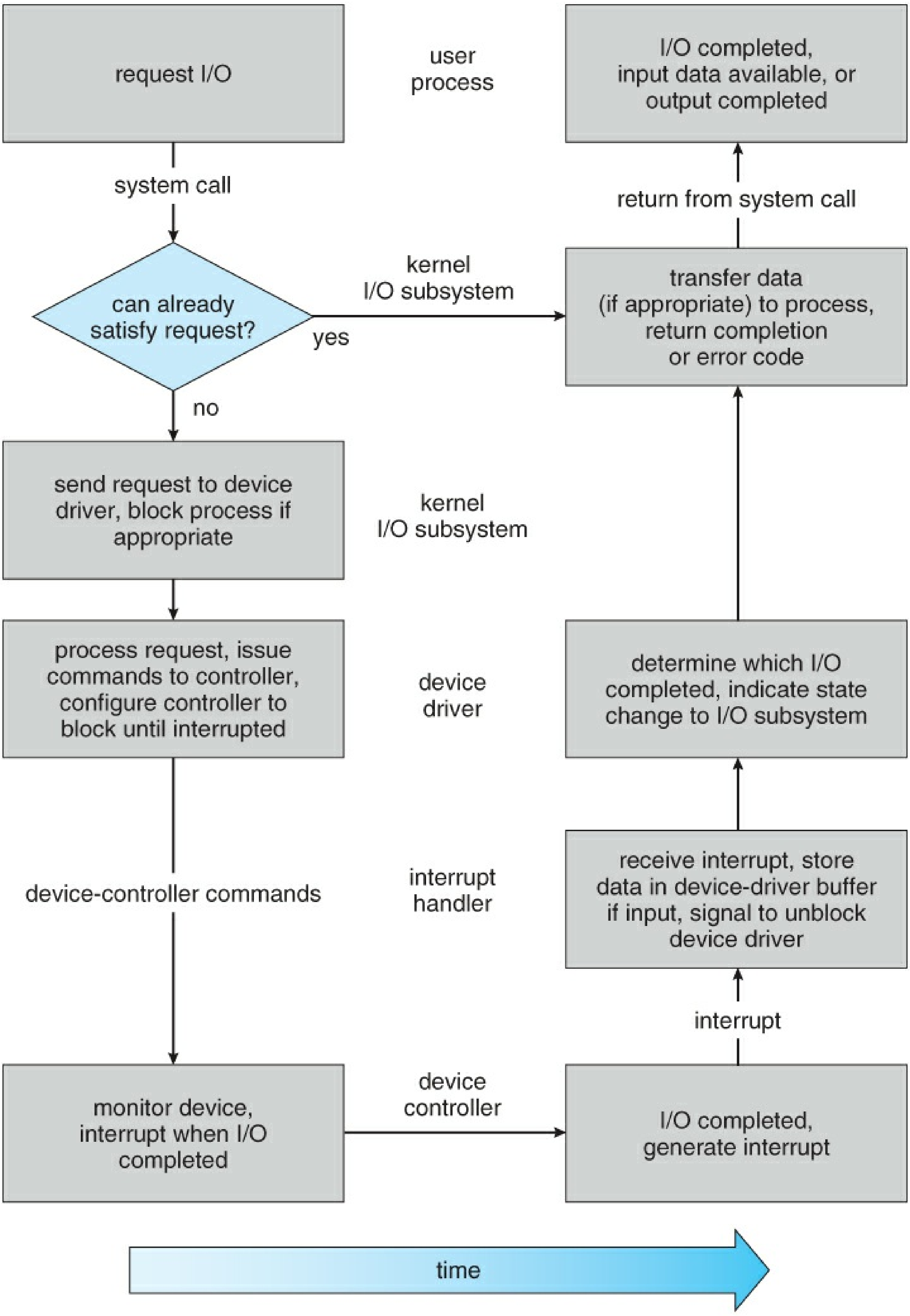
Device Drivers

What are device drivers?

- CPU is connected to I/O Bus which is connected to many hardware controllers.
- Each hardware controller has its own control and status registers (CSRs) and these differ between devices. The CSRs are used to start and stop the device, to initialize it and to diagnose any problems with it.
- The software that handles or manages a hardware controller is known as a device driver

Drivers in Linux

- Linux provides a file interface for all the devices present in the system. They can be opened, closed, read and written using the same, standard, system calls that are used to manipulate files. ex - first IDE disk in the system is represented by /dev/hda.
- Linux supports three types of hardware device: character, block and network.
- Drivers can either poll the device for the completion of a request or use interrupts. The device drivers requests ownership of an interrupt number and registers an interrupt handler for device when the driver is initialized.
- Device drivers allocate kernel, non-paged, memory to hold their data.
- User Space processes call system calls provided by the Operating System for I/O operations. These system calls are independent of the actual device and generic enough to support wide variety of devices. The OS provides system call API based on the type of devices like block or character device, and Direct Access or Sequential Access.
- The Operating System then calls the API's proveded by the device drivers. These device drivers are third party software and know how to interact with specific device.


request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
kernel I/O subsystem
yes
transfer data (if appropriate) to process, return completion or error code
no
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

Note that data has to be copied from the kernel space to user space after device has completed the I/O request

Programmed I/O

Simple I/O through polling -

```
While (STATUS == BUSY)
; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
(Doing so starts the device and executes the command)
While (STATUS == BUSY)
; // wait until device is done with your request
```

Polling is good for fast devices. For slow devices, this is inefficient due to CPU cycles wasted in polling.

- Instead of polling the device repeatedly, the OS can issue a request, put the calling process to sleep, and context switch to another task. When the device is finally finished with the operation, it will raise a hardware interrupt.
- Interrupt causes Interrupt handler registered by the device driver which then wakes the process which requested the I/O (puts the process in the ready queue)

Read Interrupt Coalescing

DMA

- Stands for Direct Memory access
- To transfer data to the device, the OS would program the DMA engine by telling it where the data lives in memory, how much data to copy, and which device to send it to
- DMA controlled takes care of the rest.
- When DMA is complete, DMA controller raises an interrupt.
- Obvious advantages over PIO.

How does the Processor communicate with the devices?

Two ways.

First is Port Mapped I/O. x86 provides the `in` and `out` instructions. Specify the register with the data in it and the port (What are ports? Ports name the device. How?). These instructions are privileged and can only be executed by the OS.

Second is Memory Mapped I/O. In this the hardware makes device registers available as if they were memory locations. Each I/O device monitors the CPU's address bus and responds to any CPU access of an address assigned to that device, connecting the data bus to the desired device's hardware register. In this case, CPU instructions used to access memory can also be used for accessing devices.

References

- http://www.tldp.org/LDP/tlk/dd/drivers.html
- http://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf
- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/13_IOSystems.html