운영체제의 기초:
I/O Devices and Device Drivers

2023년 6월 1, 6일

홍 성 수

sshong@redwood.snu.ac.kr

SNU RTOSLab 지도교수

서울대학교 전기정보공학부 교수

Agenda

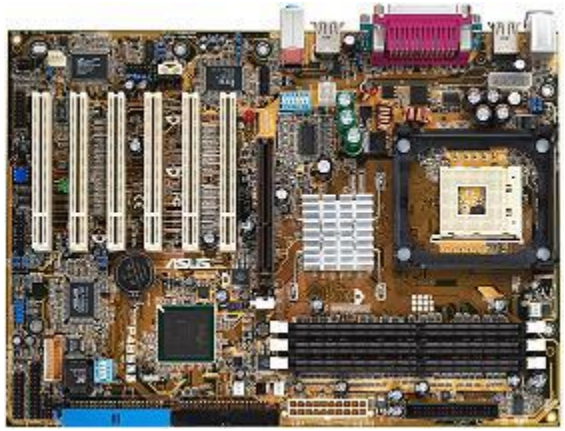
I. I/O Hardware
II. Device Drivers

I. I/O Hardware

Inside Your Computer

- Your computer is equipped with
  - CPU and memory
  - Many I/O devices
    - VGA card, network card, disk controller, …

