PES UNIVERSITY ONLINE

OPERATING SYSTEMS

I/O Management, System Protection and Security

Kakoli Bora

Department of Computer Science

OPERATING SYSTEMS

I/O Systems - Transforming I/O Requests to Hardware Operations

Kakoli Bora

Department of Computer Science

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Transforming I/O Requests to Hardware Operations

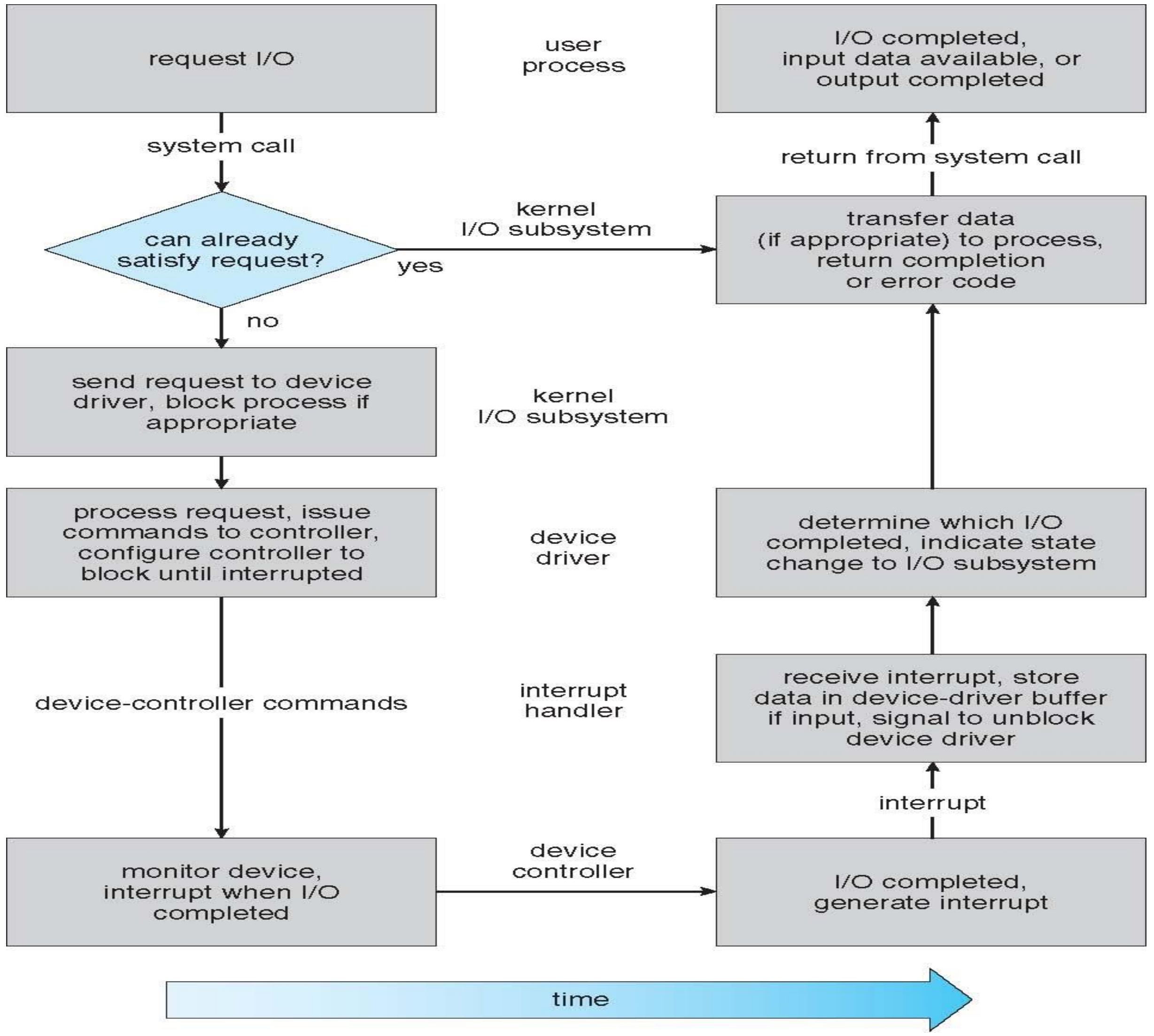
- Consider reading a file from disk for a process:
  - Determine device holding file
    - Within a disk, the file system maps from the file name through the file-system directories to obtain the space allocation of the file.
  - Translate name to device representation
    - In UNIX, the name maps to an inode number, and the corresponding inode contains the space-allocation information.
    - UNIX has a mount table that associates prefixes of path names with specific device names.
    - When UNIX looks up this name in the file-system directory structures, it finds not an inode number but a <major, minor> device number.

Transforming I/O Requests to Hardware Operations (Cont.)

- The major device number identifies a device driver that should be called to handle I/O to this device.
- The minor device number is passed to the device driver to index into a device table.
  - The corresponding device-table entry gives the port address or the memory-mapped address of the device controller.

- Physically read data from disk into buffer
- Make data available to requesting process
- Return control to process

Life Cycle of An I/O Request


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

Transforming I/O Requests to Hardware Operations

1. A process issues a blocking read() system call to a file descriptor of a file that has been opened previously.

2. The system-call code in the kernel checks the parameters for correctness. In the case of input, if the data are already available in the buffer cache, the data are returned to the process, and the I/O request is completed.

3. Otherwise, a physical I/O must be performed. The process is removed from the run queue and is placed on the wait queue for the device, and the I/O request is scheduled. Eventually, the I/O subsystem sends the request to the device driver. Depending on the operating system, the request is sent via a subroutine call or an in-kernel message.

Transforming I/O Requests to Hardware Operations

4. The device driver allocates kernel buffer space to receive the data and schedules the I/O. Eventually, the driver sends commands to the device controller by writing into the device-control registers.

5. The device controller operates the device hardware to perform the data transfer.

6. The driver may poll for status and data, or it may have set up a DMA transfer into kernel memory. We assume that the transfer is managed by a DMA controller, which generates an interrupt when the transfer completes.

7. The correct interrupt handler receives the interrupt via the interrupt vector table, stores any necessary data, signals the device driver, and returns from the interrupt.

Transforming I/O Requests to Hardware Operations

8. The device driver receives the signal, determines which I/O request has completed, determines the request's status, and signals the kernel I/O subsystem that the request has been completed.

9. The kernel transfers data or return codes to the address space of the requesting process and moves the process from the wait queue back to the ready queue.

10. Moving the process to the ready queue unblocks the process. When the scheduler assigns the process to the CPU, the process resumes execution at the completion of the system call.

THANK YOU

Kakoli Bora
Department of Computer Science Engineering
K_bora@pes.edu