHandle,INFINITE);

   /* close the thread handle */
   CloseHandle(ThreadHandle);

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

reference زي اشي بمسك منه الـobject

بقرأ من الـuser الرقم اللي بدي أجعله منا لإلك.

خليها على الـdefault

it will wait anything

بدل join

لـ اللي حستناه

Time limit (قديه بستناه) هون يعني استناه للأبد.

* لأنه الـhardware صار multicore نبعت Software لزم يستفيد منه ، مرات مشان يقللوا العبء على الـProgramer انه هو يعمل كل شي بخلوا الـCompiler لحاله يعمل threading . أو عن طريق الـrun time library .

Implicit Threading

Implicit Threading

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Five methods explored
  - Thread Pools
  - Fork-Join
  - OpenMP
  - Grand Central Dispatch
  - Intel Threading Building Blocks

Supported by Sun API's

Thread Pools

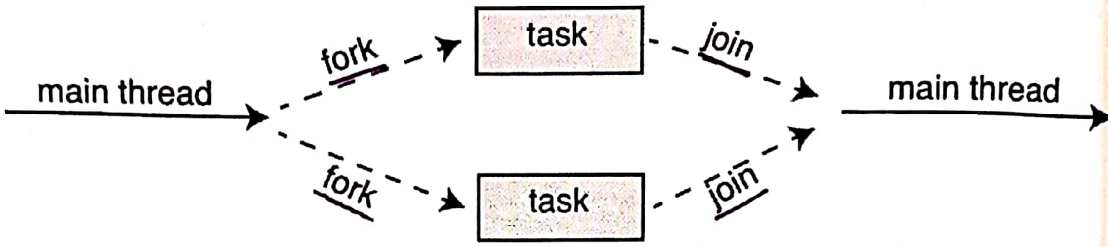
- Create a number of threads in a pool where they await work → كل ما تحتاج thread بتقام من ال pool وبس يخلص وكل اشي يرجع لل pool.
- Advantages:
  1. Usually slightly faster to service a request with an existing thread than create a new thread
  2. Allows the number of threads in the application(s) to be bound to the size of the pool → لو ال thread بال Pool خلصوا ما حيسمحلك تعمل أكثر
  3. Separating task to be performed from mechanics of creating task allows different strategies for running task → لإنك كم تفصل ال task عن ال thread اللي بتعمل فيها هونا بيعطيك flexibility.
     - i.e. Tasks could be scheduled to run periodically
- Windows API supports thread pools:

```
DWORD WINAPI PoolFunction(AVOID Param) {
   /*
   * this function runs as a separate thread.
   */
}
```

Supported by API's

Fork-Join Parallelism

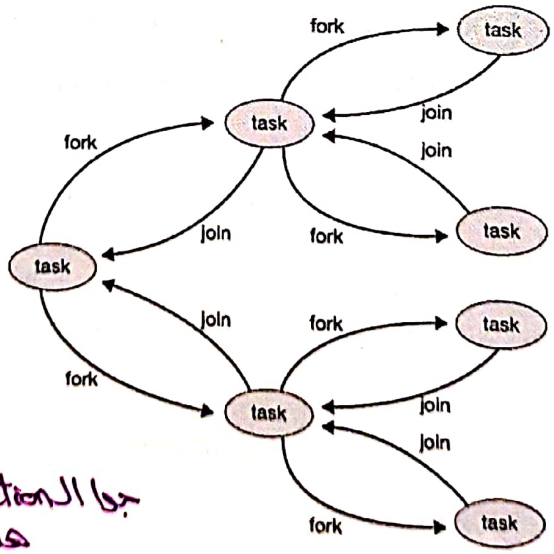
- Multiple threads (tasks) are **forked**, and then **joined**.



لـ مش أنا بعملها بتنعمل جوا الـ API .

Fork-Join Parallelism

Java



جوا الـ Functions تبع الـ API هيك بيعمل .

هون الـ multithread مش عمستوى الـ application عمستوى الـ Function داخل الـ API.

OpenMP → الشي بنعمله include جوا الـ code تبعك

- Set of compiler directives and an API for C, C++, FORTRAN
- Provides support for parallel programming in shared-memory environments
- Identifies **parallel regions** – blocks of code that can run in parallel

#pragma omp parallel

Create as many threads as there are cores

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

int main(int argc, char *argv[])
{
  /* sequential code */

  #pragma omp parallel
  {
    printf("I am a parallel region.");
  }

  /* sequential code */

  return 0;
}
```


OpenMP (Cont.)

- Run the for loop in parallel

الـ for اللي بده يعملوا parallel

```
#pragma omp parallel for
for (i = 0; i < N; i++) {
   c[i] = a[i] + b[i];
}
```

نفس الاشي هون الـ Compiler لحاله بمسك الـ for loop وبقسمه على الـ threads ويخلي الـ threads على الـ for loop تشتغل in parallel

*ممكن أو وارد يكون في مشاكل بالـ data بدك تحاول ما يصير هيك.

Grand Central Dispatch

- Apple technology for macOS and iOS operating systems
- Extensions to C, C++ and Objective-C languages, API, and run-time library
- Allows identification of parallel sections
- Manages most of the details of threading
- Block is in "^{ }" :

```
^{ printf("I am a block"); }
```

→ ال block الي بدي أعمله Parallel

- Blocks placed in dispatch queue
  - Assigned to available thread in thread pool when removed from queue

* بيشبه ال Pragma بس معمول ل macOS و iOS

Grand Central Dispatch

- * Two types of dispatch queues: → first in first out
  1. **serial** – blocks removed in FIFO order, queue is per process, called **main queue** بالترتيب بعملهم
     - Programmers can create additional serial queues within program
  2. **concurrent** – removed in FIFO order but several may be removed at a time → ممكن يمحي وحدة والي وراها
     - Four system wide queues divided by quality of service:
       - QOS_CLASS_USER_INTERACTIVE
       - QOS_CLASS_USER_INITIATED
       - QOS_CLASS_USER_UTILITY
       - QOS_CLASS_USER_BACKGROUND

Grand Central Dispatch

- For the Swift language a task is defined as a closure – similar to a block, minus the caret
- Closures are submitted to the queue using the `dispatch_async()` function:

```
let queue = dispatch_get_global_queue
   (QOS_CLASS_USER_INITIATED, 0)

dispatch_async(queue,{ print("I am a closure.") })
```

مبنية بال Intel ←

Intel Threading Building Blocks (TBB)

- Template library for designing parallel C++ programs
- A serial version of a simple for loop

```
for (int i = 0; i < n; i++) {
   apply(v[i]);
}
```

- The same for loop written using TBB with `parallel_for` statement:

← تحدد الfunction اللي بدي اعملها Parallel .

```
parallel_for (size_t(0), n, [=](size_t i) {apply(v[i]);});
```

* بهمنا نعرف انه في Intel threading بنعمل .

Threading Issues

Threading Issues

نشوفات لازم تقرأها ← بما إنه صار كنا multithreading

- Semantics of **fork()** and **exec()** system calls
- Signal handling ← Process نتبعك هو اللي سببها
  - Synchronous and asynchronous → البرنامج شغال عادي بس ال user هو اللي عمل اشي
- Thread cancellation of target thread
  - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations

* كل وحدة منهم حناخذ الفكرة العامة منها *

Child لما نخلق System call → system call

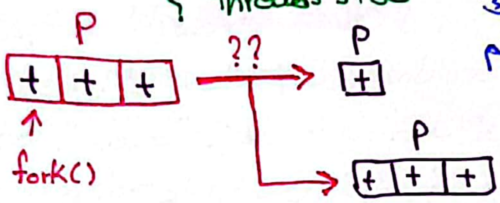
لما بدنا الـ Process تعمل load Program معين instead of الـ Program الي موجود حاليا بالـ Mem تبعها.

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 → execute الـ thread الي نادى الـ System call فإذا واحد من الـ text area خاص الي Shared للكل رح بيعمل load للـ Program فيها فكل الـ thread حيشوفوا الـ exec الي انعملت.

لازم أحدد لو thread عمل fork هل لازم أعمل Process فيها بس الي نادت الـ fork ولا أعملها مطابقة للـ Process الأم وأحط كل الـ threads ؟

الجواب هنا الـ OS بعضهم بمكنوك تعمل الخيارين تعمل واحد منهم حسب شو بدك انت.



45

أي اشي بده يتخبر فيه الـ Process تبعتي ببعت كـ Signal مثل الـ interrupt.

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 → حيصير حدث بسبب حدوث الـ Signal.
  2. Signal is delivered to a process
  3. Signal is handled by one of two signal handlers:
     1. default → لكل event في deafult
     2. user-defined → بقدر اكتب فوق الـ default الـ handler الخاص فيني
- 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

الـ Process بتتكون من thread وحدة هو الي بتعمل handling.

6

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

* إذا كان في thread معين هو اللي سبب ال Signal زي إنه يكون ال Signal ← Synchronous و thread معين اللي سبب هاي ال Signal عالأغلب ال thread لحاله هو اللي بنعمله delivering لل Signal اللي صارت.

* في حالات ثانية مثل انك علقت ال Program لل Ctrl+z تاعك معناها كل ال threads اللي جوا ال process لازم ينعمل deliver لل Signal تبعهم.

* في حالات بعض ال threads بنعمل deliver لل Signal تاعهم.

* أو بكون انت بالبرنامج عامل إنه كل الإشعارات تيجي لـ Specific thread هو اللي يعمل handling لل Signal اللي صارت.

← إنه يكون عندي thread شغال وبحاول thread ثاني يعمله Cancel وعالأغلب هو ال Parent thread.

Thread Cancellation

- Terminating a thread before it has finished
- Thread to be canceled is target thread
- Two general approaches:
  - 1 * Asynchronous cancellation terminates the target thread immediately → ممكن اعمل Cancel وال thread ما بعرف وين وصلت
  - 2 * Deferred cancellation allows the target thread to periodically check if it should be cancelled → مؤجل، مش لما تطلب ينعمل Cancel دغري حستجيب، لا
- Pthread code to create and cancel a thread:

بالحظة اللي ال thread الخارجي يطلب إنه ينعمل Cancel ← لل thread حينهي تنفيذ هنا ال 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);
```

offset ← (0) ؛ لـ اسم ال Funct (worker)

ما بعمل ال Cancel لل thread إلا لو هو طالب ينعمل Check في حد عامله Cancel أو لا ولو في بنعمل ال Cancellation

فانته بسمح لل thread يعمل كل شغله بسلام وبعدين هو بيشيك اذا في طلب Cancellation وقتها بوافق ينعمله.

Posix

3- حالة ال off : ما بقدر نعمله Cancel.

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

أنا بنقي بالـcode الـ thread يكون على أي Mode منهم

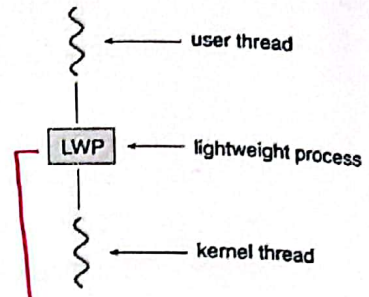
- If thread has cancellation disabled, cancellation remains pending until thread enables it → Cancellation الـthread بيستقبل ممكن يرجع
- Default type is deferred — بعد فترة معينة فلو كان طالب Cancel
  - Cancellation only occurs when thread reaches **cancellation point**
    - i.e., `pthread_testcancel()` — Enabled زمان وهو off بس يحين حيشوف هذا الطلب.
    - Then **cleanup handler** is invoked
- On Linux systems, thread cancellation is handled through signals

Thread-Local Storage

- **Thread-local storage** (**TLS**) allows each thread to have its own copy of data → Share أعرف data تكون ملكية الـthread تبعي مش
- Useful when you do not have control over the thread creation process (*i.e.*, when using a thread pool)
- Different from local variables
  - Local variables visible only during single function invocation
  - TLS visible across function invocations
- Similar to `static` data
  - TLS is unique to each thread

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Typically use an intermediate data structure between user and kernel threads – **lightweight process (LWP)**
  - Appears to be a virtual processor on which process can schedule user thread to run
  - Each LWP attached to kernel thread
  - How many LWPs to create?
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the **upcall handler** in the thread library
- This communication allows an application to maintain the correct number kernel threads



Operating System Examples

Operating System Examples

- Windows Threads
- Linux Threads

Windows Threads

- Windows API – primary API for Windows applications
- Implements the one-to-one mapping, kernel-level → هنتي بالـuser
  - Each thread contains
    - A thread id
    - Register set representing state of processor
    - Separate user and kernel stacks for when thread runs in user mode or kernel mode
    - Private data storage area used by run-time libraries and dynamic link libraries (DLLs)
- The register set, stacks, and private storage area are known as the **context** of the thread

↳ الحوائج الـ thread اما بتحكي

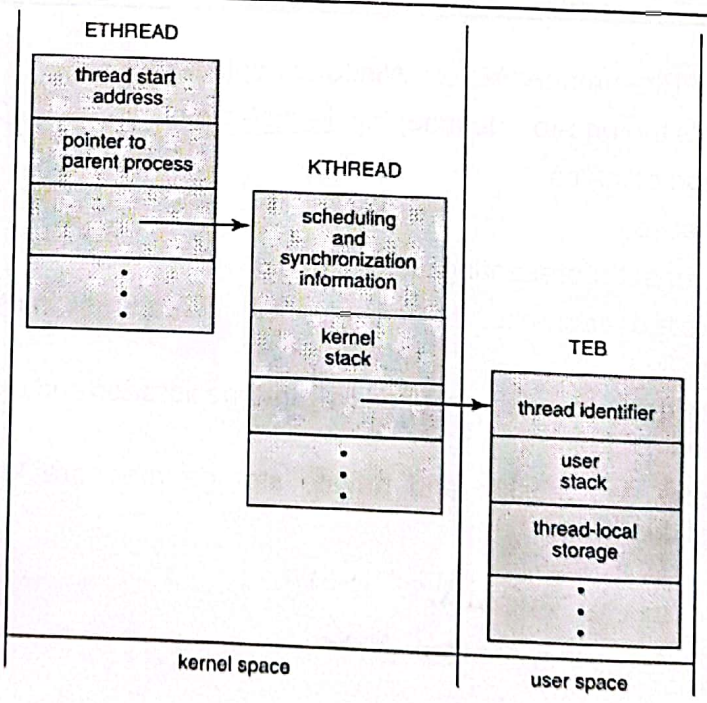
Process والحوائج الـ thread الـ كن

Process الـ بتحكي عالـ

Windows Threads (Cont.)

- The primary data structures of a thread include:
  1. ETHREAD (executive thread block) – includes:
     - Pointer to process to which thread belongs
     - Address of the routine in which the thread starts control
     - pointer to the corresponding KTHREAD
  2. KTHREAD (kernel thread block) – includes:
     - Scheduling and synchronization info
     - Kernel-mode stack
     - Pointer to TEB
  3. TEB (thread environment block) – includes:
     - Thread id
     - User-mode stack
     - Thread-local storage

Windows Threads Data Structures



Linux Threads

غير الـ API والـ Posix

- Linux refers to them as ***tasks*** rather than ***threads***
- Thread creation is done through `clone()` system call
- `clone()` allows a child task to share the address space of the parent task (process) → بتعطي flexibility أكتر من الـ fork لأنها
  - Flags control behavior الـ Sharing وبتتحكم بالـ

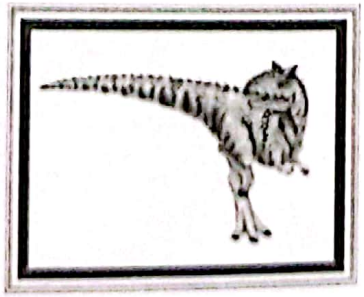
flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

أي الـ clone من الـ fork

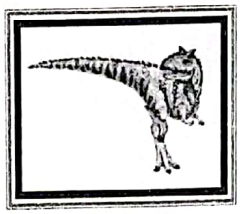
- `struct task_struct` points to process data structures (shared or unique)

process (fork) ↓ thread ↓

End of Chapter 4



Chapter 5: CPU Scheduling



Outline

- Basic Concepts (5.1)
- Scheduling Criteria (5.2)
- Scheduling Algorithms (5.3)
- Thread Scheduling (5.4)
- Multi-Processor Scheduling (5.5)
- Real-Time CPU Scheduling (5.6)
- Operating Systems Examples (5.7)
- Algorithm Evaluation (5.8)

one Process, one core

بناء عليها بحكم لو هاي الـ Scheduling منيحة ولا مو منيحة.

Objectives

- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on scheduling criteria
- Explain the issues related to multiprocessor and multicore scheduling
- Describe various real-time scheduling algorithms
- Describe the scheduling algorithms used in the Windows, Linux, and Solaris operating systems
- Apply modeling and simulations to evaluate CPU scheduling algorithms

Basic Concepts

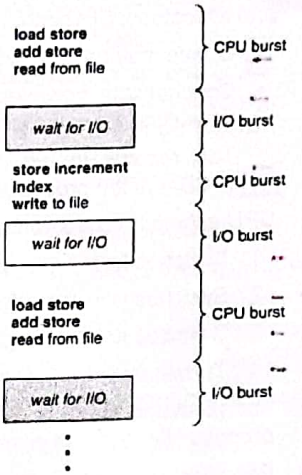
بالوضع الطبيعي، إذا عندي أكثر من برنامج شغال على الـ computer حيتهأي إنه كلهم شغالين بنفس الوقت.

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- Process execution consists of a cycle of CPU execution and I/O wait.
- Processes alternate between these two states.
- Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on.
- Eventually, the final CPU burst ends with a system request to terminate execution.

operations

*مابيصير الـ cpu يكون ideal.



5

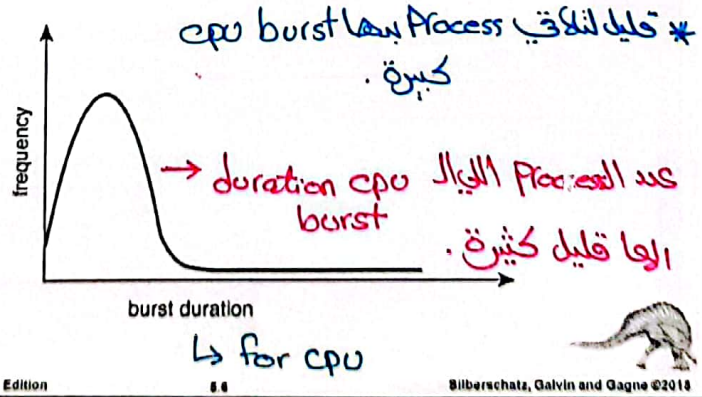
Proc esses ↑ تمثيل بياني لعدد الـ

burst duration بناء على الـ بنوعهم

Histogram of CPU-burst Times

- The durations of CPU bursts have been measured extensively
- Although they vary greatly from process to process and from computer to computer, they tend to have a frequency curve similar to that below
  - Exponential or hyper-exponential
  - Large number of short CPU bursts and a small number of long CPU bursts

* قليل لتلاقي cpu burst لها Process كبيرة.



→ duration cpu burst عدد الـ Process اللي الها قليل كثيرة.

↳ for cpu

6

* حالة رقم 3 بتحير لما الـ Process اللي انتقلت من waiting to ready إلها صلاحية أعلى لتفوت عالـ cpu فبتسحب اللي كانت بالـ cpu وبتحطها.

↳ kernal program

CPU Scheduler

* The CPU scheduler selects from among the processes in ready queue, and allocates a CPU core to one of them
  - Queue may be ordered in various ways → FIFO أشوفهم
  - Conceptually, however, all the processes in the ready queue are lined up waiting for a chance to run on the CPU
  - The records in the queues are generally process control blocks (PCBs) of the processes
* CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state → معناها طلعت من الـ cpu وراحت عالـ wait فلازم الـ Scheduler تنقي وحدة تروح مكانها عالـ cpu
  2. Switches from running to ready state
  3. Switches from waiting to ready → ممكن Schedule يحير فيه
  4. Terminates
* For situations 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution.
* For situations 2 and 3, however, there is a choice.

↳ لازم يحير فيهم (Schedule)

بختارهين next يتنفذ عالـ cpu

يعني الـ Process ماطلعت بكيفها وهون لازم وحدة تنط مكانها (مش دايما بتحير)

خلصت execute وتركت الـ cpu فالـ cpu حسا فاضي ففي مجال أحط Process جديدة.

7

إذا الـ Process بتنفذ عالـ cpu مش حتطلع إلا لتخلص الـ cpu burst تبعتها.

يعني أنا بحطك تشتغلي عالـ cpu بس مش معناه إنه رح تشتغلي cpu burst لتخلص الـ تبعتك، ممكن أقيمك وأحط حد مكانك

↳ Preemptive and Nonpreemptive Scheduling

* When scheduling takes place only under circumstances 1 and 4, the scheduling scheme is nonpreemptive or cooperative.
* Otherwise, it is preemptive. (1, 2, 3, 4)
* Under Nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state.
* Virtually all modern operating systems including Windows, MacOS, Linux, and UNIX use preemptive scheduling algorithms.

فلما تيجي اللي بعدها حتقرأ الـ data القديمة وهنا غلط.
أقطع الـ Proc وهي بتحدث الـ data اللي Update وهي بتكمل

Preemptive Scheduling and Race Conditions

- Preemptive scheduling can result in race conditions when data are shared among several processes.
- Consider the case of two processes that share data. While one process is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state.
- This issue will be explored in detail in Chapter 6.
- Preemption also affects the design of the operating-system kernel
- During the processing of a system call, the kernel may be busy with an activity on behalf of a process.
  - Such activities may involve changing important kernel data (for instance, I/O queues) → عشان مشاكل kernel لجأ مثلا لـ disable بعمل
  - Operating-system kernels can be designed as either nonpreemptive or preemptive.

→ disable interrupt

لـ هو كل الـ kernel إنه nonpreemptive

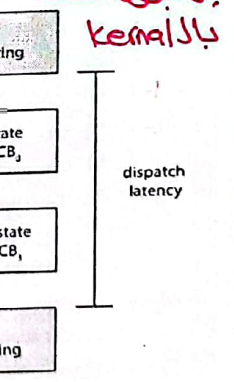
9

الـ Scheduler بس بقول مين يجي مكان الثاني، أما الـ Dispatcher هو اللي بقيم وبحط context وبعمل switching.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU scheduler; this involves:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program
- The dispatcher should be as fast as possible, since it is invoked during every context switch.
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running

لإنه بكون شغال بالـ kernel



بعتمد عالـ hardware

الوقت بين لما نوقف Process ونحول الـ Process الجديدة.

10

Scheduling Criteria

Scheduling Criteria

Many criteria have been suggested for comparing CPU-scheduling algorithms.

- 1- **CPU utilization** – keep the CPU as busy as possible
- 2- **Throughput** – # of processes that complete their execution per time unit
- 3- **Turnaround time** – amount of time to execute a particular process
  - Turnaround time is the sum of the periods spent waiting in the ready queue, executing on the CPU, and doing I/O.
- 4- **Waiting time** – amount of time a process has been waiting in the ready queue
- 5- **Response time** – amount of time it takes from when a request was submitted until the first response is produced.

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

* عادة بنحسب الـ average لكل وحدة من الـ time اللي فوق بعض الـ Systems ما بكون أدق طريقة الحساب ممكن نحسب بـ الـ Variance (التشعب) لكن للتسهيل حنستخدم الـ average.

13

Scheduling Algorithms

1. First come First Served
2. Shortest Job First
3. Round Robin
4. Priority Scheduling

14

أول ما ينحط الـ P على الـ Processor بضل لتخلص الـ CPU burst تبعته ←

أسهل وحدة ، الـ Processes بتقعدوا جوا الـ queue بشكل First in First out والـ job اللي بتوصل بتنضاف لـ Tail ولما اجي اعمل Schedule بروح أخذ من الـ head.

* اللي وصل أول Served بنعمله أول.

Non-Preemptive

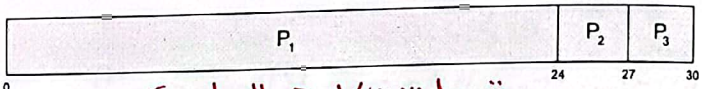
First- Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

وصل أول اشي.

cpu time قديه بيحتاج لتخلص شغلها اللي هلأ ، لايعني إنها خلصت نهائيا مش شرط.

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



قديه استنت كل P بالـ Ready

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

15

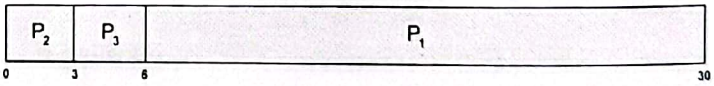
* مشكلتها انه ممكن short P يكونوا معلقين والـ long P مشكلتها الثانية ، انه انت وحظك كيف يجي ترتيب الـ Process

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
- Convoy effect - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

قافلة

بتحس لما بجيب cpu-bound كم تطلب Processor وحاجزة الـ cpu-burst طويلة

16

وراها في I/O-bound بس شغلة صغيرة من الـ Processor مشان يروحوا ينفذوا الـ I/O.

* الـ algorithm بتنقي الـ Process اللي الـ cpu burst time أقل.

Shortest next cpu burst First هي الـ Process تنزل total time مشكلة الـ

Shortest-Job-First (SJF) Scheduling

- 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

17

Shortest-Job-First (SJF) Scheduling

* Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time → بحيث إنه يبلش من الأصغر فالأكبر وهكذا، لنعمل مشكلة الـ Convoy effect
- SJF is optimal – gives minimum average waiting time for a given set of processes ← لإنه
- Preemptive version called **shortest-remaining-time-first**
- How do we determine the length of the next CPU burst?
  - Could ask the user → مش منطقي أسأل الـ User
  - Estimate → إذا اجا Process وقت أقل بيقطع اللي شغال إذا كان الوقت اللي باقي للشغال أكثر من وقت الـ Process اللي اجا.

كيف بدي أعرف الـ burst time وأنا ما بقدر أعرفه، إلا لما أنفذ الـ Process بس لازم أعرفه قبل عشان أرتبهم.

→ Prediction أو توقع أحاول أعمل توقع للي جاي (Slide 20).

18

Non-preemptive يعني

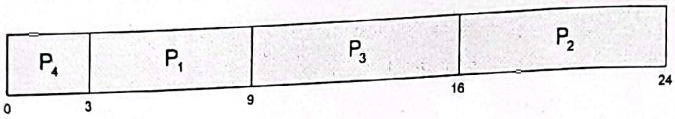
Example of SJF

يبقى بناء على الأقل

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

لو تتين نفس burst time الـ FCFO يقوتوا (First come First out)

- SJF scheduling chart



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

P_1 P_2 P_3 P_4

19

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
  1. t_n = actual length of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next CPU burst
  3. $\alpha, 0 \le \alpha \le 1$
  4. Define :
- Commonly, α set $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$.

عرفنه لاني نفذته

← CPU burst اخر

Prediction الـ ← τ

estimation الأول

← Prediction اخر

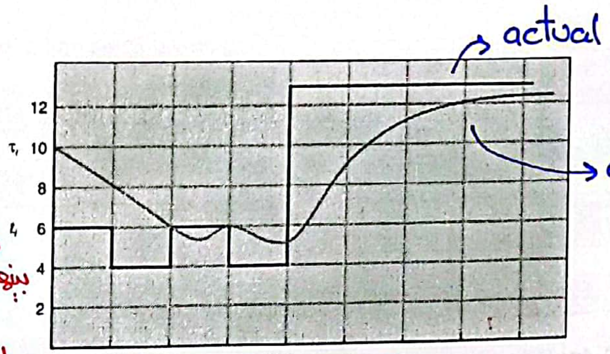
history وفيها الـ

الـ Process الوحدة

20

*ما يجيب بالزبط قديه الـ job ماخذ وقت بكون بحاول أتنبأ قدر الإمكان.

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

t_n ← ال 8 طلعت بالحقيقة 4

طلعت 6 مش 10

فبصير احسب بينهم لأوصل للشي الجديد

τ_n حطيتها fixed لأبلش فيها

حسبها بناء على ال $\alpha = \frac{1}{2}$ وفي المعادلة بتطلع 8

عالمعادلة ال next طلعت 6

← بنصير نحاول نوصل للـ actual.

Examples of Exponential Averaging

1. $\alpha = 0$ → كله بعتمد عالـ history
   - $\tau_{n+1} = \tau_n$
   - Recent history does not count
2. $\alpha = 1$
   - $\tau_{n+1} = \alpha\, t_n$
   - Only the actual last CPU burst counts → فقدت الـ history واعتمدت بس عآخر وحدة
3. If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\alpha\, t_{n-1} + \dots$$
$$+ (1-\alpha)^j \alpha\, t_{n-j} + \dots$$
$$+ (1-\alpha)^{n+1} \tau_0$$

← بجيب الـ history كلها.

* لو حطيت $\alpha = \frac{1}{2}$ بصير average عادي بس ممكن أعمل biasing لو بدي ياه أكثر complix.

- Since both α and $(1-\alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

* الفرق بينهم إنه إذا كانت Process شغالة ووصلت Process الـ burst time تبعتها أقل من اللي ضايل للي شغالة حاليا بتقطعها.

* إذا ما أعطت الـ arrival time معناه بدنا نشتغل عال non-Preemptive.
Preemptive: مش بس يعطيك الـ burst time كمان بيعطيك الـ Arrival time.

Example of Shortest-remaining-time-first

* لو كان عندي 2P اليوم نفس الـ burst time لفوقها زي كأنهم FCFS يعني بشوف الـ arrival اللي وصل أول بيعمل أول. رح تنفذ للأخير لأنه اللي وصلوا بعدها أكبر منها.

- 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

قطع P_1 ← لأنه بعد ما قطعت مقدار 1 لقيت إنه الـ burst الـ P_2 time تبعها 4 أقل من اللي ضايل لـ P_1 = 7 فقطع P_1 وحط P_2

P_1	P_2	P_4	P_1	P_3
0–1	1–5	5–10	10–17	17–26

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

waiting time: P_2 = 1-1 → لأنه أخذنا الـ arrival بعين الاعتبار.
P_4 = 5-3
P_1 = 10-1 = 9
P_3 = 17-2

* لأنه ما كانوا قاعدين من أول اجوا عند arrival time معينة.

Round Robin (RR)

بتشبه الـ FCFS ← بس الفرق إنها Preemptive، يعني ما بترك الـ Process قد ما بدها، بكون is Time quantum يعني الـ Process بتاخذ وقت معين عالـ Processor بعدين بقيمها وبيجي اللي بعدها حتى لو ما خلصت.

- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large ⇒ FIFO
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high

كأني بقسم وقت الـ Process بينهم بالتساوي.

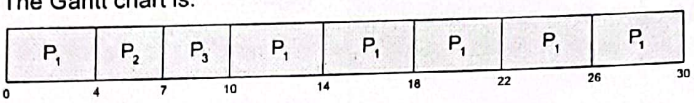
ولا تكون كثير قليلة بصير في تضارب كثير

لو خليته كثير كبير حيصير يشبه FIFO اللي بتقعد عالـ Process لياخذ وقتها بس لا أنا بدي يكون وقت يحددها.

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

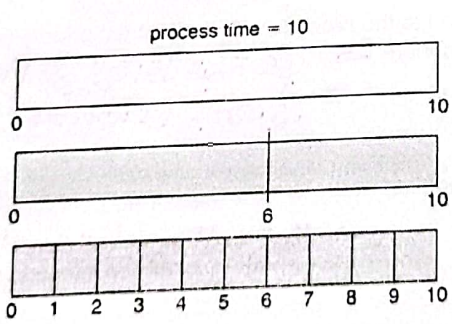
- The Gantt chart is:



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

25

Time Quantum and Context Switch Time

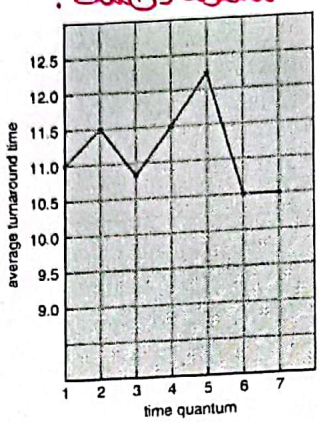


quantum	context switches
12	0
6	1
1	9

26

Turnaround Time Varies With The Time Quantum

معمولة دراسة.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

* هاي بناء على دراسة تم اجراءها.

80% of CPU bursts should be shorter than q

↓

الأفضل تخلي 80% من الـ cpu burst أقل من q

↳ Time quantum توصلوا إنه لما أزيد مش ضروري إنه يتحسن average turnaround time

27

* الـ SJF يعتبر نوع من أنواع الـ Priority Schedule (حالة خاصة)

Priority Scheduling ← بتتحدد بطرق مختلفة

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority) → الـ Process اللي إلها أقل قيمة هي أعلى Priority بالغالب.
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute ← مشكلتنا
- Solution ≡ **Aging** – as time progresses increase the priority of the process

↓

الحل إنه الـ priority سما لوطلوا لينفذوا بحيروا يزيدوا من الـ Priority تاعيتهم

إذا كانت Process كم تنفذ ووصلت Process أعلى منها بالـ Priority بتقطع هيك

أول ماتروح عالـ Process ماحد بيقطعوا حتى لو اجت Process أعلى منها Priority.

سما ممكن ما يجيهم الدور أبدًا 😢

28

زيادة الـ Priority تعني تقليل القيمة أو الرقم لإنه (Smallest integer = highest priority) في الأقلب

* لو معطيني Non-preemptive و Preemptive لازم بتقلك لونزل arrival time مختلفة.

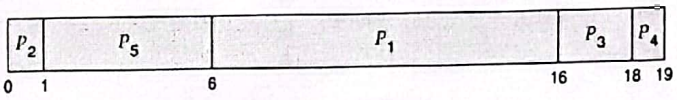
* لو مش معطيني arrival time للأكلب بنشتغل Non-Preemptive.

* لازم بنحكي بالسؤال هل الـ Priority بتكون high ولا low.

Example of Priority Scheduling

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

- Priority scheduling Gantt Chart



- Average waiting time = 8.2

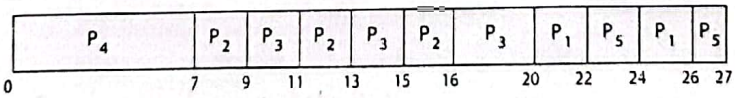
لو حتنشي "FCFS" و"RR".

لو لقيت 2 Process مشاويين بال Priority بنحكالك بالسؤال

Priority Scheduling w/ Round-Robin

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Run the process with the highest priority. Processes with the same priority run round-robin
- Gantt Chart with time quantum = 2



← waiting time: $P_4 = 0$

Same arrival time

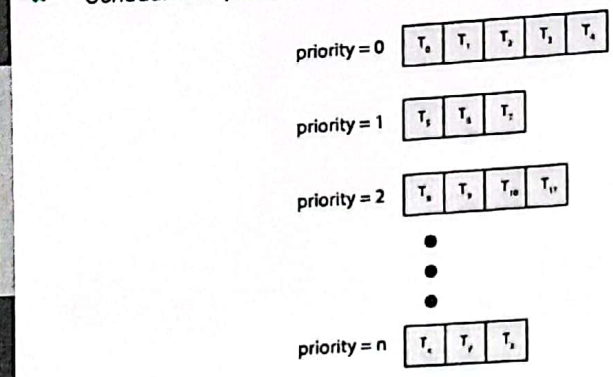
$P_2 = 7 + (11-9=2) + (15-13=2)$

* بالـ Priority Scheduling المشكلة بتكبر إنت في Search لتعرف مين الأعلى Priority لو كان كل الـ Processes معطوطين بنفس الـ queue فالطريقة الأفضل Slide 31

Priority الـ multiple مش واحد والي الهم نفس الـ queue أعلى الـ ready يكونوا بنفس الـ queue فبالتالي بحير ما باخذ من اللي تحت إلا لما اللي فوق يفضى فبتحير عملية الـ Selection أسهل.

Multilevel Queue

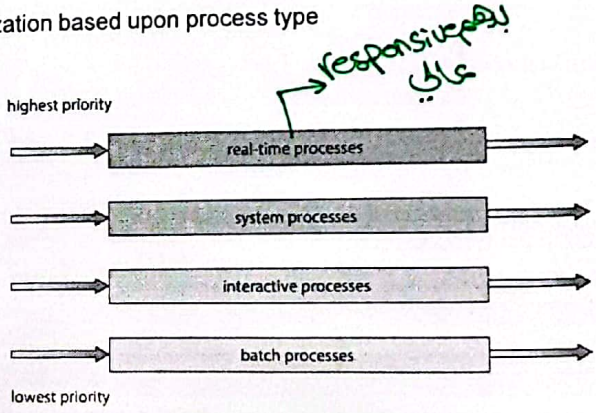
- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



31

Multilevel Queue

- Prioritization based upon process type



responsive بهم عالي

* multilevel Queue بمكنك إنك تصنف الـ Processes اللي بالـ System لأنواع وبناء على هاي الأنواع تعطيهم Priority وبالتالي بحير كل Process إلها Type.

32

لما بحير Process بكون معروف جاي Creation بحير queue رح تنقد.

* مشكلة هاي الطريقة إنها مش flexible ، ما بتسمح الـ Process إنها تنتقل من queue إلى queue (Starvation)

* لو لاحظت وجود Process بتاخذ وقت طويل بنقلها لـ low queue.
* بتسمح للـ Process يهاجروا من low level لـ high level.

الوصف منها ما أظلم تخل قاعدة بـ Process تظل low priority queue طول عمرها.

النوع الأفضل من الـ multilevel queue بس complex.

Multilevel Feedback Queue

- A process can move between the various queues.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service
- Aging can be implemented using multilevel feedback queue

كل queue بتختار اللي بدك ياه إلها

عادة اللي فوق RR لإنه طالما نفذت الـ Process ضمن الـ quantum فبتضل في الـ queue ماخلصت بتنقله للي تحته.

وبين أقوت الـ Process أول اشي ممكن أخليها بالبداية أو بالنهاية أو بالنص.

بناء عشو بقدر أنقل الـ Process من queue لـ queue من تحت لفوق

متى أحدد أنقل Process من فوق لتحت عالأغلب بناء عالـ time اللي بتاخذه.

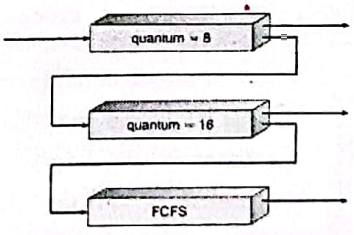
إما إنها سريعة أو إنها طولت وما إلها دور للتنفيذ.

(بتشتغل بشوي عالوقت، الـ Process اللي الهم أقل وقت بكونوا قاعدين بالـ top بس ما بتعين بين اشي والثاني لأ عادلة)

33

Example of Multilevel Feedback Queue

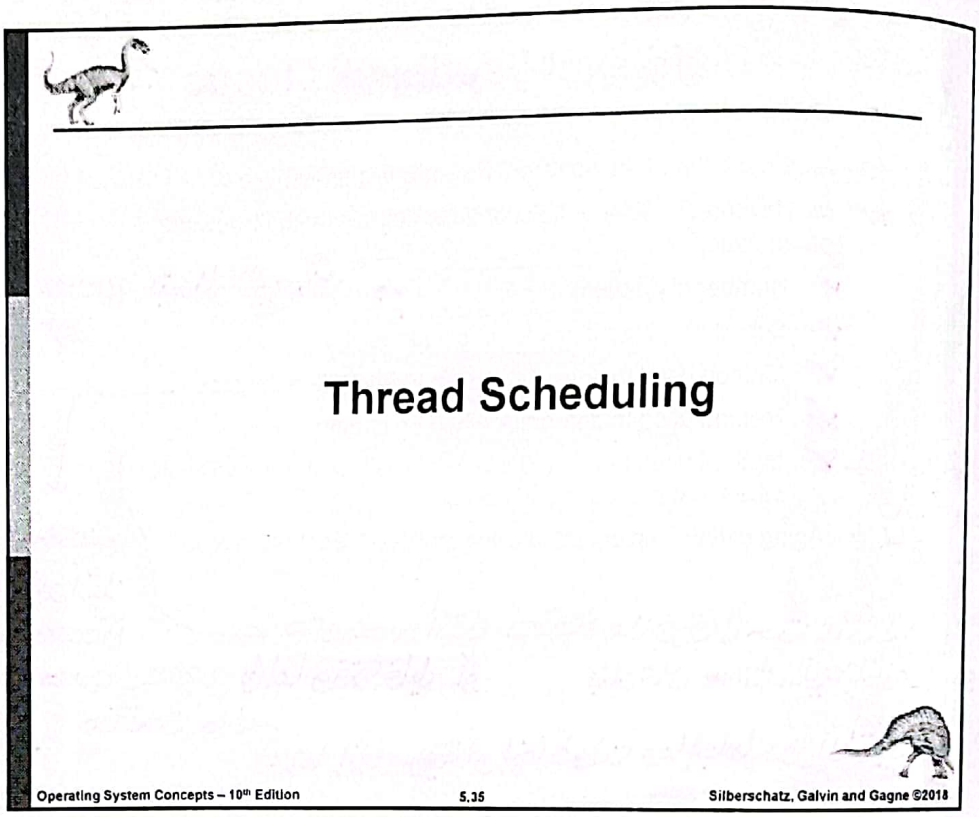
- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new process enters queue Q_0 which is served in RR
    - When it gains CPU, the process receives 8 milliseconds
    - If it does not finish in 8 milliseconds, the process is moved to queue Q_1
  - At Q_1 job is again served in RR and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2



برضه بكون مسموح الهجرة من تحت لفوق بس مو موضحة بالرسمة

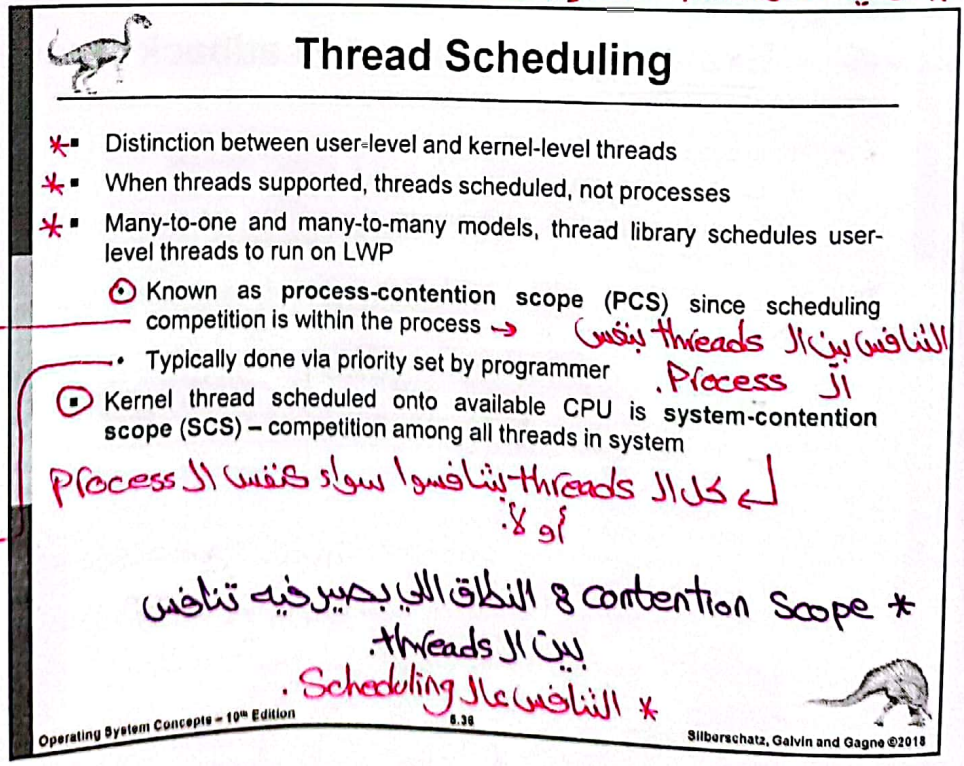
34

* قبلكنا نعتبر إنه الجهاز تبعي Single thread + Single core (بالشرح تاع Ch 5 اللي قبل)



35

* حكينا كل user thread و Kernel thread. بنحكي



كلة بظهر بال many-to-one and many-to-many

Library كيف طريقة ال

المنافسة بين ال threads بنفس ال Process.

ال threads بتنافسوا سواء كانت نفس ال Process أو لأ.

* النطاق ال Contention Scope اللي بحصل فيه تنافس بين ال threads.

* المنافسة على ال Scheduling.

36

Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- Can be limited by OS – Linux and macOS only allow PTHREAD_SCOPE_SYSTEM

↓

one-to-one لإنو النوع الـ mapping اللي فيهم هو
Kernal threads فالـ contention بكون بين الـ
اللي بالـ System جوا
. Process الـ Scheduling & Contention

37

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM");
      else
         fprintf(stderr, "Illegal scope value.\n");
   }
```

* بالـ C بعطيها Pointer وبتعبيلي جواه الاشي اللي بدي ياه لما أنادي الـ get *

لإنه Constant ←

Pointer ↗

→ بجيب معرف جوا الـ library .

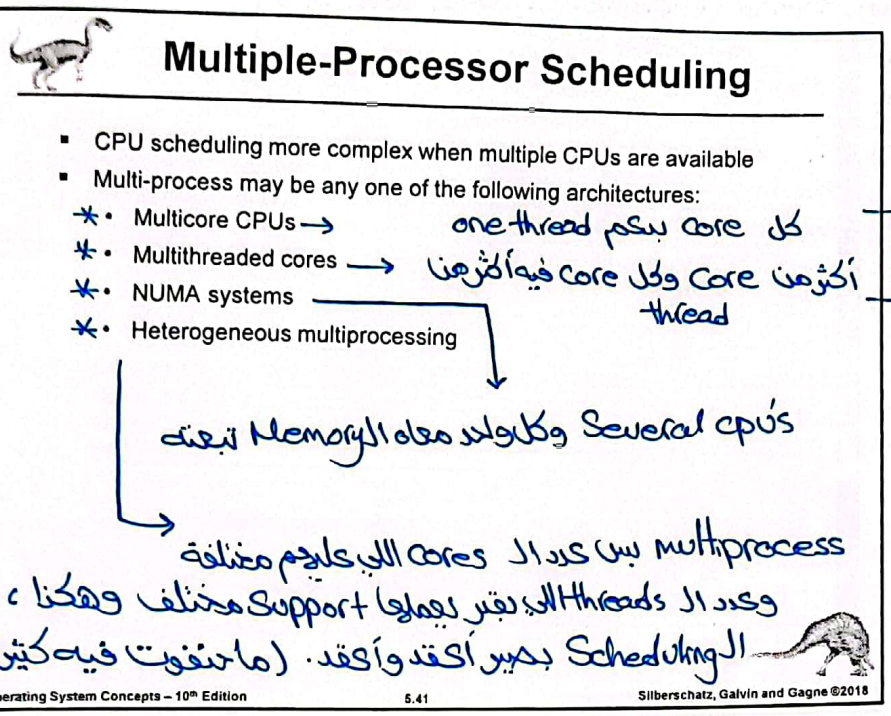
38

Pthread Scheduling API

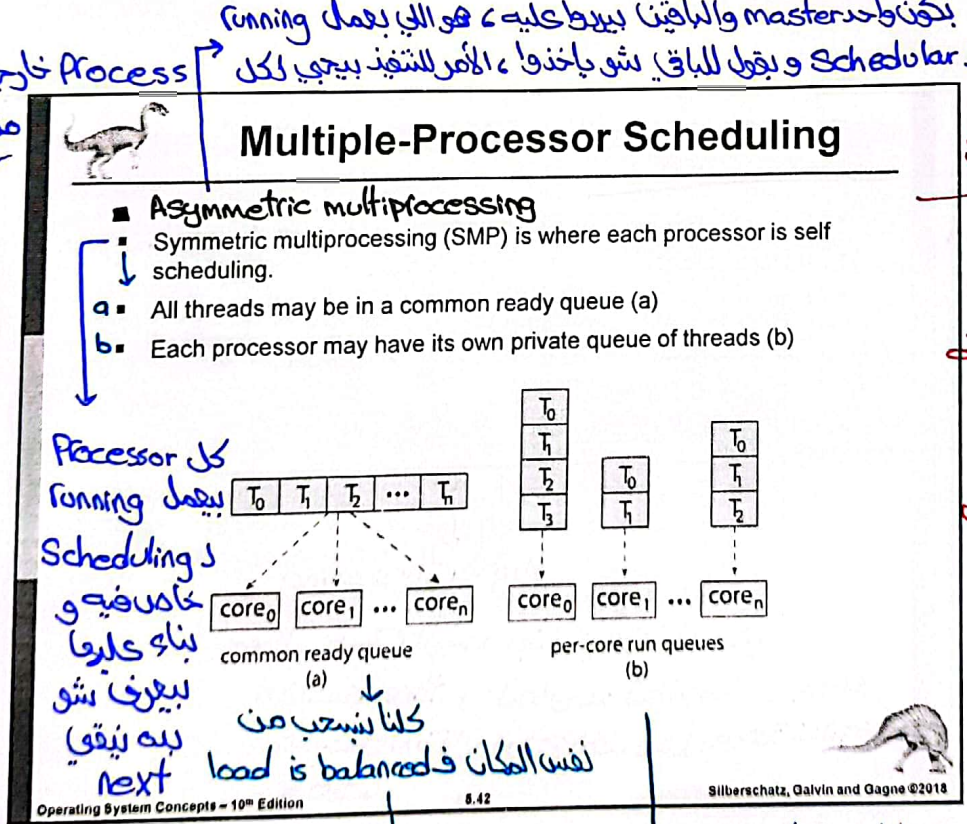
```
    /* set the scheduling algorithm to PCS or SCS */
    pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
    /* create the threads */
    for (i = 0; i < NUM_THREADS; i++)
       pthread_create(&tid[i],&attr,runner,NULL);
    /* now join on each thread */
    for (i = 0; i < NUM_THREADS; i++)
       pthread_join(tid[i], NULL);
}
/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```

* عنيا أكثر من core جوا الـ System تبعي *

Multi-Processor Scheduling



41



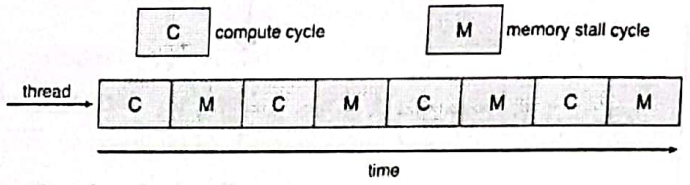
42

وأكثر من more than one register بكون فيه الـ Core الواحد ←
set.

instruction Pointer
بحيث يقدر انه ينفذ لك
أكثر من threads.

Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

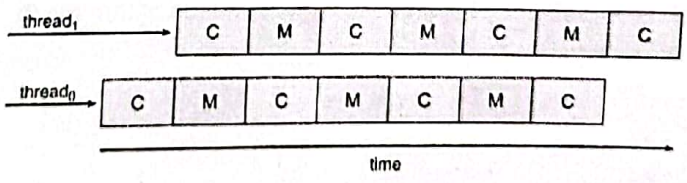


43

لإنه كل واحد ← Pointer على
Set of registers
خاصة فيه.

Multithreaded Multicore System

- Each core has > 1 hardware threads. → يعني Switching
- If one thread has a memory stall, switch to another thread!
- Figure



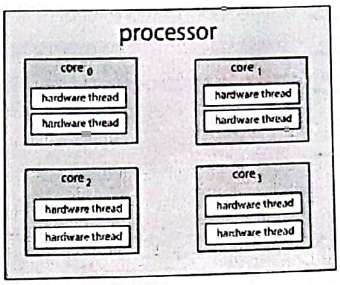
hyperthreading ↓

الـ Core الواحد بقدر يخزن عدد معين
من الـ threads وكلما وحدة تطلب من الـ Mem
ما بخلي يشتغل بروح ينفذ شغل وحدة ثانية
وهكذا.

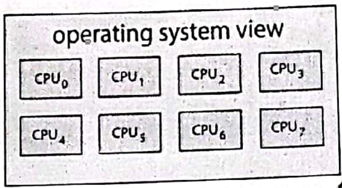
44

Multithreaded Multicore System

- **Chip-multithreading** (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyperthreading**.)



- On a quad-core system with 2 hardware threads per core, the operating system sees 8 logical processors.



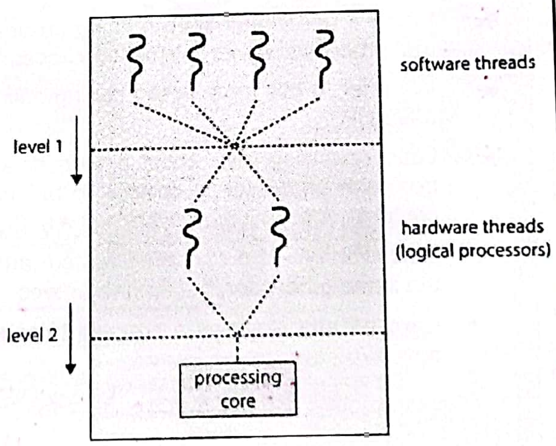
" 8 software threads "

وظيفتي عمستوى الـ OS إنه إذا أي واحد من الـ 8 انشال من الـ Processor سواء إنه خلص أو راح يستنى I/O أو غيره أقول مين دوره ينزل مكانه

Multithreaded Multicore System

- Two levels of scheduling:

1. The operating system deciding which software thread to run on a logical CPU
2. How each core decides which hardware thread to run on the physical core.



← بحتاجه بس لما يكون لكل core الماعنده خانة فيه.

← قد يكون عدد الـ task أو الـ timing اللي الـ task.

Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor

← ما في حدا مسؤول Processor أنا الـ queue اللي عندي queue انا شفت إنه العنصر تبعي فاضي وخلصت بروح بقول للـ Processor اللي جنبي أو جنب جنبي أعطيني من هندك.

يكون في Process هي المسؤولة إنها تراقب كيف توزيع الـ Processes بين الـ queues ولما تلاحظ إنه في queue (loaded) هي بتيجي بتسحب لوحدة في queue بتكون loaded أكثر من الثانية وبتحطها ع queue ثانية under loaded.

Multiple-Processor Scheduling – Processor Affinity

- When a thread has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.
- We refer to this as a thread having affinity for a processor (i.e., "processor affinity")
- Load balancing may affect processor affinity as a thread may be moved from one processor to another to balance loads, yet that thread loses the contents of what it had in the cache of the processor it was moved off of.
- **Soft affinity** – the operating system attempts to keep a thread running on the same processor, but no guarantees. ← ما في ضمانة إنه يخليها
- **Hard affinity** – allows a process to specify a set of processors it may run on. ← إذا أنا كـ Process انقلت بعد المكان خلص ما بطلع منه.

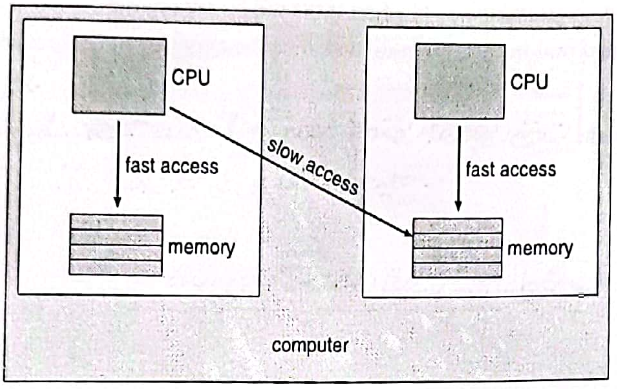
← صار في علاقة بين الـ Core والـ Process وكل الـ data محفوظة جوا الـ Cash تبع هذا الـ Core فالـ load balancing ممكن يأثر عهاي الشغلة لأنه ممكن ننقل الـ thread لمكان ثاني وهيك فقدت المعلومات اللي كانت محفوظة عندي.

* لما اعمل ال Scheduling لل threads على Processor معين بعمل حسابي انه ال code وال data تبعت ال thread أنزلها بال local Memory مش اوديها ع Mem بعيدة ، ممكن توديها ع Mem بعيدة لأنه بالنسبة لل OS كل ال Mem حيكونوا address space واحد . اما تكون

NUMA-aware يعني تكون عارفة اي جزء من ال address space هو ال local لكل وحدة.

NUMA and CPU Scheduling

If the operating system is **NUMA-aware**, it will assign memory closest to the CPU the thread is running on.



Real-Time CPU Scheduling

Real-Time CPU Scheduling

- Can present obvlous challenges
- **Soft real-time systems** – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline

لإنه يكون الخبر لو ما خلص قبل ال deadline مش كتير كبير.

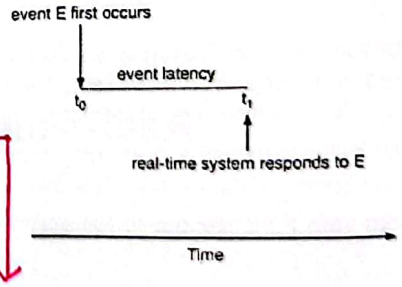
هاي ال task متى لازم تخلص، متى أقصى وقت هاي ال task لازم تخلص فيه.

ضروري تخلص قبل ال deadline

51

Real-Time CPU Scheduling

- Event latency – the amount of time that elapses from when an event occurs to when it is serviced.
- Two types of latencies affect performance
  1. **Interrupt latency** – time from arrival of interrupt to start of routine that services interrupt
  2. **Dispatch latency** – time for schedule to take current process off CPU and switch to another



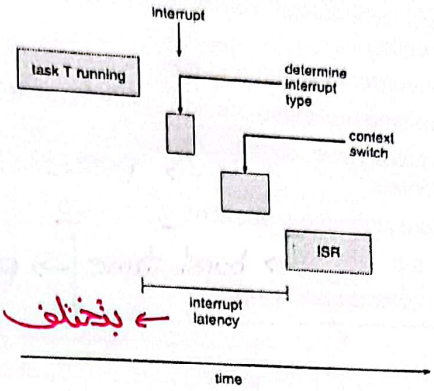
الوقت من لما حدا event ال لحد ما انعمله response.

الوقت الي لازم مشان اقطع الاشي الي شغال واعمل context switching جزء من ال Interrupt latency.

الوقت من لما يجي interrupt لحد ما يتنادى ال Service routine تبع هذا ال interrupt.

52

Interrupt Latency

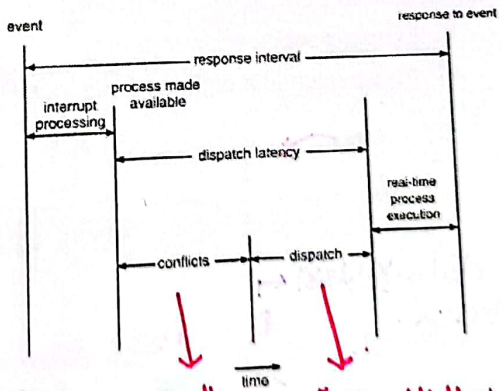


System من يختلف ← .System J

53

Dispatch Latency

- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes



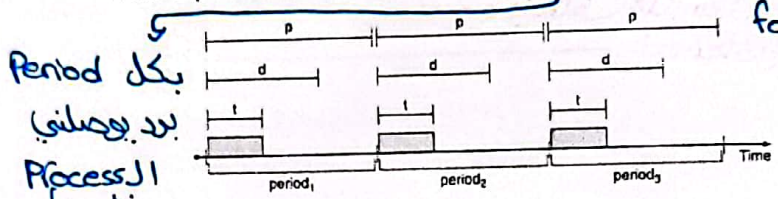
الجزء هذا الوقت كبين ما أقدر أقطع اللي شغال هلأ.

الجزء الثاني الوقت اللي بعمل فيه Switching للموجود وأحط مكانه الجديد.

54

Priority-based Scheduling

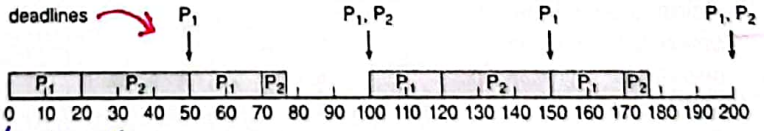
- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: periodic ones require CPU at constant intervals
  - Has processing time t, deadline d, period p
  - $0 \le t \le d \le p$
  - Rate of periodic task is $1/p$



55

Rate Monotonic Scheduling (Static)

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.



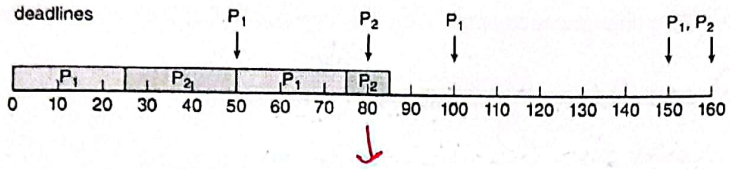
	P_1	P_2
t	20	35
d	50	100
p	50	100

56

$$\frac{t_{P_1}}{dP_1} + \frac{t_{P_2}}{dP_2}$$

② Missed Deadlines with Rate Monotonic Scheduling

- Process P_2 misses finishing its deadline at time 80



deadline الـ تفوت P_2

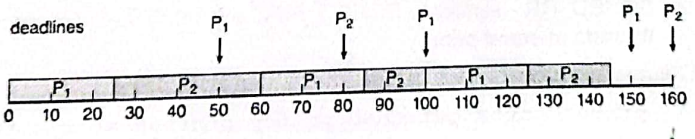
	P_1	P_2
t	25	35
d	50	80
p	50	100

57

③ Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:
  - The earlier the deadline, the higher the priority
  - The later the deadline, the lower the priority

الـ Priorities بتكون static مش dynamic
مشروطيا عندي
Priority الـ نفس



اللي الـdeadline تبعها مقرب أكثر اشي هي اللي تاخذ
. Scheduling الـ

58

بقسم الـ cpu time لـ t بقسموا بين الـ task بحيث كل واحد ياخذ حسب Share Priority الـ تبعه.

Proportional Share Scheduling

- *T* shares are allocated among all processes in the system
- An application receives *N* shares where *N* < *T*
- This ensures each application will receive *N* / *T* of the total processor time

بصير الشي بشبه الـ Round Robin بس ماحد بياخذ مكان حدا، يعني لو وحدة بتاخذ 5 time وحدة 2 بس تخلص الـ 5 ما بترجع تفوت إلا لكل اللي بعدها يخلصوا تنفيذ.

59

جزء من الـ API بخص الـ Real-time

POSIX Real-Time Scheduling

- The POSIX.1b standard
- API provides functions for managing real-time threads
- Defines two scheduling classes for real-time threads: → بتختار اللي بدك ياه
  1. SCHED_FIFO - threads are scheduled using a FCFS strategy with a FIFO queue. There is no time-slicing for threads of equal priority
  2. SCHED_RR - similar to SCHED_FIFO except time-slicing occurs for threads of equal priority
- Defines two functions for getting and setting scheduling policy:
  1. `pthread_attr_getsched_policy(pthread_attr_t *attr, int *policy)`
  2. `pthread_attr_setsched_policy(pthread_attr_t *attr, int policy)`

Time Slicing ما فيها

بترجعلك شو نوع الـ Policy اللي مستخدم هلأ هذا الـ Variable

انت بتحط الـ Policy وبتفرك للـ Policy اللي انت عطيتها ايها.

60

POSIX Real-Time Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[])
{
   int i, policy;
   pthread_t tid[NUM_THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* get the current scheduling policy */
   if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
      fprintf(stderr, "Unable to get policy.\n");
   else {
      if (policy == SCHED_OTHER) printf("SCHED_OTHER\n");
      else if (policy == SCHED_RR) printf("SCHED_RR\n");
      else if (policy == SCHED_FIFO) printf("SCHED_FIFO\n");
   }
```

هش بكل الـ sys معرفت لم

61

POSIX Real-Time Scheduling API (Cont.)

```
   /* set the scheduling policy - FIFO, RR, or OTHER */
   if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) != 0)
      fprintf(stderr, "Unable to set policy.\n");
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);
   /* now join on each thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```

62

Operating System Examples

Operating System Examples

- Linux scheduling
- Windows scheduling
- Solaris scheduling

(("Linux Scheduling Through Version 2.5"))

* قبل Version 2.5 كان بيستخدم Scheduling Algorithm موجودة بال Unix مشكلتها كانت تاخذ order of n time (O(n)) . الـ Scheduling عليها إني أنقي مين الـ Process الي حتنفذ next.
↳ number of processes ↳ كانت بطيئة كثير وسيئة "ومش مناسبة لل Symmetric multi process لإنه بنستخدم كل الـ processes ب shared q واحد.

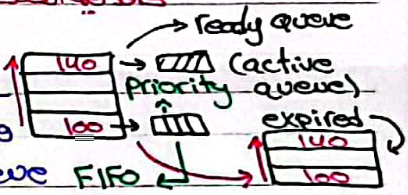
* بعد Version 2.5 ظهرت Algorithm ← O(1) ↲ constant ، لا تعتمد على عدد الـ Processes

*linux → Real time processes → Priority أعلى
*linux → Normal processes → Priority أقل
↓ أنواع الـ task داخلها

بتوصف أي Process فيها عملية Input/out كثير مع الـ
Interactive → Process الـ I/O وفي معظم الـ time أقل وغالبية وقتها مع الـ cpu
Batch → in the background الي بتشتغل Process أي أغلب وقتها بالـ cpu ما إلها تفاعل مع الـ User والـ I/O .

Priority الرقم الأقل هو الأعلى * Priority 0-99   Priority 100-140 or 100-139

* هاي الـ Algo بتشتغل بحيث إنه فيها نوعين من الـ queues & active queue / expired queue
فالـ active فيها الـ processes الي لسا بيشتغلوا ولسا ما أخذوا الـ time slot ← تاعهم كامل (ركزنا عالـ Normal مش الـ Real)



بنبلش ننقي من أول الـ non empty highest priority queue
وإذا خلصت الـ Process خلص ولو ما خلصت بتنزل على non empty highest priority expired queue FIFO وهكذا. فالـ Process إلها Time slot معين خلصت فيه خلص ما خلصت بتنزل عالـ expired ، بعد ما تخلص كل الي بالـ active رح يوقف وكل الي ما خلصوا عالـ expired راحوا ، فبعمل Switching بصير الـ active هو الـ expired والـ expired هو الـ active.

* هاي الاستراتيجية multi level لإنها level الأول الـ active والـ level الثاني هو الـ expired وجواهم بوضع في غير الـ priority للأشياء الي جواها إني بتقوم بناء عالـ priority . هون بضمن عالأغلب الكل حيجيه الدور فهاي الطريقة تمنع الـ Starvation الي بصير لما يكون عندي task إلها lowest priority ما بيجيها الدور بالمرة.
* هون بنحتاج لـ 2 Steps & الأولى إنو نلاقي lowest number queue with at least 1 task (يعني تلاقي بالـ active الـ highest queue الي مش فاضي ، الثانية بيها ننقي أول task من الـ queue الي مش فاضي وهاي ثابت الـ time نبعها بس الفكرة كيف ألاقي أول الـ non-empty queue عن طريق الـ bit map وهي String of bits كل واحد من الـ active Q بكون إله Position فيه برجعلك لو هو فاضي (0) ولا لأ (1) وعن طريق الـ instruction & Find First bit Set بتلاقي أول bit بهاي الـ String كان (1) بناخذ الـ Index تبعه وبنتصل عالـ queue access ، ما بعد أدور لإنه هذا بياخذ وقت ، يعني لو كان الـ String of bits = 0111... فأول queue مش فاضية هي الـ 101 بدخل عليها.

* حتى الـ Priority الي بنعطوها للـ tasks من 100 لـ 140 ← can be assigned statically or dinamically
↳ hearstics ← لحالها تعمل تحديث الـ Algo استشكالات معينة بتصير
الـ Priority ، بعطي الـ default = 120 وبصير dinamic نستدل من طبيعة الـ task انا هي Interactive أو Batch إلها تقلل الـ Priority أو تزيدها والـ I/O هي الي بنعطيها الـ Priority تاعها فبصير براقب لو الـ task أغلبها I/O بعطيها بونص لتصير higher priority ولو أغلبية وقتها بالـ cpu بعطيها بونص عكسي ليخليها lower Priority.

↳ الـ task أول ما تشتغل الـ Priority إلها هو الـ default الوسط = 120
فيه بك تزيح عن الـ 120 ← N nice-n
عشان تاخذ highest priority
N [-20, 20]
أو 19 ↲ ↳ بنقص لأخليه high priority
↳ ازيد لأخليه low Priority

* شوفي الورقة أول *

أول ما بلش كان كبارة عن امتداد لـ unix
بعدين صار له كيان مستقل

Linux Scheduling Through Version 2.5

- Prior to kernel version 2.5, ran variation of standard UNIX scheduling algorithm
- Version 2.5 moved to constant order O(1) scheduling time
  - Preemptive, priority based
  - Two priority ranges: time-sharing and real-time → Normal → -20, 20
  - **Real-time** range from 0 to 99 and **nice** value from 100 to 140 → بنفس فهم عن طريق الـ nice
  - Map into global priority with numerically lower values indicating higher priority
  - Higher priority gets larger q → quantum
  - Task run-able as long as time left in time slice (**active**) → حالما هو مخلص بضل بالـ active
  - If no time left (**expired**), not run-able until all other tasks use their slices → ضايل عندهم اشي ما نقدوه لإنه خلص وقتهم
  - All run-able tasks tracked in per-CPU **runqueue** data structure
    - Two priority arrays (active, expired)
    - Tasks indexed by priority
    - When no more active, arrays are exchanged
  - Worked well, but poor response times for interactive processes → لإنه ثقيلة فنخلوا لـ CFS

65

خلت مبادئ اللي فوق بس الفرق إنه طريقة الـ Scheduling ما فيها ثقل وتعقيد .

Linux Scheduling in Version 2.6.23 +

- **Completely Fair Scheduler (CFS)**
- **Scheduling classes**
  - Each has specific priority
  - Scheduler picks highest priority task in highest scheduling class
  - Rather than quantum based on fixed time allotments, based on proportion of CPU time
  - Two scheduling classes included, others can be added
    1. default
    2. real-time

66

Linux Scheduling in Version 2.6.23 + (Cont.)

- Quantum calculated based on **nice value** from -20 to +19
  - Lower value is higher priority
  - Calculates **target latency** – interval of time during which task should run at least once
  - Target latency can increase if say number of active tasks increases
- CFS scheduler maintains per task **virtual run time** in variable `vruntime`
- CFS scheduler maintains per task **virtual run time** in variable `vruntime`
  - Associated with decay factor based on priority of task – lower priority is higher decay rate
  - Normal default priority yields virtual run time = actual run time
- To decide next task to run, scheduler picks task with lowest virtual run time

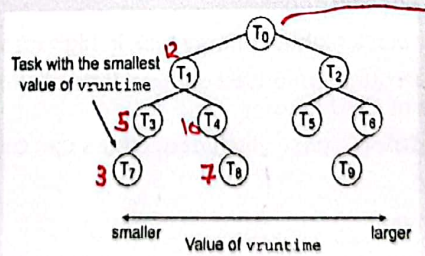
بمعدوفت خلاله لازم
إنه الـ task يتنفذ على الأقل one
كل ما زاد الـ task كل ما انخفض priority low
لإنه كل ما خلصتي معناه إنت cpu bound
highest priority
يعني بتاخدي على cpu وقت أقل.

67

كيف أرتبهم لأوقت الأعلى priority بسرعة

CFS Performance

The Linux CFS scheduler provides an efficient algorithm for selecting which task to run next. Each runnable task is placed in a red-black tree—a balanced binary search tree whose key is based on the value of `vruntime`. This tree is shown below:



كلما رحت لليمين وفوق يزيد Virtual run time اللي ما بتخلص بتروح عاليا أكبر منها أو بتتنقل لليمين وفوق.

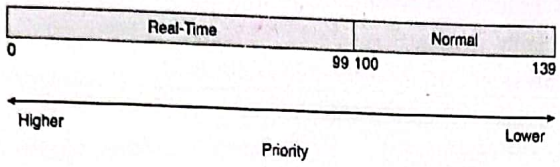
أعلى وحدة

When a task becomes runnable, it is added to the tree. If a task on the tree is not runnable (for example, if it is blocked while waiting for I/O), it is removed. Generally speaking, tasks that have been given less processing time (smaller values of `vruntime`) are toward the left side of the tree, and tasks that have been given more processing time are on the right side. According to the properties of a binary search tree, the leftmost node has the smallest key value, which for the sake of the CFS scheduler means that it is the task with the highest priority. Because the red-black tree is balanced, navigating it to discover the leftmost node will require $O(\lg N)$ operations (where N is the number of nodes in the tree). However, for efficiency reasons, the Linux scheduler caches this value in the variable `rb_leftmost`, and thus determining which task to run next requires only retrieving the cached value.

68

Linux Scheduling (Cont.)

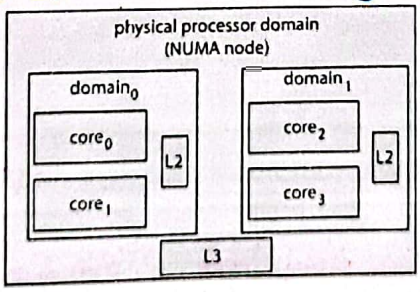
- Real-time scheduling according to POSIX.1b
  - Real-time tasks have static priorities
- Real-time plus normal map into global priority scheme
- Nice value of -20 maps to global priority 100
- Nice value of +19 maps to priority 139



69

Linux Scheduling (Cont.)

- Linux supports load balancing, but is also NUMA-aware.
- **Scheduling domain** is a set of CPU cores that can be balanced against one another.
- Domains are organized by what they share (i.e., cache memory.) Goal is to keep threads from migrating between domains.



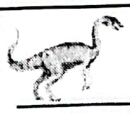
70

Windows Scheduling

- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- **Dispatcher** is scheduler
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time
- 32-level priority scheme
- **Variable class is 1-15, real-time class is 16-31**
- Priority 0 is memory-management thread
- Queue for each priority
- If no run-able thread, runs **idle thread**

(highest number highest priority)

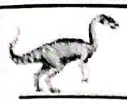
71



Windows Priority Classes

- Win32 API identifies several priority classes to which a process can belong
  - REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS
  - All are variable except REALTIME
- A thread within a given priority class has a relative priority
  - TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL, BELOW_NORMAL, LOWEST, IDLE
- Priority class and relative priority combine to give numeric priority
- Base priority is NORMAL within the class
- If quantum expires, priority lowered, but never below base

72



Windows Priority Classes (Cont.)

- If wait occurs, priority boosted depending on what was waited for
- Foreground window given 3x priority boost
- Windows 7 added **user-mode scheduling** (UMS)
  - Applications create and manage threads independent of kernel
  - For large number of threads, much more efficient
  - UMS schedulers come from programming language libraries like C++ **Concurrent Runtime** (ConcRT) framework

73

Windows Priorities

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

74

Algorithm Evaluation

↓

بدنا نعرف مين أحسن من الثانية

75

Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- ① **Deterministic modeling** → منيحة لو ال workload صغير
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

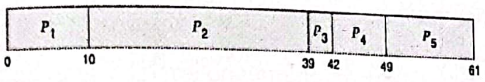
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

76

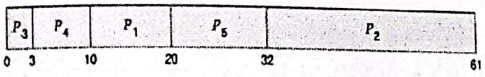
← حتيجي بالامتحان حتعطيك ال Sample وتحكيلك احسب average waiting time اذا استخدمنا هذه ال Algo ↓ أيًا كانت

Deterministic Evaluation

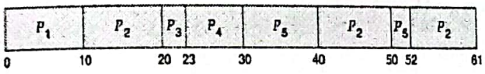
- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCS is 28ms:



  - Non-preemptive SFJ is 13ms:



  - RR is 23ms: ← time slot = 10 ...



77

work load ← test ... منيحة لو ال ... كبيرة

② Queueing Models

- Describes the arrival of processes, and CPU and I/O bursts probabilistically — arrival Rate = 1/arrival
  - Commonly exponential, and described by mean
  - Computes average throughput, utilization, waiting time, etc.
- Computer system described as network of servers, each with queue of waiting processes
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc.

78

③ Little's Formula

- n = average queue length
- W = average waiting time in queue
- λ = average arrival rate into queue → معدل وصول الـ processes
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:
  $n = \lambda \times W$
  - Valid for any scheduling algorithm and arrival distribution
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

* مايكون عدد اللي بوصلوا قد عدد اللي بطلعو ما بتكون صح المعادلة ١٠٠٪.

* لو عدد اللي بوصلوا قد عدد اللي بغادروا فالمعادلة صح ١٠٠٪.

79

اكتب برنامج يعمل Simulation لعملية Scheduling بعطيه work load الـ وهو يعمل Scheduling ويحسب الوقت ويقارن بين الـ Algo مين أعطت وقت أفضل.

④ Simulations

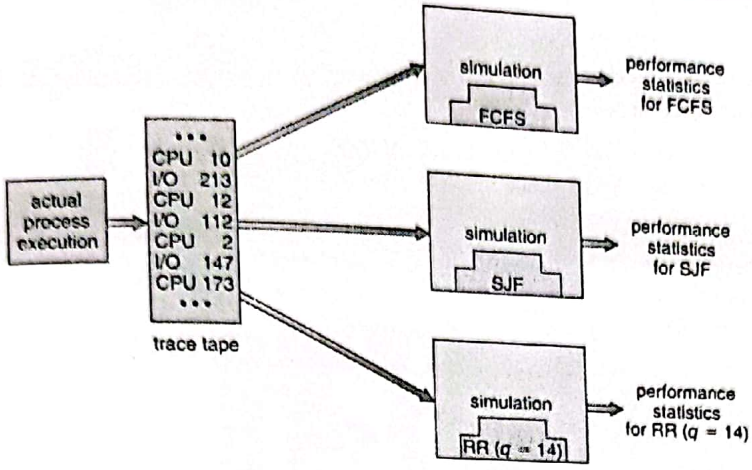
- Queueing models limited
- **Simulations** more accurate
  - Programmed model of computer system
  - Clock is a variable
  - Gather statistics indicating algorithm performance
  - Data to drive simulation gathered via
    - Random number generator according to probabilities ← مشان اشي حقيقي لهاي
    - Distributions defined mathematically or empirically
    - Trace tapes record sequences of real events in real systems ← بكون مسجلة اشي فعلي.

Fixed

من مشاهدتك لـ task بعملها بطريقة معينة متشابهة الواقع.

80

Evaluation of CPU Schedulers by Simulation



81

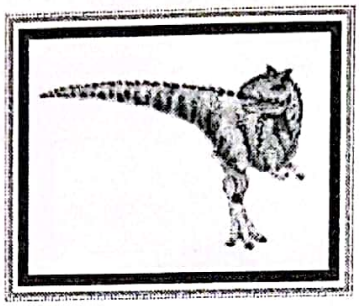
⑤ Implementation

→ Inside operating System (at actual Implementation time)

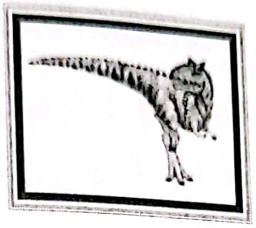
- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
  - High cost, high risk
  - Environments vary
- Most flexible schedulers can be modified per-site or per-system
- Or APIs to modify priorities
- But again environments vary

82

End of Chapter 5



Chapter 6: Synchronization Tools



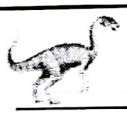
Outline

- Background (6.1)
- The Critical-Section Problem (6.2)
- Peterson's Solution (6.3)
- Hardware Support for Synchronization (6.4)
- Mutex Locks (6.5)
- Semaphores (6.6)
- Monitors (6.7)
- Liveness (6.8)

Objectives

- Describe the critical-section problem and illustrate a race condition
- Illustrate hardware solutions to the critical-section problem using memory barriers, compare-and-swap operations, and atomic variables
- Demonstrate how mutex locks, semaphores, monitors, and condition variables can be used to solve the critical section problem
- Evaluate tools that solve the critical-section problem in low-, Moderate-, and high-contention scenarios

3



Background

4

* Concarency & Switching بين الـ Processes بحيث يوحيلك إنوكله تنفذ بنفس الوقت . حتى لوكان الـ hardware عليه Single core

* Parallism & فعليا بكون في أكثر من Task تشغال بنفس الوقت لإنه الـ hardware بدعم multiple cores أو multiple threads .

Background

- Processes can execute concurrently
  - May be interrupted at any time, partially completing execution
- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- We illustrated in chapter 4 the problem when we considered the Bounded Buffer problem with use of a counter that is updated concurrently by the producer and consumer,. Which lead to race condition.

الـ Race condition بتطلع مشكلة الـ Synchroniation* اللي كنا نواجها ، بيمنع أي 2 processes إنهم يعملوا update لنفس الـ data بنفس الوقت كشان ما يحيس في قراءات أو كتابات خاطئة .

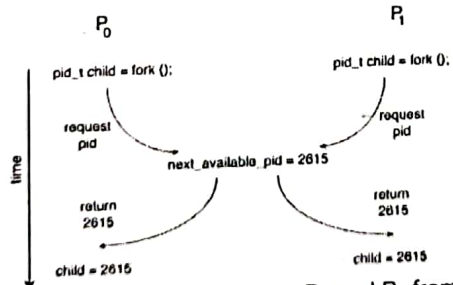
5

مثال كبير مثال الـ Consumer / Producer

Race Condition

- Processes P_0 and P_1 are creating child processes using the `fork()` system call
- Race condition on kernel variable `next_available_pid` which represents the next available process identifier (pid)

→ global variable (shared)

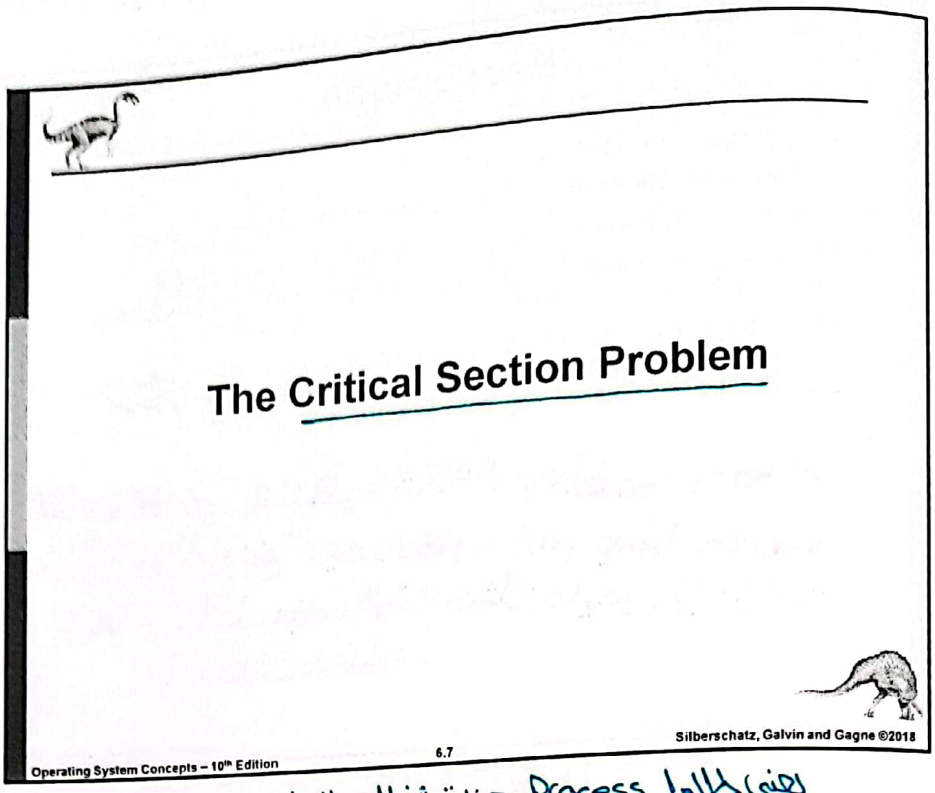


- Unless there is a mechanism to prevent P_0 and P_1 from accessing the variable `next_available_pid` the same pid could be assigned to two different processes!

6

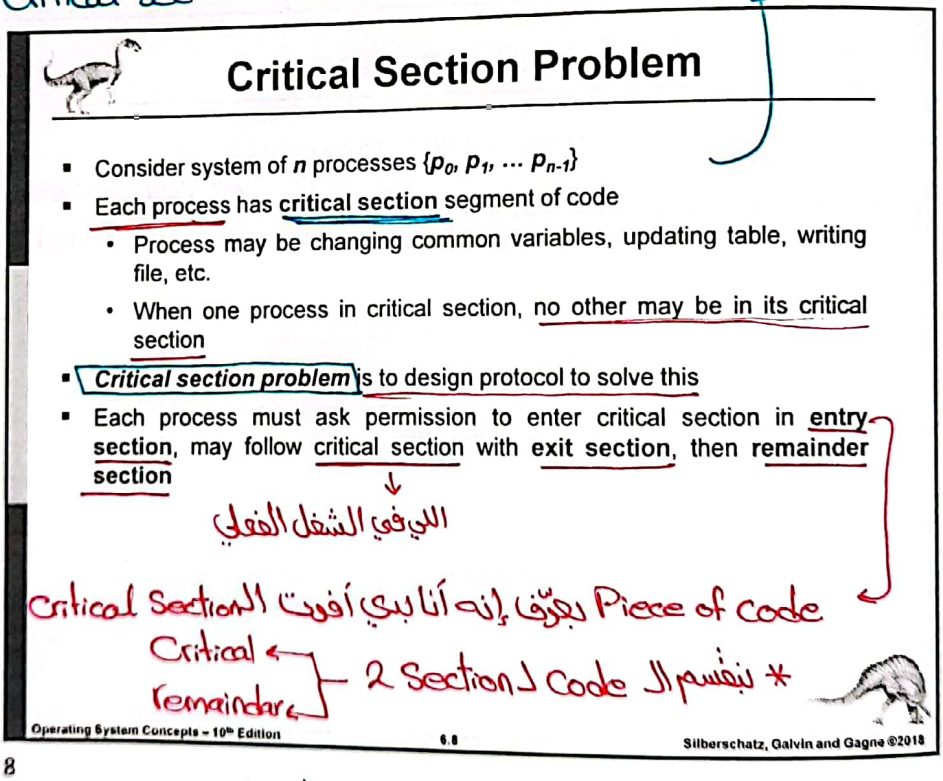
→ بتحيس المشكلة لما يعملوا 2 process بنفس الوقت fork فالتين حياخدوا 2615 وهذا غلط لإنه مفروض أي process يكون إلها id مختلف عن غيرها .

فلحا حندرس كيف نقدر نمنع هذا الأشي إنه يحيس .



7

يعني طالما Process معينة شغالة بالـ Critical Section ممنوع أي Process ثانية تحاول تفوت عالـ Critical Section تبقى لحد ما هيك تخلص.



كشان هيك كانة بيصل update/sharing لـ data بالـ Critical Section كشان ما يصير في Race condition

اللي في الشغل الفعلي

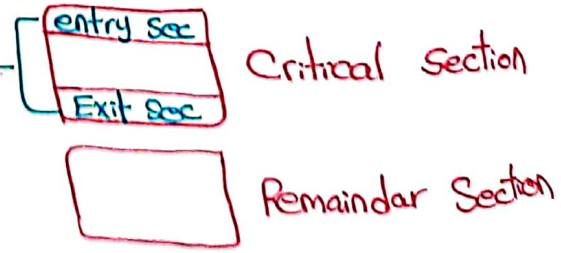
Algorithm بختلف من التانية

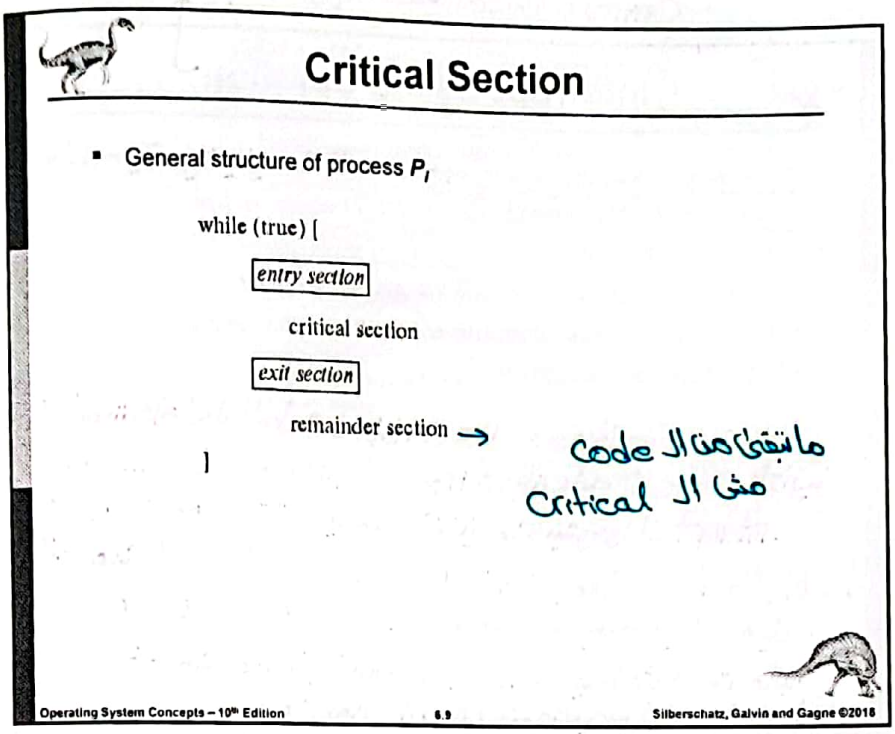
Critical Section بعرّف إنه أنا بدي أفوت Piece of code

Critical ← 2 Section لـ Code الـ بنقسم الـ *
Remainder ←

8

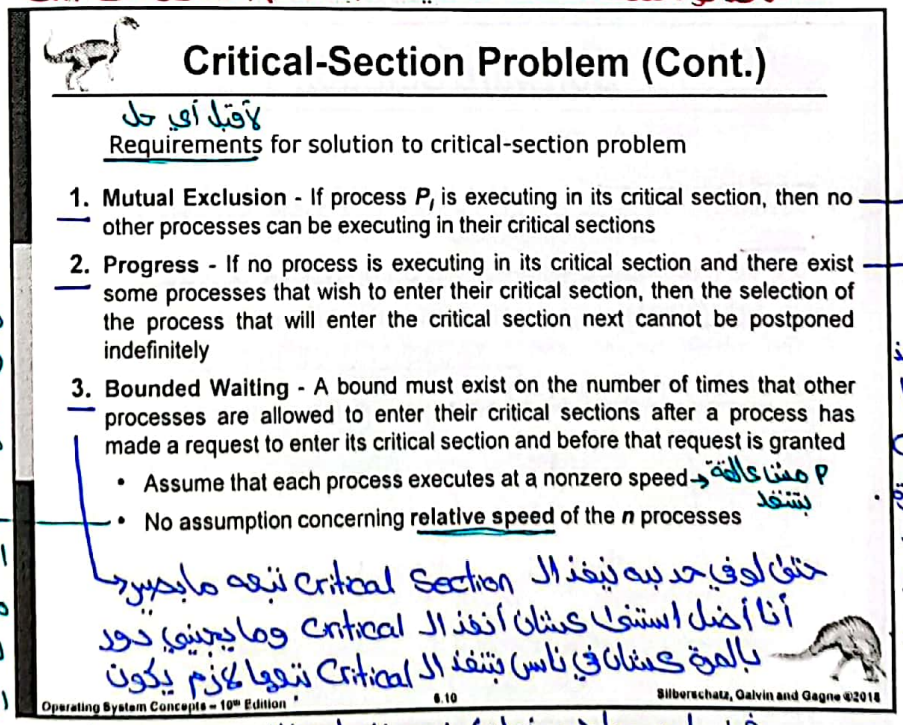
↑ Critical Section جزء بنظم بخولي وبخرجي من الـ بحيث لما أفوت حدا يفوت بنفس الوقت ولما أطلع الباقي يقدر يفوت.





