PRINCIPLES OF OPERATING SYSTEMS

LECTURE 34

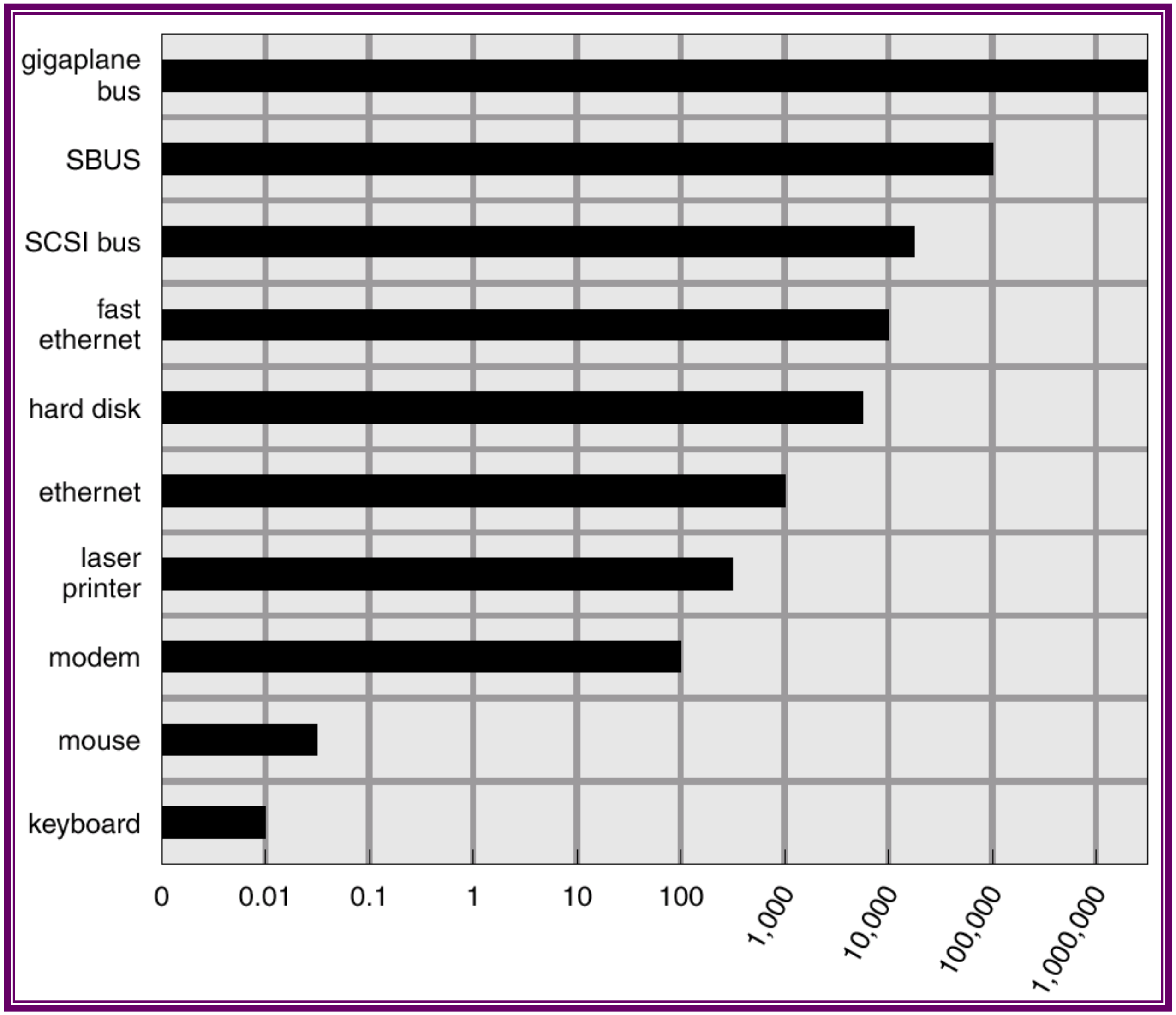
KERNEL, TRANSFORMING I/O REQUESTS & PERFORMANCE ISSUES

Kernel I/O Subsystem

- See A Kernel I/O Structure slide - Fig 13.6
- Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness

- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch **- de-couples application from device action**
  - To cope with device transfer size mismatch
  - To maintain "copy semantics" **- guarantee that the version of data written to device from a buffer is identical to that which was there *at the time of the "write call"* - even if on return of the system call, the user modifies buffer - OS copies data to kernel buffer before returning control to user.**
  - **Double or "ping-pong" buffers - write in one and read from another - decouples devices and applications**
    … idea can be extended to multiple buffers accesses in a circular fashion

Sun Enterprise 6000 Device-Transfer Rates



Kernel I/O Subsystem - (continued)

- Caching - fast memory holding ***copy*** of data
  - Always just a copy
  - Key to performance
  - **How does this differ from a buffer?**
- Spooling - **a *buffer*** hold**ing** output**/(input too)** for a device
  - If device can serve only one request at a time
  - **Avoids queuing applications making requests.**
  - **Data from an application is saved in a unique file associated with the application AND the particular request. Could be saved in files on a disk, or in memory.**
  - Example: Printing
- Device reservation - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock - **why?**

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports
- **CRC checks - especially over network transfers of a lot of data, for example video in real time.**

Kernel Data Structures

- Kernel keeps ***state info*** for I/O components, including ***open file tables***, network connections, character device state
  - ☞ **used by device drivers in manipulating devices and data transfer, and in for error recovery**
  - ☞ **data that has images on the disk must be kept in synch with disk copy.**
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- Some use object-oriented methods and message passing to implement I/O
  - ☞ **Make data structures object oriented classes to encapsulate the low level nature of the “device” - UNIX provides a seamless interface such as this.**

