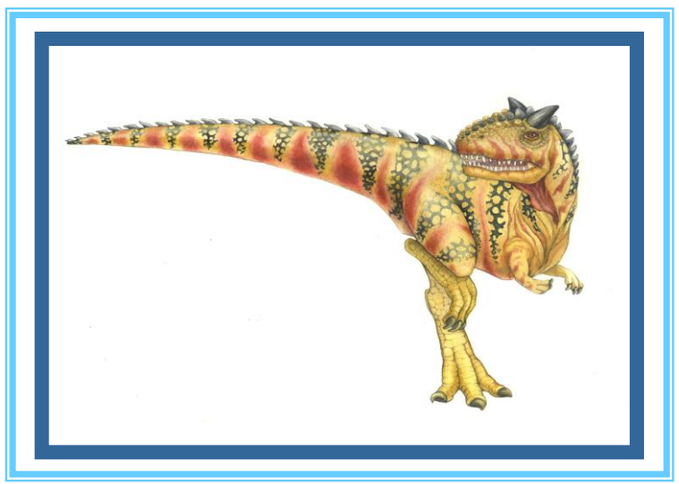
Chapter 13: I/O Systems

هذا التشابتر يشرح كيف يتعامل نظام التشغيل مع أجهزة الإدخال والإخراج (Input/Output Devices) مثل:

الكيبورد والماوس
الشاشة والطابعة
الهاردسك و SSD
كروت الشبكة
أجهزة USB وغيرها

الهدف الأساسي هو فهم:
كيف يربط نظام التشغيل البرامج بالهاردوير
كيف تتم عملية نقل البيانات بين الجهاز والذاكرة والمعالج
كيف يحسن النظام الأداء والكفاءة أثناء عمليات الـ I/O)



Silberschatz, Galvin and Gagne ©2009



Overview

هذا الجزء يعطي مقدمة عامة عن إدارة الإدخال والإخراج (I/O Management) داخل نظام التشغيل، ويشرح لماذا تعتبر من أهم أجزاء الـ OS.

- I/O management is a major component of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly
  - Various methods to control them
  - Performance management
  - New types of devices frequent

لماذا الـ I/O مهم جدًا؟
إدارة الـ I/O تعتبر جزء أساسي من تصميم وتشغيل نظام التشغيل لأن الكمبيوتر دائمًا يتعامل مع أجهزة خارجية مثل:
الهاردسك، الكيبورد، الشاشة، الشبكات.
بدون نظام إدارة I/O لن تستطيع البرامج التواصل مع هذه الأجهزة.

المشاكل والتحديات في الـ I/O
1) تنوع الأجهزة
أجهزة الـ I/O تختلف بشكل كبير:
بعضها للتخزين (Storage)
بعضها لنقل البيانات (Transmission)
بعضها لتفاعل المستخدم (Human Interface)

2) اختلاف طرق التحكم
كل جهاز قد يحتاج:
أوامر مختلفة
طريقة نقل بيانات مختلفة
لذلك نظام التشغيل يحتاج طبقة تنظم هذه الاختلافات.

3) إدارة الأداء
عمليات الـ I/O غالبًا أبطأ من الـ CPU.
لذلك النظام يحاول:
تقليل وقت الانتظار
تسريع نقل البيانات
تنظيم الطلبات

4) ظهور أجهزة جديدة باستمرار
كل فترة تظهر:
أجهزة جديدة
تقنيات جديدة
لذلك تصميم نظام الـ I/O يجب أن يكون مرنًا.

Ports, Buses, Controllers
هذه هي المكونات الأساسية التي تربط الأجهزة بالكمبيوتر.

- Ports, busses, device controllers connect to various devices

- Device drivers encapsulate device details
  - Present uniform device-access interface to I/O subsystem

ما هو الـ Device Driver؟
برنامج داخل نظام التشغيل مسؤول عن التواصل مع جهاز معين.
مثال:
Driver للطابعة
Driver لكرت الشاشة





I/O Hardware

I/O Hardware
هذا الجزء يشرح الهاردوير الخاص بالإدخال والإخراج.

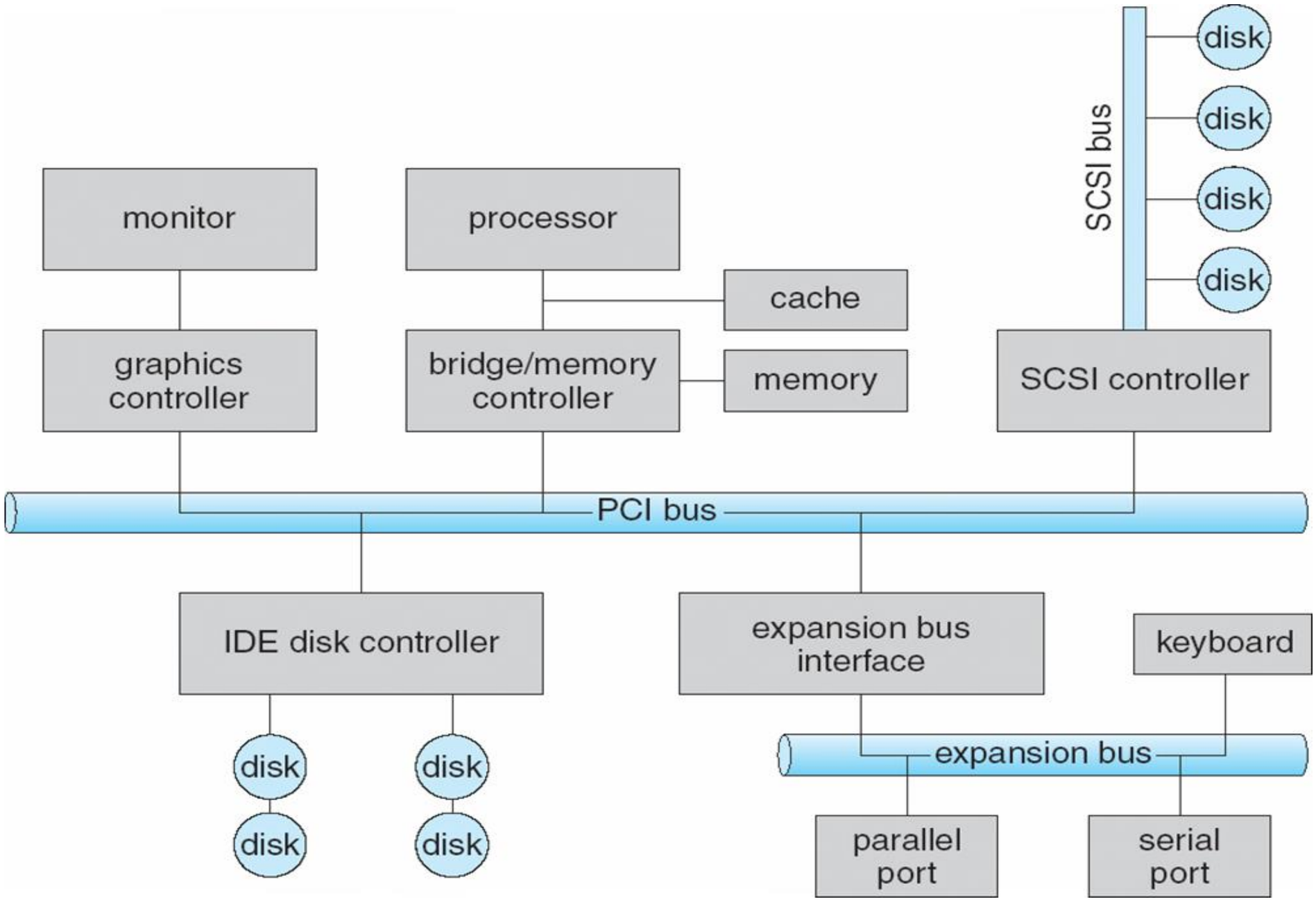
- Incredible variety of I/O devices
  - Storage
  - Transmission
  - Human-interface
- Common concepts – signals from I/O devices interface with computer
  - Port – connection point for device
  - Bus - daisy chain or shared direct access
  - Controller (host adapter) – electronics that operate port, bus, device
    - Sometimes integrated
    - Sometimes separate circuit board (host adapter)
    - Contains processor, microcode, private memory, bus controller, etc
      - Some talk to per-device controller with bus controller, microcode, memory, etc

أنواع أجهزة الـ I/O
1) Storage Devices
أجهزة التخزين:
HDD
SSD
USB Flash
2) Transmission Devices
أجهزة نقل البيانات:
Network Cards
Modems
3) Human-Interface Devices
أجهزة تفاعل المستخدم:
Keyboard
Mouse
Touchscreen

كيف تتواصل الأجهزة مع الكمبيوتر؟
كل الأجهزة ترسل Signals وإشارات للنظام عبر:
Ports
Buses
Controllers



A Typical PC Bus Structure (P.496)





I/O Hardware (Cont.)

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
  - Direct I/O instructions
  - **Memory-mapped I/O**
    - Device data and command registers mapped to processor address space
    - Especially for large address spaces (graphics)

I/O Instructions
المعالج يستخدم تعليمات خاصة للتحكم بالأجهزة.
هذه التعليمات:
ترسل أوامر
تقرأ البيانات
تتابع حالة الجهاز
Registers in I/O Devices
الأجهزة تحتوي **Registers** يتعامل معها الـ **Driver**.
أنواع الـ **Registers**
1) Data-in Register
يخزن البيانات القادمة من الجهاز للكمبيوتر.
2) Data-out Register
يخزن البيانات الخارجة من الكمبيوتر للجهاز.
3) Status Register
يوضح حالة الجهاز:
جاهز؟
مشغول؟
4) Control Register
يستقبل أوامر التحكم من الـ **Driver**.
FIFO Buffer
بعض الأجهزة تستخدم **Buffer** يعمل بنظام:
First In First Out
أي أول بيانات تدخل هي أول بيانات تخرج.