9

Critical Section Problem الـ 3 قواعد لو تحققت معناها أنا حليت



ممنوع إذا فتت عالـ Critical section حد تاني يفوت بنفس الوقت معي

ما بجبر يكون الكود معلق بحيث إنه إذا ما كان في Process بتنفذ بالـ Critical section وأنا بدي أنفذ الـ Critical section تبعي وهي قادرة لازم طالما ما في حد بنفذ عالـ Critical section أقدر أنا أنفذ لو بدي أنفذ.

P معناها قادرة بتنفذ

قديه بدها وقت بال Critical ما بيهمني

المهم إنها ما تكون بتقاطعوا الـ Critical Section

حتى لو في حد بده ينفذ الـ Critical Section تبعه ما بجبر أنا أخلي استنى عشان أنفذ الـ Critical وما بجيبي دور بالمرة عشان في ناس بتنفذ الـ Critical تبعها لازم يكون

في الـ bound (حد أعلى) لعدد المرات اللي مسموح غيرها بأي Process إنها تنفذ الـ Critical Section تبعها بالوقت اللي أنا بستناها تخلص عشان أنفذ تبعي

10

أسهل طريقة لعمل Critical Section

Interrupt-based Solution

- Entry section: disable interrupts → Processor ماحد حيقدر يشيلني من ال
- Exit section: enable interrupts
- Will this solve the problem?
  - What if the critical section is code that runs for an hour?
  - Can some processes starve – never enter their critical section.
  - What if there are two CPUs?

* المشكلة لهاد الحل 1- إذا كان عنا multicore بس بتعمل disable على الـ core اللي الـ Process شغالة عليه الـ Cores التانين ما بيكون الو علاقة حيكملوا تنفيذ ما حيشوفوا الـ interrupt.
2- مش كل أنواع الـ Processes بقدروا يعملوا disable/Enable للـ interrupt، عادة الـ user processes ما بقدروا يعملوا هيك.
3- مش منطقي تسمح للـ Critical بضل شغال قد ما بده لحد ما يعمل Enable فممكن يصير فيه Starvation

ما بديني دور إني أفوت أبدًا

عا افتراض عندي فقط 2 processes

Software Solution 1

- Two process solution
- Assume that the `load` and `store` machine-language instructions are atomic; that is, cannot be interrupted
- The two processes share one variable: بقلي دور مين من الـ P
  - `int turn;`
- The variable `turn` indicates whose turn it is to enter the critical section
- initially, the value of `turn` is set to *i*

يعني لو الـ Process عملت load أو store ما بنقطع نهائي إلا لتخلص الـ store/load اللي عملته.

P_i P_j

turn = i turn = j

Algorithm for Process P_i

لو كان Turn=j ما حعمل اشي حضل معلقة استناه يخلص.

```
while (true){

      while (turn = = j);

      /* critical section */

      turn = j;

      /* remainder section */

}
```

← Entry Section (while (turn = = j);)

→ لو قتا هون معناه turn ≠ j (critical section)

→ Exit (turn = j;) يعني دور P_i مش P_j

فبنفذ ال Critical تبع P_i ولما أخلصه بخلي ال turn=j عشان تاخذ دورها.

* أول حد حيتنفذ هو i ولما خلص بعطيها لـ j وهكذا.

لو بدي اكتب code لـ j حيكون نفسه بس while(turn==i); و turn=i ;

Correctness of the Software Solution

- Mutual exclusion is preserved ✓ → حلها لإنه لو i بنفذ ال Critical تبعها j مستحيل تقدر تنفذ تبعها.

  P_i enters critical section only if:

  turn = i

  and turn cannot be both 0 and 1 at the same time
- What about the Progress requirement? ✗
- What about the Bounded-waiting requirement? ✓

← حلها لإنه كل واحد مسموح له يفوت بالترتيب مرة بعد الثاني فما حد حيستنى كثير وما يتنفذ.

← ما حلها لإنه لو بدي أفوت ال Critical وما في حد ثاني بده يفوت ال Critical رح تمنعني أفوت لإنه بتقول انه بعدما i يخلص لازم j يفوت بعدها.

فافرض j ما كان بده يفوت ال Critical وقتها مثلاً؟ حتضل i تستنى لحد ما j تقبل تفوت ال Critical.

Peterson's Solution

15

2 Processes بشتغل بس ↓

Peterson's Solution

- Two process solution
- Assume that the `load` and `store` machine-language instructions are atomic; that is, cannot be interrupted
- The two processes share two variables:
  - `int turn;`
  - `boolean flag[2]` → Flag[i] for P_i ، Flag[2] for P_j
- The variable `turn` indicates whose turn it is to enter the critical section
- The `flag` array is used to indicate if a process is ready to enter the critical section.
  - `flag[i] = true` implies that process P_i is ready!

↓ بتوضحلي لو الـ Process جاهز يدخل الـ Critical Section

16

الـ Solution المفضل

Algorithm for Process P_i

```
while (true){
        flag[i] = true;
        turn = j;
        while (flag[j] && turn = = j)
              ;
        /* critical section */
        flag[i] = false;
        /* remainder section */
}
```

Memory-barrier

Entry Section

Exit Section

P_i تبع

بتقول P_i أنا جاهزة أفوت الـ Critical بس لو P_j بعدها تفوت قبلي أعطيها الدور

* لو فقط طلعت ز اللي بتفوت بتشتي مابعمل شي ولو ما بعدها تفوت أنا بفوت.

* لو صار إنه اثنين بعدهم يفوتوا بنفس الوقت، اثنين حيكتبوا عالـ turn بس واحد فيوم حيكتب بعد الثاني فبناء عآخر قيمة بتنكتب واحد فيوم بفوت عالـ Critical.

الكود تاع P_j حيكون نفسه بس بدل كل مكان في ز حنحط ز وبدل كل مكان فيه ز حنحط ز.

17

Correctness of Peterson's Solution

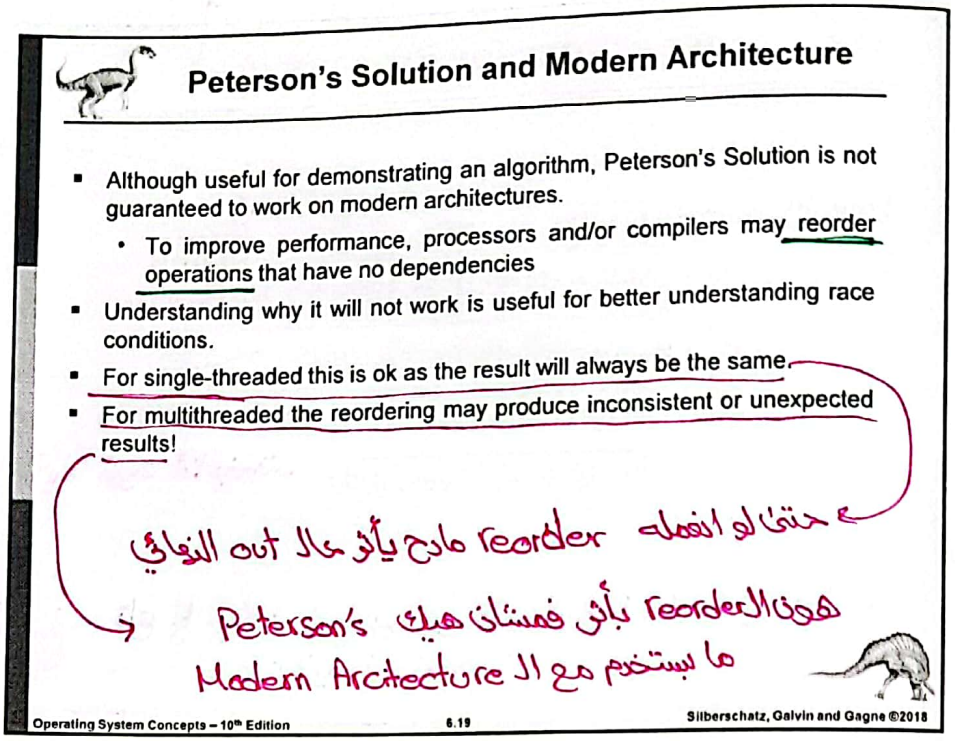
- Provable that the three CS requirement are met:
  1. Mutual exclusion is preserved
     P_i enters CS only if:
     either `flag[j] = false` or `turn = i`
  2. Progress requirement is satisfied
  3. Bounded-waiting requirement is met

حلوها لإنه مدام واحد بينفذ الثاني ما بينفذ بنفس الوقت.

انحلت لإنه لو بدي أفوت وماحد بده يفوت غيري بفوت عادي ما بحتاج استنى.

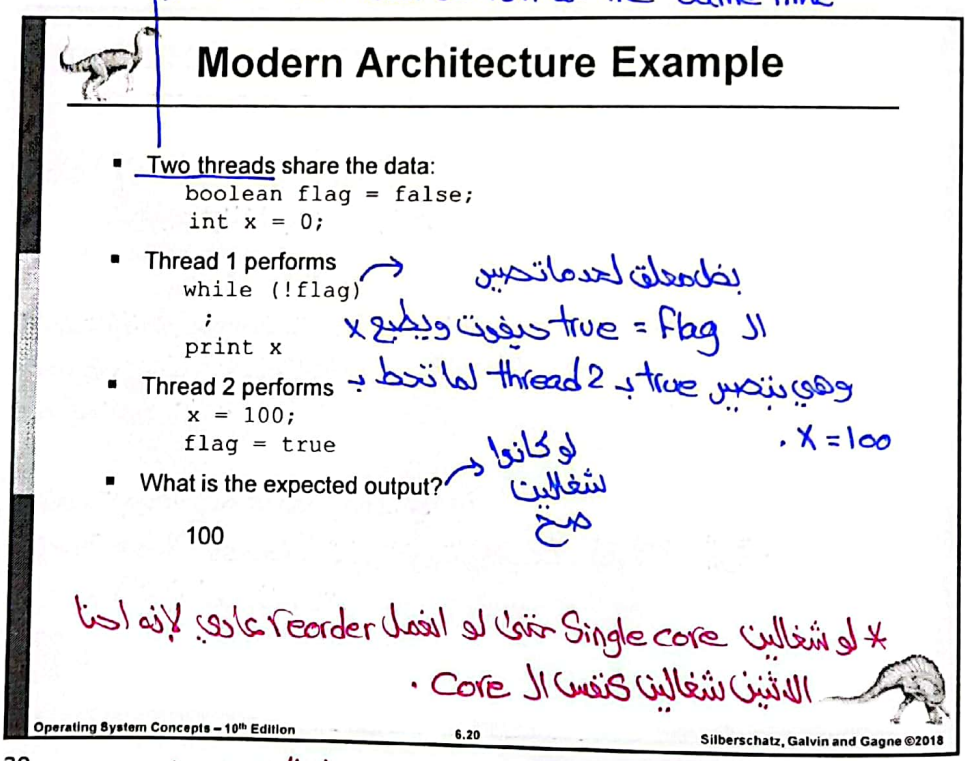
انحلت برضه لإنه عالأغلب مستحيل وحدة من الـ Process تظل تفوت وراء بعض لمرات كتيرة. والثانية ما تفوت.

18



19

Two task can be run at the same time



20

in parallel

* الخوف إذا ينفذوا بنفس الوقت وانعملوم reordering يعني بدّل بين X = 100; و flag=true قبل ما نكتب بـ X فطبع ناتج غلط.

Modern Architecture Example (Cont.)

- However, since the variables `flag` and `x` are independent of each other, the instructions:

```
flag = true;
x = 100;
```

for Thread 2 may be reordered

- If this occurs, the output may be 0!

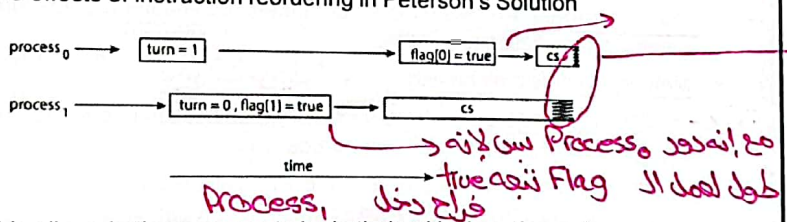
21

Processes هون بوضحلك لو انفعل Peterson's solution لـ reorder وكان الـ
بنفذوا بنفس الوقت (Parallel) شو حيصير؟ يم

Peterson's Solution Revisited

هون بكتشف إنه Process₀ ضلّ فبقوت عالـ CS

- The effects of instruction reordering in Peterson's Solution



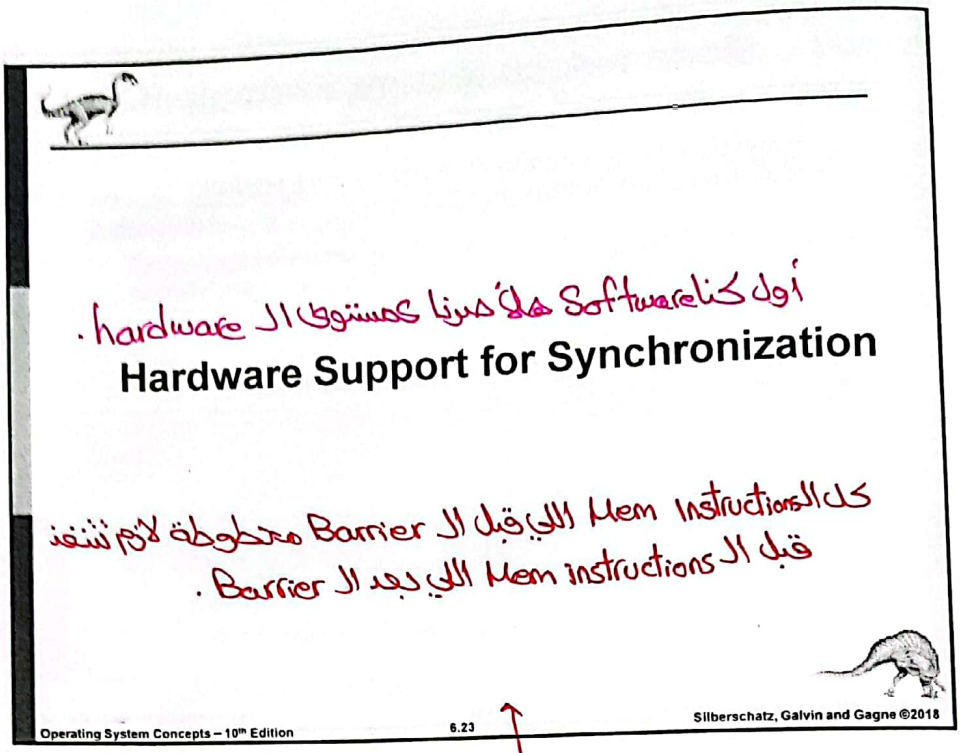
مع إنه Process₀ بس لإنه
طول لما الـ Flag تبعه true
Process₁ فراح دخل

فاتوا الاثنين عالـ CS
فبطل في معنى
Mutual
Exclusion.

- This allows both processes to be in their critical section at the same time!
- To ensure that Peterson's solution will work correctly on modern computer architecture we must use **Memory Barrier**.

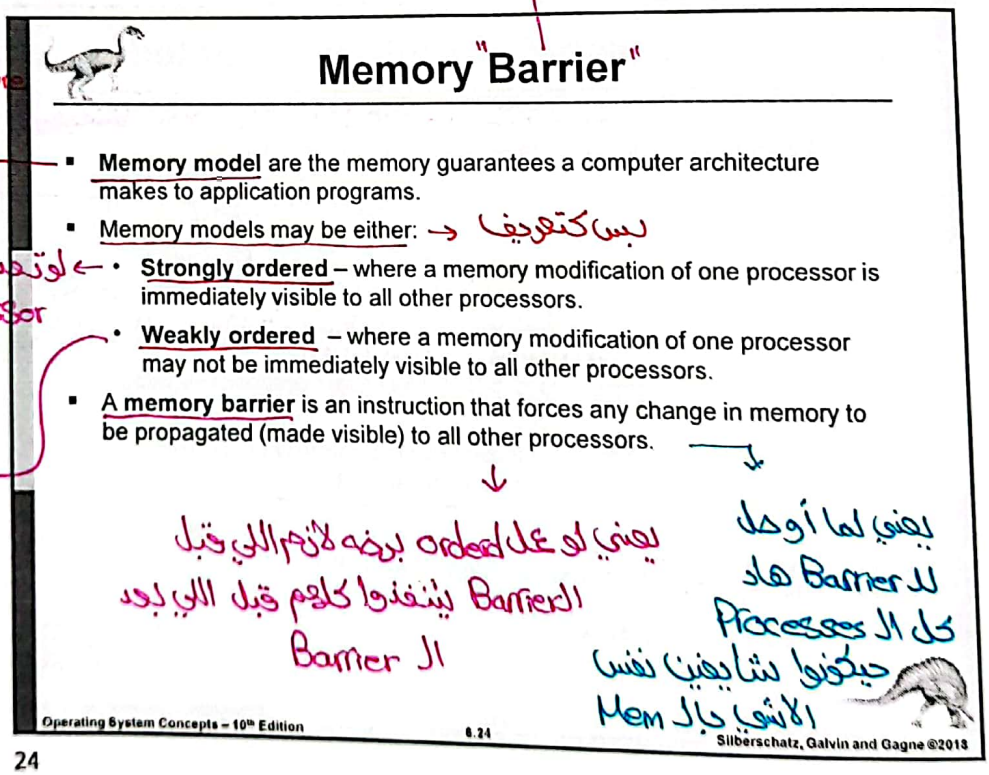
لـ عشان نحل هالمشكلة بنحتاج لمساعدة
hardware

22



23

حاجز



24

الضمانات اللي بيعطيها الـ Mem لـ Computer Architecture لـ Applications تبعتي

لو تعدل Mem من قبل الـ cores كل الـ Processor اللي باقين بشوفوا التعديل بسرعة

لو صار في تعديل مش ضروري بسرعة بيشوفوه

(بتذكي كيه وزام)

أي load أو store قبل الـ Barrier لازم كلها تنفذ قبل ما تنفذ لما بعد الـ Barrier.

Memory Barrier Instructions

- When a memory barrier instruction is performed, the system ensures that all loads and stores are completed before any subsequent load or store operations are performed.
- Therefore, even if instructions were reordered, the memory barrier ensures that the store operations are completed in memory and visible to other processors before future load or store operations are performed.

Memory Barrier Example

- Returning to the example of slides 6.17 - 6.18
- We could add a memory barrier to the following instructions to ensure Thread 1 outputs 100:
- Thread 1 now performs

```
while (!flag)
  memory_barrier();
print x
```

function call

- Thread 2 now performs

```
x = 100;
memory_barrier();
flag = true
```

يعني ماح يسمح للـ 2 processes انهم ينفذوا الأ لما الجنة اللي فوق flag=true و Print x الـ memory_barrier() يتحقق.

- For Thread 1 we are guaranteed that that the value of `flag` is loaded before the value of `x`.
- For Thread 2 we ensure that the assignment to `x` occurs before the assignment `flag`.

Memory_barrier فيك ما حيقدر يبدل بينهم لأنه في

x في الـ Peterson's عشان أحل مشكلة الـ reorder

بحط Memory_Barrier بين أول سطرين في كود الـ Peterson's، شوفي المكان بسلايد 6.17

Synchronization Hardware

- Many systems provide hardware support for implementing the critical section code.
- Uniprocessors – could disable interrupts
  - Currently running code would execute without preemption
  - Generally too inefficient on multiprocessor systems
    - Operating systems using this not broadly scalable
- We will look at three forms of hardware support:

1. Hardware instructions
2. Atomic variables

27

Hardware Instructions

- Special hardware instructions that allow us to either *test-and-modify* the content of a word, or to *swap* the contents of two words atomically (uninterruptedly.)
  - **Test-and-Set** instruction
  - **Compare-and-Swap** instruction

معرفين كمستوى ← Processor ال
هنا الاسم العام للهم

28

هاي الcodes توضيح لآلية
علهاي ال hardware inst

The test_and_set Instruction

- Definition

```
boolean test_and_set (boolean *target)
  {
        boolean rv = *target;
        *target = true;
        return rv:
  }
```

(value / register → بدك تعمله test ; target → address بمرّق ; *target = true → set عملتله)

- Properties
  - Executed atomically (يعني مستوى الhardware ماحدا رح يقطعه (uninterrupted))
  - Returns the original value of passed parameter → بترجع القيمة الأصلية
  - Set the new value of passed parameter to **true**

Solution Using test_and_set()

- Shared boolean variable lock, initialized to false
- Solution:

```
do {
      while (test_and_set(&lock))
      ; /* do nothing */

            /* critical section */

    lock = false;
            /* remainder section */
  } while (true);
```

(← بمرّق ال address)

بس يدخل بخلي lock true عشان ماحد يفوت وبس يخلص يعمل lock = false عشان الباقي يقدروا يفوتوا

- Does it solve the critical-section problem?

it's Not solve the bounding waiting

Starvation لـ مافي ضمانة إنو ما يصير

* الـ test_and_set بتشتغل كالعادة بس Compare-and-Swap بينما boolean بتستقبل أي Value يعني int value.

The compare_and_swap Instruction

- Definition

```
int compare_and_swap(int *value, int expected, int new_value)
{
   int temp = *value;
   if (*value == expected)
       *value = new_value;
   return temp;
}
```

إذا ماكانوا متساوين بترجعلك القيمة القديمة بدون ما تكتب وإذا متساوين بترجع وبتكتب.

- Properties
  - Executed atomically
  - Returns the original value of passed parameter `value`
  - Set the variable `value` the value of the passed parameter `new_value` but only if `*value == expected` is true. That is, the swap takes place only under this condition.

Solution using compare_and_swap

- Shared integer `lock` initialized to 0;
- Solution:

```
while (true){
    while (compare_and_swap(&lock, 0, 1) != 0)
        ; /* do nothing */

    /* critical section */

    lock = 0;

    /* remainder section */
}
```

لو كانت lock صفر فاكتب فيها واحد وطلع من الـ loop وأفوت بس لو ماكانت صفر حترجعلي Method الـ Critical رقم واحد وهاتستحلي أحط قيمة جديدة فخلي ألف بال loop

- Does it solve the critical-section problem?

لسا ماحلت الـ Starvation بس حنحلها باستخدام الـ Code الجاي

Bounded-waiting with compare-and-swap

Array boolean global

```
while (true) {
   waiting[i] = true;
   key = 1;
   while (waiting[i] && key == 1)
      key = compare_and_swap(&lock,0,1);
   waiting[i] = false;
   /* critical section */
   j = (i + 1) % n;
   while ((j != i) && !waiting[j])
      j = (j + 1) % n;
   if (j == i)
      lock = 0;
   else
      waiting[j] = false;
   /* remainder section */
}
```

Entry Part

Exit Part

أول مرة الـ while حتكون true

بشتغل circular

معناها إنه في واحد غيري بستنى

Starvation زيادة بس بضمنلنا في overhead *

33

كل Process إلها (time: n-1) لحد ما يجيلها دور، مستحيل وحدة يجيلها الدور مرتين قبل ما كل الـ وحدة يجيلها الدور مرة.

Atomic Variables

بنحتاجها لما يكون Multicore عندنا Variables معرفين جوا الـ API بتشتغل تعمل كل العمليات عليها بحيث الـ Variable إنها تكون Atomic (uninterruptible)

- Typically, instructions such as compare-and-swap are used as building blocks for other synchronization tools.
- One tool is an **atomic variable** that provides *atomic* (uninterruptible) updates on basic data types such as integers and booleans.
- For example:
  - Let `sequence` be an atomic variable
  - Let `increment()` be operation on the atomic variable `sequence`
  - The Command:

    `increment(&sequence);`

    ensures `sequence` is incremented without interruption:

وأنا بعمل increment ما حد يقدر يفوت مكانه بنفس الوقت.

34

Atomic Variables

- The `increment()` function can be implemented as follows:

```
void increment(atomic_int *v)
{
    int temp;
    do {
        temp = *v;
    }
    while (temp != (compare_and_swap(v,temp,temp+1));
}
```

لو تساوت العيقتين بطلع ولو لا بضل بلف جوا loop

* لو أي حد تاني حاول يعمل عملية على temp حيضل معلق لحد ما أنا أخلص.

هاد يصير لو حد فجأة فلت عالخط.

* Starvation * ممكن يصير فيه

Mutex Locks

Software tools ← مستوى الـ OS

الجزء اللي مابيحد يشتغل معي فيه باخذه الـ lock
وبس أخلص بترك الـ lock وهذ بصير بخليتا .

Mutex Locks

- Previous solutions are complicated and generally inaccessible to application programmers
- OS designers build software tools to solve critical section problem
- Simplest is mutex lock
  - Boolean variable indicating if lock is available or not
- Protect a critical section by
  1. First `acquire()` a lock → while (! available) ;
  2. Then `release()` the lock → available = false ;
- Calls to `acquire()` and `release()` must be atomic
  - Usually implemented via hardware atomic instructions such as compare-and-swap.
- But this solution requires busy wating → Variable كنتي كلما قيمته مش زي ما انا بدي يدخل ألف (في Process شغالة) ما الها overhead بس ممكن تاخذ time كبير .
  - This lock therefore called a spinlock → بلف يشيك على اشي ، كل اللي عليناه قبل يعتبر Spinlock

available = true ;

Solution to CS Problem Using Mutex Locks

```
while (true) {
        acquire lock

            critical section

        release lock

    remainder section
}
```

فكرتها بتكون إنه يا إما حد ماخذ الـ lock يا إما حد مش ماخذ الـ lock (on/off) مو شرط نروح لـ 1/0 بس ، يترجع قيم عادي.

Semaphores

↓

يشبه Mutex lock بس more general

39

Semaphores

* Synchronization tool that provides more sophisticated ways (than Mutex locks) for processes to synchronize their activities.
* Semaphore *S* – integer variable
  - Can only be accessed via two indivisible (atomic) operations
    - `wait()` and `signal()`
      - Originally called `P()` and `V()`
  - Definition of the `wait()` operation

```
wait(S) {
    while (S <= 0)
        ; // busy wait
    S--;
}
```

* ما لازم تنزل S عن صفر.

↳ بضل استنى

  - Definition of the `signal()` operation

```
signal(S) {
    S++;
}
```

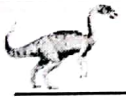
لما تحرر S أكبر برجع بنقصها

40

Semaphores (Cont.)

- **Counting semaphore** – integer value can range over an unrestricted domain
- **Binary semaphore** – integer value can range only between 0 and 1
  - Same as a **mutex lock**
- Can implement a counting semaphore *S* as a binary semaphore
- With semaphores we can solve various synchronization problems

41



Semaphore Usage Example

- Solution to the CS Problem
  - Create a semaphore "`mutex`" initialized to 1

```
wait(mutex);
    CS
signal(mutex);
```

- Consider P_1 and P_2 that with two statements S_1 and S_2 and the requirement that S_1 to happen before S_2
  - Create a semaphore "`synch`" initialized to 0

```
P1:
    S1;
    signal(synch);
P2:
    wait(synch);
    S2;
```

42

Implementation كيف معمولة من جوا ↓

Semaphore Implementation

- Must guarantee that no two processes can execute the `wait()` and `signal()` on the same semaphore at the same time
- Thus, the implementation becomes the critical section problem where the `wait` and `signal` code are placed in the critical section → كمان هيك بحطهم فيه
- ① Could now have **busy waiting** in critical section implementation → جوا الـ Signal
  - But implementation code is short
  - Little busy waiting if critical section rarely occupied → اعمل loop
- Note that applications may spend lots of time in critical sections and therefore this is not a good solution

لو وحدة بتعمل wait ما بصير التانية تعمل wait بنفس الوقت وبنفس الاشي والـ Signal بزبه بصير فيا Race condition

43

Semaphore Implementation with no Busy waiting

② With each semaphore there is an associated waiting queue

- Each entry in a waiting queue has two data items:
  - Value (of type integer) operation من الـ Process تقيم الـ
  - Pointer to next record in the list وتحطو جوا الـ Queue
- Two operations: ↑ تصير بالـ (wait())
  - **block** – place the process invoking the operation on the appropriate waiting queue (sleep)
  - **wakeup** – remove one of processes in the waiting queue and place it in the ready queue

بتفيق الـ Process وبتقيمو من الـ queue بتصير بالـ (signal())

44

Implementation with no Busy waiting (Cont.)

- Waiting queue

```
typedef struct {
   int value;
   struct process *list;
} semaphore;
```

list of Processes

45

Implementation with no Busy waiting (Cont.)

بعمل decrement قبل ما يفوت

```
wait(semaphore *S) {
   S->value--;
   if (S->value < 0) {
      add this process to S->list;
      block();
   }
}
```

جيب الـvalue اللي بـ S واعمل decrement، ممكن تصل قيمة سالبة.

فعليا بنضاف الـ PCB تاعها.

يعني في اشي خلّص

```
signal(semaphore *S) {
   S->value++;
   if (S->value <= 0) {
      remove a process P from S->list;
      wakeup(P);
   }
}
```

يعني في حد بستنى

Function call

بتروح عالصف بتفوق وحدة وبتقديها عالصف Ready.

46

الترتيب المنطقي لليما نعمل signal بعدين wait

Problems with Semaphores

- Incorrect use of semaphore operations:
  - `signal(mutex)` …. `wait(mutex)`
  - `wait(mutex)` … `wait(mutex)`
  - Omitting of `wait (mutex)` and/or `signal (mutex)`
- These – and others – are examples of what can occur when semaphores and other synchronization tools are used incorrectly.