Network card



CPU

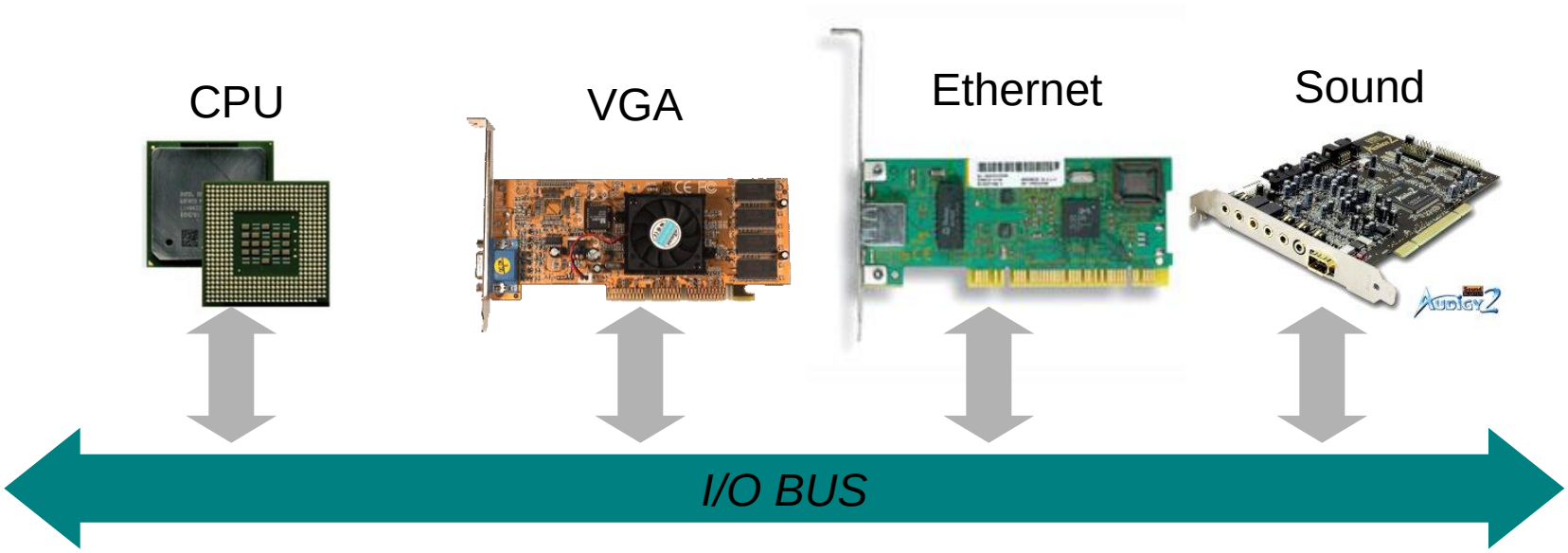
Hard disk

Main board

Memory

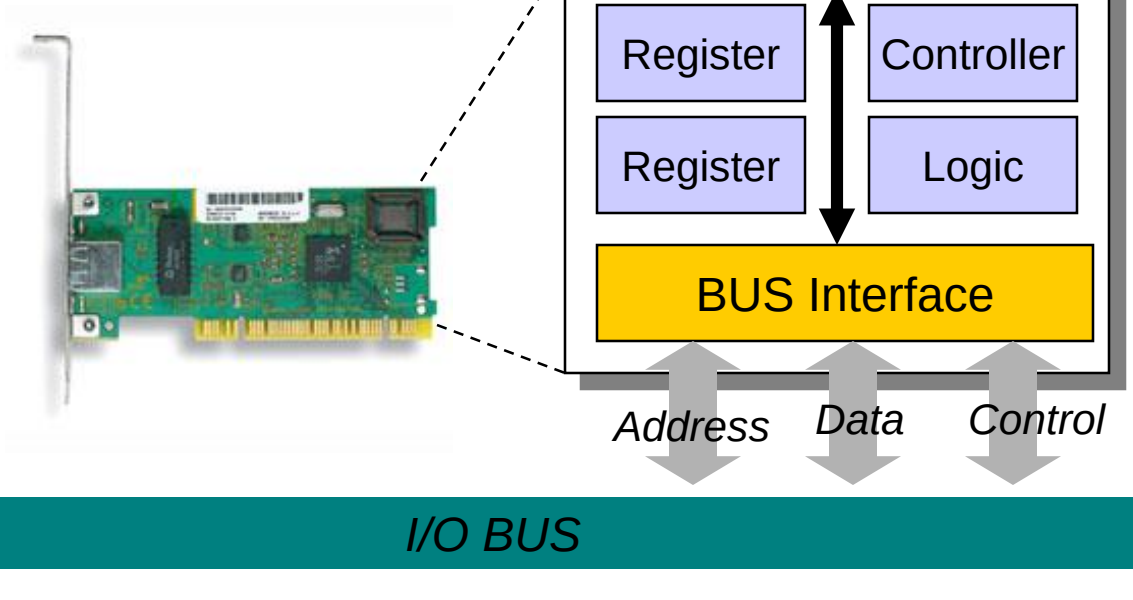
Device Connection

- Devices are attached to I/O bus
  - ISA, PCI, EISA, SCSI, …



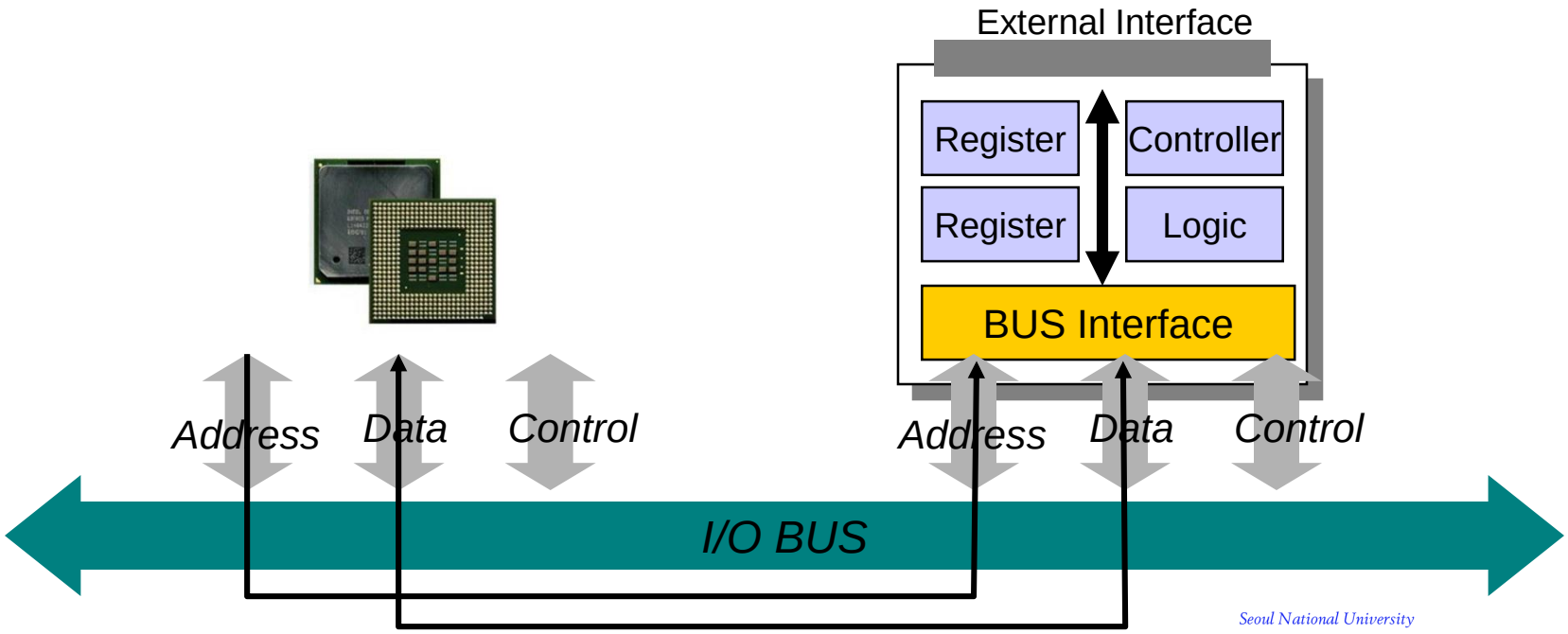
Inside I/O Device

- I/O device consists of
  - Controller, logic
  - Register set
  - Bus interface



Device Control

- How CPU controls devices
  - CPU writes register address on the bus
    - Address space is assigned to each device
  - Device write/read data on the bus



I/O Devices

- Character devices
  - Mouse
  - Terminal
- Block devices
  - Disk drive
  - Flash drive
- Network devices
  - NIC (network interface card)

Character Device: Mouse (1)

- Example

Source: http://commons.wikimedia.org/wiki/File%3AMicrosoft-wireless-mouse.jpg

Character Device: Mouse (2)

- Brief history
  - First mechanical mouse with a roller ball
    - Bill English at Xerox PARC in the early 1970s
  - Introduced by Apple Macintosh in 1984
    - They have helped to completely redefine the way we use computers since then
  - Became the PC-human interface of choice quickly when Windows 3.1 made Graphical User Interface (GUI) a standard
  - Optical Mouse
    - Gary Gordon at Agilent Laboratories in 1999

Character Device: Mouse (3)

- Mechanism – Optical mouse
  - Tiny camera takes 1500-7080 images per second
    - Camera = laser + a CMOS sensor
  - Images sent for analysis to a DSP operating typically at 18 MIPS
  - DSP detects patterns in images and thus estimates motion
  - Data ports are used for two-way communication
  - Upon mouse movement, a 3/5-byte packet is sent to the port
    - Typical description of the data
      - (x_s, y_s), (x_d, y_d), mouse-up/down
  - This data packet is decoded by the mouse driver and its internal co-ordinates are updated

Character Device: Mouse (4)

- PC mouse system (data transfer chain)
  - Sensors (CMOS)
  - Mouse Controller (DSP)
  - Communication link (Cable/Wireless)
  - Data interface (Serial, PS/2, USB)
  - Device driver
  - Application

Character Device: Mouse (5)

- How OS collaborates with mouse driver
  - *Applications* wait for mouse movement or click
  - When a mouse movement occurs, (1) the *mouse driver* informs (2) the *event manager* of OS about the event
    - *Mouse driver* automatically tracks the mouse and displays the cursor as the user moves the mouse
    - *Event manager* determines whether to queue the event or not
  - When a mouse-up or mouse-down event occurs, the *event manager* records the action in the event queue and informs (3) the *active application* about it
  - The *active application* decides what action is to be taken
    - Ex: Show the mouse cursor, hide the cursor, and draw something onto the screen, etc.

Character Device: Terminal (1)

- Example
  - DEC VT100, Heathkit Z19

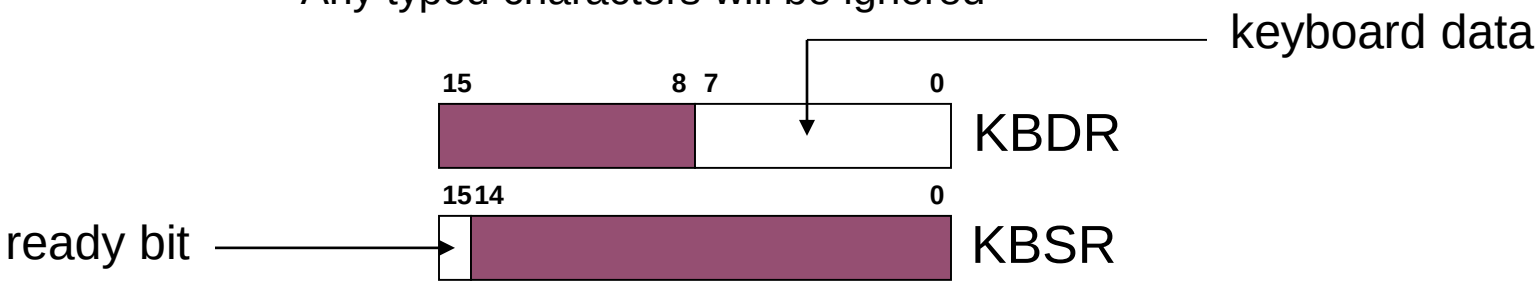
Source: http://upload.wikimedia.org/wikipedia/commons/6/6f/Terminal-dec-vt100.jpg

Character Device: Terminal (2)

- Mechanism
  - Terminal = keyboard + display
    - Keyboard and display are handled independently in most systems (no automatic echo, full duplex serial link)
  - I/O registers are connected to the host via serial line
    - Keyboard *data/status* registers
    - Display *data/status* registers
  - *One interrupt per character*
    - One character (8-bit data or control function) is sent at a time
  - ASCII encoding is used: ‘A’ is 0x65
  - Slow speed
    - 10-1800 characters per second
    - Measure: Baud rate (bits per second)

Character Device: Terminal (3)

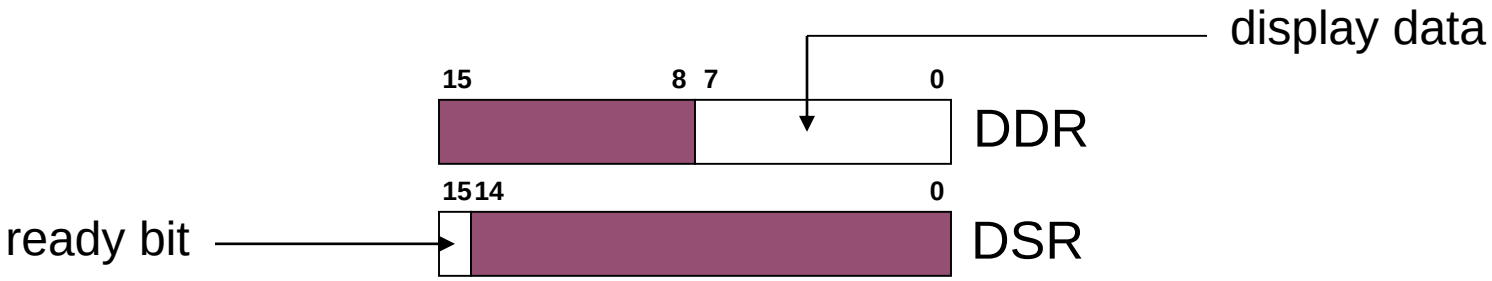
- Keyboard input
  - When a character is typed:
    - Its ASCII code is placed in bits [7:0] of keyboard data register
    - The "*ready bit*" of keyboard status register is set to zero
      - Keyboard is *disabled*
        - Any typed characters will be ignored



  - When KBDR is read:
    - The "*ready bit*" of KBSR is set to one
    - Keyboard is enabled

Character Device: Terminal (4)

- Display output
  - When monitor is ready to display another character
    - The "*ready bit*" of display status register is set to one



  - When data is written to DDR:
    - The "*ready bit*" of DSR is set to zero
      - Any other character data written to DDR is *ignored*
    - Character in DDR is displayed
      - The "*ready bit*" of DSR is set to one

Character Device: Terminal (5)

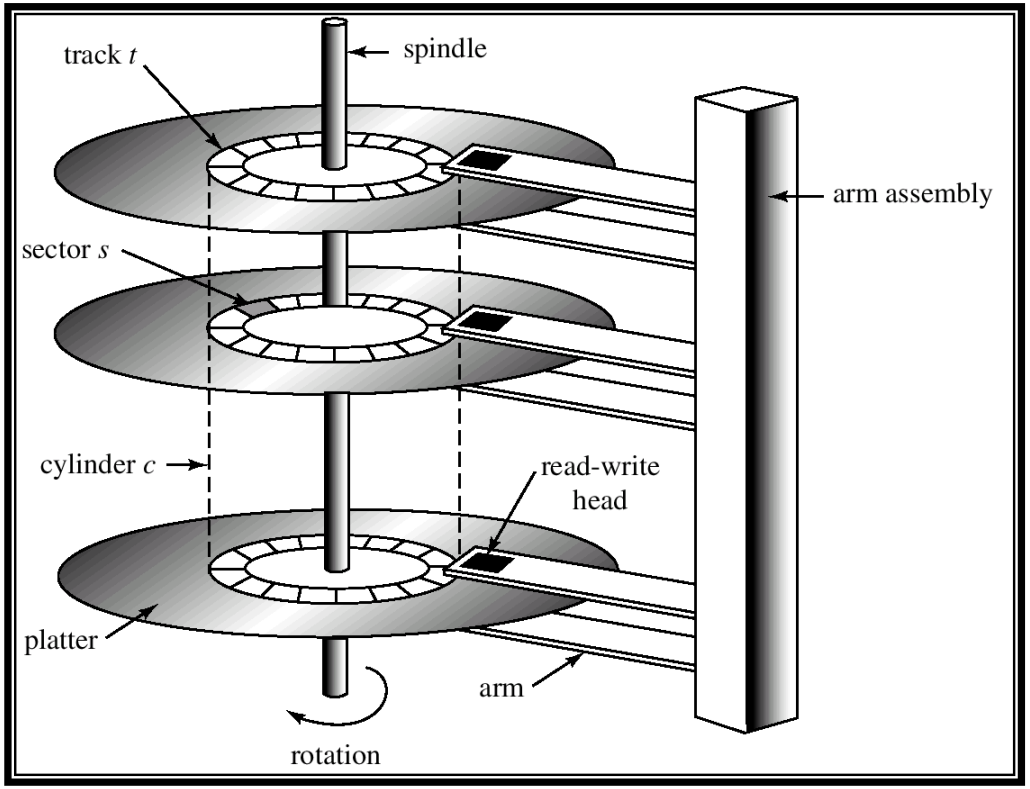
- Keyboard echo
  - Usually, the input character is also printed to the screen automatically
  - User gets feedback on character typed and knows it's ok to type the next character

Block Device: Disk Drive (1)

- Hard disk drive

Block Device: Disk Drive (2)

- Moving-head disk mechanism



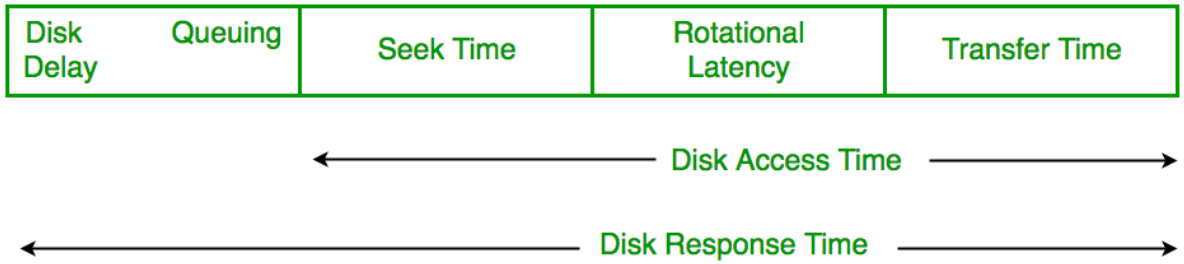
Source: Silberschatz, Galvin and Gagne, Operating System Concepts, 2005

Block Device: Disk Drive (3)

- Disk characteristics (technology of 2012)
  - 2-6 heads (platters x 2)
  - Platter diameter between 0.8" and 8"
  - 16,383 tracks (cylinders) per surface
  - 63 sectors per track
  - Sector size of 512 to 4096 bytes
    - 4KB physical emulated at 512-byte sectors
  - Capacity ranges up to 4 TB

Block Device: Disk Drive (4)

- Disk operation
  - Select desired read/write head
  - Move heads to the correct track ("*seek*")
    - Seek time
  - Wait for disk to rotate desired sector into position
    - Rotational latency (delay)
  - Read and write sector while it spins by
    - Transfer time



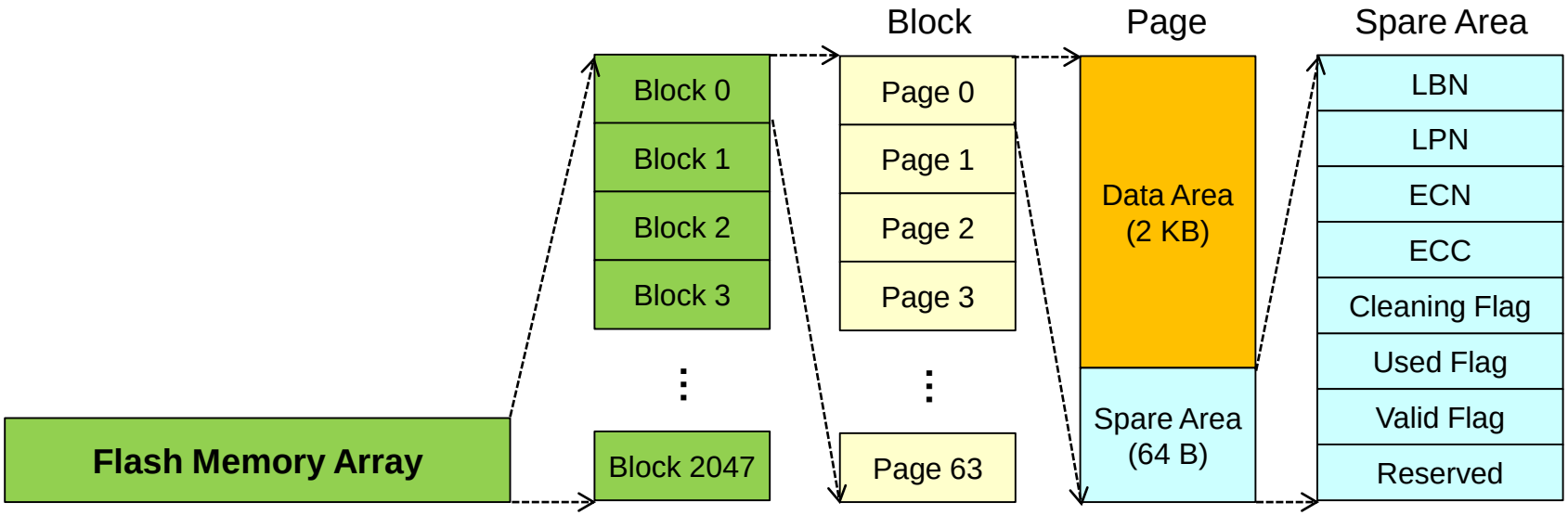
Source: https://www.geeksforgeeks.org/disk-scheduling-algorithms

Block Device: Disk Drive (5)

- Disk performance numbers
  - Seek time: 0~50 ms (average 10~20 ms)
  - Rotational delay: 0~16 ms
    - Typical drive spins at 3600~5400 RPM
  - Transfer time: 8~40 μs
    - Maximum transfer rate is 25MB/s~125 MB/s

Block Device: Flash Drive (1)

- Array structure of flash memory
  - Page (2 KB, 4 KB, 8 KB)
  - Block (64 pages, 128 pages)

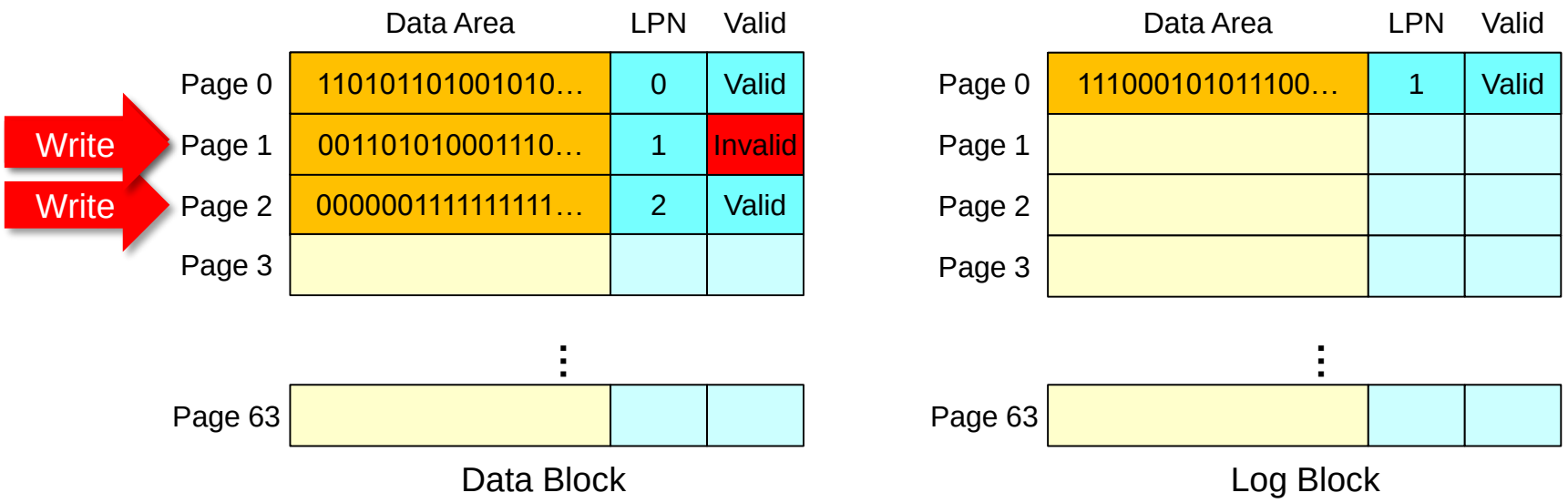


Block Device: Flash Drive (2)

- Flash operations
  - Read from page: takes 20 μs
  - Write/Program onto page: takes 200 μs
  - Erase block: takes 2000 μs
- Constraints on flash operations
  - Erase-before-write
  - Worn-out

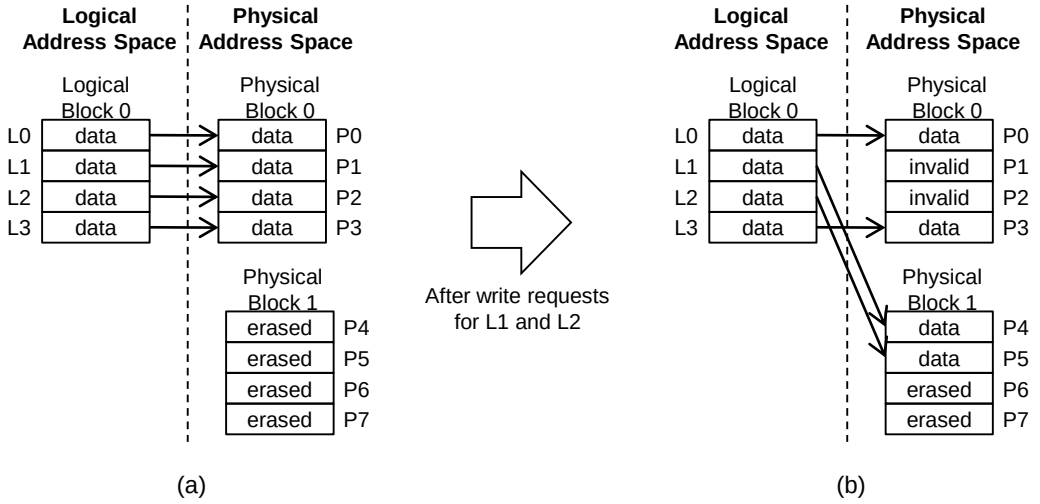
Block Device: Flash Drive (3)

- Log structure due to “erase-before-write” constraints



Block Device: Flash Drive (4)

- Log-structured file system
  - Page remapping after update
    - No need for file system scanning after sudden power-off
      - Ideal characteristics for hand-held smart devices



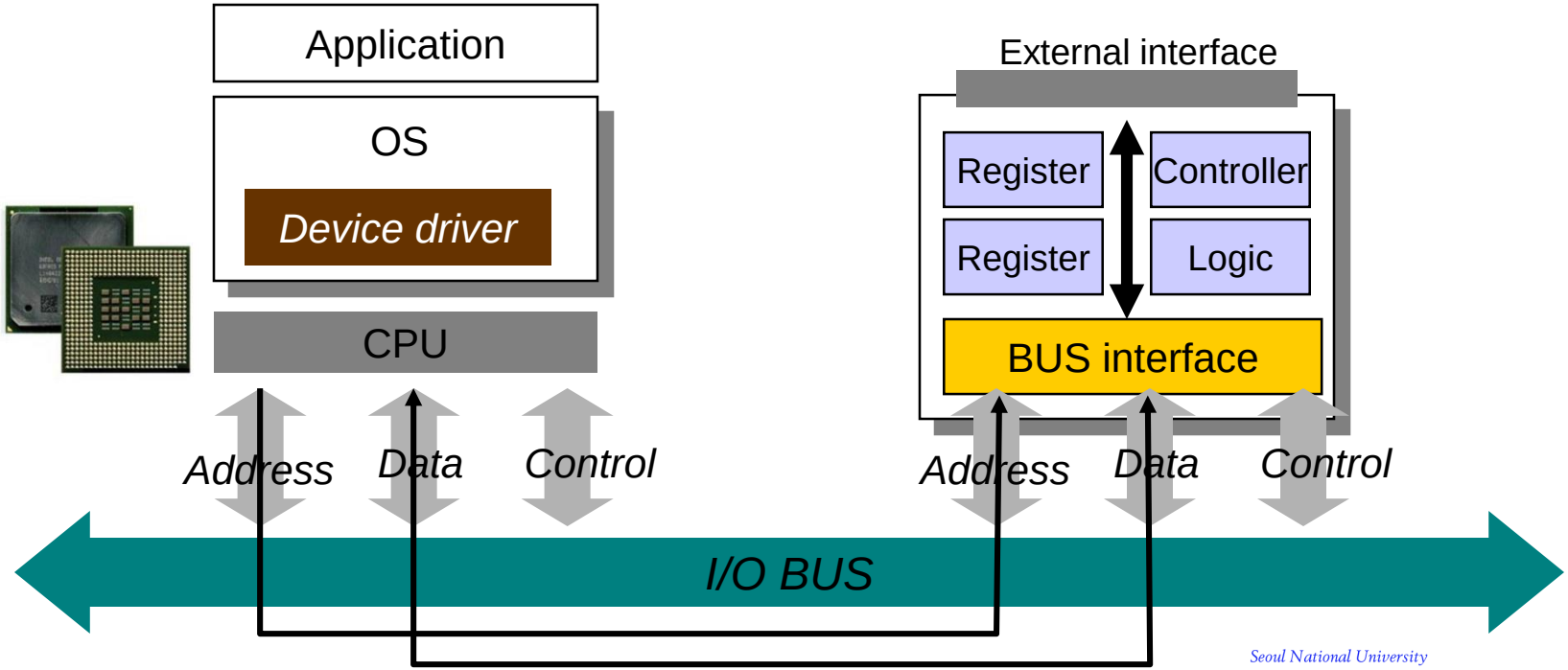
II. Device Drivers

What is Device Driver? (1)

- A computer program that enables another program (typically an OS) to interact with a hardware device

What is Device Driver? (2)

- Software layer between applications and devices
- Implements operations exposed to users
  - (Ex) open, close, read, write, ioctl, …



Types of Linux Devices (1)

- Character device
  - Transfer unit: character (byte)
  - Can be accessed like a file (open, close, read, write, …)
  - (Ex) console, keyboard, mouse, … (/dev/tty1, /dev/lp1, …)
- Block device
  - Transfer unit: block (usually some kilobytes)
  - Can be accessed like a file (open, close, read, write, …)
  - External view to users is same as character devices
  - Internally, block buffer is used for efficiency (contrast to character devices)
  - (Ex) hard disk drive, CD-ROM drive, … (/dev/hda1, …)

Types of Linux Devices (2)

- Network interface
  - Can't be easily mapped to either character or block device
  - (Ex) eth0

Accessing I/O Device Drivers (1)

- Two related questions
  - How to name an I/O device?
  - How to gain access to the device driver routines?

- Name an I/O device
  - Device file
    - Special file containing attributes of the I/O device
    - Represents an I/O device
    - Ex: "**`/dev/tty0`**": first serial port
    - Two device files can represent the same I/O devices (but may be implemented in a different way)
      - Ex: **`/dev/psaux`**, **`/dev/psmouse`**: serial mouse

Accessing I/O Device Drivers (2)

- Device file contains such attributes as below
  - Type: block or character
  - Major number: specifies device driver
  - Minor number: argument to device driver, kernel don't care

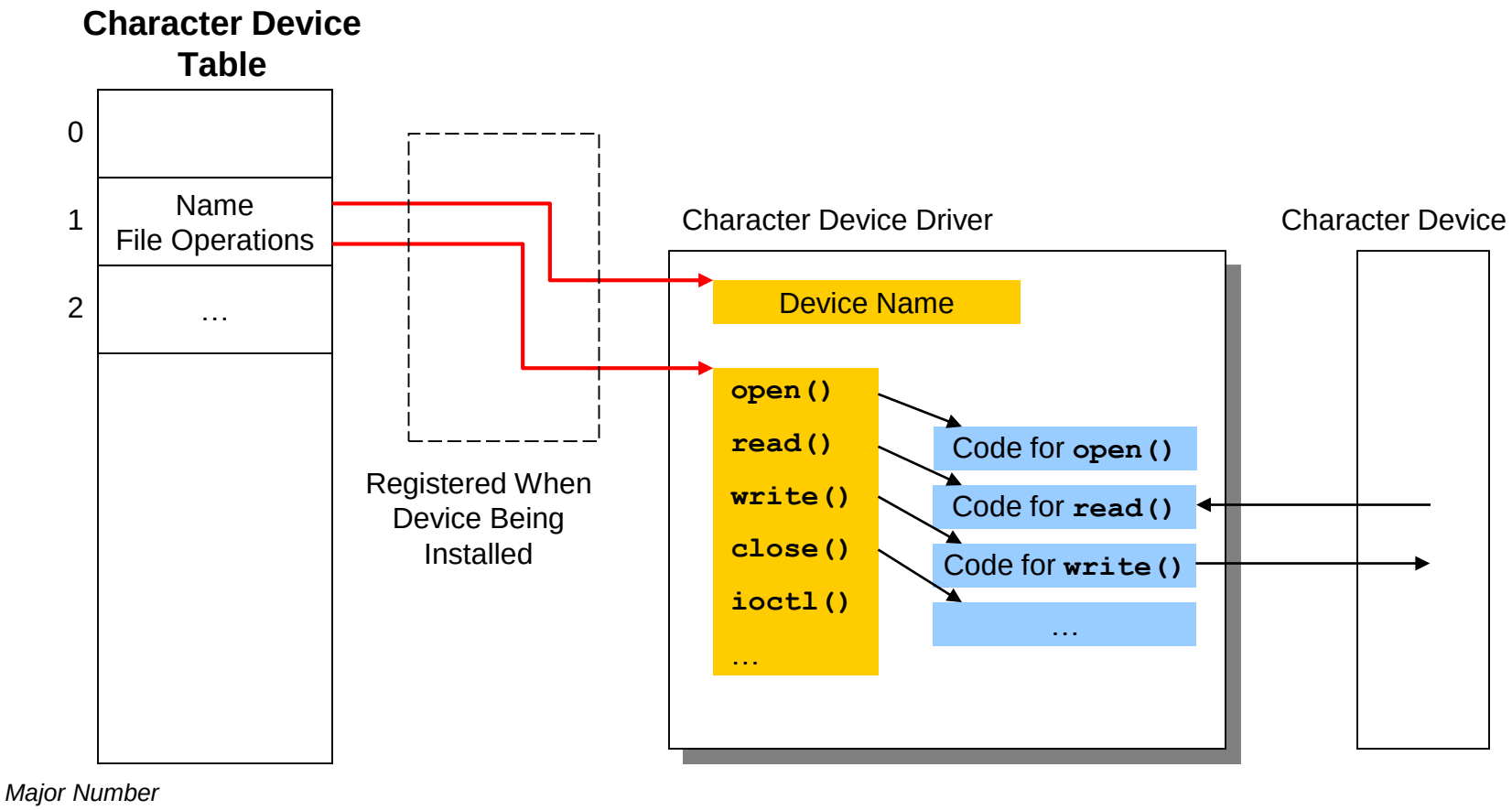
Name	Type	Major	Minor	Desc
/dev/fd0	Block	2	0	Floppy Disk
/dev/hda	Block	3	0	First IDE Disk
/dev/hda2	Block	3	2	Second Partition of First IDE Disk
/dev/hdb	Block	3	64	Second IDE Disk
/dev/console	Char	5	1	Console
/dev/null	Char	1	3	Null Device

Accessing I/O Device Drivers (3)

- Two related questions
  - How to name an I/O device?
  - How to gain access to the device driver routines?

- Access device driver routines of an I/O device
  - Thru the attributes contained in a device file
    - Major number
      - Locates the access information of the I/O device
    - Minor number
      - Identifies a particular instance of the given I/O device

Structure of Character Device Drivers



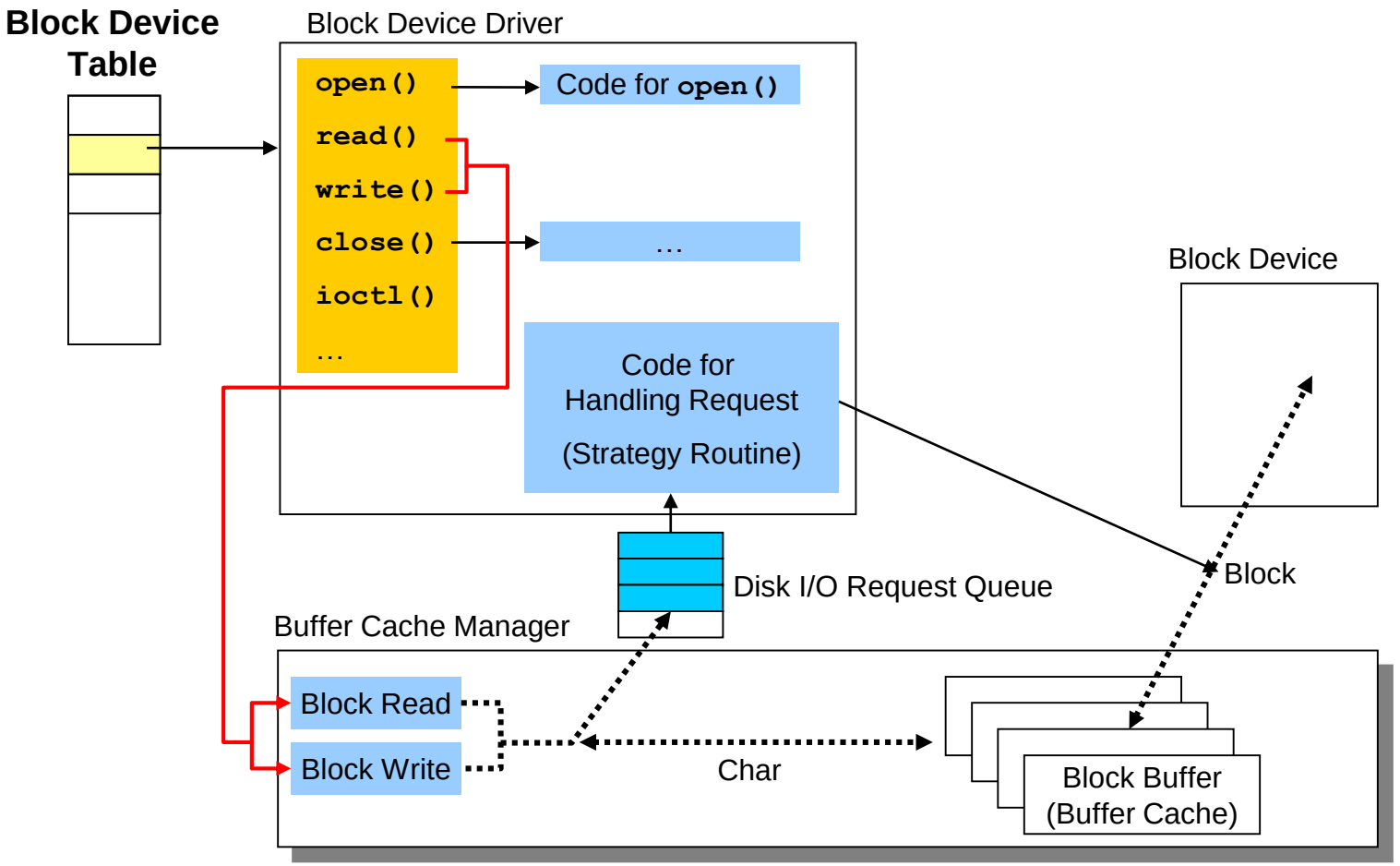
Character Device Drivers

- To applications
  - Device file = Regular file
- To file system
  - Regular file: read from or write to disk drive
  - Device file: invoke device driver operations
    - **`open()`**: initialize device
    - **`read()`**: device data to user buffer
    - **`write()`**: user buffer to device
    - **`close()`**: called when removed from device table
    - **`ioctl()`**: device-specific control
      - (Ex) baud rate change, access permission set

Block Device Drivers

- Differences with character device drivers
  - Actual transfer unit: *block* (some kilobytes)
    - Transfer between “device driver” and “device” cannot be character-based, rather block-based
- How to implement character oriented read/write?
  - Kernel provides general *block conscious* read/write functions
    - Translate character I/O to block I/O
  - Device driver only need to implement *handle block request*
    - Sometimes called as “*strategy routine*”

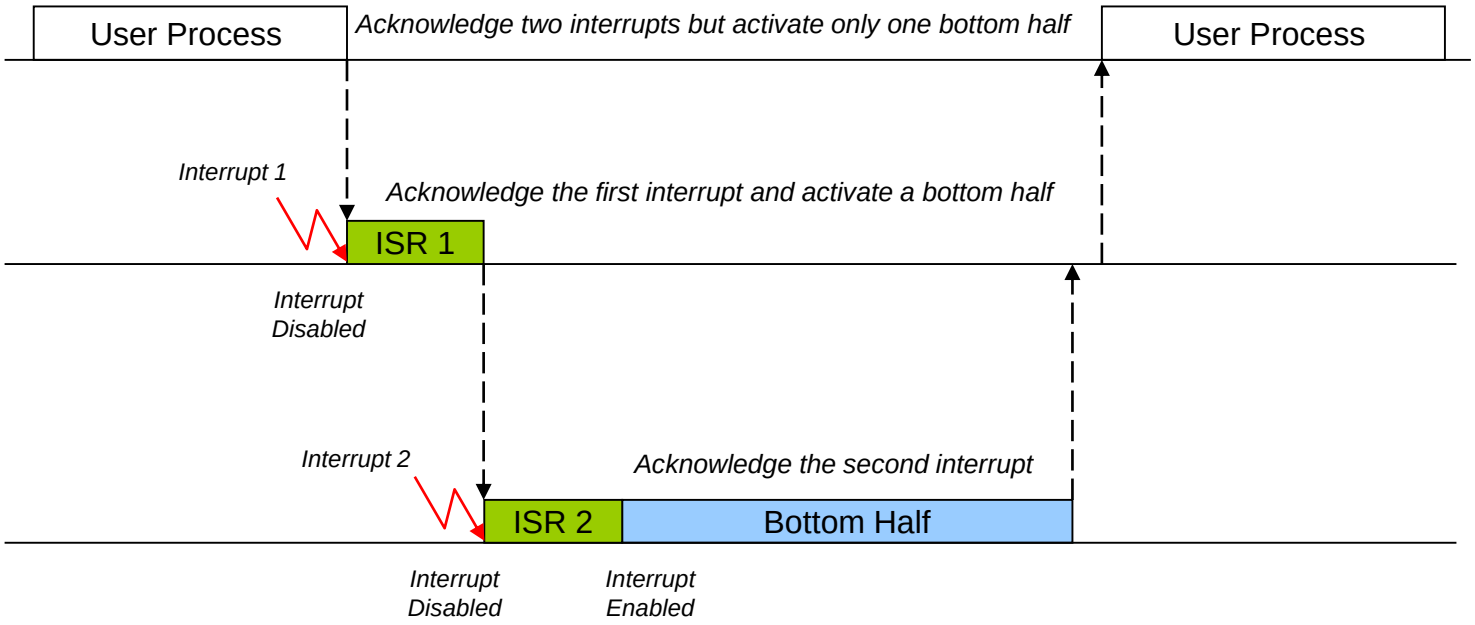
Structure of Block Device Drivers



Interrupt Handling in Device Drivers

- Register “interrupt handling routine” when opening
- Kernel invokes registered handler when interrupt occurs
- Linux interrupt handling (two-level processing)
  - Interrupt handler
    - High priority function (such as acknowledging to PIC)
    - Run with *interrupt disabled*
    - Invoked every time interrupt occurred
    - Marks a bottom half as active
  - Bottom half
    - Low priority function (such as transferring data from device)
    - Run with *interrupt enabled*
    - Execution may be deferred

Bottom Half


User Process
Acknowledge two interrupts but activate only one bottom half
User Process
Interrupt 1
Acknowledge the first interrupt and activate a bottom half
ISR 1
Interrupt
Disabled
Interrupt 2
Acknowledge the second interrupt
ISR 2
Bottom Half
Interrupt
Disabled
Interrupt
Enabled

General Device Driver Routines (1)

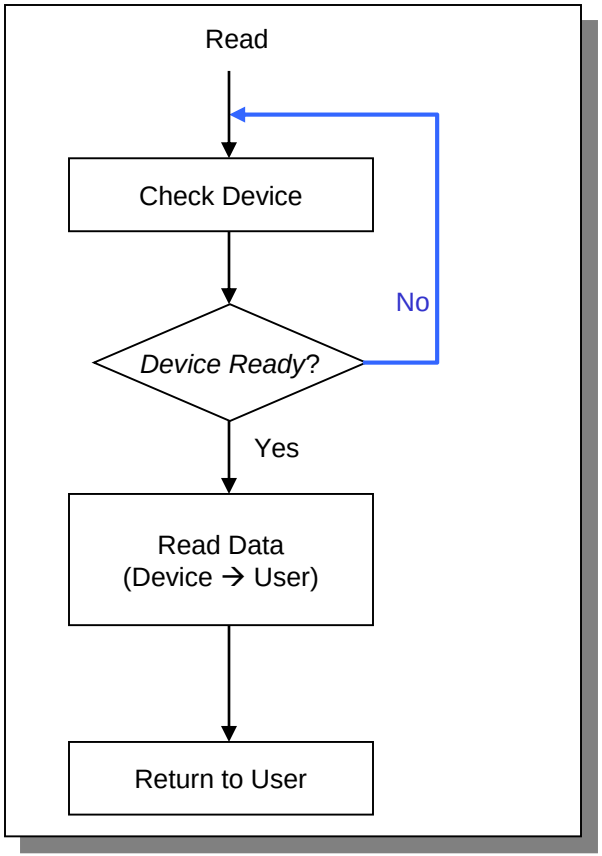
- Open
  - Reads minor number
  - Initializes the appropriate device
  - Initializes the device driver internal data structures
  - If needed, changes the file operation table (according to the minor number)
  - Increases the usage counter
  - If needed, registers the interrupt handler and enable IRQ
- Close
  - Frees dynamically allocated data structures
  - Decreases the usage counter
  - Un-registers the interrupt handler

General Device Driver Routines (2)

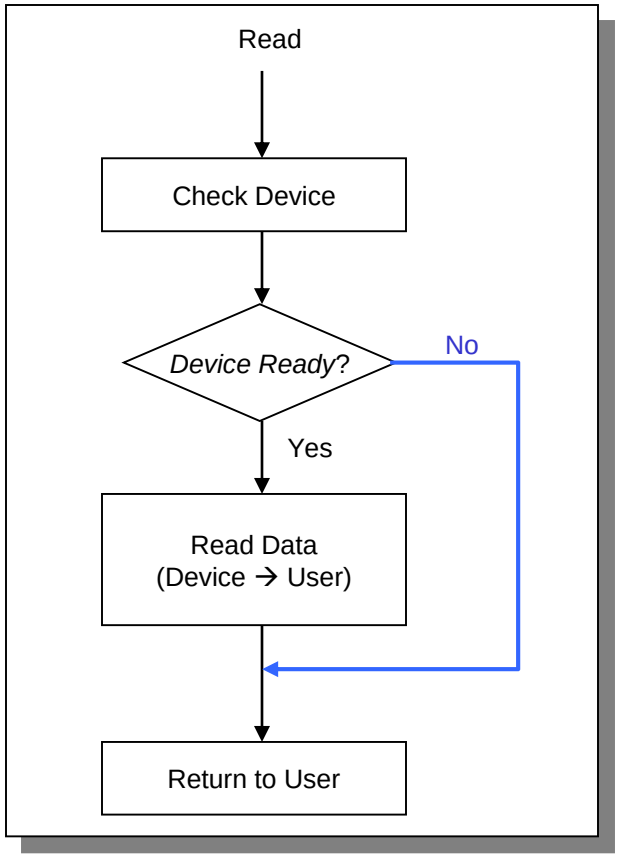
- Read/Write
  - Polling vs. interrupt-driven
    - Polling
      - Checks the device if data can be read or written
      - Usually used for character devices
    - Interrupt-driven
      - Sleeps the process until data can be read or written
      - Usually used for character and block devices
  - Blocking vs. non-blocking
    - Blocking
      - If data cannot be read or written, waits until ready
    - Non-blocking
      - If data cannot be read or written, immediately returns

General Device Driver Routines (3)

Character* Device, *Polling* and *Blocking I/O

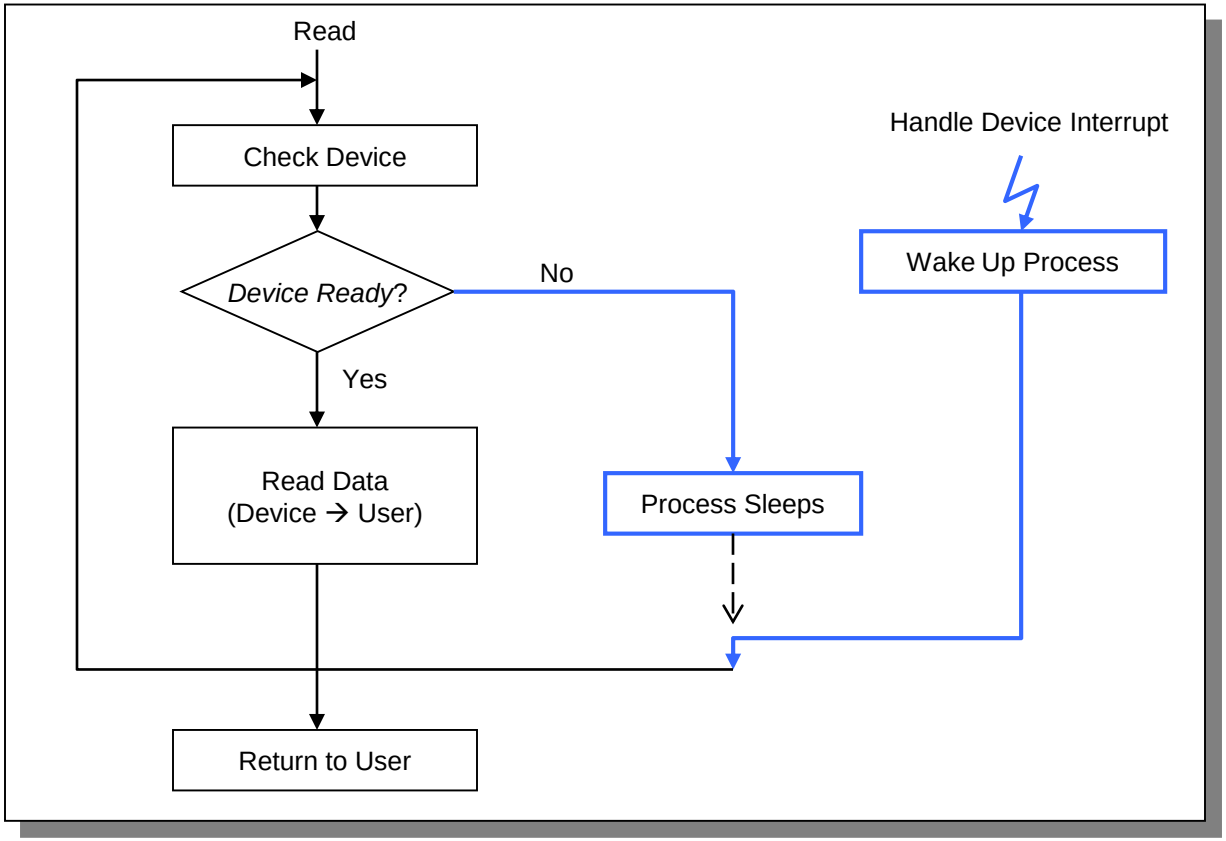


Character* Device, *Polling* and *Non-Blocking I/O



General Device Driver Routines (4)

***Character* Device, *Interrupt-Driven* and Blocking I/O**



General Device Driver Routines (5)

Block* Device, *Interrupt-Driven,* and *Blocking I/O