UNIX I/O Kernel Data Structure

Refer to chapter 11 and 12 on files

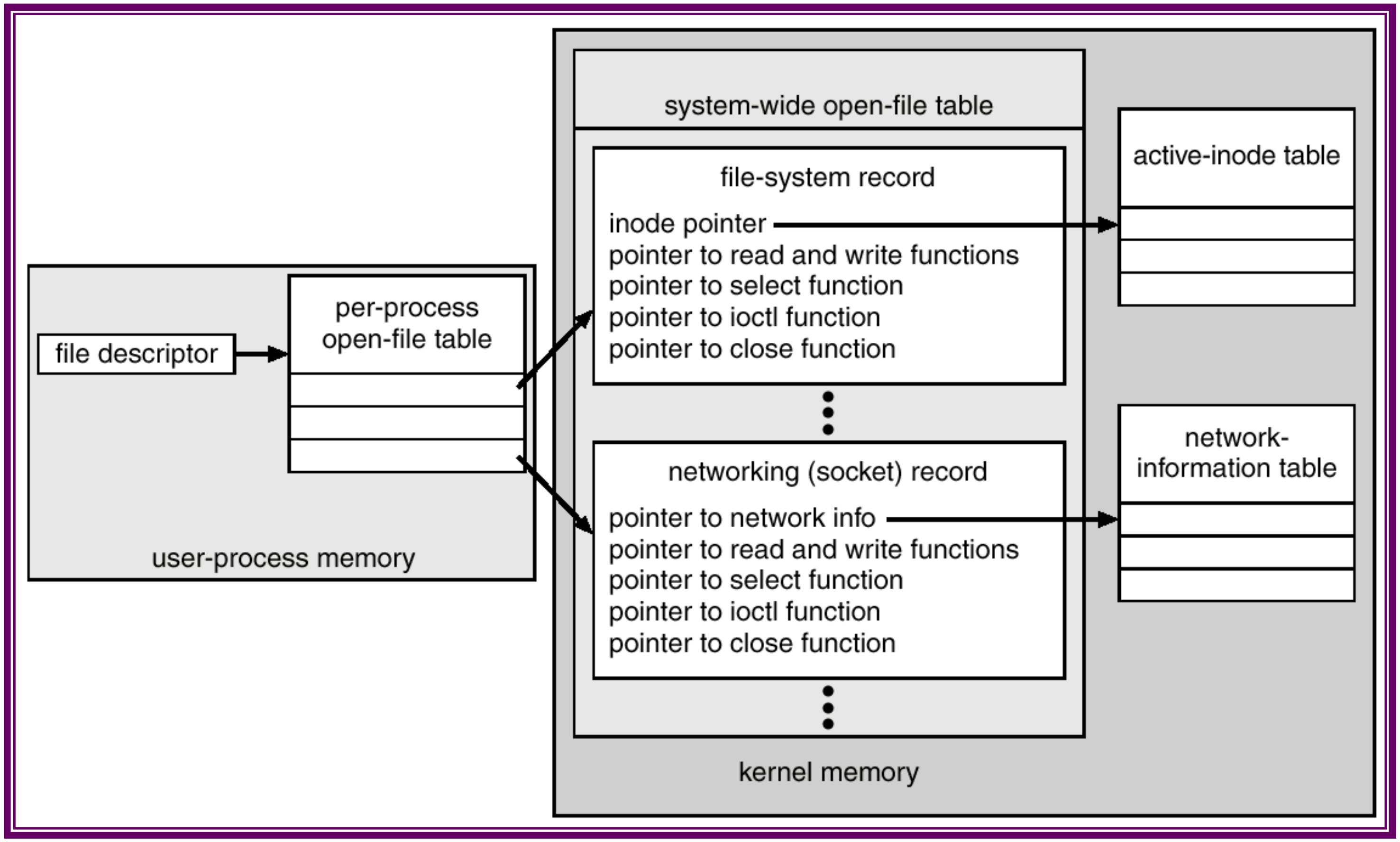


Fig. 13.9

Mapping I/O Requests to Hardware Operations

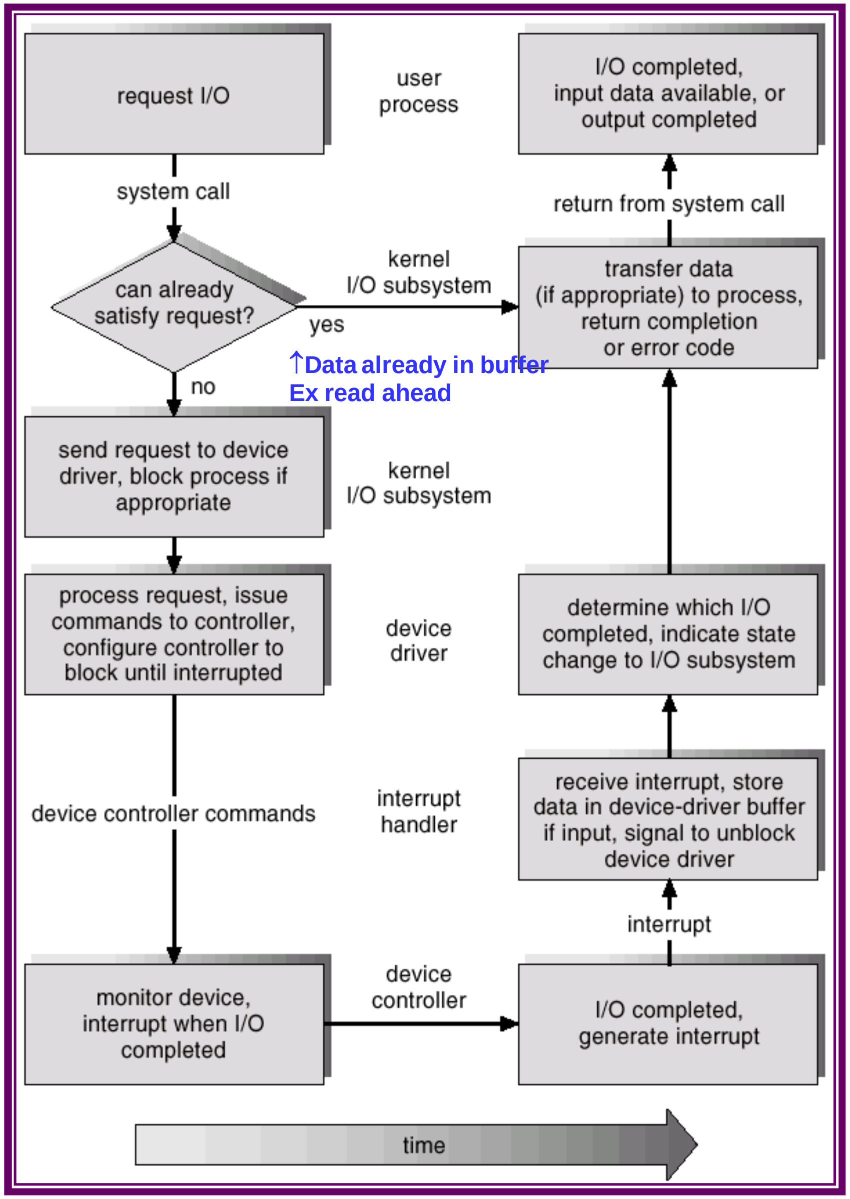
- Consider reading a file from disk for a process:

  How is connection made from file-name to disk controller:

  - 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

- **See the 10 step scenario on pp. 479-481 (Silberschatz, 6th ed.) for a clear description.**

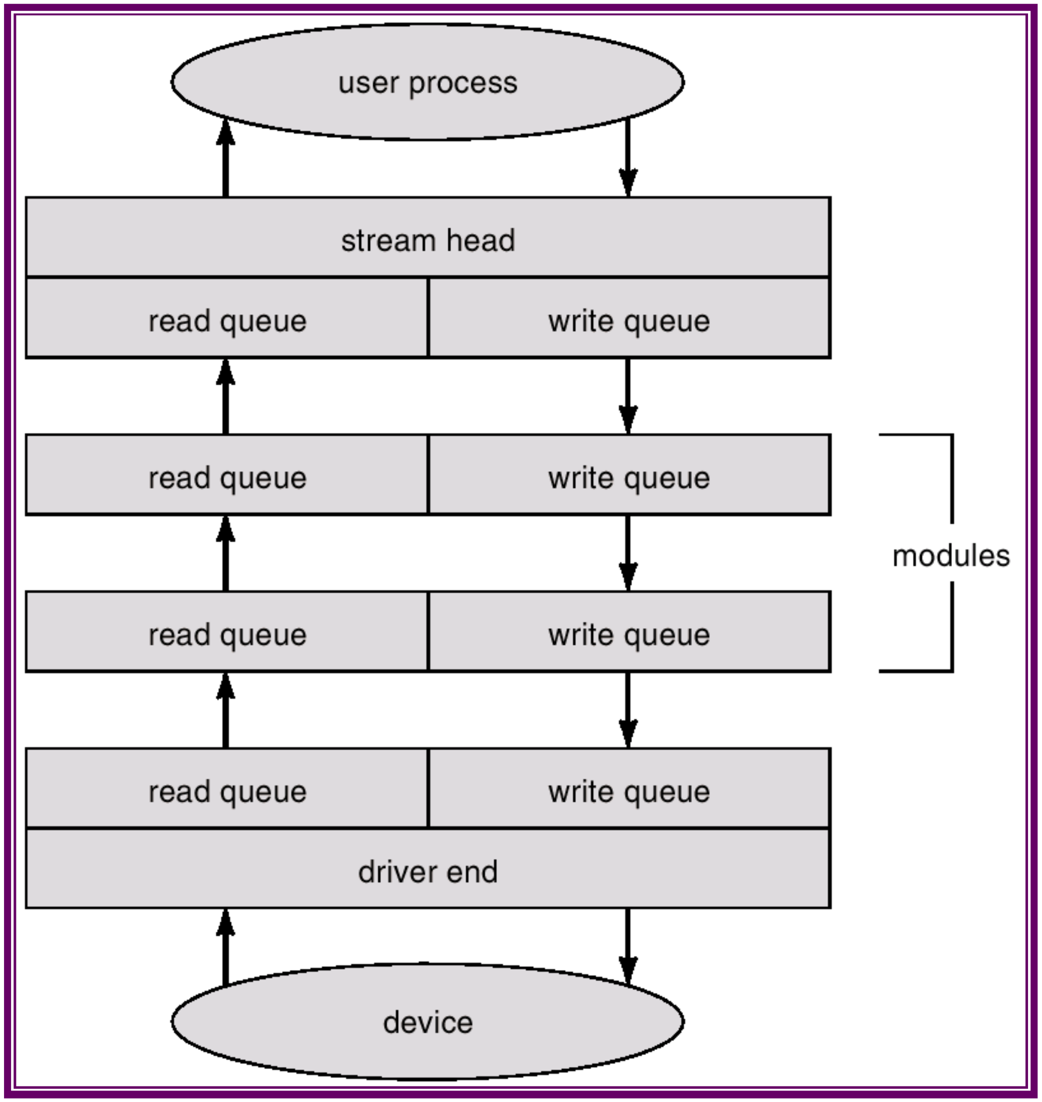
Life Cycle of An I/O Request



STREAMS (?)