شو بصير لو عمل هيك؟

بسمح أكثر من Process تفوت جوا الـ Critical

أو

Processes الـ بكونوا blocked للأبد

لازم الـ Programmer ينتبه وهو يكتب الترتيب

47

Monitors

48

Monitors

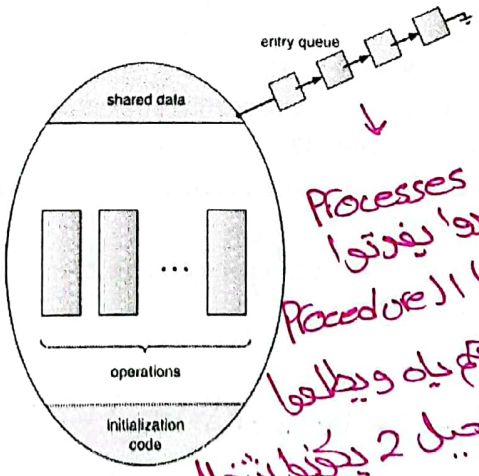
- A high-level abstraction that provides a convenient and effective mechanism for process synchronization
  - *Abstract data type*, internal variables only accessible by code within the procedure
  - Only one process may be active within the monitor at a time
  - Pseudocode syntax of a monitor:

```
monitor monitor-name
{
  // shared variable declarations
  procedure P1 (…) { …. }

  procedure P2 (…) { …. }

  procedure Pn (…) {……}

  initialization code (…) { … }
}
```

2procedure ما في يشتغلوا at the same time.

49

Schematic view of a Monitor



زي data type

Processes الـ بدخلوا يفوتوا Procedure الـ بناتوا اللي بيهمهم يطلبوها ومستحيل 2 يكونوا شغالين بنفس الوقت بس Process وحدة بتشغل بـ Procedure واحدة at the same time

50

Monitor Implementation Using Semaphores

- Variables

```
semaphore mutex
mutex = 1
```

binary Semaphore

- Each procedure *P* is replaced by

```
wait(mutex);
    ...
    body of P;
    ...
signal(mutex);
```

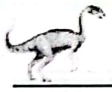
بتعمل block لما يكون الـ Semaphore ≤ 0

لما تكون 1 يعني ماحد شغال هلأ.

Mutex incremented

- Mutual exclusion within a monitor is ensured

* المشكلة إنه إذا الـ Process وهي جوا بتنفذ الـ Procedure وصارلها block حتنضل تشتغل الـ Monitor فالفاضي فالـ design مش flexibilty

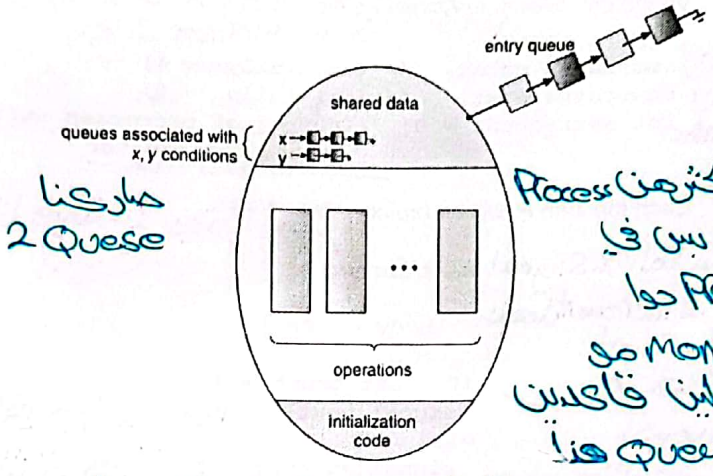


Condition Variables

- `condition x, y;`
- Two operations are allowed on a condition variable:
  - `x.wait()` – a process that invokes the operation is suspended until `x.signal()`
  - `x.signal()` – resumes one of processes (if any) that invoked `x.wait()`
    - If no `x.wait()` on the variable, then it has no effect on the variable

وظيفتهم إذا في Process جوا الـ Monitor ماهم تنفذ إنه ما يدخل الـ Monitor معك فالفاضي.

Monitor with Condition Variables



هاركنا 2 Queue

ماقي أكثر من Process بشغالة بس في Processes جوا الـ Monitor مو شغالين قاعدين بـ Queues هذا

بيعطينا Flexibility وما بعلق الـ Monitor كالغاني

53

Usage of Condition Variable Example

- Consider P_1 and P_2 that that need to execute two statements S_1 and S_2 and the requirement that S_1 to happen before S_2
  - Create a monitor with two procedures F_1 and F_2 that are invoked by P_1 and P_2 respectively
  - One condition variable "x" initialized to 0
  - One Boolean variable "done"

P1 →
```
F1:
    S1;
    done = true;
    x.signal();
```

كشان أضمن إنو S2 دايمًا تنفذ قبل

P2 →
```
F2:
    if done = false
       x.wait()
    S2;
```


54

الفرق هاد كندي Condition Variables .

Monitor Implementation Using Semaphores

- Variables

```
binary semaphore mutex;  // (initially = 1)
binary semaphore next;   // (initially = 0)
int next_count = 0;      // number of processes waiting
                         //   inside the monitor
```

- Each function *P* will be replaced by

```
wait(mutex);
...
body of P;
...
if (next_count > 0)
    signal(next)
else
    signal(mutex);
```

- Mutual exclusion within a monitor is ensured

55

Implementation – Condition Variables

- For each condition variable *x*, we have:

```
semaphore x_sem; // (initially = 0)
int x_count = 0;
```

- The operation `x.wait()` can be implemented as:

```
x_count++;
if (next_count > 0)
    signal(next);
else
    signal(mutex);
wait(x_sem);
x_count--;
```


56

execution

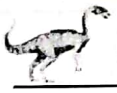
Process وظيفتها تروح على صف x تشوف لوقيا
بها ثفوت الـ monitor وبتدخلهم واحد واحد وبا بعض.

Implementation (Cont.)

- The operation `x.signal()` can be implemented as:

```
if (x_count > 0) {
   next_count++;
   signal(x_sem);
   wait(next);
   next_count--;
}
```

57



Resuming Processes within a Monitor

- If several processes queued on condition variable `x`, and `x.signal()` is executed, which process should be resumed?
- FCFS frequently not adequate → أطروحة احيت هي بتفوت
- Use the **conditional-wait** construct of the form

  `x.wait(c)` ← بيعطيني مين تنفذ أول.

  where:
  - `c` is an integer (called the priority number)
  - The process with lowest number (highest priority) is scheduled next

58

Single Resource allocation

- Allocate a single resource among competing processes using priority numbers that specifies the maximum time a process plans to use the resource

```
R.acquire(t);
    ...
  access the resource;
    ...

R.release;
```

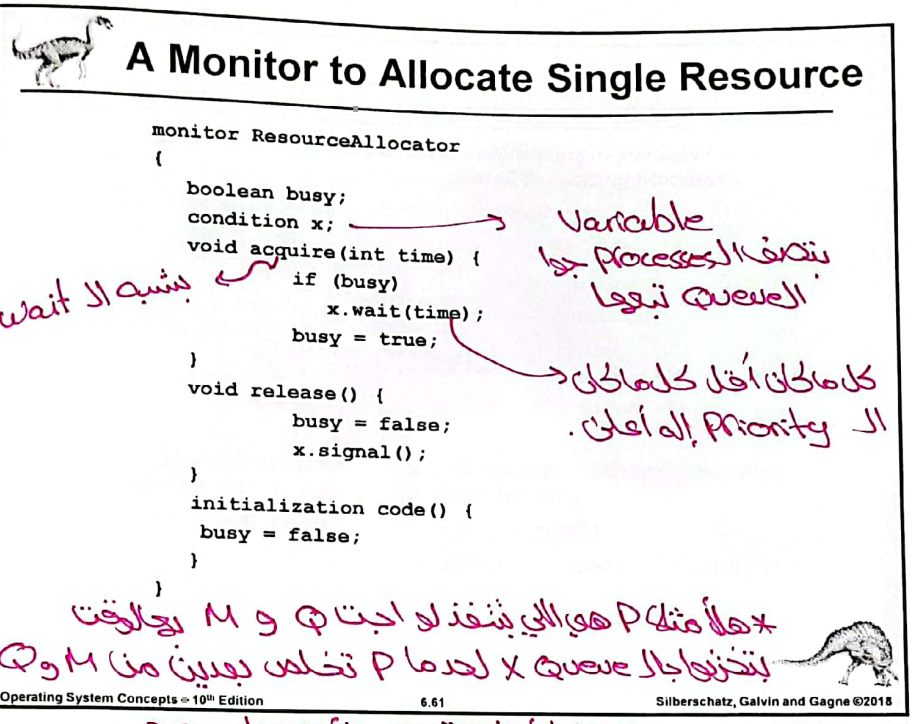
- Where R is an instance of type `ResourceAllocator`

Single Resource allocation

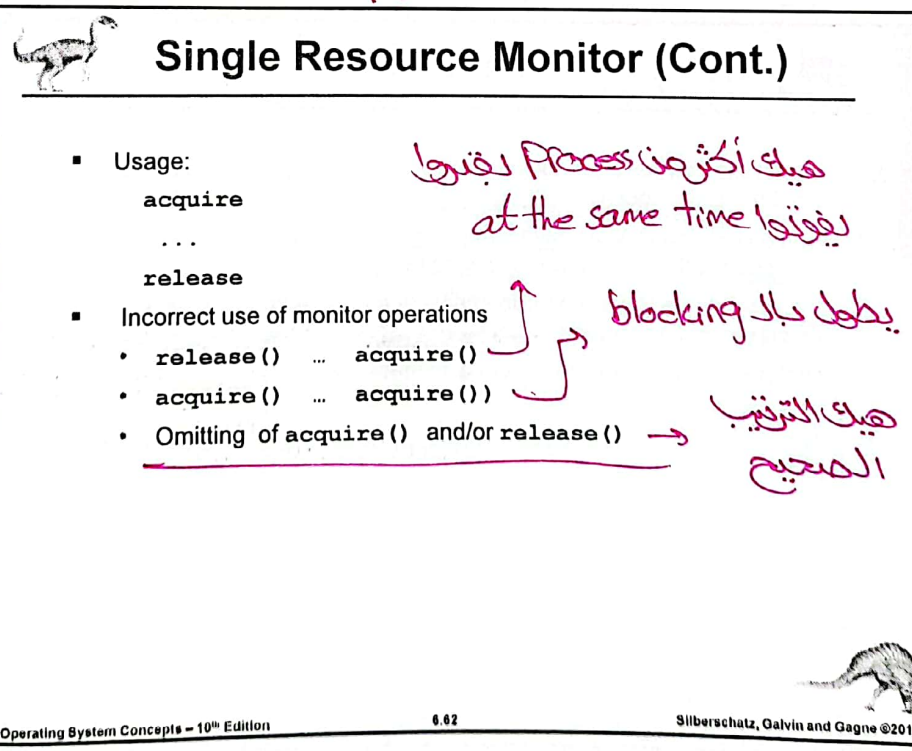
- Allocate a single resource among competing processes using priority numbers that specifies the maximum time a process plans to use the resource
- The process with the shortest time is allocated the resource first
- Let R is an instance of type `ResourceAllocator` (next slide)
- Access to `ResourceAllocator` is done via:

```
R.acquire(t);
    ...
  access the resource;
    ...
R.release;
```

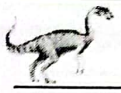
- Where `t` is the maximum time a process plans to use the resource



61

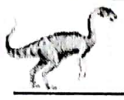


62



Liveness

63



Liveness

- Processes may have to wait indefinitely while trying to acquire a synchronization tool such as a mutex lock or semaphore.
- Waiting indefinitely violates the progress and bounded-waiting criteria discussed at the beginning of this chapter.
- **Liveness** refers to a set of properties that a system must satisfy to ensure processes make progress. → System لازم يحققها عشان الـ
- Indefinite waiting is an example of a liveness failure. يحققها لحتى يكون صحيح وماقيه Starvations

64

مشاكل بتعيق ال liveness

Liveness

- **Deadlock** – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes
- Let S and Q be two semaphores initialized to 1

P_0	P_1
`wait(S);`	`wait(Q);`
`wait(Q);`	`wait(S);`
`...`	`...`
`signal(S);`	`signal(Q);`
`signal(Q);`	`signal(S);`

كل وحدة فيهم بتستنى الثانية فصار فيه تعليق بالـ System.

- Consider if P_0 executes wait(S) and P_1 wait(Q). When P_0 executes wait(Q), it must wait until P_1 executes signal(Q)
- However, P_1 is waiting until P_0 execute signal(S).
- Since these signal() operations will never be executed, P_0 and P_1 are **deadlocked**.

65

Liveness

- Other forms of deadlock:
- **Starvation** – indefinite blocking
  - A process may never be removed from the semaphore queue in which it is suspended
- **Priority Inversion** – Scheduling problem when lower-priority process holds a lock needed by higher-priority process
  - Solved via **priority-inheritance protocol**

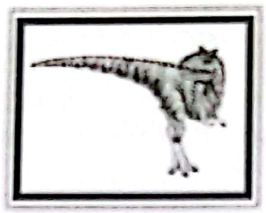
بكون كنبي Process بنفذ الـ Priority إلها أقل من الباقي وما بتسمح للي الـ Priority إلها أعلى إنها تقطعها وتنفذ ومكانك med تنفذ قبل ما يخلص الـ high

P1 2 → high
P2 5 → med
P3 7 → low → بنفذ هلا

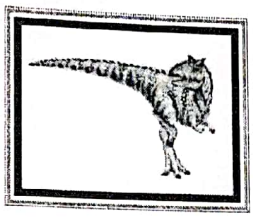
66

* مشاكل الـ System بحيث فيها هذا المشاكل.

End of Chapter 6



Chapter 7: Synchronization Examples



1

Outline

- Classical Problems of Synchronization (7.1)
- Synchronization within the Kernel (7.2)
- POSIX Synchronization (7.3)

2

Classical Problems of Synchronization

3

Classical Problems of Synchronization

- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

4

يعني الـ capacity معينة، بقدر أحط فيه عدد معين من الـ items.

حكيناه قبل

Bounded-Buffer Problem

locations كل بقدر أحط فيه one item

- *n* buffers, each can hold one item
- Semaphore **mutex** initialized to the value 1 → binary Semaphore → 1 → ما حدا ماسكه / في حدا ماسكه → 0 / ما حدا بفوت عالخط
- Semaphore **full** initialized to the value 0 → قديه عدد الـ buffers المشغولين
- Semaphore **empty** initialized to the value n → قديه عدد الـ buffers اللي فاضيين

Counting Semaphore (not binary)

5

* wait للـ Semaphore: طالما الـ Semaphore ≥ 0 بضل استنى غير ذلك بحيي بجيب item جديد.

Bounded Buffer Problem (Cont.)

- The structure of the producer process → الـ Producer بجيب

```
while (true) {
    ...
    /* produce an item in next_produced */
    ...
  wait(empty);
  wait(mutex);
    ...
    /* add next produced to the buffer */
    ...
  signal(mutex);
  signal(full);
}
```

wait → decrement بتعمل دايما لو كانت أصلا 0

wait(mutex) ← لازم يكون 1 عشان أقدر أفوت

signal → بتعمل increment عالـ Semaphore دايما

6

Criticals حلها Signal / wait *

Bounded Buffer Problem (Cont.)

- The structure of the consumer process بشتغل بالعكس

```
while (true) {
   wait(full);
   wait(mutex);
      ...
    /* remove an item from buffer to next_consumed */
      ...
   signal(mutex);
   signal(empty);
      ...
     /* consume the item in next consumed */
      ...
}
```

7

فكرة إنه يكون عندي data base مشتركة بين الـ Processes.

Readers-Writers Problem

- A data set is shared among a number of concurrent processes
  - * **Readers** – only read the data set; they do ***not*** perform any updates → ما بكون مشكلة لو أكثر من Process بتقرأ سوا
  - * **Writers** – can both read and write
- Problem – allow multiple readers to read at the same time
  - Only one single writer can access the shared data at the same time
- Several variations of how readers and writers are considered – all involve some form of priorities

المشكلة لو في Process بتعدل عالـ data و بنفس الوقت Process ثانية بتقرأ أو بتكتب

أكثر من حل حسب بعطي الـ Reader الـ Priority و الـ Writer.

8

.بيعطي أولوية للـ readers مش الـ writers

First Readers-writers problem

Readers-Writers Problem (Cont.)

- Shared Data
  - Data set
  - Semaphore `rw_mutex` initialized to 1
  - Semaphore `mutex` initialized to 1
  - Integer `read_count` initialized to 0

إذا في أي process بدها تفوت تكتب عالـ data بتاخذ هنا الـ lock

binary → not aquired

binary

counter int

عدد الـ Processes اللي بتنفذ

عشان نعمله Synchronize لعملية الـ read_count

Readers-Writers Problem (Cont.)

- The structure of a writer process

```
while (true) {
    wait(rw_mutex);
        ...
    /* writing is performed */
        ...
    signal(rw_mutex);
}
```

لو كانت 1 بتفوت بتعمل document وبتنفذ وبس تخلص بتعمل Signal فبعمل increment للـ rw_mutex عشان ترجع not aquired

* لازم كل الـ readers يقفوا لحد ما يخلصوا بعدين نسمح لـ writers يكتبوا.

Readers-Writers Problem (Cont.)

- The structure of a reader process

```
while (true) {
    wait(mutex);
    read_count++;
    if (read_count == 1) /* first reader */
        wait(rw_mutex);
    signal(mutex);
    ...
    /* reading is performed */
    ...
    wait(mutex);
    read_count--;
    if (read_count == 0) /* last reader */
        signal(rw_mutex);
    signal(mutex);
}
```

Critical Section

عشان أضمن مافي أكثر من حد يعمل read غيري أنا

shared with writers

عادي لو أكثر من process دخلوا بنفس الوقت هدا لأنه ضمنت إنه أول مرة أنا قلت (بس أول واحد الي بستنا)

هو بس الي بيعمل signal

* في حالة الـ starvation لـ writers لازم ممكن يستنوا كتير

11

لما هيك بخلصوا الـ readers فسمحوا لـ writers إنهم يفوتوا.

Readers-Writers Problem Variations

- The solution in previous slide can result in a situation where a writer process never writes. It is referred to as the "First reader-writer" problem.
- The "Second reader-writer" problem is a variation the first reader-writer problem that state:
  - Once a writer is ready to write, no "newly arrived reader" is allowed to read.
- Both the first and second may result in starvation. leading to even more variations
- Problem is solved on some systems by kernel providing reader-writer locks

↳ بعملوا lock لـ read و lock لـ write منفصلين مش زي أول شوف.

12

Dining-Philosophers Problem

- N philosophers' sit at a round table with a bowel of rice in the middle.

المشكلة بيوم هي ال chopsticks فبتحين المشكلة لو كان حدا كل فيكون آخذ اللي جنبه وشاكي فما بعرف بشو آكل.

- They spend their lives alternating thinking and eating.
- They do not interact with their neighbors.
- Occasionally try to pick up 2 chopsticks (one at a time) to eat from bowl
  - Need both to eat, then release both when done
- In the case of 5 philosophers, the shared data
  - Bowl of rice (data set)
  - Semaphore chopstick [5] initialized to 1

↳ Array of Semaphores binary

13

كل ما آكل أتأكد إنه → الـ 2 chopsticks فاضيين ما حد ماخذهم هيك أقدر آكل.

Dining-Philosophers Problem Algorithm

- Semaphore Solution
- The structure of Philosopher i :

كشان يكون Circular Array

```
while (true){
    wait (chopstick[i] );
    wait (chopStick[ (i + 1) % 5] );

    /* eat for awhile */

    signal (chopstick[i] );
    signal (chopstick[ (i + 1) % 5] );

    /* think for awhile */

}
```

أتأكد إنه الاثنين فاضيين ما حد ماخذهم

- What is the problem with this algorithm? deadlock may occure

بيشتغل 2 processes بوقت وحد بقدر يشتغل فيهم

14

في حلين ١٨ - أقل عدد الفيلسوفين واحد بس هيك ما بكون استخدمت كل الـ capacity

2 - الـ condition انه ما ينلقطوا 2 chopsticks الا سوا يا لا ممنوع واحد بس ينلقط

3 - أو أحط خيار انهم كلهم يلقطوا اليد اليمين أول مرة مثلاً

صف، انه كلهم قرروا ياكلوا بنفس الوقت ويمدوا wait لـ chopstick بنفس الوقت فكلهم لقطوا وحدة من اليمين . صاروا كل الـ chopsticks مأخوذين بس كل فيلسوف ماخذ وحدة، فوحد بقدر ياكل

« استخدمنا الحل الثالث »

Monitor Solution to Dining Philosophers

```
monitor DiningPhilosophers
{
   enum { THINKING; HUNGRY, EATING) state [5] ;
   condition self [5];

   void pickup (int i) {
          state[i] = HUNGRY;
          test(i);
          if (state[i] != EATING) self[i].wait;
   }

   void putdown (int i) {
          state[i] = THINKING;
                 // test left and right neighbors
          test((i + 4) % 5);
          test((i + 1) % 5);
   }
```

كل واحد من ال 5 اله state

it's not eating but want to eat

بحكي لجيرانه إنه أنا خلصت أكل
لو حد منكم hungry بيقدر ياخذ وياكل.

Solution to Dining Philosophers (Cont.)

very important

```
   void test (int i) {
          if ((state[(i + 4) % 5] != EATING) &&
          (state[i] == HUNGRY) &&
          (state[(i + 1) % 5] != EATING) ) {
               state[i] = EATING ;
          self[i].signal () ;
          }
   }

   initialization_code() {
      for (int i = 0; i < 5; i++)
      state[i] = THINKING;
      }
}
```

بتأكد لو أشماله ويمينه ما بياكلوا وهو جوعان بتغير حالته لـ eating كذلك بدخل ال CS لحد ما يخلص eating

ما إلها معنى لأنه أصلاً أنا مو waiting وما ناديت wait (هذا في حال فعلاً كانت waiting) فهي مهمة لما حد من اللي بياكلوا يترك ال chopstick ويمكن في حد جوعان.

Solution to Dining Philosophers (Cont.)

- Each philosopher "i" invokes the operations `pickup()` and `putdown()` in the following sequence:

```
DiningPhilosophers.pickup(i);

    /** EAT **/

DiningPhilosophers.putdown(i);
```

لو مش دوك إنك تاكل م حتظل معلق عندها لحد مايجي دورك بتاكل بعين ترجع ال chopsticks

- No deadlock, but starvation is possible

17

حناخده بشكل عام

Synchronization within the Kernel

18

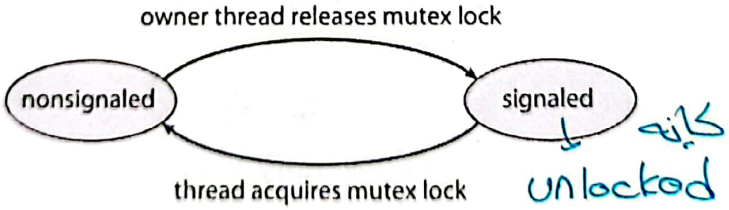
Kernel Synchronization - Windows

- Uses interrupt masks to protect access to global resources on uniprocessor systems
- Uses **spinlocks** on multiprocessor systems
  - Spinlocking-thread will never be preempted
- Also provides **dispatcher objects** user-land which may act mutexes, semaphores, events, and timers
  - **Events**
    - An event acts much like a condition variable
  - Timers notify one or more thread when time expired
  - Dispatcher objects either **signaled-state** (object available) or **non-signaled state** (thread will block)

19

Kernel Synchronization - Windows

- Mutex dispatcher object



20

Linux Synchronization

- Linux:
  - Prior to kernel Version 2.6, disables interrupts to implement short critical sections
  - Version 2.6 and later, fully preemptive
- Linux provides:
  1. Semaphores
  2. Atomic integers
  3. Spinlocks
  4. Reader-writer versions of both
- On single-CPU system, spinlocks replaced by enabling and disabling kernel preemption

Linux Synchronization

- Atomic variables

  `atomic_t` is the type for atomic integer
- Consider the variables

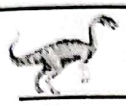
```
atomic_t counter;
int value;
```

Atomic Operation	*Effect*
`atomic_set(&counter,5);`	`counter = 5`
`atomic_add(10,&counter);`	`counter = counter + 10`
`atomic_sub(4,&counter);`	`counter = counter - 4`
`atomic_inc(&counter);`	`counter = counter + 1`
`value = atomic_read(&counter);`	`value = 12`

POSIX Synchronization

POSIX Synchronization

- POSIX API provides
  1. mutex locks
  2. semaphores
  3. condition variable
- Widely used on UNIX, Linux, and macOS



POSIX Mutex Locks

- Creating and initializing the lock

```
#include <pthread.h>

pthread_mutex_t mutex;

/* create and initialize the mutex lock */
pthread_mutex_init(&mutex,NULL);
```

- Acquiring and releasing the lock

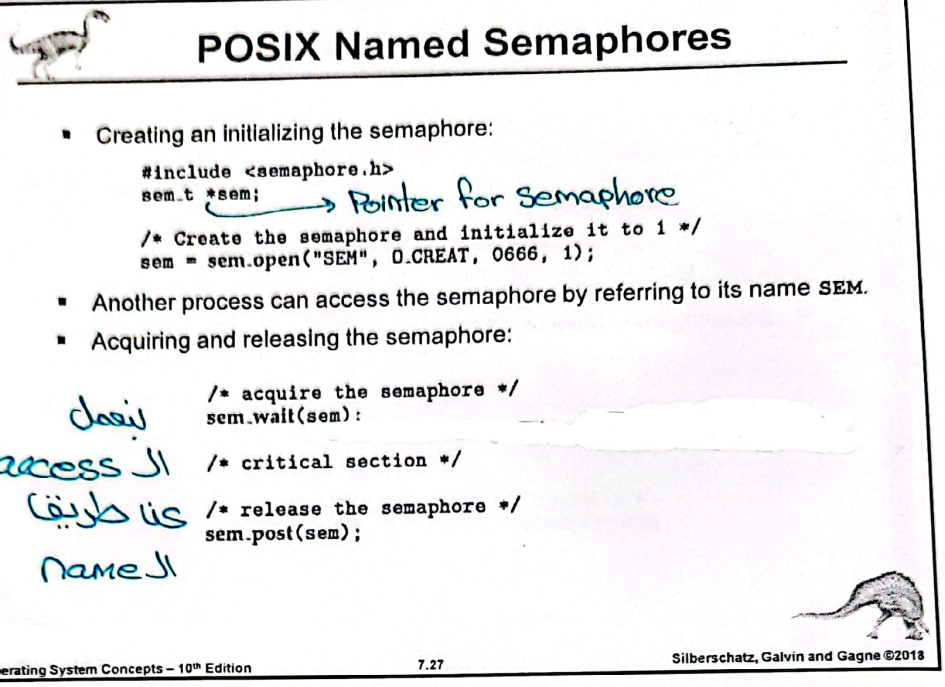
```
/* acquire the mutex lock */
pthread_mutex_lock(&mutex);

/* critical section */

/* release the mutex lock */
pthread_mutex_unlock(&mutex);
```

25

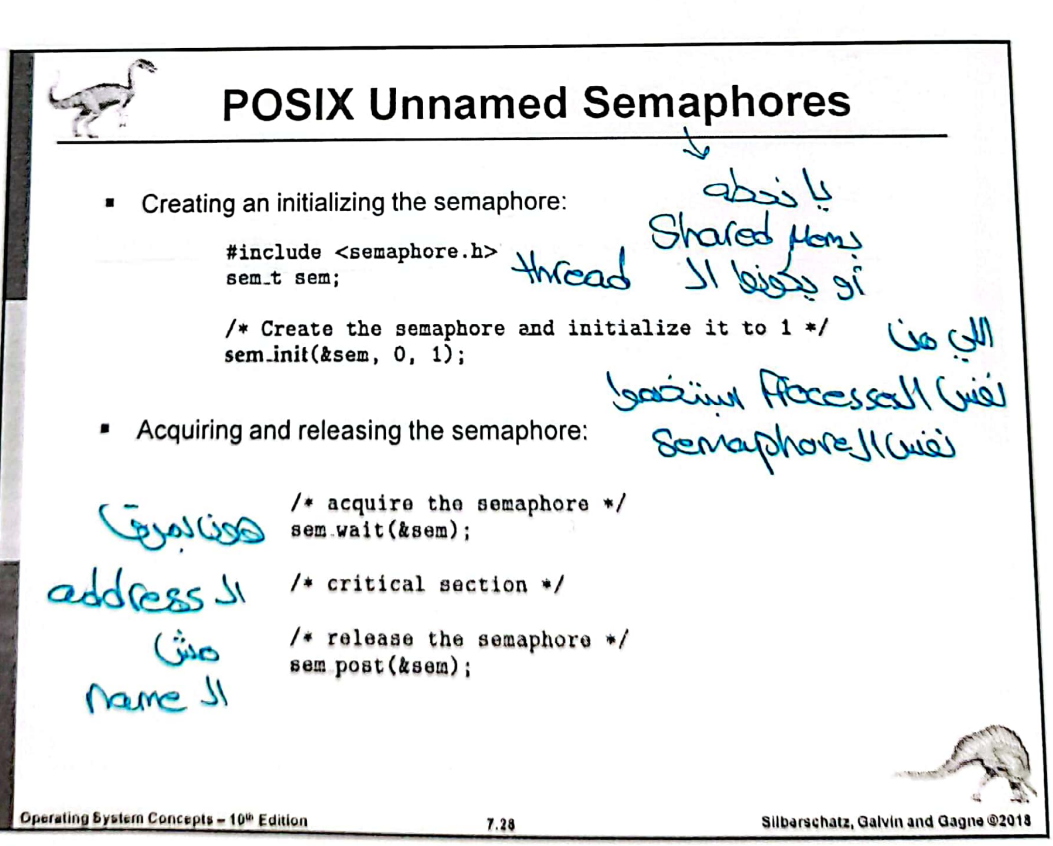
POSIX Semaphores

- POSIX provides two versions – named and unnamed.
- Named semaphores can be used by unrelated processes, unnamed cannot.

26



27



28

monitors ما في

POSIX Condition Variables

- Since POSIX is typically used in C/C++ and these languages do not provide a monitor, POSIX condition variables are associated with a POSIX mutex lock to provide mutual exclusion: Creating and initializing the condition variable:

```
pthread_mutex_t mutex;
pthread_cond_t cond_var;

pthread_mutex_init(&mutex,NULL);
pthread_cond_init(&cond_var,NULL);
```

} initialization with default value

29

POSIX Condition Variables

- Thread waiting for the condition `a == b` to become true:

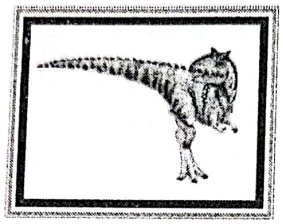
```
pthread_mutex_lock(&mutex);
while (a != b)
    pthread_cond_wait(&cond_var, &mutex);

pthread_mutex_unlock(&mutex);
```

- Thread signaling another thread waiting on the condition variable:

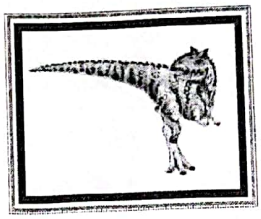
```
pthread_mutex_lock(&mutex);
a = b;
pthread_cond_signal(&cond_var);
pthread_mutex_unlock(&mutex);
```

30

End of Chapter 7



Chapter 8: Deadlocks



1

Outline

- System Model (8.1)
- Deadlock in Multithreaded Applications (8.2)
- Deadlock Characterization (8.3)
- Methods for Handling Deadlocks (8.4)

2

Chapter Objectives

- Illustrate how deadlock can occur when mutex locks are used
- Define the four necessary conditions that characterize deadlock
- Identify a deadlock situation in a resource allocation graph

3

System Model

4

مواصفات الـ System اللي ممكن يصير
deadlock فيه

System Model

1. System consists of resources
   - Resource types $R_1, R_2, \ldots, R_m$
     - *CPU cycles, memory space, I/O devices*
2. Each resource type R_i has W_i instances.
3. Each process utilizes a resource as follows:
   - a. **request** : The thread requests the resource. If the request cannot be granted then the requesting thread must wait until it can acquire the resource.
   - b. **use** : The thread can operate on the resource.
   - c. **Release:** The thread releases the resource.

يختلف تعريفهم حسب الـ Resource
بين الصف والـ .

5

مثال على resource بصير فيها deadlock

Deadlock with Semaphores

- Data:
  - A semaphore `S1` initialized to 1
  - A semaphore `S2` initialized to 1
- Two processes P1 and P2
- `P1:`

```
wait(s1)   S1=0
wait(s2)
```

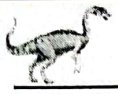
- `P2:`

```
wait(s2)   S2=0
wait(s1)
```

الاثنين بيستنوا بعض وما حدا
بقدر يشتغل.

مثال على الـ deadlock بيوقع Dining-philosopher

6



Deadlock in Multithreaded Applications

7

Deadlock in Pthread Program

- The pthread_mutex_init() function initializes an unlocked mutex.
- Mutex locks are acquired and released using pthread_mutex_lock() and pthread_mutex_unlock(), respectively.
- If a thread attempts to acquire a locked mutex, the call to pthread_mutex_lock() blocks the thread until the owner of the mutex lock invokes pthread_mutex_unlock().
- Two mutex locks are created and initialized in the following code example:

```
pthread_mutex_t first_mutex;
pthread_mutex_t second_mutex;

pthread_mutex_init(&first_mutex,NULL);
pthread_mutex_init(&second_mutex,NULL);
```

8

مش حنفوت بتفاصيله كامل.

Deadlock in Pthread Program (Cont.)

- Next, two threads: thread_one and thread_two are created.
- Both these threads have access to both mutex locks.
- thread_one and thread_two run in the functions do_work_one() and do_work_two(), respectively.
- thread_one attempts to acquire the mutex locks in the order (1) first mutex, (2) second mutex.
- At the same time, thread_two attempts to acquire the mutex locks in the order (1) second mutex, (2) first mutex.

```
/* thread_one runs in this function */
void *do_work_one(void *param)
{
   pthread_mutex_lock(&first_mutex);
   pthread_mutex_lock(&second_mutex);
   /**
    * Do some work
    */
   pthread_mutex_unlock(&second_mutex);
   pthread_mutex_unlock(&first_mutex);

   pthread_exit(0);
}

/* thread_two runs in this function */
void *do_work_two(void *param)
{
   pthread_mutex_lock(&second_mutex);
   pthread_mutex_lock(&first_mutex);
   /**
    * Do some work
    */
   pthread_mutex_unlock(&first_mutex);
   pthread_mutex_unlock(&second_mutex);

   pthread_exit(0);
}
```

9

* مش ضروري دايما يصير الـ deadlock، حسب الترتيب اللي ماشي فيه البرنامج بتحدد لو فيه deadlock أو لأ.

Deadlock in Pthread Program (Cont.)

- Deadlock is possible if thread one acquires first mutex while thread two acquires second mutex.
- Note that, even though deadlock is possible, it will not occur if thread one can acquire and release the mutex locks for first mutex and second mutex before thread two attempts to acquire the locks.
- The order in which the threads run depends on how they are scheduled by the CPU scheduler.

10

يختلف عالdeadlock بانه بصير فيه ثريد أو كم ثريد قدرة كل الهم
كجزء معين بس ما بنعمل block كامل.

Livelock

- **Livelock** is another form of liveness failure.
- Livelock occurs when a thread continuously attempts an action that fails.
- Livelock typically occurs when threads retry failing operations at the same time.
  - It thus can generally be avoided by having each thread retry the failing operation at random times.
- Livelock can be illustrated with the Pthreads pthread_mutex_trylock() function, which attempts to acquire a mutex lock without blocking.

لما ما يعمل block وبنتج عالآخر.

11

Livelock (Example)

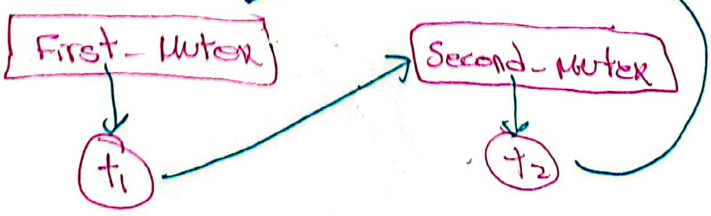
```
/* thread_one runs in this function */
void *do_work_one(void *param)
{
   int done = 0;

   while (!done) {
      pthread_mutex_lock(&first_mutex);
      if (pthread_mutex_trylock(&second_mutex)) {
         /**
          * Do some work
          */
         pthread_mutex_unlock(&second_mutex);
         pthread_mutex_unlock(&first_mutex);
         done = 1;
      }
      else
         pthread_mutex_unlock(&first_mutex);
   }

   pthread_exit(0);
}
```

```
/* thread_two runs in this function */
void *do_work_two(void *param)
{
   int done = 0;

   while (!done) {
      pthread_mutex_lock(&second_mutex);
      if (pthread_mutex_trylock(&first_mutex)) {
         /**
          * Do some work
          */
         pthread_mutex_unlock(&first_mutex);
         pthread_mutex_unlock(&second_mutex);
         done = 1;
      }
      else
         pthread_mutex_unlock(&second_mutex);
   }

   pthread_exit(0);
}
```

Code of thread 1 Code of thread 2

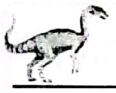
12

هيك بصير الـ deadlock بسلايد ٩



Deadlock Characterization

13



Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

لازم يكونوا موجودين ليحدث الـdeadlock

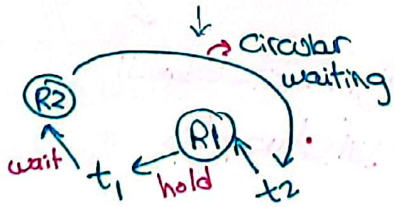
1- ▪ **Mutual exclusion:** only one process at a time can use a resource

2- ▪ **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes

3- ▪ **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task

4- ▪ **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, ..., P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

*R → Resource



14

هاد الـProcess اما يستخدم الـResource او بيستنى لحاله

مستحيل ينقطع على الـProcesses هاي (threads)

* اتجاه السهم من الـResource لـthread معناها هذا الـResource الها اصلا thread.

* اتجاه السهم من الـthread للـResource يعني أنا بستنى هذا الـResource

Resource-Allocation Graph

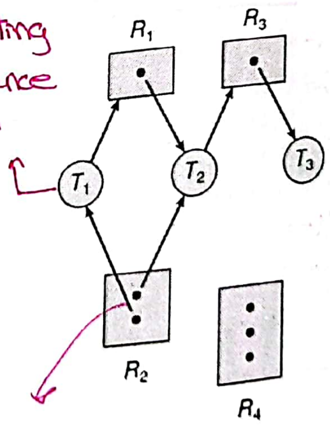
A set of vertices V and a set of edges E.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system (or threads ←)
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$

Resource Allocation Graph Example

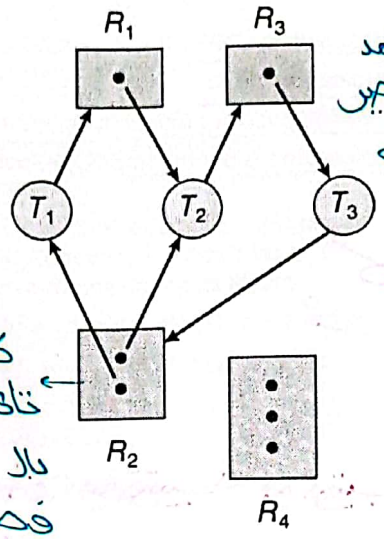
- One instance of R1
- Two instances of R2
- One instance of R3
- Three instance of R4
- T1 holds one instance of R2 and is waiting for an instance of R1
- T2 holds one instance of R1, one instance of R2, and is waiting for an instance of R3
- T3 holds one instance of R3

Requesting one instance of R1



ال assignment Edge السهم بطلع من
holding ال instance اي بحسابي جوا

Resource Allocation Graph with a Deadlock



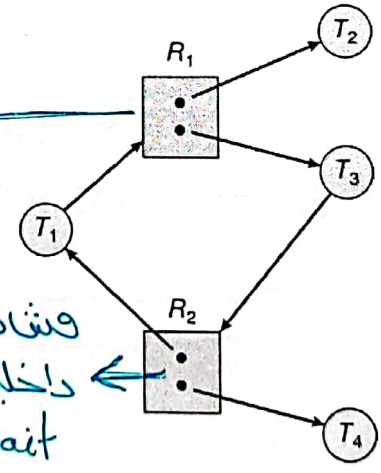
كل وحدة بتعتمد
والتانية فحيبن
في deadlock

instances كل ال
تانين R2 داخلين

Circular wait بال
فحدا في
deadlock

17

Graph with a Cycle But no Deadlock

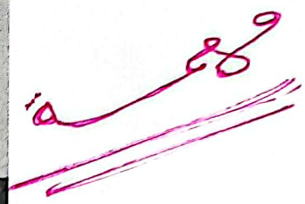


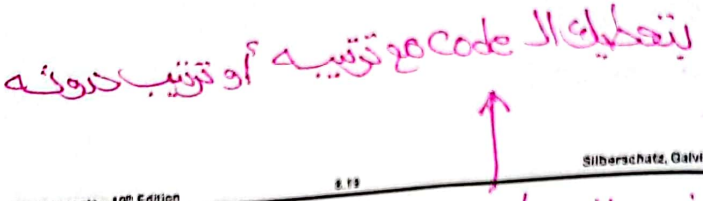
instances وشاكل ال
Circular داخلين بال
wait فحاق كنا
deadlock دون

18

Basic Facts

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock





19

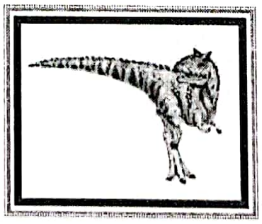
Methods for Handling Deadlocks

20

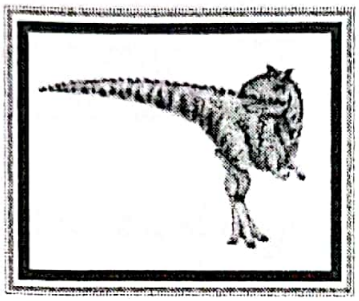
Methods for Handling Deadlocks

1. Ensure that the system will never enter a deadlock state:
   - a. Deadlock prevention
     - Provides a set of methods to ensure that at least one of the necessary conditions cannot hold.
   - b. Deadlock avoidance
     - Requires that the operating system be given additional information in advance concerning which resources a thread will request and use during its lifetime.
2. Allow the system to enter a deadlock state and then recover
3. Ignore the problem and pretend that deadlocks never occur in the system.

End of Chapter 8



Chapter 9: Main Memory



Chapter 9: Memory Management

- Background (9.1)
- Contiguous Memory Allocation (9.2)
- Paging (9.3)
- Structure of the Page Table (9.4)
- Swapping (9.5)
- Example: The Intel 32 and 64-bit Architectures (9.6)

Objectives

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques,
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging

3

Background

4

ال cpu عمره مابروح لل disk عشان يجيب معلوماته وينفذ اشي معين

Background

Instruction cycle:
1- Fetch
2- decode
3- load
4- execute
5- Store

- * Program must be brought (from disk) into memory and placed within a process for it to be run
- * Main memory and registers are only storage CPU can access directly
  - Memory unit only sees a stream of:
    - a. addresses + read requests, or
    - b. address + data and write requests
  - Register access is done in one CPU clock (or less)
  - Main memory can take many cycles, causing a **stall**
  - **Cache** sits between main memory and CPU registers → لتسرع ال access عال Main Mem
  - Protection of memory required to ensure correct operation

Register → أسرع اشي ال cpu بقدر ال access عليه

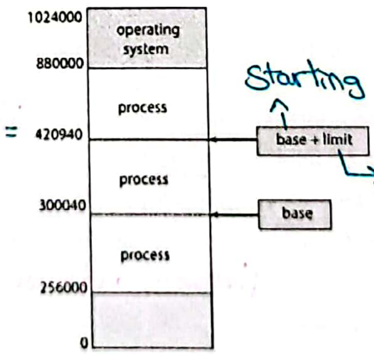
→ يمنع أي Process إنها تعمل access لإلا address space تبعها

بتختلف من OS للثانية

Protection

- * Need to censure that a process can access only access those addresses in its address space.
- * We can provide this protection by using a pair of **base** and **limit registers** define the logical address space of a process

limit ↑ 300040+120900 = 420940

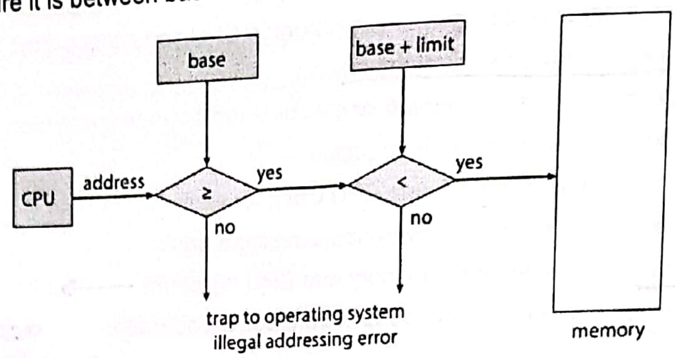


Starting address ↑ (base + limit)

بحدد ال Size تبع ال address space تبع هاي ال Process

Hardware Address Protection

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



- The instructions to loading the base and limit registers are privileged

لـ kernal instruction ← بقوم فيها الـ OS ← الـ User
ما بتقدر تعدل عهاي الـ Values

* الـ OS ماخذ صلاحيات الملاحيات كيف يتعامل مع الـ Main Memory.

Address Binding

- Programs on disk, ready to be brought into memory to execute form an **input queue**
  - Without support, must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000
  - How can it not be?
- Addresses represented in different ways at different stages of a program's life
  - Source code addresses usually symbolic like: Count, array, ...
  - Compiled code addresses **bind** to relocatable addresses → وين مكان هنا (Relative addresses)
    - i.e., "14 bytes from beginning of this module" الـ Var بالنسبة لل block
  - Linker or loader will bind relocatable addresses to absolute addresses
    - i.e., 74014 → Physical addresses → الموقع الحقيقي لل Var جوا الـ Mem
  - Each binding maps one address space to another

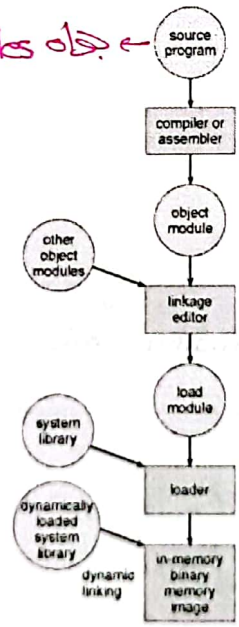
Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  1. **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  2. **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  3. **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
     - Need hardware support for address maps (e.g., base and limit registers)

Operating System Concepts – 10th Edition 9.9 Silberschatz, Galvin and Gagne ©2018

9

Multistep Processing of a User Program



Operating System Concepts – 10th Edition 9.10 Silberschatz, Galvin and Gagne ©2018

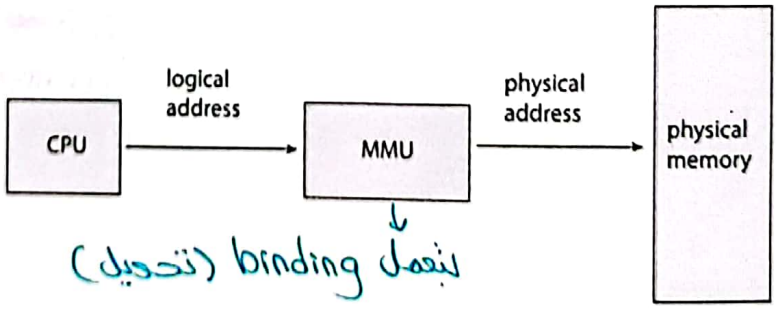
10

Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program (actual space)

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address



(تحويل) binding بنعمل

- Many methods possible, covered in the rest of this chapter

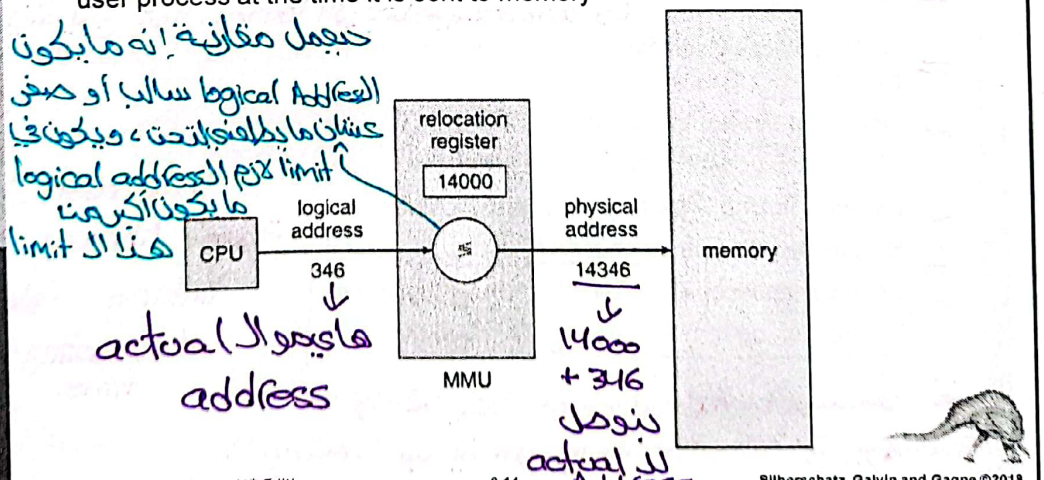
Memory-Management Unit (Cont.)

- Consider simple scheme. which is a generalization of the base-register scheme.
- The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

13

Memory-Management Unit (Cont.)

- Consider simple scheme. which is a generalization of the base-register scheme.
- The base register now called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory



14

ما بعمل load لكل الـ routine بعمل بس الـ main routine
وبعدين بحيث أعمل load للشغلات اللي بتتطلب.

Dynamic Loading

- The entire program does need to be in memory to execute
- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format → physical مش
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading

حالة بتكون مش من الـ OS support
مثال API.

15

Dynamic Linking

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- Dynamic linking –linking postponed until execution time
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
  - Versioning may be needed

وظيفته يتبع شوال الـ library اللي بعملوم invocation
ولما أعمل invocation لـ library معينة linking بعمل at executing (run) time.

لأنه لما يكون Multiple processes بيستخدموا نفس الـ library بس بيعملوا مرة واحدة بتتخزن shared مرة واحدة linking

16

لكل ما يخطر لعقل كمان مرة.

① Contiguous Memory Allocation

الطرق اللي بنعمل فيهم binding :

① Contiguous Memory Allocation
② Paging

17

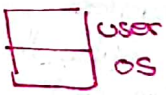
يعني متجاور، جنب بعض

يكون كل بمكان واحد بالـ Memory متقسمة

Finite (محدودة)

Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory



18

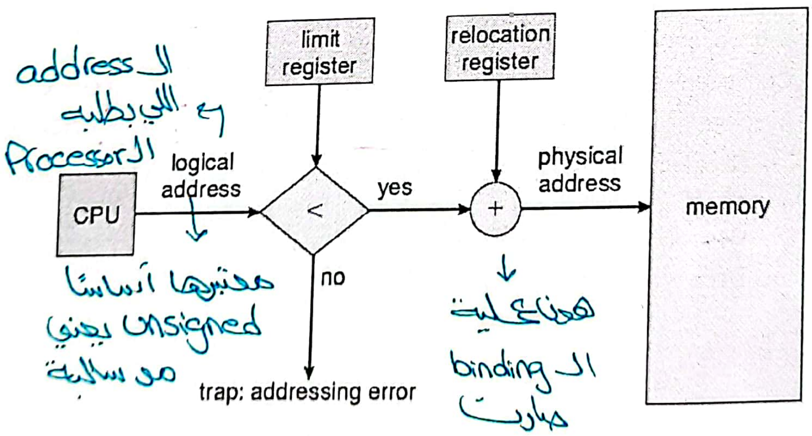
Contiguous Allocation (Cont.)

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** and kernel changing size

كل Process إلها Space معينة خاصة فيها بالـ Mem
وهنا يختلف من Process لـ Process.

*limit → قديه الـ Size تاع الـ block في الـ Mem
* relocation reg → الـ Start address اللي بنبلش منه

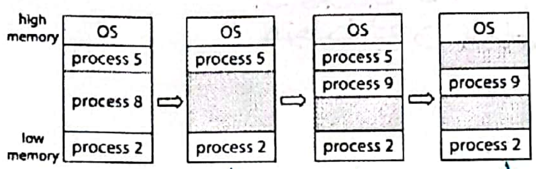
Hardware Support for Relocation and Limit Registers



ولا ضروري تكون نفس الـ Size الـ processes التابعين
كل Process تيجي مش ضروري ينعملها loading بنفس المكان

Variable Partition

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory (Free spaces)
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



وهكذا ...
خلصت process 8 — اتحطت process 9 — خلصت process 5

21

Dynamic Storage-Allocation Problem

أي hole استخدم لـ process اللي انعملها loading

How to satisfy a request of size *n* from a list of free holes?

1. **First-fit**: Allocate the ***first*** hole that is big enough
2. **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size → لضرورة تكفي الـ process عشان
   - Produces the smallest leftover hole (ما يكون في فراغ كبير (مش طبيعي أحسن))
3. **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
   - Produces the largest leftover hole → نبدور على أكبر وحدة

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

بيكون الـ OS معاه زي Pointer واصل فيه لآخر مكان عملت فيه allocation فلما تيجي process جديدة يكمل تدوير لأول next hole بالـ Mem وأول hole بتكفي بحطها فيها.

بس حيضل عنا مشكلتين مع هذا النوع من الـ Allocation اللي هم 8 External and internal Fragmentation

22

معناها بدخل كتير من Fragments الـ Memory مابنقدر نستخدمها.

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

اجزاء فاضية مابقدر استخدمها لإنها مشاصب بعض تتاتيف كلواحد بجوة

مثلاً Process بكون بيها 1428 بيعطيها 1440 فبضل في تتاتيف جوا مو مستخدمة ومابنقدر نستخدمها

مهمة

↳ 1/3 blocks الـ unallocated منها ضايع و

23

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

الحل إنه بدون كل الـ allocated جنب بعض والـ unallocated جنب بعض

لو كان كنا I/O ما بقدر أعمل Compaction بنفس الوقت، الحل بخلي الـ I/O بصير دا الـ OS اللي ما بتلعب فيها أساسًا.

↓

الحل المسؤول كنا عاملينه Realocation و Swapping الـ

Flexible وبتاخد عالية overhead الـ Compaction بس الـ

24

فبروح لشغلة الـ Paging

② Paging

25

Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a page table to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

26

Address Translation Scheme

- Address generated by CPU is divided into:
  - ① Page number (p) – used as an index into a **page table** which contains base address of each page in physical memory
  - ② Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit

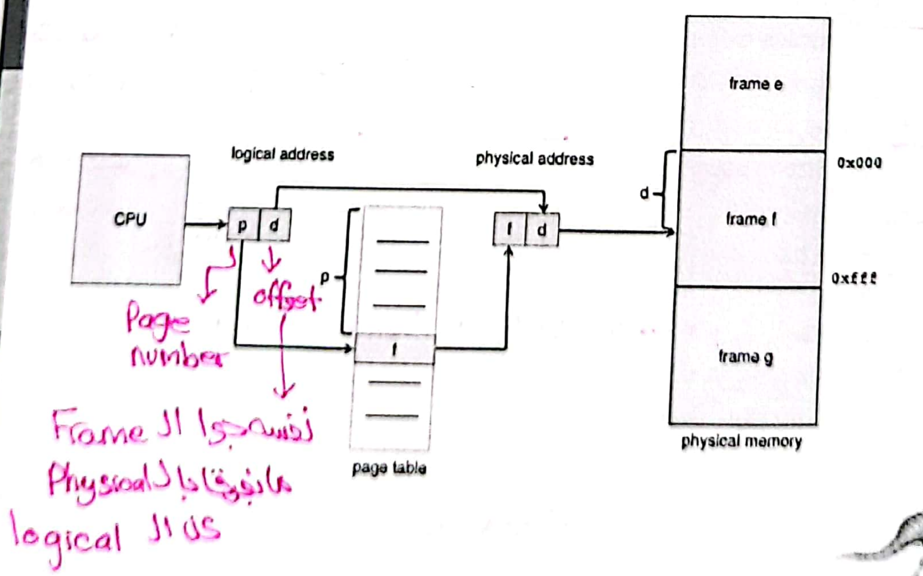
page number	page offset
p	d
m - n	n → Page size يحددلنا ال

- For given logical address space 2^m and page size 2^n

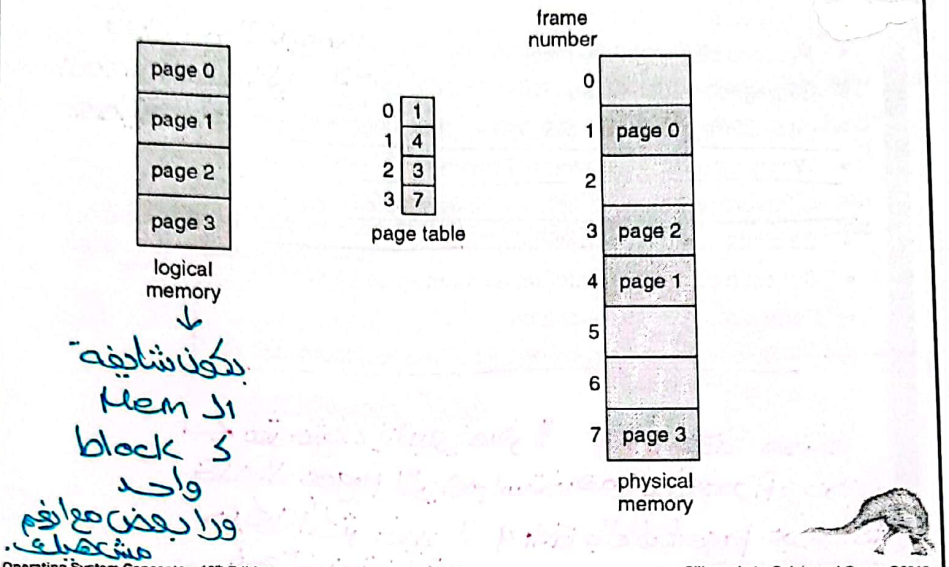
logical address (Mem الي طالعتلي) = m bits

⇒ logical space = 2^m

Paging Hardware



Paging Model of Logical and Physical Memory

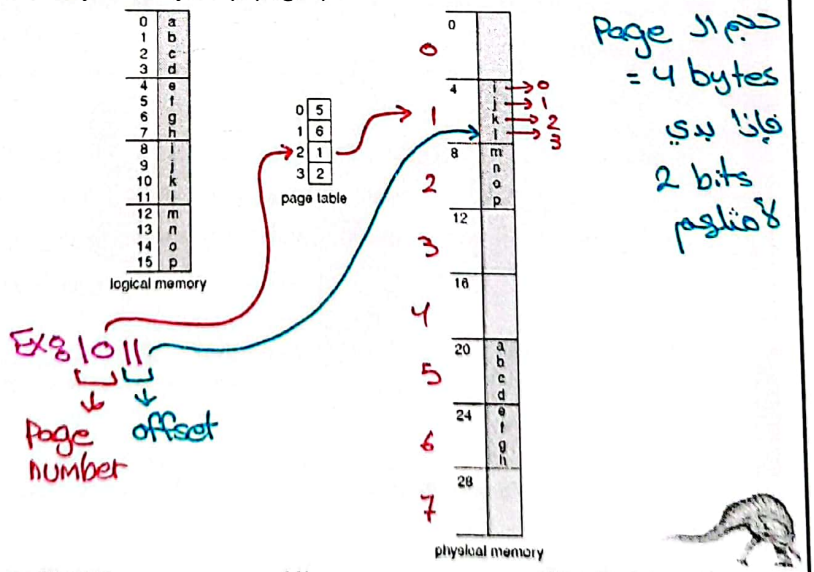


بكون شايفة Mem ال block ك واحد وزا بعض معانوم مشعبك.

29

Paging Example

- Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)



Ex: 8 | 0 | 1 | 1
Page number | offset

حجم ال Page = 4 bytes فإذا بدي 2 bits وعلوم

30

Paging -- Calculating internal fragmentation

- Page size = 2,048 bytes
- Process size = 72,766 bytes
- 35 pages + 1,086 bytes
- Internal fragmentation of 2,048 - 1,086 = 962 bytes
- Worst case fragmentation = 1 frame – 1 byte
- On average fragmentation = 1 / 2 frame size
- So small frame sizes desirable?
- But each page table entry takes memory to track
- Page sizes growing over time
  - Solaris supports two page sizes – 8 KB and 4 MB

قديه الجزء الضايع من ال Fixed Size اللي كطيتوا ياه

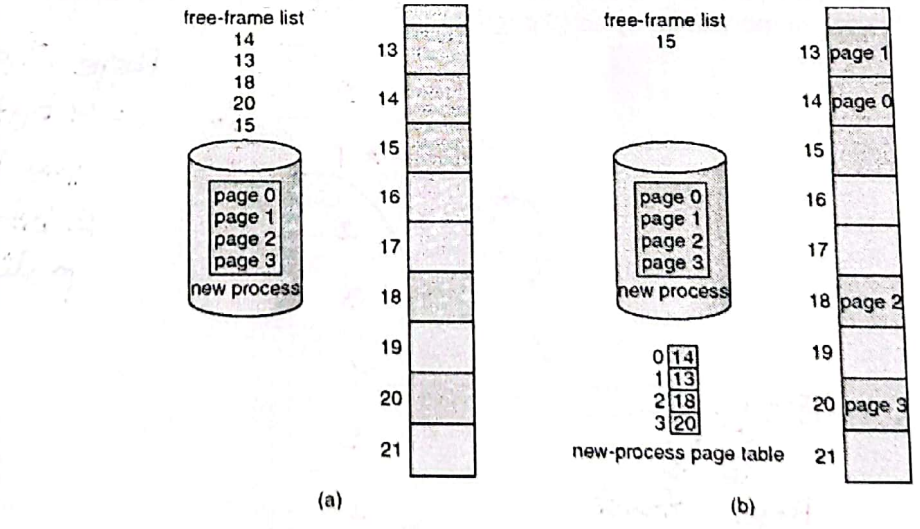
لو الProcess صغير بستخدموا ال Frame size الصغير ولو كبير بستخدم الكبير.

مش شرط، لو بدي أصغر ال Frame Size معناها عدد ال Pages اللي كم بستخدمهم لعمل Process واحد حيكون أكتر فعدد ال Page table بال Entry حيكون أكتر فرح آخذ وقت أكتر

31

ال OS لازم يكون حافظ حتى في Free Frames.

Free Frames



Before allocation

After allocation

32

Implementation of Page Table

- Page table is kept in main memory
  - **Page-table base register (PTBR)** points to the page table
  - **Page-table length register (PTLR)** indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called **translation look-aside buffers (TLBs)** (also called **associative memory**).

وين بداية الـ Page table تبعي.

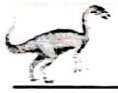
وينهاية الـ Page table تبعي

لهيك، انو كل Process الها Page table منفصل.

2 memory access بحتاج overhead فوقا

حلّه بالـ buffers

33



Translation Look-Aside Buffer

- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process
  - Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
  - Replacement policies must be considered
  - Some entries can be **wired down** for permanent fast access

لما تصير المعلومات تاعت الـ TLB's بتتغير بتتعمل invalid

فبالعادة بتحتفظ خانة

حالة تنقل بالـ Cash ما تنقل

عادة بتتعمل kernal بالـ

بتحتفظ المعلومات لـ Process معينة address-space identifier يعني

(totally supported by hardware)

34

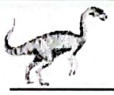
Hardware

- Associative memory – parallel search

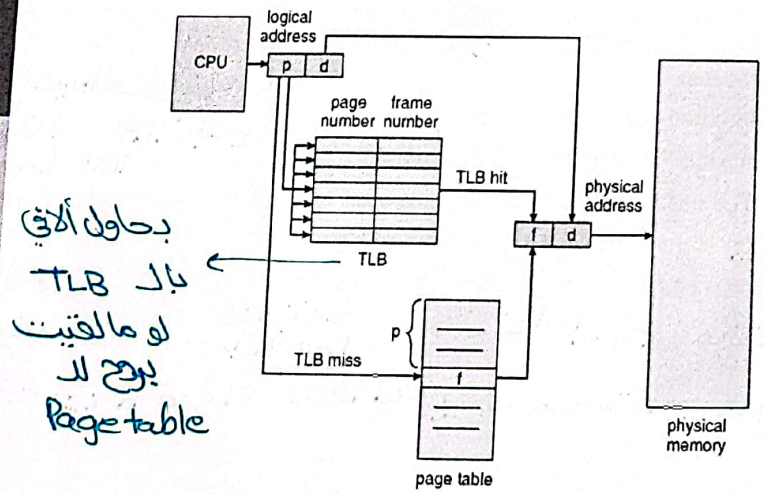
Page #	Frame #

- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

35



Paging Hardware With TLB



بحاول ألاقي TLB بال لو ما لقيت بروح لـ Page table

36

نسبة المرات اللي لقيت فيها الـ TLB حاجة اللي بدي إياه

Effective Access Time

- Hit ratio – percentage of times that a page number is found in the TLB
- An 80% hit ratio means that we find the desired page number in the TLB 80% of the time.
- Suppose that 10 nanoseconds to access memory.
  - If we find the desired page in TLB then a mapped-memory access take 10 ns
  - Otherwise we need two memory access so it is 20 ns
- **Effective Access Time (EAT)**

  EAT = 0.80 x 10 + 0.20 x 20 = 12 nanoseconds

  implying 20% slowdown in access time
- Consider amore realistic hit ratio of 99%,

  EAT = 0.99 x 10 + 0.01 x 20 = 10.1ns

  implying only 1% slowdown in access time.

37

Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on
- **Valid-invalid** bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space
  - Or use **page-table length register (PTLR)**
- Any violations result in a trap to the kernel

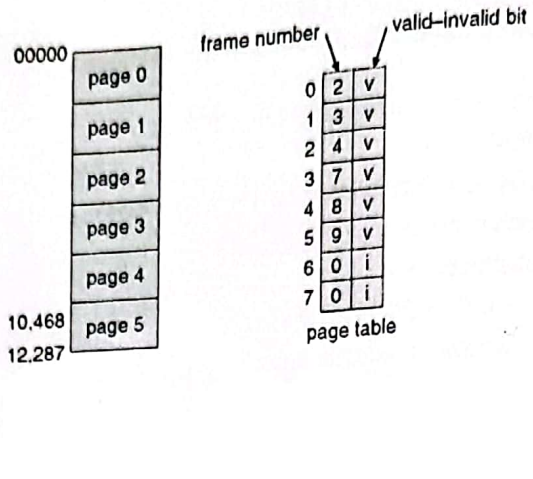
إنه يكون لكل Page الـ Protection خاص + المسؤول عن ذلك الـ hardware

لإنه خلص بيوف نهاية الـ Page table تلقائي فما بحتاج هدول الـ bits.

38

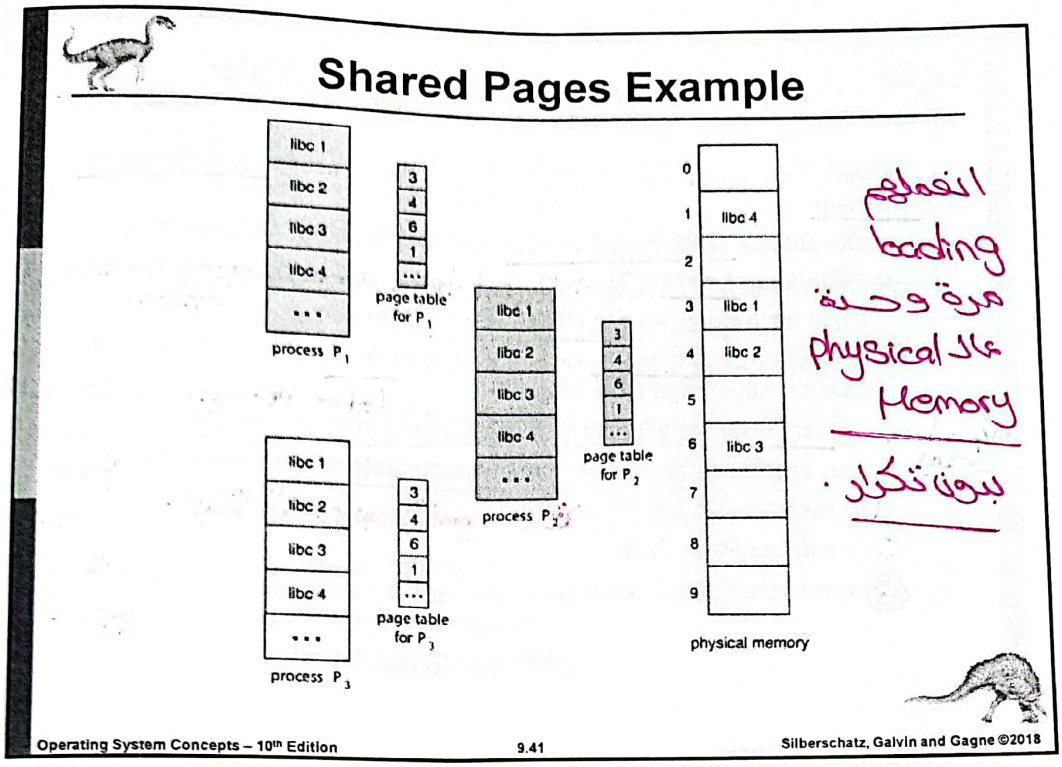
عشان أعرف إذا بقدر أعمل access لكل الـ Page وهي منهم الـ Entry ما بقدر.

Valid (v) or Invalid (i) Bit In A Page Table

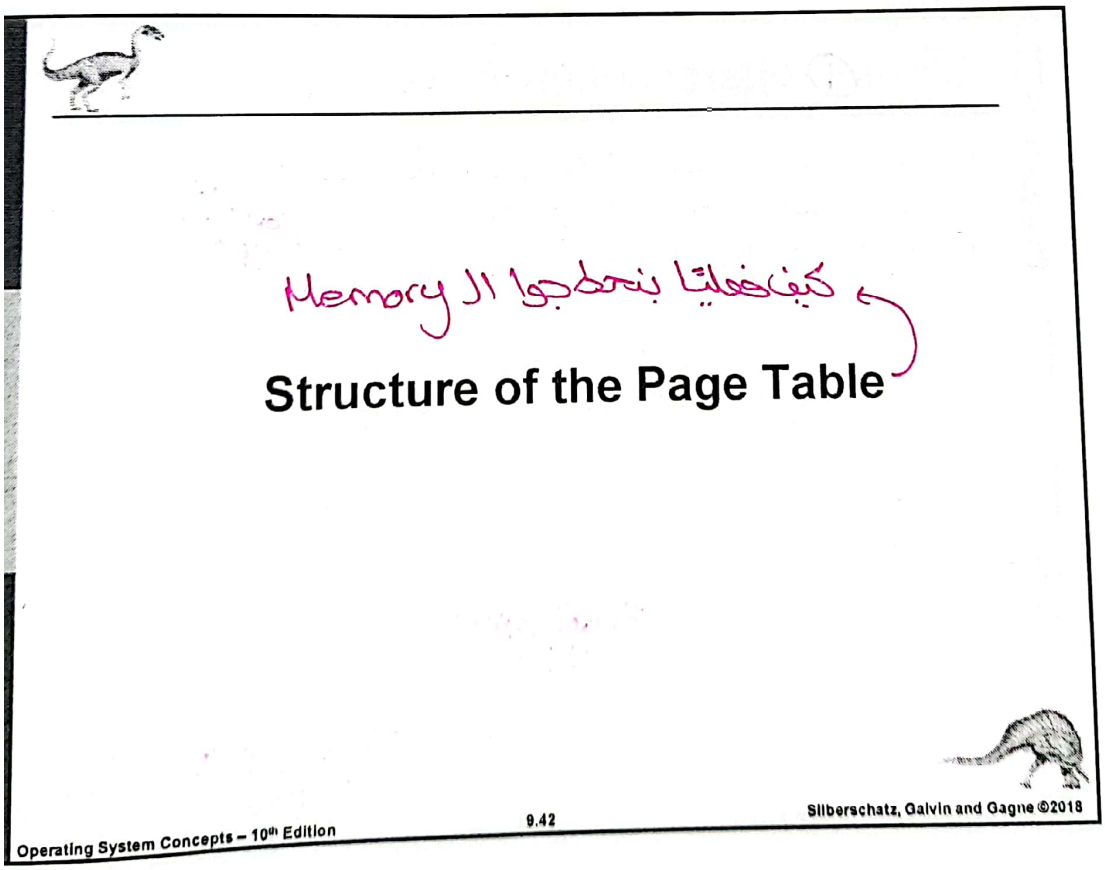


Shared Pages

- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space



41



2

Structure of the Page Table

- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12}) → 12 bit from address goes to the offset.
  - Page table would have 1 million entries ($2^{32} / 2^{12}$)
  - If each entry is 4 bytes → each process 4 MB of physical address space for the page table alone
    - Per Process
  - Don't want to allocate that contiguously in main memory
  - One simple solution is to divide the page table into smaller units
    1. Hierarchical Paging
    2. Hashed Page Tables
    3. Inverted Page Tables

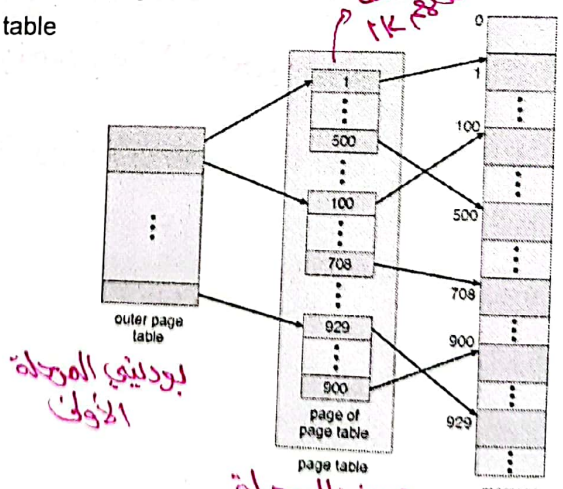
ما أحله بمكان واحد

ما بتقلل الحجم بس بتقسمه لمجلدات

20 بتا هم تاكون Page table وهذا اشي كتير كبير

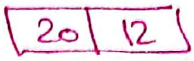
1 Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table



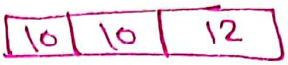
Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits



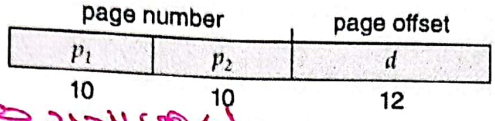
بدون Two-level

- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number
  - a 10-bit page offset



مع Two-level

- Thus, a logical address is as follows:

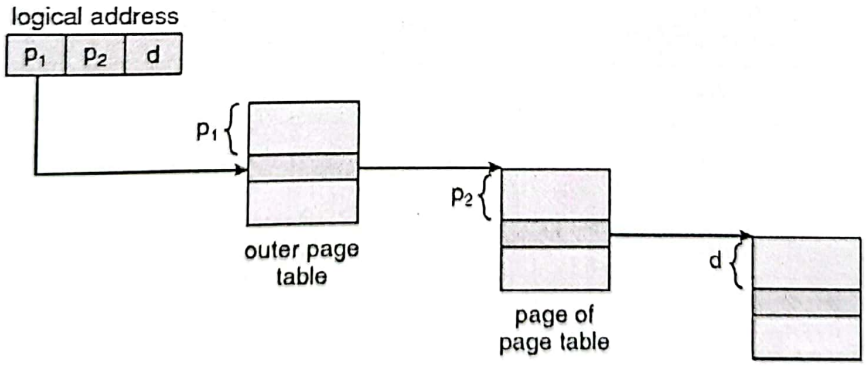


← هي بالعدد جمعا أول.

- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

45

Address-Translation Scheme



46

64-bit Logical Address Space

- Even two-level paging scheme not sufficient
- If page size is 4 KB (2^{12})
  - Then page table has 2^{52} entries
  - If two level scheme, inner page tables could be 2^{10} 4-byte entries
  - Address would look like

outer page	inner page	offset
p_1	p_2	d
42	10	12

كبير كبير لسا

  - Outer page table has 2^{42} entries or 2^{44} bytes
  - One solution is to add a 2^{nd} outer page table
  - But in the following example the 2^{nd} outer page table is still 2^{34} bytes in size
    - And possibly 4 memory access to get to one physical memory location

Three-level Paging Scheme

outer page	inner page	offset
p_1	p_2	d
42	10	12

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

حيصير كتير!

Logical address Memory accesses

· Physical addressed

بنساعدني أقل الوقت اللازم لعملية ال Search

2 Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
- Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match → linked list
  - If a match is found, the corresponding physical frame is extracted
- Variation for 64-bit addresses is **clustered page tables** → خاصة لو كانت الـ Memory كبيرة
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for sparse address spaces (where memory references are non-contiguous and scattered)

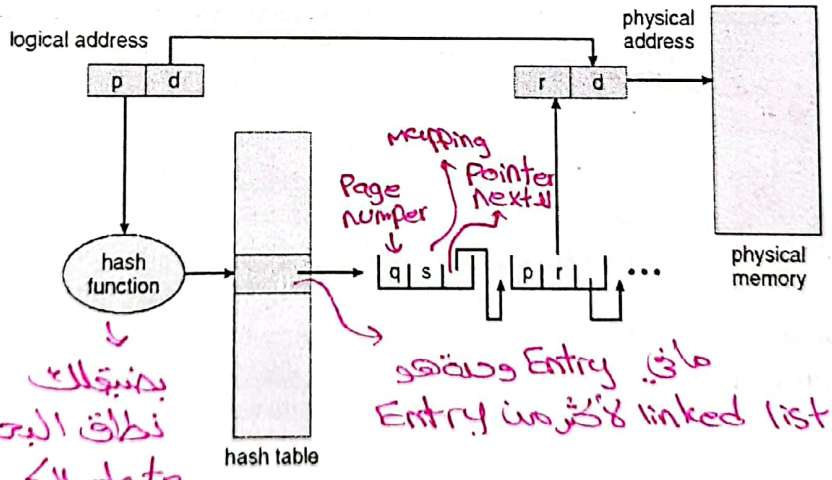
متفرقة وقليلة ومتعبية الـ Memory

كل Entry في أكثر من Page Frame أكثر من mapped

49

الهدف ما أكون استخدم كل Page number كل Entry ممكن يكون فيه أكثر من Page بطريقة معينة

Hashed Page Table → عادي مش Clustered



بنضيق نطاق البحث الـ data الكبيرة

ماني وصفهو Entry Entry أكثر من linked list

50

الي أخذناه قبل افترضنا إنه لكل Process الها Page table خاص فيها
بس هون بيعطينا إمكانية إنه يكون Page table واحد لكل الProcesses
ويتم التفريق عن طريق id
ما بكون في Mem زيادة محجوزة بدون ما تكون مستخدمة

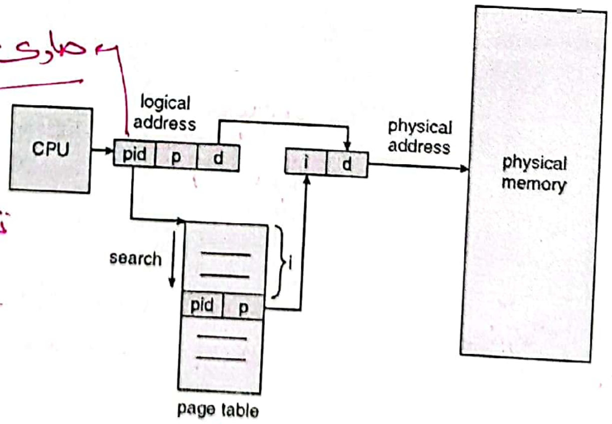
③ Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

ما بنضطر نعمل Context Switching أو Flashing لأنه صار عندي معروف id لكل Process

بهاي الحالة صعب يكون عنا Shared Mem زي قبل بس بدي لكل Virtual بده يوصل للPhysical الShared بده يكون اله Mapping منفصل.

Inverted Page Table Architecture



صار عندي id
ال offset تاعه هو ال Frame number

مخزن فيش لل Process وحدة، لكل ال Processes الي موجودين عندي

Scanned with CamScanner

Oracle SPARC Solaris

- Consider modern, 64-bit operating system example with tightly integrated HW
  - Goals are efficiency, low overhead
- Based on hashing, but more complex
- Two hash tables → Framed Page كلوحة فيها بتحطلي
  - One kernel and one for all user processes
  - Each maps memory addresses from virtual to physical memory
  - Each entry represents a contiguous area of mapped virtual memory,
    - More efficient than having a separate hash-table entry for each page
  - Each entry has base address and span (indicating the number of pages the entry represents)

Oracle SPARC Solaris (Cont.)

- TLB holds translation table entries (TTEs) for fast hardware lookups
  - A cache of TTEs reside in a translation storage buffer (TSB)
    - Includes an entry per recently accessed page
- Virtual address reference causes TLB search → Second level كانه
  - If miss, hardware walks the in-memory TSB looking for the TTE corresponding to the address → hardware بالمعنى مشهود → buffer بخزن نطاق البحث
    - If match found, the CPU copies the TSB entry into the TLB and translation completes → cash
    - If no match found, kernel interrupted to search the hash table → العملية صارت أعقد
      - The kernel then creates a TTE from the appropriate hash table and stores it in the TSB, Interrupt handler returns control to the MMU, which completes the address translation.

Swapping

Swapping

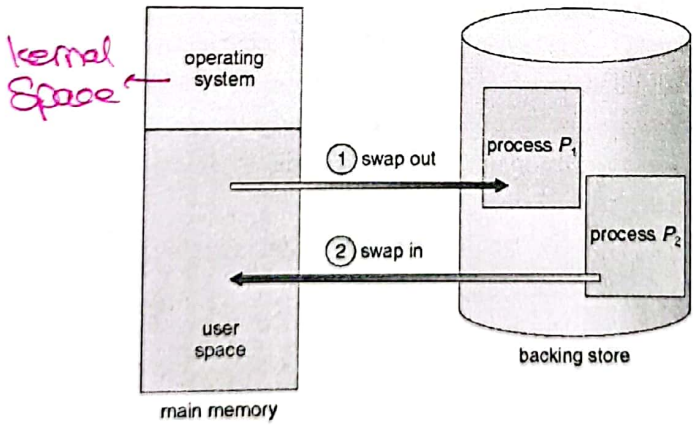
- A process can be **swapped** temporarily out of memory to a backing store, and then brought **back** into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store –** fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images → Backing store الي موجود فيها ال Mem ال حجز*
- **Roll out, roll in –** swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed (lower ينقل ال Backing store ونحط مكانه ال higher)
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses? No
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

57

Schematic View of Swapping



58

Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms → 100/50
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 seconds)
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used
  - System calls to inform OS of memory use via `request_memory()` and `release_memory()`

Context Switch Time and Swapping (Cont.)

- Other constraints as well on swapping
  - Pending I/O – can't swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

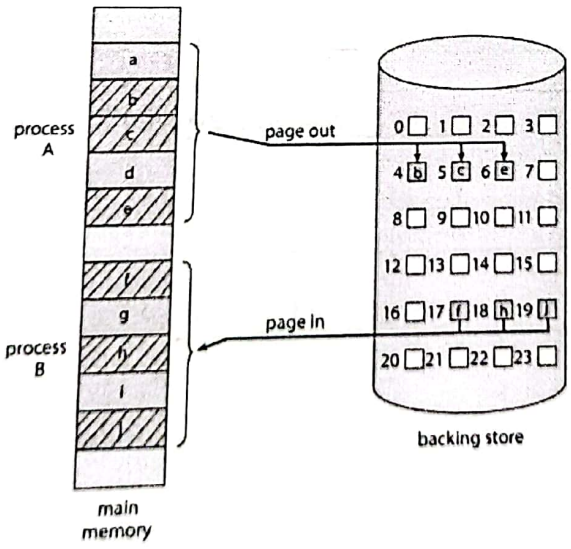
Swapping on Mobile Systems

- Not typically supported
  - Flash memory based → مشتركة بال harddisk
    - Small amount of space
    - Limited number of write cycles → مش قد هابك بتفل
    - Poor throughput between flash memory and CPU on mobile platform
- Instead use other methods to free memory if low
  - iOS ***asks*** apps to voluntarily relinquish allocated memory
    - Read-only data thrown out and reloaded from flash if needed → code أو data ما بتتغير جوا الله
    - Failure to free can result in termination
  - Android terminates apps if low free memory, but first writes **application state** to flash for fast restart
  - Both OSes support paging as discussed below

→ بخليوها عال Flash وما بعملوا load إلا لما احتاجوا.

61

Swapping with Paging



62

الـ OS هي المسؤولة عن عملية الـ Paging و الـ Swapping و الـ loading

Example: The Intel 32 and 64-bit Architectures

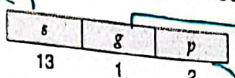
- Dominant industry chips
- Pentium CPUs are 32-bit and called IA-32 architecture
- Current Intel CPUs are 64-bit and called IA-64 architecture
- Many variations in the chips, cover the main ideas here

Example: The Intel IA-32 Architecture

- Supports both segmentation and segmentation with paging
  - Each segment can be 4 GB
  - Up to 16 K segments per process → Segment register = SR + offset
  - Divided into two partitions
    - First partition of up to 8 K segments are private to process (kept in **local descriptor table (LDT)**)
    - Second partition of up to 8K segments shared among all processes (kept in **global descriptor table (GDT)**)

Example: The Intel IA-32 Architecture (Cont.)

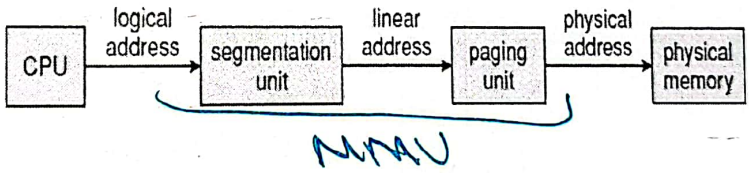
- CPU generates logical address
  - Selector given to segmentation unit
    - Which produces linear addresses



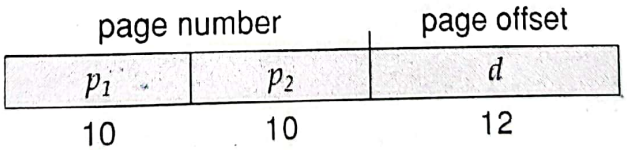
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
    - Pages sizes can be 4 KB or 4 MB

65

Logical to Physical Address Translation in IA-32

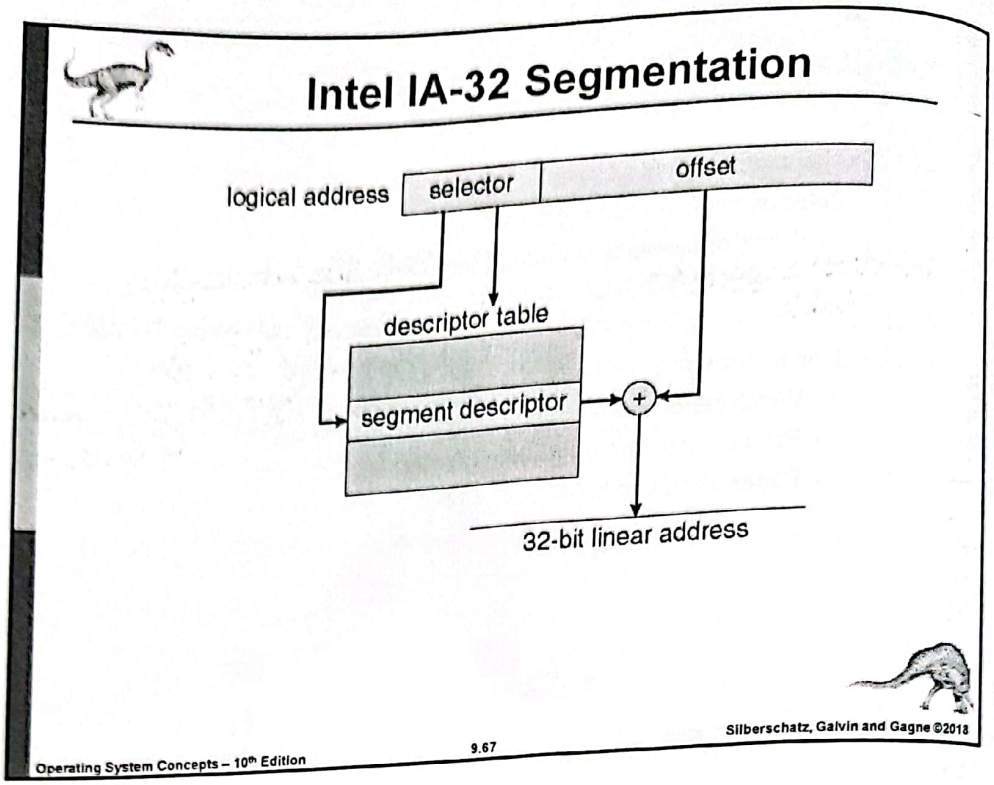


MMU

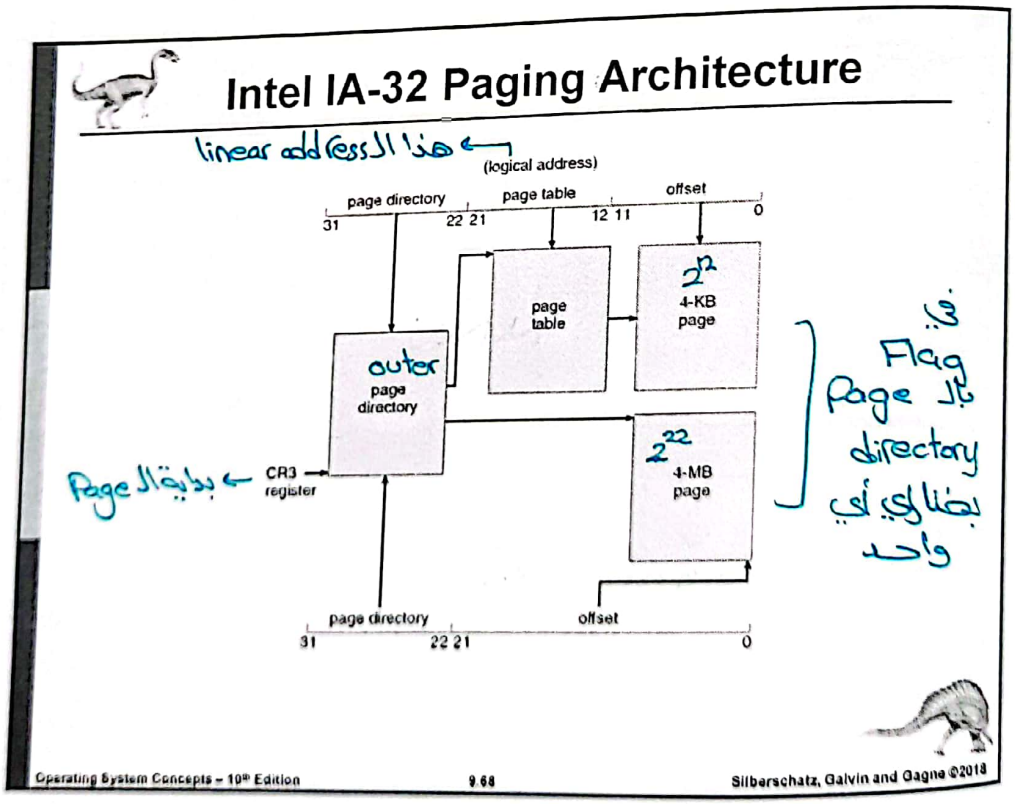


outer inner

66



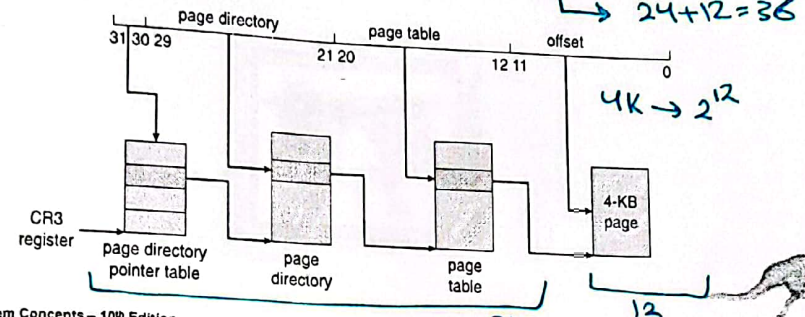
67



68

Intel IA-32 Page Address Extensions

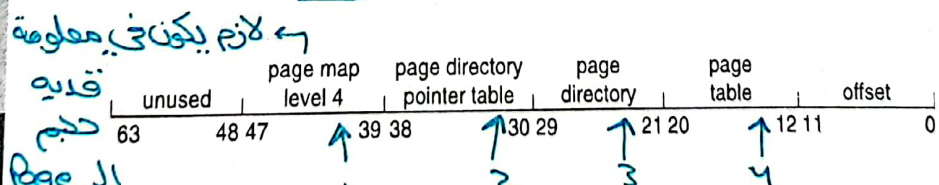
- 32-bit address limits led Intel to create **page address extension (PAE)**, allowing 32-bit apps access to more than 4GB of memory space
  - Paging went to a 3-level scheme
  - Top two bits refer to a **page directory pointer table**
  - Page-directory and page-table entries moved to 64-bits in size
  - Net effect is increasing address space to 36 bits – 64GB of physical memory



69

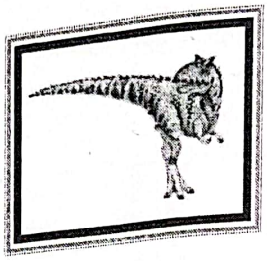
Intel x86-64

- Current generation Intel x86 architecture
- 64 bits is ginormous (> 16 exabytes)
- In practice only implement 48 bit addressing
  - Page sizes of 4 KB, 2 MB, 1 GB
  - Four levels of paging hierarchy
- Can also use PAE so virtual addresses are 48 bits and physical addresses are 52 bits

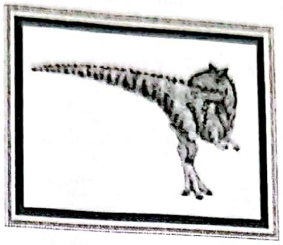


70

End of Chapter 9



Chapter 10: Virtual Memory



1

Chapter 10: Virtual Memory

- Background (10.1)
- Demand Paging (10.2)
- Copy-on-Write (10.3)
- Page Replacement (10.4)
- Allocation of Frames (10.5)
- Thrashing (10.6)

2

Objectives

- Define virtual memory and describe its benefits.
- Illustrate how pages are loaded into memory using demand paging.
- Apply the FIFO, optimal, and LRU page-replacement algorithms.
- Describe the working set of a process, and explain how it is related to program locality.

3

Background

4

Background

- Code needs to be in memory to execute, but entire program rarely used
  - Error code, unusual routines, large data structures
- Entire program code not needed at same time
- Consider ability to execute partially-loaded program
  - Program no longer constrained by limits of physical memory
  - Each program takes less memory while running -> more programs run at the same time
    - Increased CPU utilization and throughput with no increase in response time or turnaround time
  - Less I/O needed to load or swap programs into memory -> each user program runs faster

5

* أحيانا مش كل أجزاء الـ Program بتشغل خاصة لو البرنامج كبير كبير مرات بكون في :

Virtual memory

- Virtual memory – separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes

6

Processor الـ Mem اللي بتتعامل معها

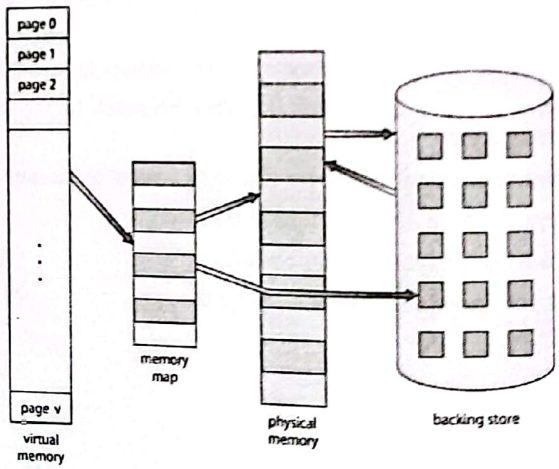
actual address space اللي بتشغل فعلا in the Memory

كيف الـ Process بتشوف الـ Memory تبعتها

Virtual memory (Cont.)

- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
- Virtual memory can be implemented via:
  1. Demand paging
  2. Demand segmentation

Virtual Memory That is Larger Than Physical Memory



بيشوفها الـ CPU كأنها contiguous

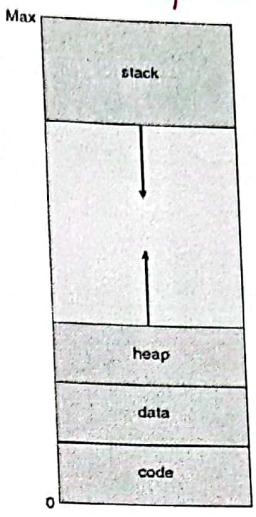
بيشوفهم كأنهم كلهم موجودين، بس بالخلف هنا مش كلهم بالـ Mem ممكن جزء بالـ backing store.

لازم يسمح لـ Stack و heap يكبروا لأي مايوهم طالما في واسع كتير.

أنا كـ Process هيك شايفة.

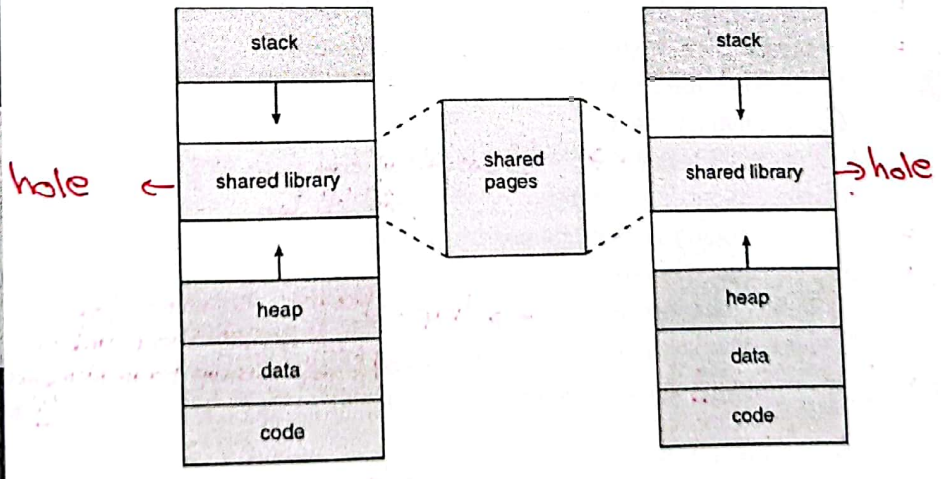
Virtual-address Space

- Usually design logical address space for stack to start at Max logical address and grow "down" while heap grows "up"
  - Maximizes address space use
  - Unused address space between the two is hole
    - No physical memory needed until heap or stack grows to a given new page
- Enables **sparse** address spaces with holes left for growth, dynamically linked libraries, etc.
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during `fork()`, speeding process creation



9

Shared Library Using Virtual Memory



10

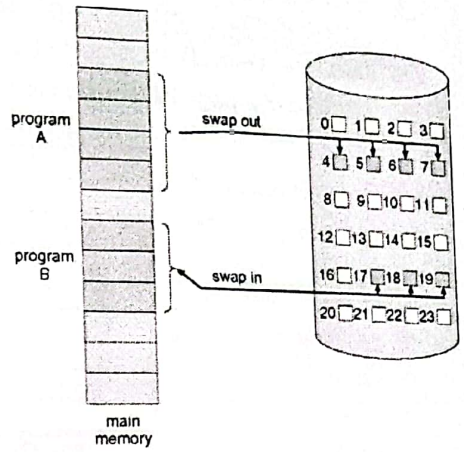
Demand Paging

Demand Paging

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Similar to paging system with swapping (diagram on right)
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a pager

Demand Paging

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Similar to paging system with swapping (diagram on right)



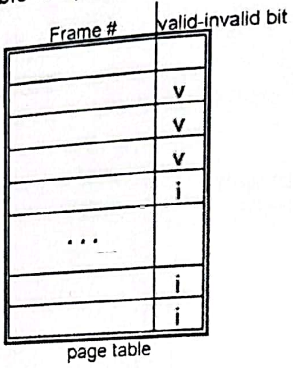
Basic Concepts

- With swapping, pager guesses which pages will be used before swapping out again
- Instead, pager brings in only those pages into memory
- How to determine that set of pages?
  - Need new MMU functionality to implement demand paging
- If pages needed are already **memory resident** → بحتاج Pages بالأساس لا Mem ما بحتاج أعمل loading لاشي
  - No difference from non demand-paging
- If page needed and not memory resident
  - Need to detect and load the page into memory from storage
    - Without changing program behavior
    - Without programmer needing to change code

لازم تكون كنبي لأقرأ عل اللي حكيناه ورا.

Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (v ⇒ in-memory – **memory resident**, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



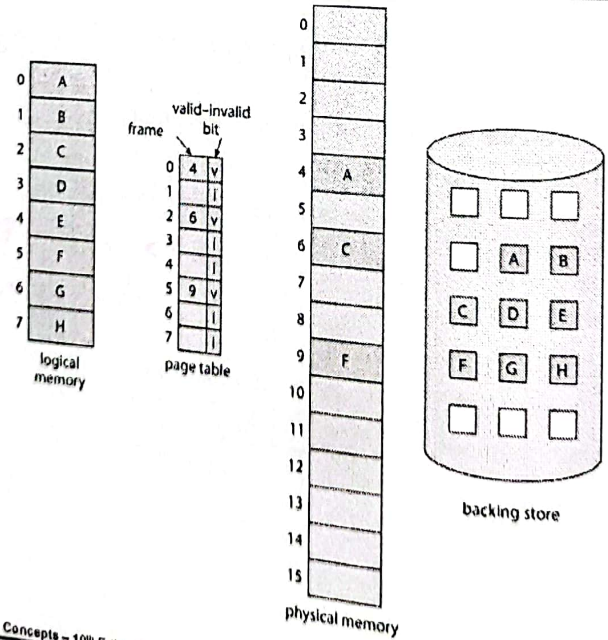
Valid page بس مش موجودة بالـMem

أو هاي مش Valid Page

- During MMU address translation, if valid–invalid bit in page table entry is **i** ⇒ page fault

يعني الـ Page هاي مش موجودة بالـ Mem بدك تعملها loading من الـ Secondary Storage

Page Table When Some Pages Are Not in Main Memory

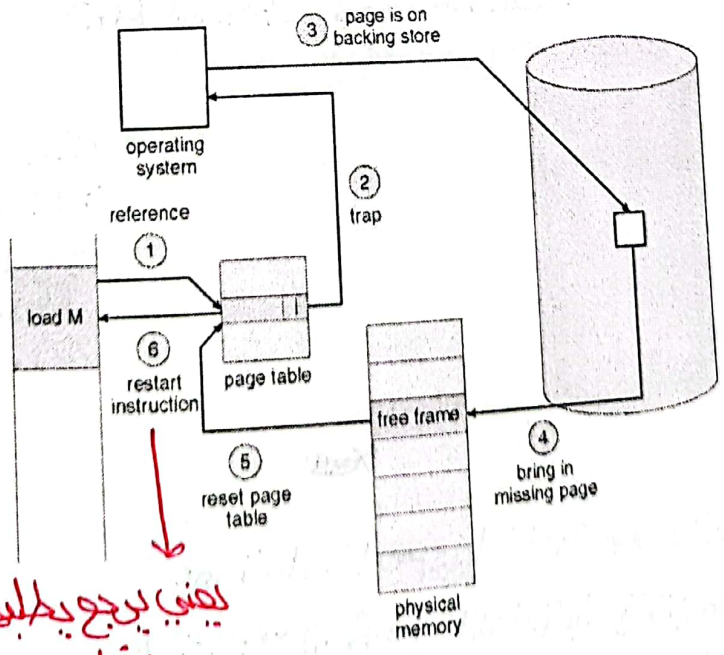


Steps in Handling Page Fault

1. If there is a reference to a page, first reference to that page will trap to operating system
   - Page fault
2. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
3. Find free frame
4. Swap page into frame via scheduled disk operation
5. Reset tables to indicate page now in memory
   Set validation bit = v
6. Restart the instruction that caused the page fault

17

Steps in Handling a Page Fault (Cont.)



18

Aspects of Demand Paging

- Extreme case – start process with *no* pages in memory
  - OS sets instruction pointer to first instruction of process, non-memory-resident -> page fault
  - And for every other process pages on first access
  - **Pure demand paging**
- Actually, a given instruction could access multiple pages -> multiple page faults
  - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
  - Pain decreased because of locality of reference
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory (swap device with **swap space**)
  - Instruction restart

Min = 2 page fault
Max = 4 page fault

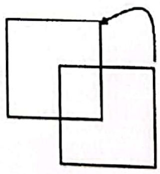
لو بنطلب inst حنفذ ال inst البرنامج وانا بنطلب data حنكون بنطلب من نفس ال Page.

Processor ال Supporting نفسه

Instruction Restart

can be load a block of Memory ال instruction ال with another block of Mem and store to a block of Memory

- Consider an instruction that could access several different locations
  - Block move



  - Auto increment/decrement location
  - Restart the whole operation?
    - What if source and destination overlap?

معناها وانا عم بنفذ قاعدة جمل عال Source

* ببقى بيصير Page fault للكل بالبداية قبل ما اكتب

اي اشي عال Source بعد هيك ما بيصير Page fault

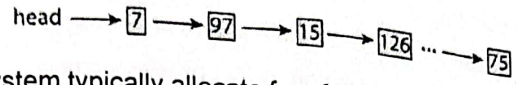
لاني عملت للكل من البداية.

أو

have registers to store all copy on data (locations)

Free-Frame List

- When a page fault occurs, the operating system must bring the desired page from secondary storage into main memory.
- Most operating systems maintain a **free-frame list** -- a pool of free frames for satisfying such requests.



- Operating system typically allocate free frames using a technique known as **zero-fill-on-demand** -- the content of the frames zeroed-out before being allocated.
- When a system starts up, all available memory is placed on the free-frame list.

Stages in Demand Paging – Worse Case

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
   a) Wait in a queue for this device until the read request is serviced
   b) Wait for the device seek and/or latency time
   c) Begin the transfer of the page to a free frame

Stages in Demand Paging (Cont.)

6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

Performance of Demand Paging

- Three major activities
  - Service the interrupt – careful coding means just several hundred instructions needed
  - Read the page – lots of time
  - Restart the process – again just a small amount of time
- Page Fault Rate $0 \le p \le 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

EAT = (1 – p) x memory access
+ p (page fault overhead
+ swap page out
+ swap page in)

Demand Paging Example

- Memory access time = 200 nanoseconds → 0.2 micro sec.
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (1 – p x 200 + p x 8,000,000
  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then EAT = 8.2 microseconds.
  This is a slowdown by a factor of 40!!
- If want performance degradation < 10 percent
  - 220 > 200 + 7,999,800 x p
    20 > 7,999,800 x p
  - p < .0000025
  - < one page fault in every 400,000 memory accesses

Demand Paging Optimizations

- Swap space I/O faster than file system I/O even if on the same device
  - Swap allocated in larger chunks, less management needed than file system
- Copy entire process image to swap space at process load time
  - Then page in and out of swap space
  - Used in older BSD Unix
- Demand page in from program binary on disk, but discard rather than paging out when freeing frame → dirty bit
  - Used in Solaris and current BSD
  - Still need to write to swap space
    - Pages not associated with a file (like stack and heap) – **anonymous memory**
    - Pages modified in memory but not yet written back to the file system
- Mobile systems
  - Typically don't support swapping
  - Instead, demand page from file system and reclaim read-only pages (such as code)

*Swapping → Main ↔ Secondary Storage

Copy-on-Write

كلما الـ Parent والـ child ماعليهم تعديل فبخليهم يتشاركوا نفس الـ Page memory لحد ما واحد منهم يعمل تعديل بصير كل واحد نسخة خاصة فيه

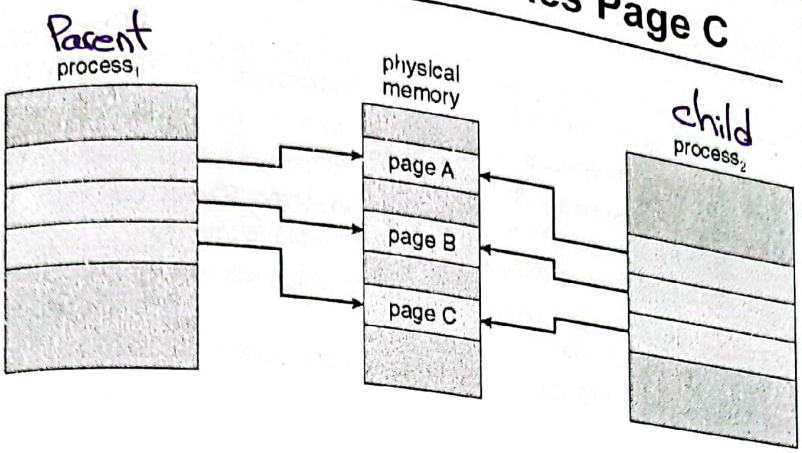
Copy-on-Write

- **Copy-on-Write** (COW) allows both parent and child processes to initially ***share*** the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- In general, free pages are allocated from a **pool** of **zero-fill-on-demand** pages
  - Pool should always have free frames for fast demand page execution
    - Don't want to have to free a frame as well as other processing on page fault
    - لو ماكان عندي حفظ ↳ Swapping اعمل
  - Why zero-out a page before allocating it?
- `vfork()` variation on `fork()` system call has parent suspend and child using copy-on-write address space of parent
  - Designed to have child call `exec()`
  - Very efficient

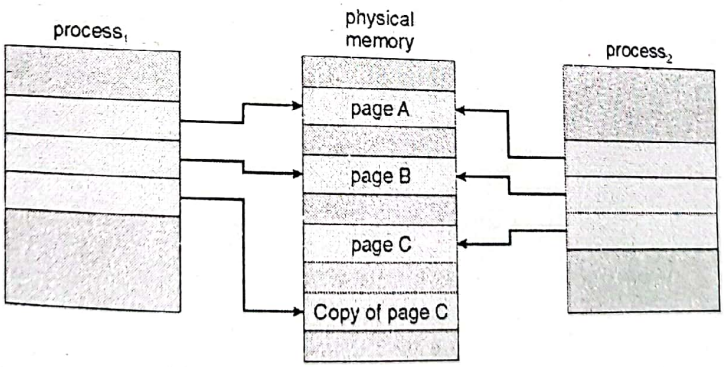
هاي اللي بتعمل Copy-on-write مش الـ fork() هنا أكثر efficient من الـ fork()

* بالتوقيع لما يصير في تعديل حيبطلوا sharing ويصير كل واحد إله نسخة خاصة فيه.

Before Process 1 Modifies Page C



After Process 1 Modifies Page C



هون كل ولد اخذه نسخته الخاصة
لإنه صار بده يعتل فما بدي الثاني يتأثر
بتعديل الأول.

* الـ Process لما تخلص تنفيذ بصير الـ Pages تاعتها Free.

What Happens if There is no Free Frame?

- Used up by process pages
- Also in demand from the kernel, I/O buffers, etc
- How much to allocate to each?
- Page replacement – find some page in memory, but not really in use, page it out
  - Algorithm – terminate? swap out? replace the page?
  - Performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

31

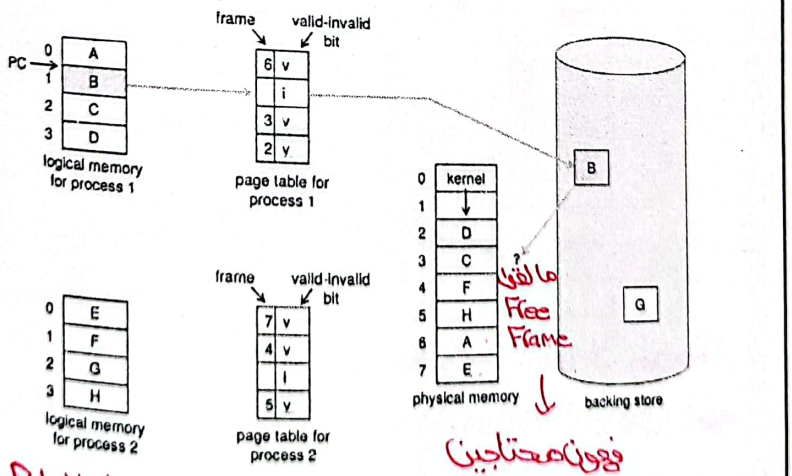
Page Replacement

32

Page Replacement

- Prevent **over-allocation** of memory by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Need For Page Replacement



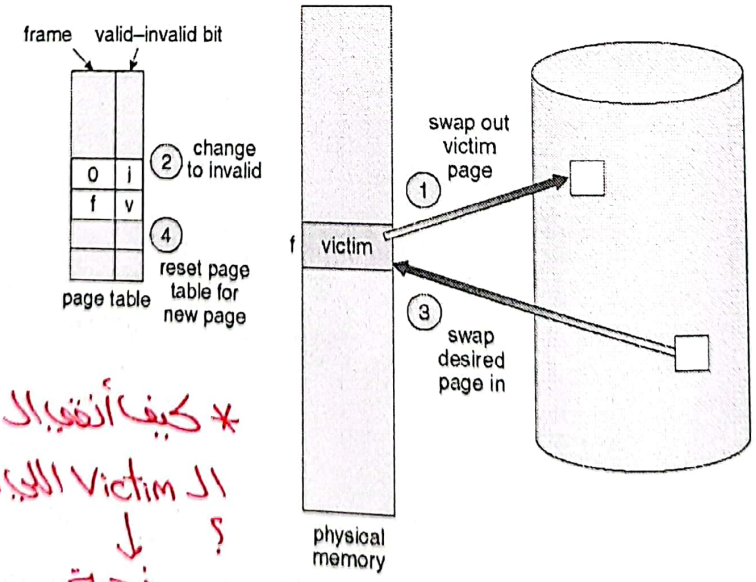
Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

لو كانت الـ dirty bit = 0 ما فيا حكي إني أكتب فحتاج page واحد بس.

Page Replacement



* كيف أختار الـ Page الـ Victim اللي بدي أطلعها ؟ خوارزمية

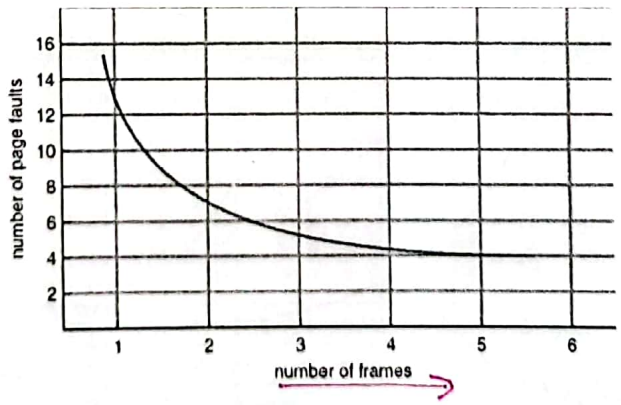
Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

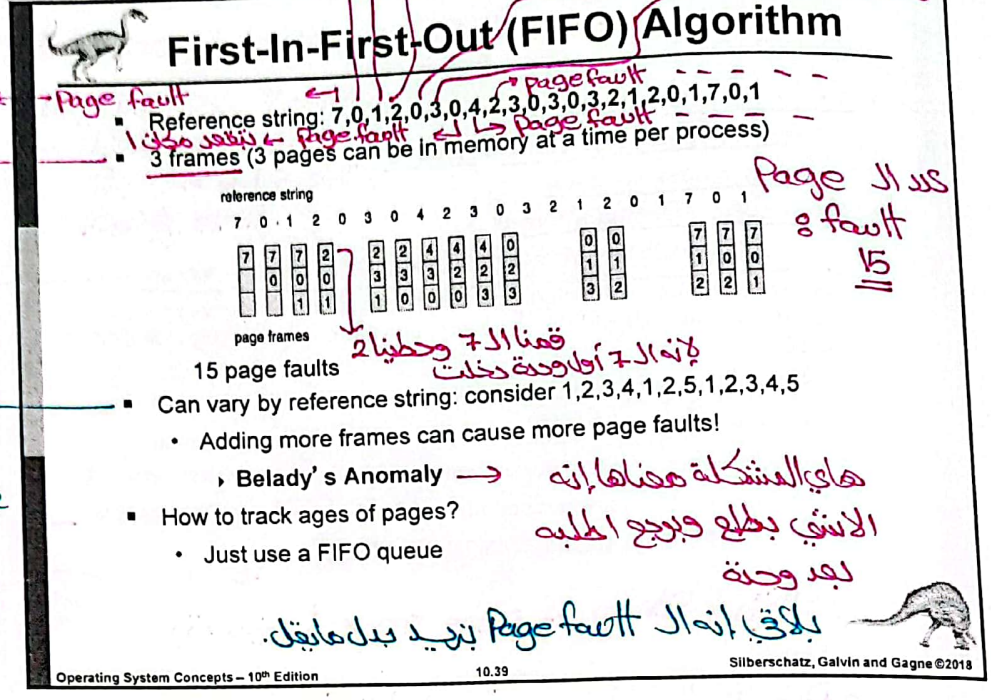
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

37

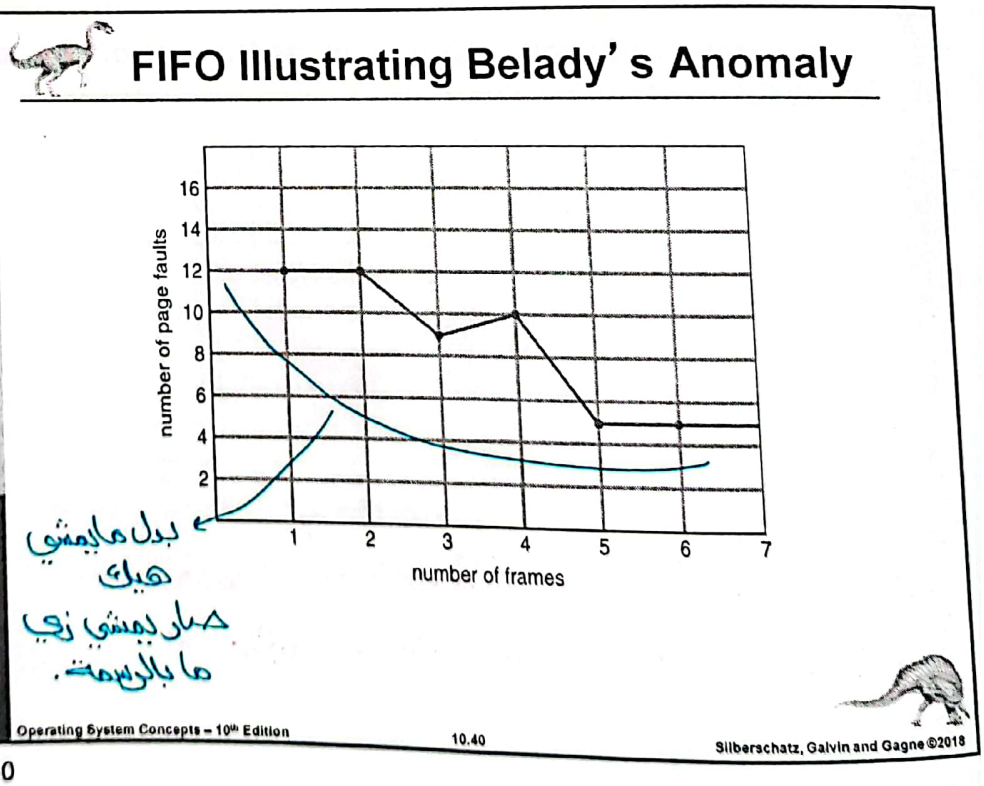
Graph of Page Faults Versus the Number of Frames



38



39



40

Algorithm لل Performance في مرجع بقارن فيها الـ
مع أحسن اشي ممكن.
ماني اشي بنقدر نعمله هيك لإنه
هاد بالمستقبل Future.

Optimal Algorithm

بيستخدموا بس للمقارنة

- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs

optimal page fault → أحسن
Page fault أقل ← اشي بقدر أحصله

reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	3	3	3	3	1	1	1

page frames

أبعد واحد حينطلب هو اللي
FIFO ببدلمكانه مش

41

Least Recently Used (LRU) Algorithm

عالأغلب اللي ما انطلبت حديثاً ماح تنطلب Soon

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page

بحتاج لمعلومة اضافية هنا
آخر مرة انطلبت هاي الـ process

ابعد وحدة انطلبت 2 بدلمكانها

reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	4	4	4	0	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	3	3	3	3	2	2	2	2	2	7	7	7
		1	1	1	3	3	2	2	2	2	2	2	3	3	3	3	3	0	0

ابعد وحدة انطلبت هي 1 فببدلمكانها Page fault هذا

page frames

ابعد وحدة انطلبت هي الـ 7 فببدلمكانها

وبستمر زي هيك

- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

بطريقتين بنستعمل
Counter - 1
Stack - 2

42