Device Addressing
لكل جهاز **Address** خاص يستخدمه النظام للوصول إليه.
طريقتان للتعامل مع الأجهزة
1) Direct I/O Instructions
المعالج يستخدم تعليمات خاصة مباشرة للتعامل مع الجهاز.
2) Memory-Mapped I/O
طريقة شائعة جدًا.
فكرة الـ **Memory-Mapped I/O**:
الخاصة بالجهاز تُعامل كأنها مواقع في الذاكرة **Registers**.
يعني المعالج يقرأ ويكتب عليها مثل الـ **RAM**.
لماذا **Memory-Mapped I/O** مهم؟
خصوصًا للأجهزة الكبيرة مثل:
كروت الشاشة
Graphics Systems
لأنها تحتاج مساحة **Address** كبيرة وسرعة عالية.





Device I/O Port Locations on PCs (partial)

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)





Polling

الـ Polling هو طريقة يستخدمها الـ CPU لمعرفة هل جهاز الـ I/O جاهز أم لا.
الفكرة بسيطة:
المعالج يظل يسأل الجهاز باستمرار:
"هل انتهيت؟ هل أصبحت جاهز؟"

- For each byte of I/O
  1. Read busy bit from status register until 0
  2. Host sets read or write bit and if write copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done

- Step 1 is **busy-wait** cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data overwritten / lost

خطوات الـ Polling
1)
الـ CPU يقرأ الـ busy bit من الـ status register.
إذا كانت:
1 → الجهاز مشغول
0 → الجهاز جاهز
فيستمر المعالج بالانتظار حتى تصبح 0.
2)
إذا كانت العملية كتابة (Write):
المعالج ينسخ البيانات إلى الـ data-out register
ثم يحدد:
Write أو Read
3)
الـ CPU يفعّل:
command-ready bit
أي يخبر الجهاز:
"ابدأ التنفيذ"
4)
الـ Controller:
يفعّل busy bit
ويبدأ عملية نقل البيانات
5)
بعد انتهاء النقل:
الـ Controller يمسح:
busy bit
error bit
command-ready bit
ويصبح الجهاز جاهز مرة أخرى.
مشكلة الـ Polling
الـ CPU يظل ينتظر فقط بدون تنفيذ شيء مفيد.
وهذا يسمى:
Busy Waiting
متى يكون Polling جيد؟
إذا كان الجهاز سريع جداً.
متى يكون سيئ؟
إذا كان الجهاز بطيئاً لأن:
الـ CPU سيضيع وقته بالانتظار.
لماذا لا ينفذ الـ CPU مهمة أخرى؟
لأن البيانات قد تضيع إذا لم يرجع بسرعة لفحص الجهاز.





Interrupts

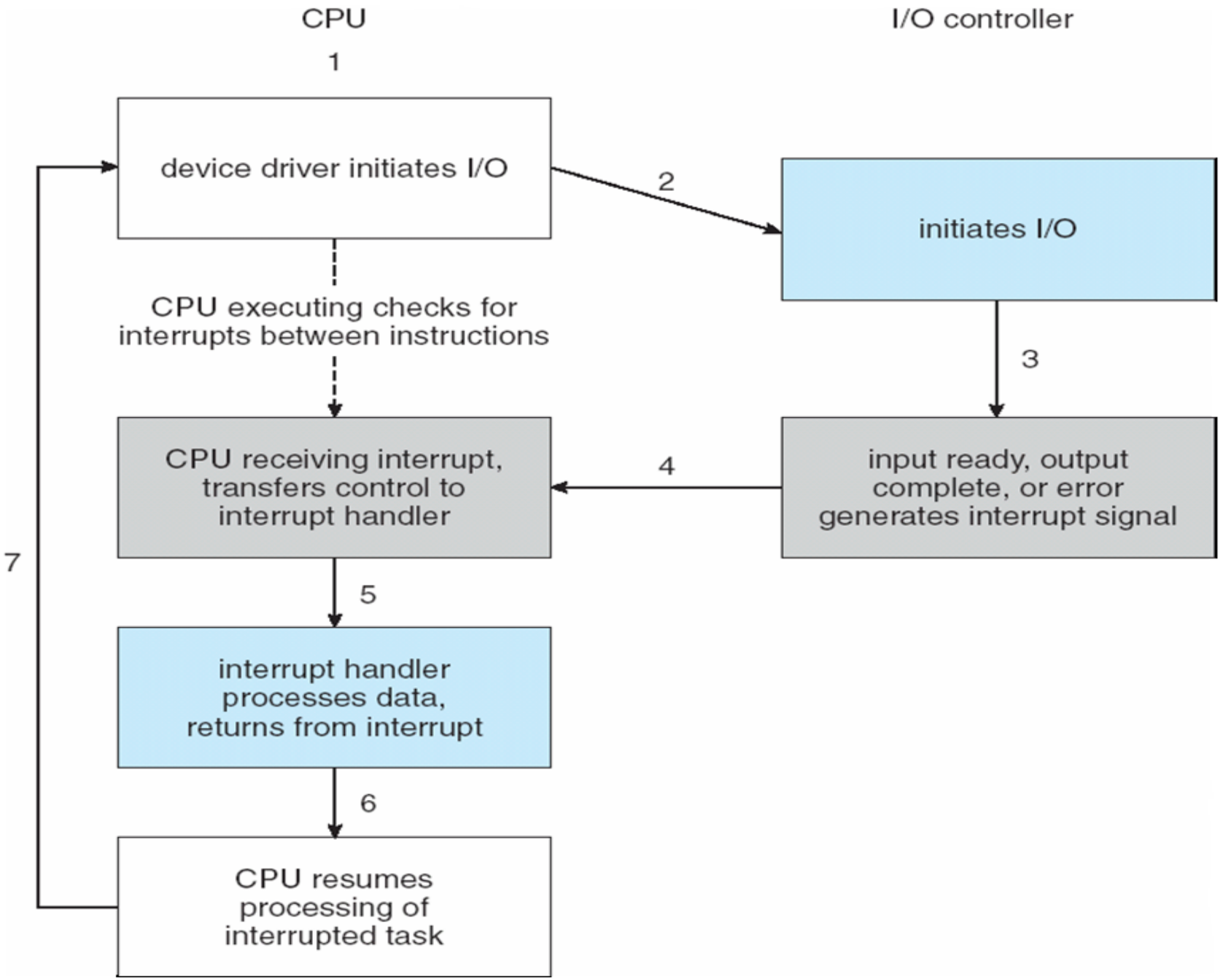
- Polling can happen in 3 instruction cycles
  - Read status, logical-and to extract status bit, branch if not zero
  - How to be more efficient if non-zero infrequently?
- CPU Interrupt-request line triggered by I/O device
  - Checked by processor after each instruction
- Interrupt handler receives interrupts
  - Maskable to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some nonmaskable
  - Interrupt chaining if more than one device at same interrupt number

الـ Interrupts
الـ Interrupts هي الحل لمشكلة Polling.
بدل أن يسأل الـ CPU باستمرار:
الجهاز نفسه يرسل تنبيه عند الانتهاء.
كيف يعمل الـ Interrupt؟
الجهاز يرسل:
Interrupt Request (IRQ)
إلى المعالج.
ماذا يفعل المعالج؟
بعد كل Instruction:
يفحص هل يوجد Interrupt أم لا.
إذا وجد:
يوقف المهمة الحالية مؤقتًا
ويشغل Interrupt Handler
Interrupt Handler
برنامج داخل الـ OS وظيفته:
معرفة أي جهاز أرسل الـ Interrupt
وتنفيذ الخدمة المطلوبة
Interrupt Vector
جدول يحتوي عناوين جميع الـ Handlers.
كل Interrupt له رقم معين،
والنظام يذهب للـ Handler المناسب.
Context Switch
عند بداية الـ Interrupt:
يتم حفظ حالة العملية الحالية
وعند الانتهاء:
يتم استرجاعها
وهذا يسمى: Context Switching
أنواع الـ Interrupts
Maskable Interrupts
يمكن للنظام:
تجاهلها
أو تأجيلها مؤقتًا
Nonmaskable Interrupts
Interrupts خطيرة لا يمكن تجاهلها.
مثل:
أعطال الهاردوير
Interrupt Chaining
إذا أكثر من جهاز يستخدم نفس رقم الـ Interrupt:
النظام يفحص الأجهزة واحدًا واحدًا لمعرفة المصدر الحقيقي.





Interrupt-Driven I/O Cycle







Intel Pentium Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts





Interrupts (Cont.)

- Interrupt mechanism also used for exceptions
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via trap to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

استخدامات الـ Interrupts
1) أجهزة الـ I/O
مثل:
الكيبورد
الشبكة
الهاردسك
2) Exceptions
مثل:
القسمة على صفر
أخطاء الهاردوير
3) Page Fault
عند محاولة الوصول لذاكرة غير موجودة حاليًا.
4) System Calls
البرنامج يطلب خدمة من الـ Kernel باستخدام:
Trap
وهو نوع خاص من الـ Interrupt.

لماذا الـ Interrupts أفضل؟
لأن الـ CPU:
لا يضيع وقته بالانتظار
يستطيع تنفيذ مهام أخرى





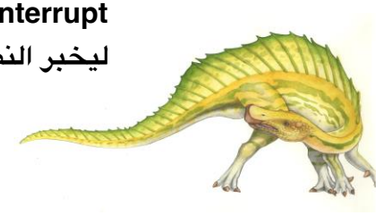
Direct Memory Access

الـ DMA يستخدم لنقل كميات كبيرة من البيانات بسرعة.

- Used to avoid programmed I/O (one byte at a time) for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
  - When done, interrupts to signal completion

المشكلة بدون DMA
في Programmed I/O:
الـ CPU ينقل البيانات Byte by Byte
وهذا بطيء جدًا.
فكرة الـ DMA
بدل أن ينقل الـ CPU البيانات:
الـ DMA Controller ينقل البيانات مباشرة بين:
جهاز الـ I/O
والذاكرة
بدون تدخل مستمر من الـ CPU.

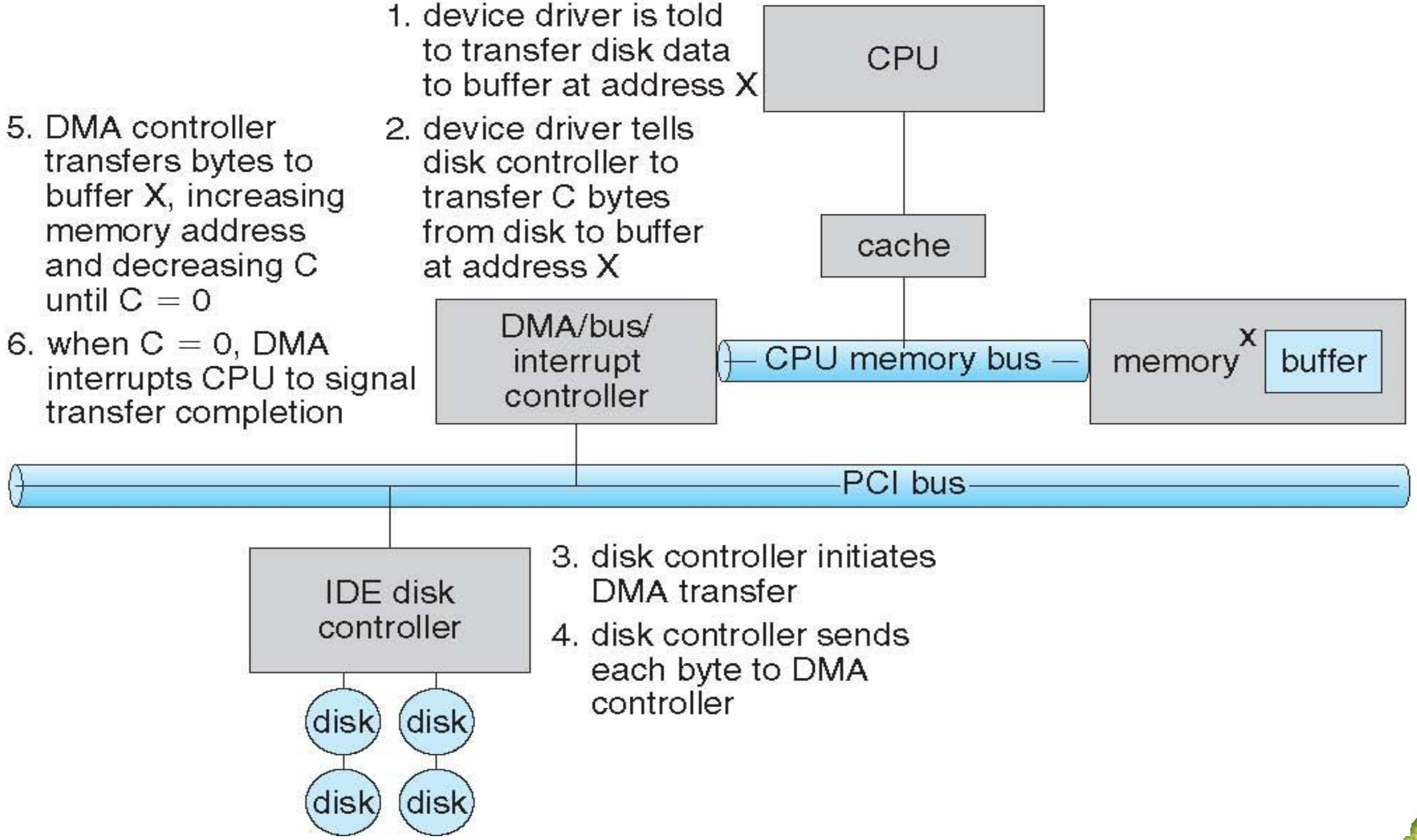
ماذا يفعل الـ OS؟
ينشئ:
DMA Command Block
ويحتوي على:
Source Address
Destination Address
Read/Write Mode
عدد الـ Bytes
ثم ماذا يحدث؟
الـ OS يعطي عنوان الـ Command Block إلى:
DMA Controller
Bus Mastering
الـ DMA Controller يأخذ التحكم بالـ Bus مؤقتًا من الـ CPU،
ثم يبدأ نقل البيانات.
بعد انتهاء النقل
الـ DMA Controller يرسل:
Interrupt
ليخبر النظام أن العملية انتهت.





Six Step Process to Perform DMA Transfer

1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion







Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

واجهة الإدخال والإخراج (Application I/O Interface)
هذا الجزء يشرح كيف يوفر نظام التشغيل طريقة موحدة للتعامل مع أجهزة الـ I/O المختلفة.
الفكرة الأساسية:
البرنامج لا يتعامل مباشرة مع الهاردوير، بل يستخدم أوامر جاهزة من نظام التشغيل مثل:
read()
write()
open()
close()
ونظام التشغيل يتولى التواصل مع الجهاز الحقيقي.

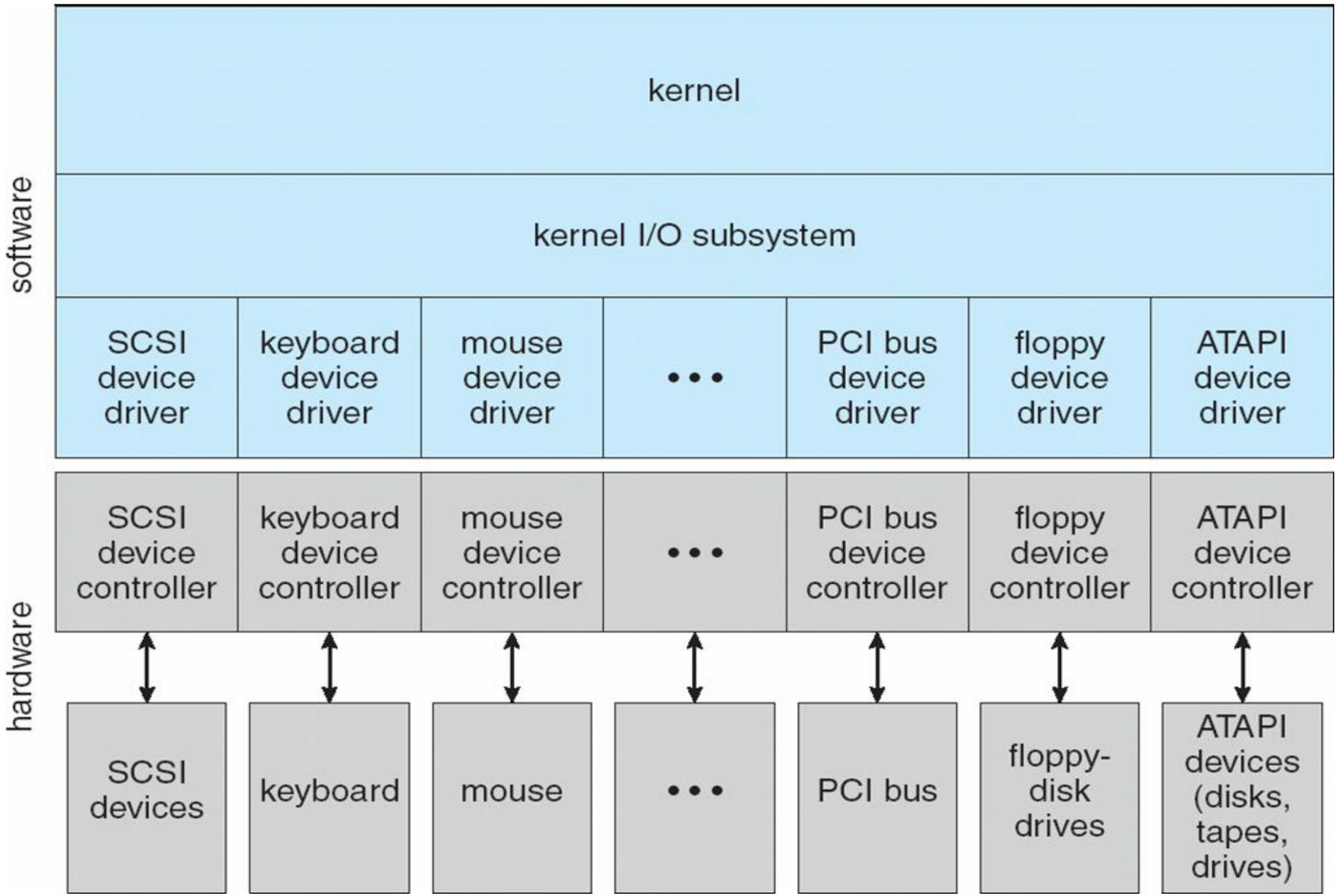
لماذا نحتاج Interface موحد؟
لأن أجهزة الـ I/O مختلفة جدًا، مثل:
الهاردسك
الكيبورد
الطابعة
الشبكة
وكل جهاز له طريقة عمل مختلفة.
لذلك الـ OS يخفي هذه التفاصيل حتى تصبح البرمجة أسهل.

الأجهزة تختلف بعدة أشياء:
1) Character Stream أو Block
- Character Devices
تنقل البيانات حرفًا حرفًا أو Stream.
أمثلة:
Keyboard
- Block Devices
تنقل البيانات على شكل Blocks.
أمثلة:
HDD
2) Sequential أو Random Access
—Sequential Access
يجب قراءة البيانات بالترتيب.
مثال:
Tape Drive
—Random Access
يمكن الوصول لأي مكان مباشرة.
مثال:
Hard Disk
3) Synchronous أو Asynchronous
Synchronous I/O
البرنامج ينتظر حتى تنتهي العملية.
Asynchronous I/O
البرنامج يكمل عمله، ويتم إشعاره لاحقًا عند انتهاء العملية.
4) Sharable أو Dedicated
—Sharable Device
عدة Processes تستطيع استخدامه.
مثل: الملفات
—Dedicated Device
جهاز يستخدمه Process واحد فقط في نفس الوقت.
مثل: بعض الطابعات
5) سرعة الأجهزة
الأجهزة تختلف في السرعة:
سريعة RAM
أبطأ Disk
6) نوع القراءة والكتابة
بعض الأجهزة:
Read & Write، Read Only، Write Only





A Kernel I/O Structure







Characteristics of I/O Devices

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk





Characteristics of I/O Devices (Cont.)

- Subtleties of devices handled by device drivers

كيف يتعامل الـ OS مع هذه الاختلافات؟
التفاصيل الدقيقة يتعامل معها:
Device Driver
وليس البرنامج نفسه.

- Broadly I/O devices can be grouped by the OS into
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets

Block Devices—
تنقل البيانات على شكل Blocks
مثل HDD و SSD
تدعم DMA و Random Access
Character Devices—
تنقل البيانات كـ Stream
مثل Keyboard و Mouse
Memory-Mapped Files—
الملف يُعامل كأنه جزء من الذاكرة
Network Devices—
تستخدم Sockets

- For direct manipulation of I/O device specific characteristics, usually an escape / back door
  - Unix `ioctl()` call to send arbitrary bits to a device control register and data to device data register





Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Raw I/O, direct I/O, or file-system access
  - Memory-mapped file access possible
    - File mapped to virtual memory and clusters brought via demand paging
  - DMA

- Character devices include keyboards, mice, serial ports
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing

Block I/O
مثل:
الأقراص الصلبة
الأوامر المشهورة:
Read
Write
Seek
مميزات الـ Block Devices
تدعم Random Access
تستخدم DMA
تدعم Memory-Mapped Files

Character I/O (Streams)
مثل:
Keyboard
Mouse
Serial Ports
أوامر الـ Character Devices
مثل:
get()
put()
Libraries فوق Character Devices
بعض المكتبات تضيف ميزات مثل:
تعديل السطر





Network Devices

- Varying enough from block and character to have own interface
- Unix and Windows NT/9*x*/2000 include **socket** interface
  - Separates network protocol from network operation
  - Includes `select()` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Network Devices
أجهزة الشبكة مختلفة عن:
Block Devices
Character Devices
لذلك لها Interface خاص.
Socket Interface
أنظمة مثل:
Unix
Windows
تستخدم:
Sockets
للتعامل مع الشبكات.
وظيفة الـ Socket
يفصل بين:
بروتوكولات الشبكة
وعمليات الإرسال والاستقبال
select()
دالة تسمح بمتابعة عدة اتصالات بنفس الوقت.
طرق أخرى للتواصل
مثل:
Pipes
FIFOs
Streams
Queues
Mailboxes





Clocks and Timers

الـ Clocks والـ Timers تُستخدم داخل نظام التشغيل لمعرفة:
الوقت الحالي
الزمن المستغرق
تنفيذ العمليات بعد مدة معينة

- Provide current time, elapsed time, timer
- Normal resolution about 1/60 second
- Some systems provide higher-resolution timers
- Programmable interval timer used for timings, periodic interrupts
- `ioctl()` (on UNIX) covers odd aspects of I/O such as clocks and timers



وظائف الـ Clocks و Timers
1) Current Time
إعطاء الوقت الحالي للنظام.
مثل: الساعة والتاريخ
2) Elapsed Time
حساب الزمن الذي استغرقته عملية معينة.
مثل: كم استغرق البرنامج للتنفيذ
3) Timer
تنفيذ حدث بعد فترة زمنية محددة.
مثل: التنبيه أو Sleep
دقة التوقيت (Resolution)
الدقة الطبيعية غالبًا:
1/60 ثانية
لكن بعض الأنظمة توفر:
High-Resolution Timers
بدقة أعلى.

Programmable Interval Timer
وهو Timer يمكن برمجته لإرسال:
دورية Interrupts
ويستخدم في:
Scheduling
Time Sharing
قياس الزمن
Timers و Clocks مع ioctl()
في Unix يمكن استخدام:
ioctl()





Blocking and Nonblocking I/O

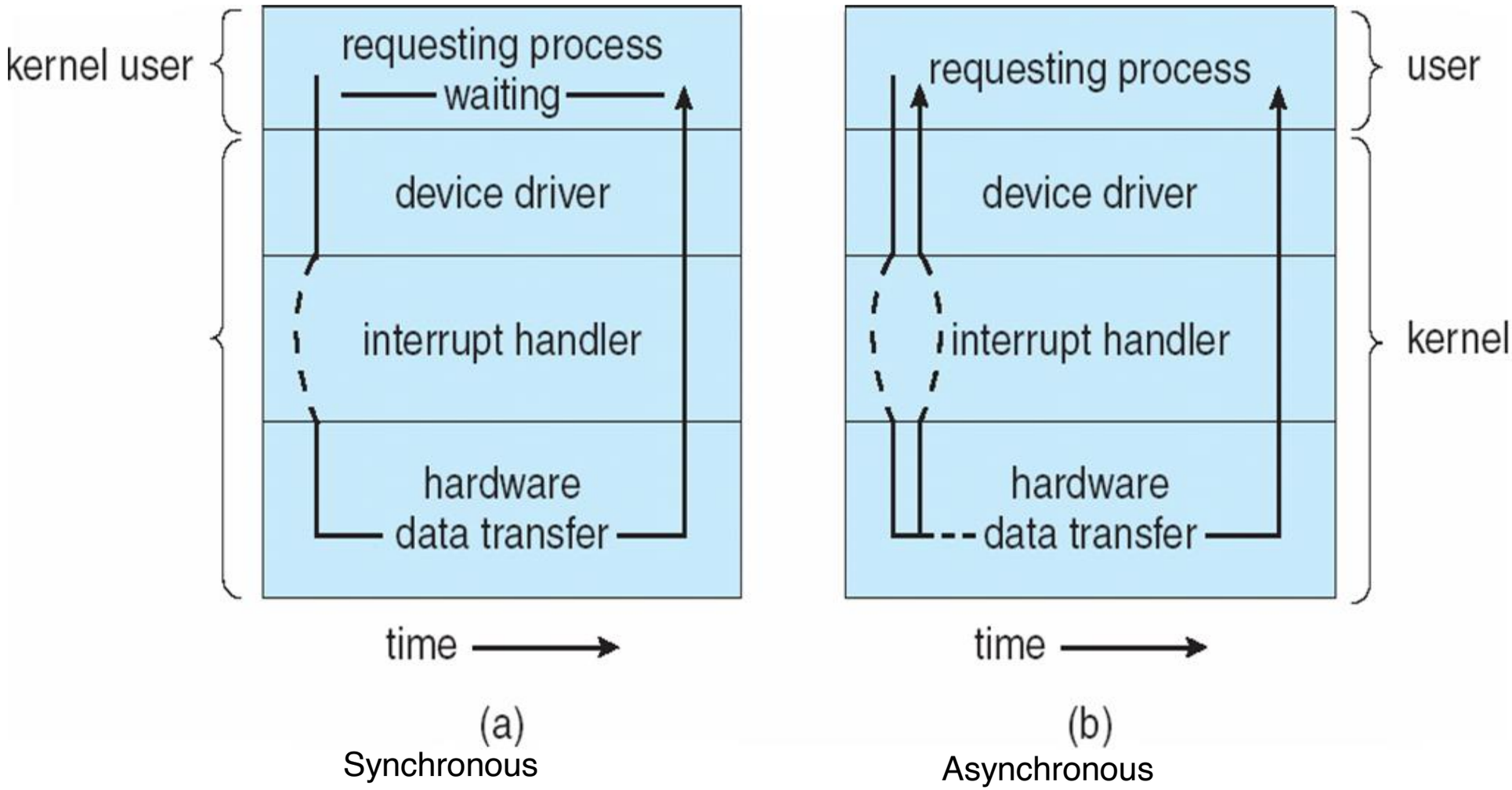
هذا الجزء يشرح كيف يتعامل الـ Process مع عمليات الـ I/O.

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
  - `select()` to find if data ready then `read()` or `write()` to transfer
- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

أولًا: Blocking I/O
الـ Process يتوقف
وينتظر حتى تنتهي عملية الـ I/O
مثال
إذا نفذ البرنامج: read()
ولم تصل البيانات بعد،
فالـ Process ينتظر.
مميزاته : سهل الفهم والبرمجة
عيوبه: يضيع الوقت
ثانيًا: Nonblocking I/O
الـ Process لا ينتظر
العملية ترجع مباشرة
كيف يعمل؟
النظام يعيد البيانات المتوفرة حاليًا فقط حتى لو كانت قليلة.
مميزاته: أسرع ومناسب للـ User Interface
كيف يُنفذ؟ باستخدام Multithreading
ماذا تُرجع العملية؟
ترجع عدد الـ Bytes التي تمت قراءتها أو كتابتها
select()
تُستخدم لمعرفة: هل البيانات جاهزة أم لا
ثم يتم استخدام:
read() أو write()
ثالثًا: Asynchronous I/O
الـ Process يكمل عمله
بينما عملية الـ I/O تعمل بالخلفية
عند انتهاء العملية
الـ OS يرسل إشعار للـ Process.
مميزاته : كفاءة عالية، لا يوجد انتظار
عيوبه : أصعب بالبرمجة، معقد



Two I/O Methods (P.510)





Kernel I/O Subsystem

هذا الجزء يشرح كيف يدير الـ Kernel عمليات الـ I/O داخليًا.

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
  - Some implement Quality Of Service (i.e. IPQOS)
- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain "copy semantics"
  - Double buffering – two copies of the data
    - Kernel and user
    - Varying sizes
    - Full / being processed and not-full / being used
    - Copy-on-write can be used for efficiency in some cases

1) Scheduling
الـ OS يرتب طلبات الـ I/O داخل Queue لكل جهاز.
لماذا؟ لتحسين الأداء، السرعة.

Fairness
بعض الأنظمة تحاول إعطاء كل Process فرصة عادلة لاستخدام الجهاز.

Quality Of Service (QoS)
بعض الأنظمة تعطي أولوية لبعض الطلبات.
مثل: الفيديو، الصوت، المكالمات
حتى لا يحدث تأخير.

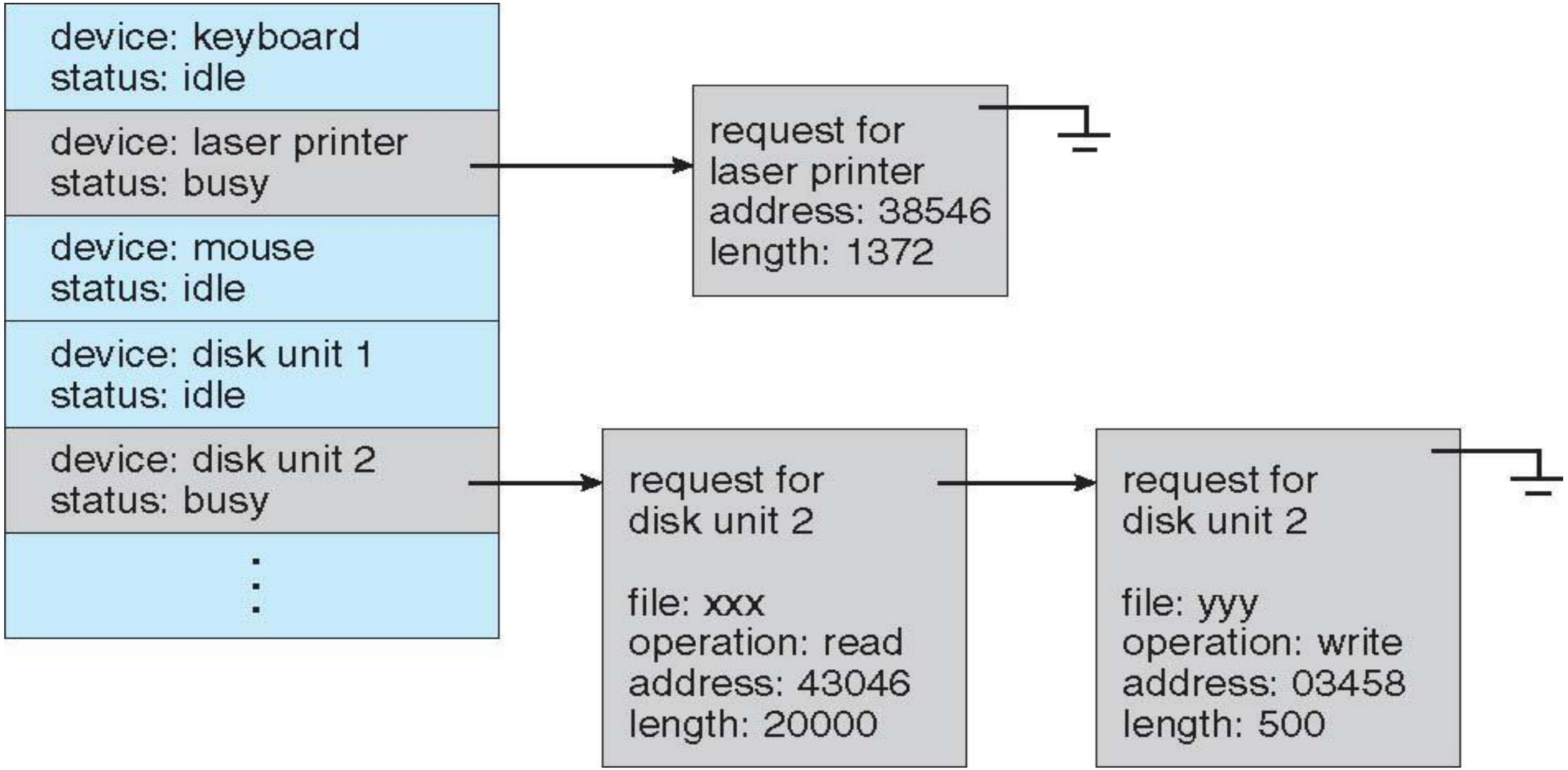
2) Buffering
تخزين البيانات مؤقتًا في الذاكرة أثناء نقلها بين الأجهزة.
لماذا نستخدم Buffering؟
1) اختلاف السرعات
بعض الأجهزة سريعة، وبعضها بطيء.
فالـ Buffer يساعد على التوازن.
2) اختلاف حجم البيانات
قد يرسل جهاز: Byte واحد
بينما جهاز آخر يحتاج: Block كامل
فالـ Buffer ينظم عملية النقل.

3) Copy Semantics
للحفاظ على البيانات مستقرة أثناء النقل،
حتى لو تغيرت النسخة الأصلية.

Double Buffering
نسختين من البيانات
نسخ البيانات في Double Buffering قد توجد:
نسخة في الـ Kernel
ونسخة عند المستخدم
Copy-on-Write
تقنية لتحسين الكفاءة.
بدل نسخ البيانات مباشرة:
يتم النسخ فقط عند التعديل الحقيقي.

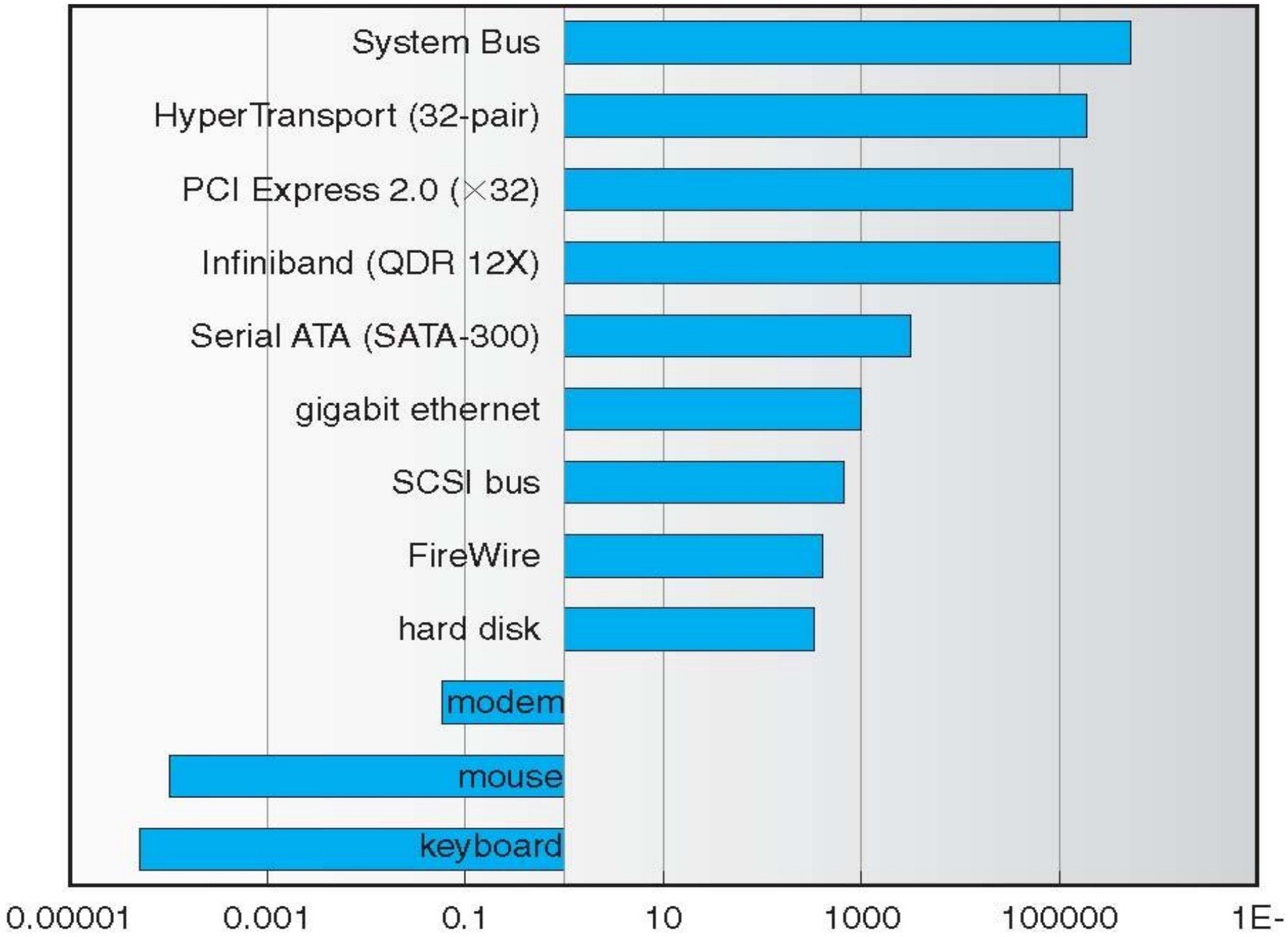


Device-status Table





Sun Enterprise 6000 Device-Transfer Rates







Kernel I/O Subsystem

- **Caching** - faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering

1) Caching
حفظ نسخة من البيانات في جهاز أسرع
غالبًا داخل: **RAM**
بدل قراءة البيانات كل مرة من جهاز بطيء مثل: **Hard Disk**
الـ **Cache** يحتوي **Copy** فقط
وليس البيانات الأصلية.
لماذا الـ **Caching** مهم؟
لأنه يسرّع الوصول للبيانات، ويحسن الأداء بشكل كبير

- **Spooling** - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing

2) Spooling
تخزين طلبات الإخراج مؤقتًا
حتى يأتي دور الجهاز لتنفيذها.
لماذا نستخدم **Spooling**؟
لأن بعض الأجهزة تستطيع تنفيذ طلب واحد فقط في نفس الوقت
أشهر مثال : الطباعة
إذا أرسل عدة مستخدمين ملفات للطباعة: النظام يخزن الطلبات
داخل **Queue** ثم يطبعها واحدًا واحدًا.

- **Device reservation** - provides exclusive access to a device
  - System calls for allocation and de-allocation
  - Watch out for deadlock

3) Device Reservation
إعطاء **Process** واحد وصول حصري للجهاز
بحيث لا يستطيع **Process** آخر استخدامه مؤقتًا.
لماذا نحتاج **Device Reservation**؟
بعض الأجهزة لا تتحمل الاستخدام المتزامن.
مثل: بعض الطابعات، أجهزة خاصة
كيف يتم ذلك؟ باستخدام **System Calls**
من أجل:
حجز الجهاز → **Allocation**
تحرير الجهاز → **De-allocation**
مشكلة مهمة: **Deadlock**
عند حجز الأجهزة يجب الانتباه إلى:
Deadlock
ما هو الـ **Deadlock**؟
حالة تتوقف فيها العمليات بسبب انتظار الموارد من بعضها.
مثال
Process A حجزت **Printer** وتنتظر **Scanner**
Process B حجزت **Scanner** وتنتظر **Printer**
النتيجة:
كل **Process** تنتظر الأخرى
ولا أحد يكمل





Error Handling

- OS can recover from disk read, device unavailable, transient write failures
  - Retry a read or write, for example
  - Some systems more advanced – Solaris FMA, AIX
    - Track error frequencies, stop using device with increasing frequency of retry-able errors
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

Error Handling
هذا الجزء يشرح كيف يتعامل نظام التشغيل مع أخطاء الـ I/O.
أثناء التعامل مع الأجهزة قد تحدث مشاكل مثل:
فشل قراءة من القرص
الجهاز غير متوفر
لذلك الـ OS يجب أن يكتشف الأخطاء ويحاول إصلاحها.

كيف يتعامل الـ OS مع الأخطاء؟
Retry)
إعادة المحاولة مرة أخرى

لماذا تنجح إعادة المحاولة أحيانًا؟
لأن بعض الأخطاء:
مؤقتة (Transient Errors)
مثل:
انقطاع بسيط
بطء مؤقت بالجهاز
أنظمة متقدمة

بعض الأنظمة مثل: Solaris FMA و AIX
تراقب: معدل تكرار الأخطاء
ماذا يحدث إذا كثرت الأخطاء؟
النظام قد:
يعتبر الجهاز غير موثوق
ويتوقف عن استخدامه
خصوصًا إذا أصبحت الأخطاء تتكرر كثيراً.

Error Codes
إذا فشلت عملية الـ I/O،
فإن النظام يعيد:
Error Code أو Error Number
حتى يعرف البرنامج نوع المشكلة.

System Error Logs
النظام يحفظ تقارير الأخطاء داخل: Error Logs
حتى يستطيع المسؤول: معرفة المشاكل، تتبع الأعطال، إصلاحها





I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

I/O Protection

هذا الجزء يشرح كيف يحمي نظام التشغيل الأجهزة من البرامج.

لماذا نحتاج I/O Protection؟

لأن بعض البرامج قد ترتكب أخطاء

الحل

جميع أوامر الـ I/O تعتبر:

Privileged Instructions

ماذا يعني Privileged؟

يعني:

لا يسمح بتنفيذها إلا داخل الـ Kernel Mode

وليس User Mode.

كيف ينفذ البرنامج عمليات I/O إذًا؟

عن طريق: System Calls

مثل:

write() أو read()

والـ OS ينفذ العملية نيابةً عنه.

حماية Memory-Mapped I/O

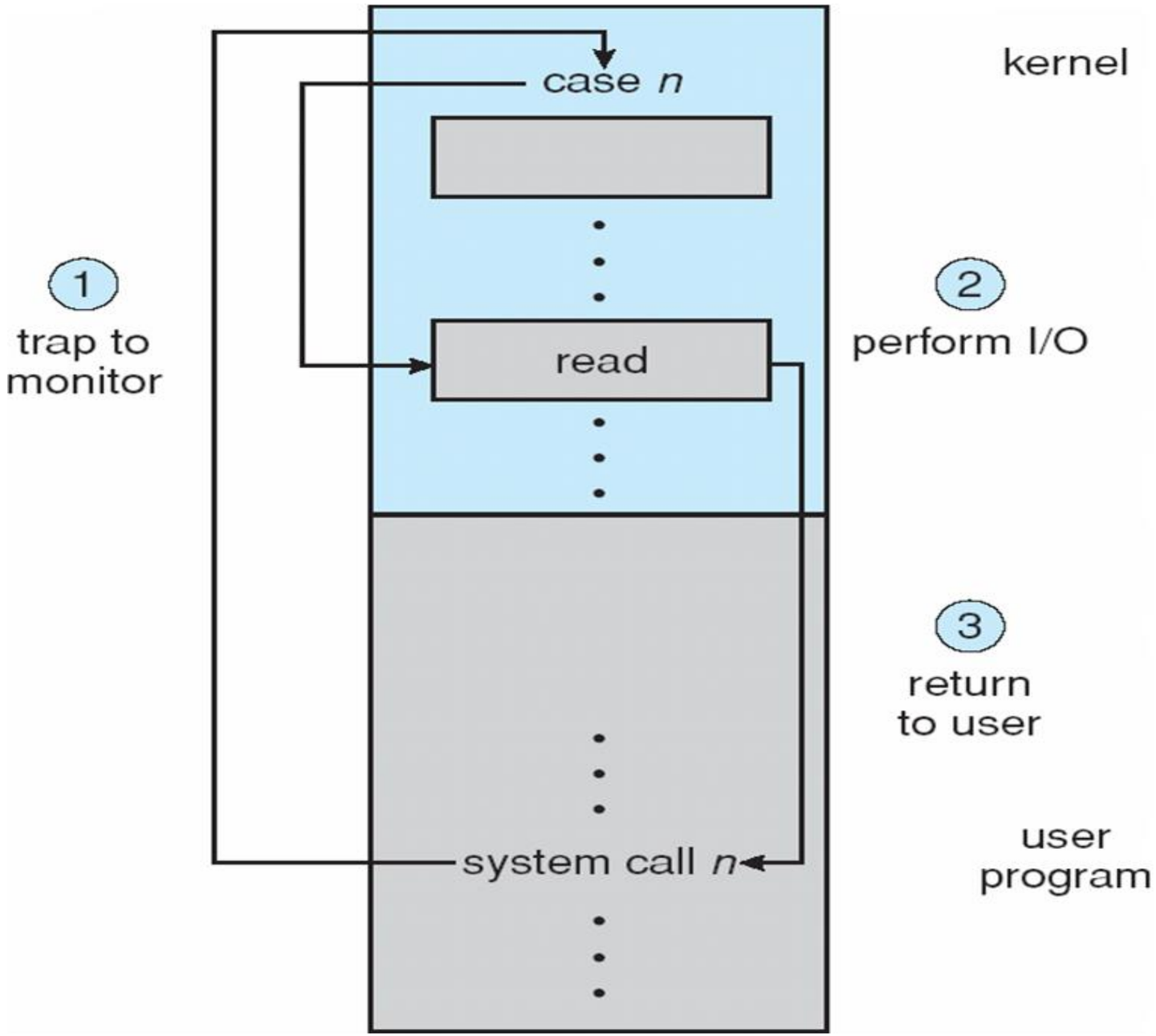
حتى عناوين الأجهزة الموجودة داخل الذاكرة:

يجب حمايتها أيضًا

حتى لا يصل إليها أي برنامج مباشرة.



Use of a System Call to Perform I/O





Kernel Data Structures

الـ Kernel يحتاج تخزين معلومات كثيرة عن أجهزة الـ I/O.

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- Some use object-oriented methods and message passing to implement I/O
  - Windows uses message passing
    - Message with I/O information passed from user mode into kernel
    - Message modified as it flows through to device driver and back to process

أمثلة على المعلومات المخزنة
Open File Tables
Network Connections
Character Device State

لماذا هذه البيانات مهمة؟
حتى يعرف النظام:
من يستخدم الجهاز
ما حالة الجهاز

أين توجد البيانات
الـ OS يحتوي تراكيب بيانات كثيرة لإدارة:
Buffers
Memory Allocation
Dirty Blocks

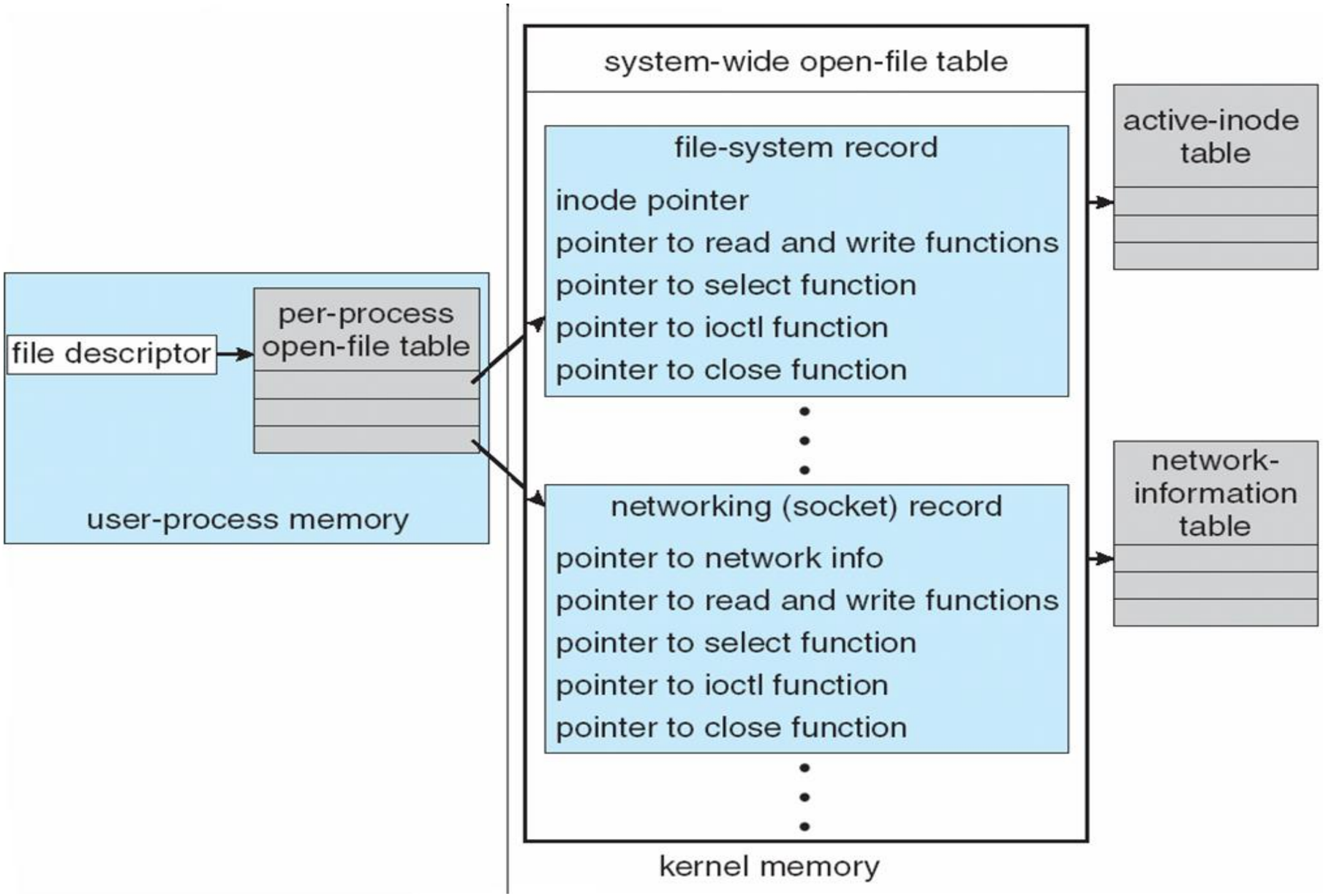
ما هي Dirty Blocks؟
تم تعديلها في الذاكرة Blocks
لكن لم تُكتب بعد على القرص.

Message Passing و Object-Oriented
بعض الأنظمة تستخدم:
Object-Oriented Design
Message Passing
في إدارة الـ I/O.

مثال: Windows
ويندوز يستخدم:
Message Passing
كيف يعمل؟
البرنامج يرسل Message تحتوي معلومات الـ I/O
الرسالة تدخل إلى الـ Kernel
تنتقل إلى الـ Device Driver
ثم تعود النتيجة للبرنامج



UNIX I/O Kernel Structure



هذا الجزء يشرح الرحلة الكاملة لطلب الـ I/O.



I/O Requests to Hardware Operations

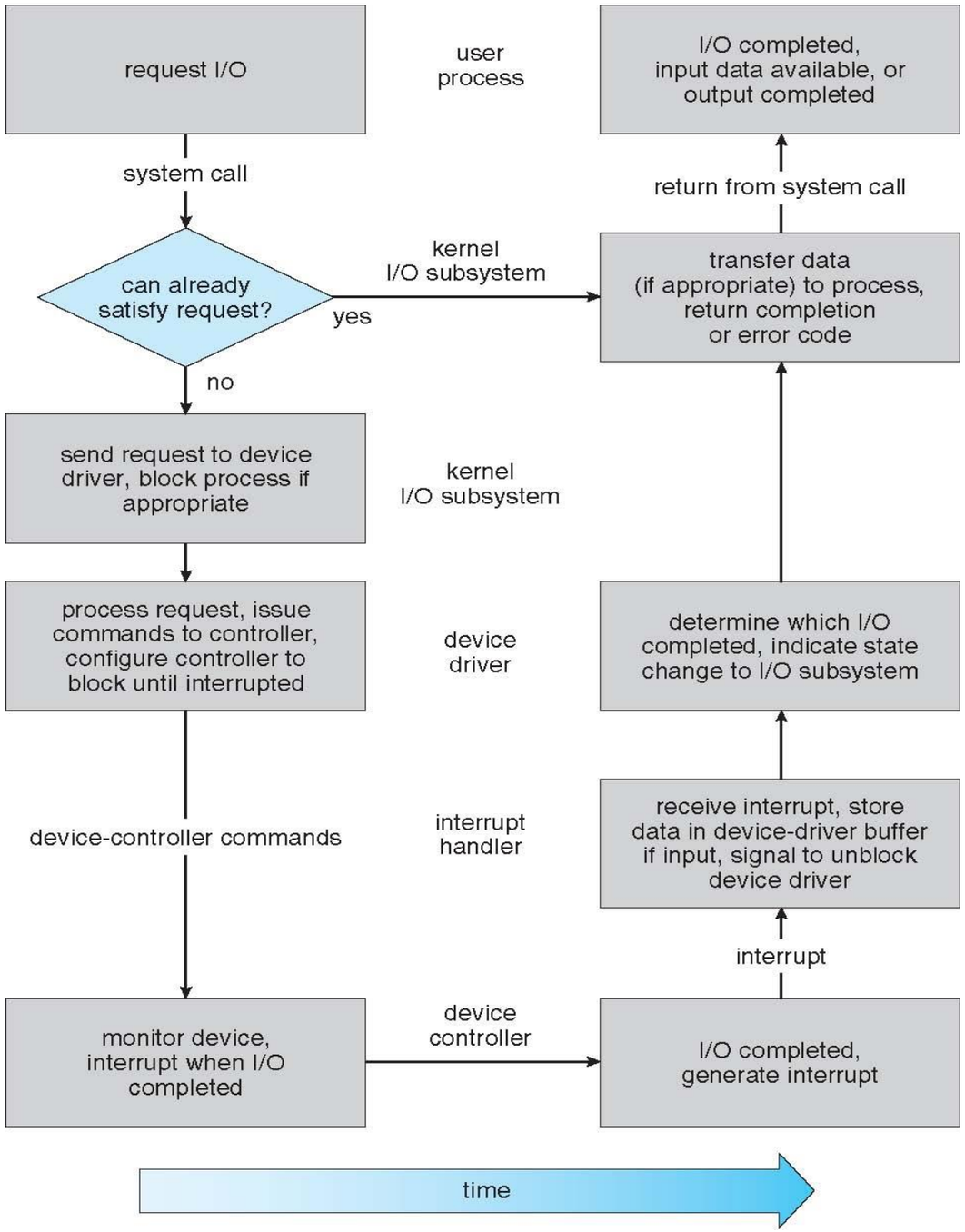
- Consider reading a file from disk for a process:
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process

مثال: قراءة ملف من القرص
الخطوة 1
النظام يحدد:
أي جهاز يحتوي الملف
مثل:
أي Hard Disk
الخطوة 2
تحويل اسم الملف إلى:
Device Representation
أي الشكل الذي يفهمه النظام والجهاز.
الخطوة 3
قراءة البيانات فعليًا من القرص إلى:
Buffer
داخل الذاكرة.
الخطوة 4
جعل البيانات متاحة للـ Process الذي طلبها.
الخطوة 5
إرجاع التحكم مرة أخرى للبرنامج.
الفكرة الأساسية
عملية بسيطة مثل:
read()
تمر بمراحل كثيرة داخل الـ OS قبل الوصول للهاردوير.





Life Cycle of An I/O Request







STREAMS

- **STREAM** – a full-duplex communication channel between a user-level process and a device in Unix System V and beyond

- A STREAM consists of:
  - STREAM head interfaces with the user process
  - driver end interfaces with the device
  - zero or more STREAM modules between them

- Each module contains a **read queue** and a **write queue**

- Message passing is used to communicate between queues
  - **Flow control** option to indicate available or busy

- Asynchronous internally, synchronous where user process communicates with stream head

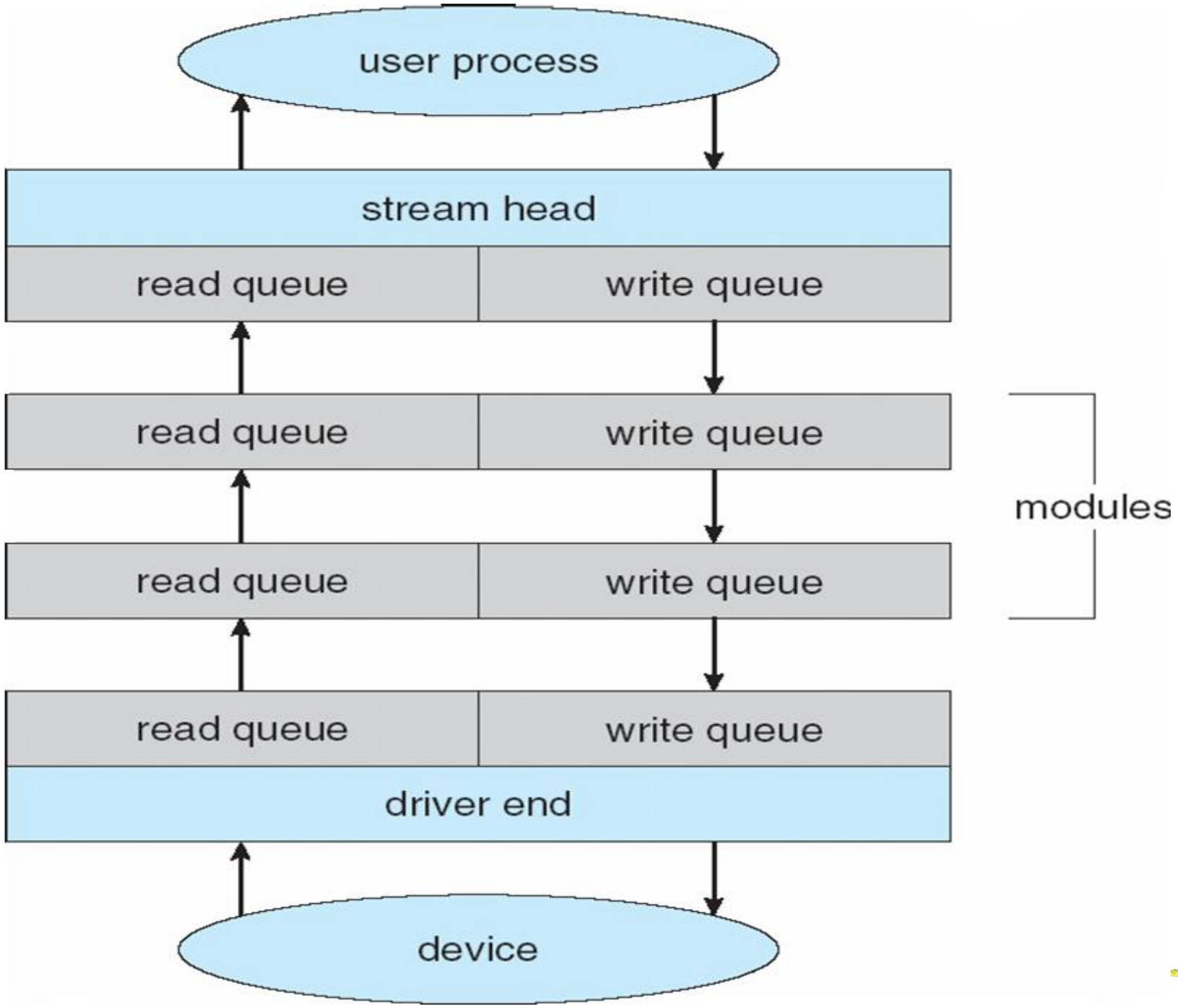
STREAM هو قناة اتصال ثنائية الاتجاه بين:
برنامج على مستوى المستخدم
والجهاز داخل نظام Unix System V وما بعده
يتكوّن الـ STREAM من:
1) STREAM head
الجزء الذي يتعامل مباشرة مع برنامج المستخدم.
2) Driver end
الجزء الذي يتعامل مع الجهاز نفسه.
3) Modules
قد يوجد بينهما:
صفر
أو أكثر من الوحدات
كيف تعمل؟
كل Module يحتوي على:
Read Queue
Write Queue
ويتم التواصل بين هذه الـ queues باستخدام:
Message Passing
Flow Control
أحيانًا يوجد خيار للتحكم في التدفق، لمعرفة:
هل الطرف الآخر جاهز؟
أم مشغول؟

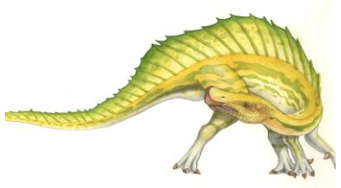
فكرة الـ STREAMS باختصار
من الداخل: Asynchronous
بين المستخدم و stream head: Synchronous
يعني البرنامج يتعامل بشكل مرتب وواضح، بينما النظام ينفذ داخليًا بطريقة غير متزامنة.





The STREAMS Structure







Performance

- I/O a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Performance

الـ I/O يؤثر كثيراً على أداء النظام لأنه يستهلك:

المعالج CPU

وقت الـ kernel

عمليات تغيير السياق Context Switches

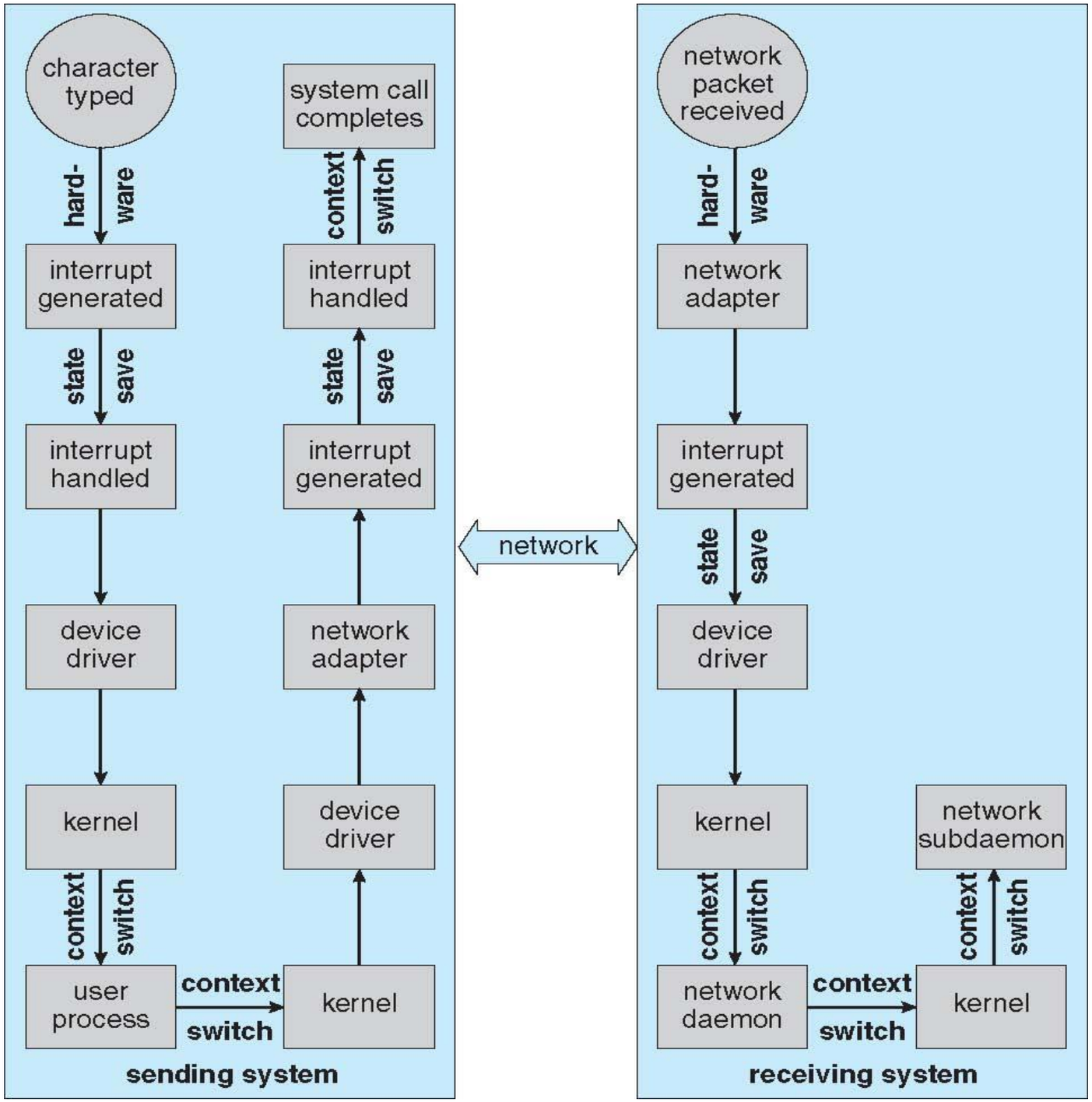
نسخ البيانات

والـ Network traffic يكون مرهقًا جدًا





Intercomputer Communications







Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Use smarter hardware devices
- Balance CPU, memory, bus, and I/O performance for highest throughput
- Move user-mode processes / daemons to kernel threads

لتحسين الأداء، يمكن النظام أن يقوم بـ:

1) تقليل عدد الـ **Context Switches**

كلما قلّت، صار النظام أسرع.

2) تقليل نسخ البيانات

لأن النسخ المتكرر يستهلك وقتًا وذاكرة.

3) تقليل الـ **Interrupts**

ويتم ذلك باستخدام:

Transfers أكبر

Smart Controllers

Polling في بعض الحالات

4) استخدام **DMA**

لنقل البيانات مباشرة بين الجهاز والذاكرة.

5) استخدام **Hardware** أذكى

أجهزة أفضل تعني تدخل أقل من المعالج.

6) موازنة الأداء بين:

CPU

Memory

Bus

I/O

حتى يصل النظام إلى أعلى **Throughput**.

7) نقل بعض الـ **user-mode processes** أو **daemons** إلى **kernel threads**

حتى تكون أسرع وأقرب إلى نواة النظام.



Device-Functionality Progression