- **STREAM** – a full-duplex communication channel between a user-level process and a device
- 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

The STREAMS Structure

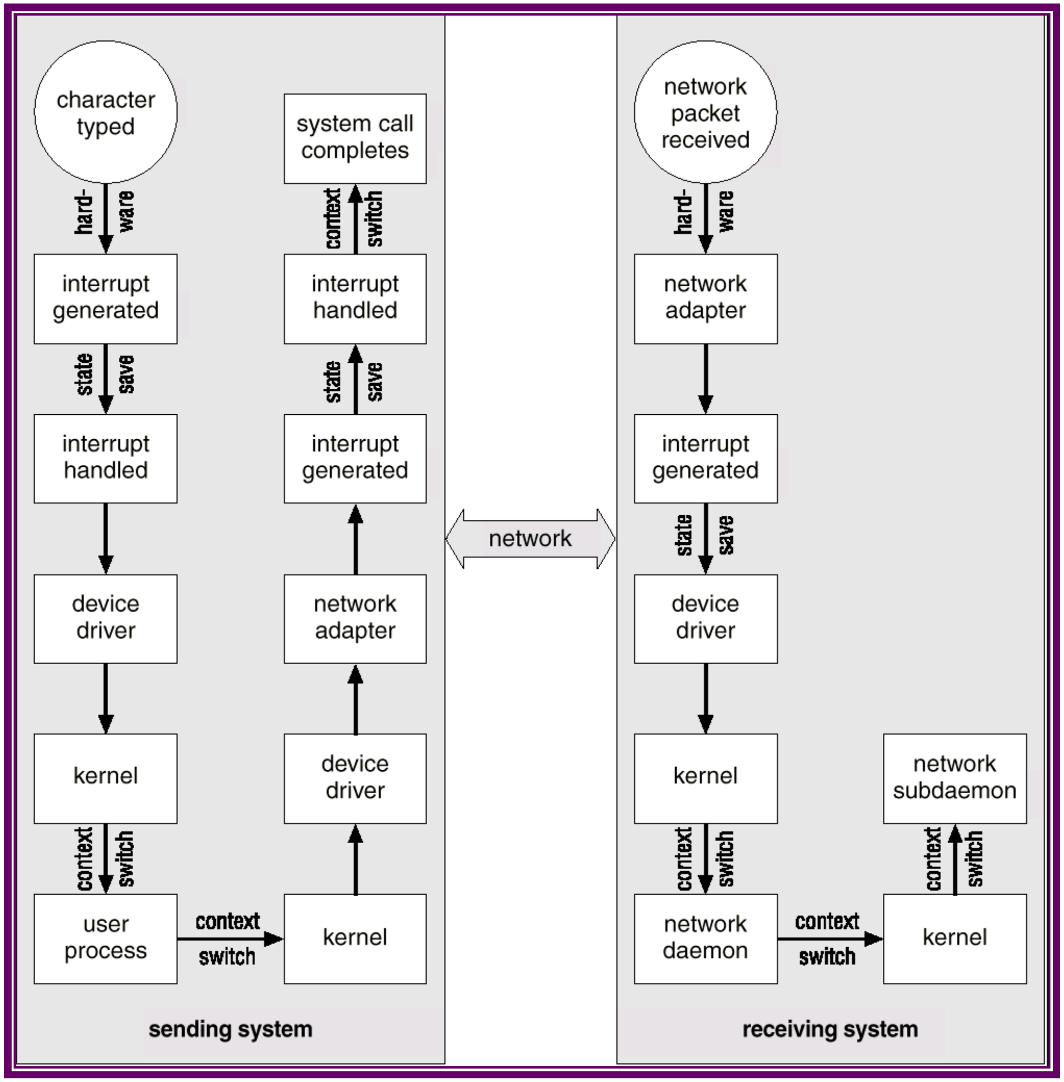


Performance

- I/O a major factor in system performance:

  - Places demands on CPU to execute device driver, kernel I/O code
    - resulting in context switching
    - interrupt overhead
  - Data copying **- loads down memory bus**
  - Network traffic especially stressful
  - **See bulleted list on page 485 (Silberschatz, 6th ed.)**
- **Improving Performance**
  See bulleted list on page 485 (Silberschatz, 6th ed.)
  - Reduce number of context switches
  - Reduce data copying
  - Reduce interrupts by using large transfers, smart controllers, polling
  - Use DMA
  - Move proccessing primitives to hardware
  - Balance CPU, memory, bus, and I/O performance for highest throughput

Intercomputer Communications- omit for now



Device-Functionality Progression

Where should I/O functionality be implemented? Application level … device hardware

Decision depends on trade-offs in the design layers: