

CS241 System Programming

Input/Output II

Klara Nahrstedt

Lecture 19

3/3/2006



Content

- Administrative announcements
- Software I/O principles
- Device Drivers

- Summary

Administrative

- Homework 1 is posted – due 3/6/06 – Individual effort
- Quiz 5 – 3/3/06

Device-Independent I/O Software (1)

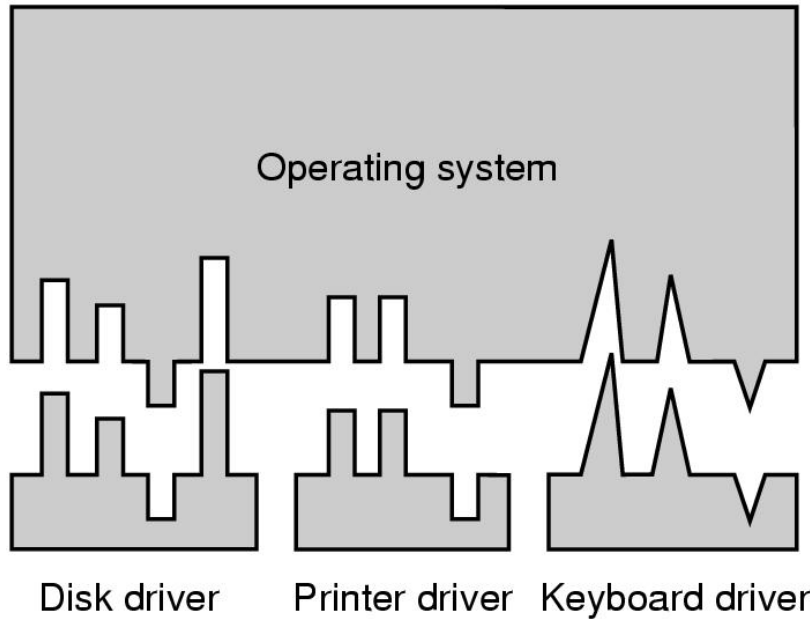
Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicate devices
Providing a deice-independent block size

Functions of the device-independent I/O software

Uniform Interfacing for Device Drivers

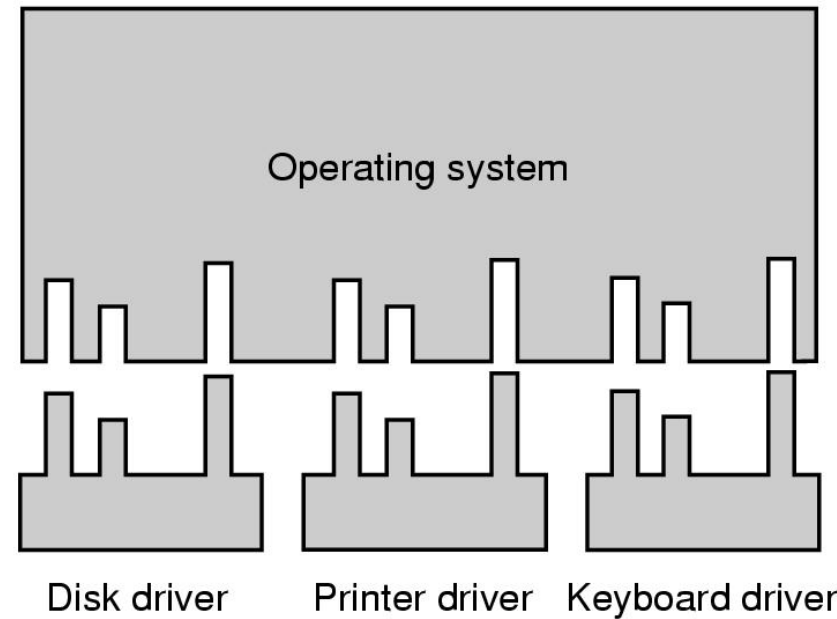
- **Goal:** how to make all IO devices and drivers look similar
- **Problem:** interface between drivers and the rest of OS might differ, which might mean that each new driver would require a lot of new programming effort
- **Solution:** There are specifications which functions and kernel calls to use in device drivers to make the drivers pluggable. Not all devices are absolutely identical, but usually there are only small number of device types.
- **Example:** block and character devices, but they also have many functions common

Uniform Interface



(a)

(a) Without a standard driver interface



(b)

(b) With a standard driver interface

Naming of I/O devices

- The device-independent software takes care of mapping symbolic device names onto the proper driver.
- In UNIX/Linux each device is modeled as a special file and therefore easier integrated into the overall file system
 - Each I/O device is assigned a path name, usually /dev
 - Example: the symbolic device name /dev/disk0 uniquely specifies the disk, /dev/net – network, /dev/lp – printer
 - These special files can be accessed the same way as any other files

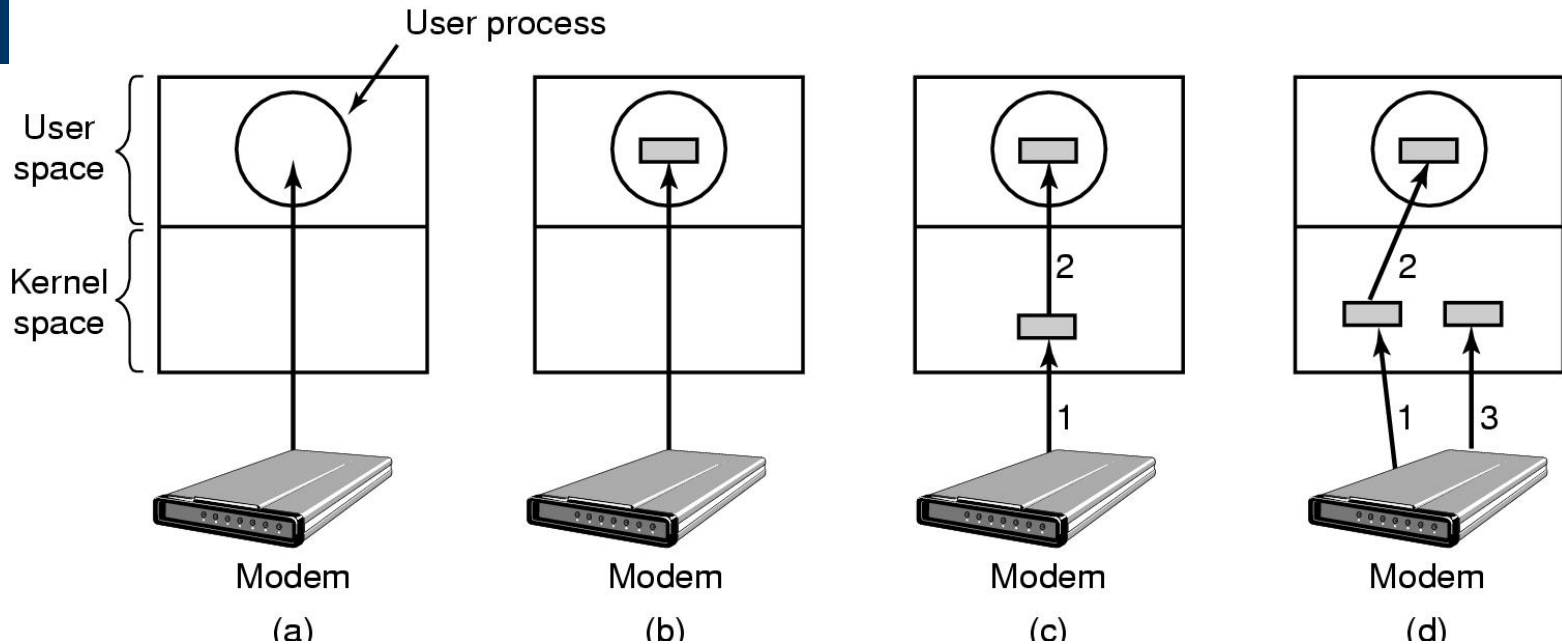
Buffering

- Buffer is a memory area that stores data while they are transferred between two devices or between a device and an application.
- **Reasons of buffering:**
 - cope with speed mismatch between the producer and consumer of a data stream - use *double buffering*
 - adapt between devices that have different data-transfer sizes
 - support of copy semantics for application I/O - application writes to an application buffer and the OS copies it to the kernel buffer and then writes it to the disk.
 - Unbuffered input strategy is ineffective, as the user process must be started up with every incoming character.
 - Buffering is also important on output.
- **Caching**
 - cache is region of fast memory that holds copies of data and it allows for more efficient access.
- Difference?

Buffering Issues

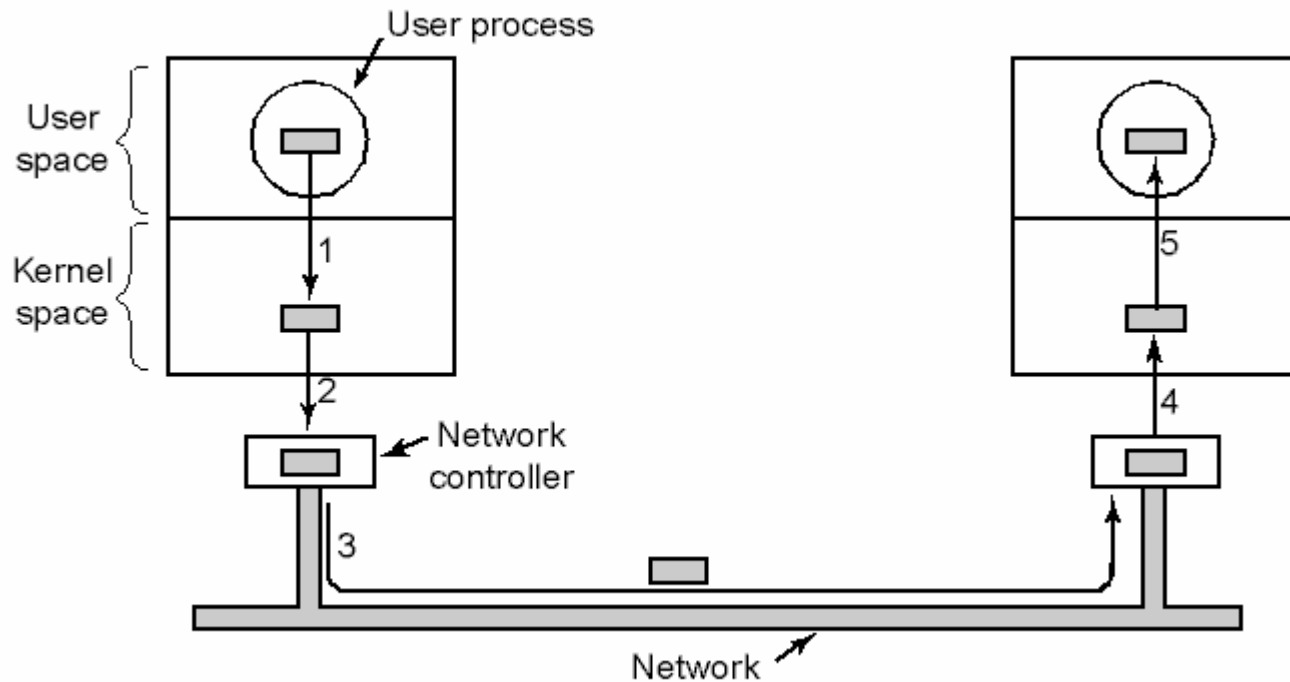
- Pros:
 - Unbuffered input strategy is ineffective, as the user process must be started up with every incoming character.
- Buffering in user space
 - if the buffer is paged out, where should the character be put? One solution would be to pin the pages into the real memory, but this could degrade performance if many processes start to lock pages in memory.
- Buffering in kernel followed by copying to user space:
 - far more efficient than the previous methods, but again the problem occurs if the kernel buffer becomes full and the user buffer page is paged out. In this case again there is no space to put the incoming data.
- Double buffering in the kernel
 - as long as one kernel buffer takes incoming data, the other buffer is copied to the user space.
- Note: if data gets buffered too many times, performance suffers.

Buffering strategies



- (a) Unbuffered input
- (b) Buffering in user space
- (c) Buffering in the kernel followed by copying to user space
- (d) Double buffering in the kernel

Buffering in multiple places



Networking may involve many copies

Error Reporting

- Programming IO Errors
 - these errors occur when a process asks for something impossible (e.g., write to an input device such as keyboard, or read from output device such as printer).
- Actual IO Errors
 - errors occur at the device level (e.g., read disk block that has been damaged, or try to read from video camera which is switched off)
- The device-independent IO software detects the errors and responds to them by reporting to the user-space IO software.

Allocating and Releasing Dedicated Devices

- Some devices can be used only by single process at any given time
 - e.g., CD-ROM recorders, some audio devices
- Two approaches:
 - Perform open on the special file for devices directly. If device is not available, open fails.
 - A process requests a device. If it is not available, the process is blocked, instead failing, and waits until the device become available.

Device Independent Block Size

- Different disks have different sector sizes.
- The device independent IO software should hide this fact and provide uniform block size to higher layers.
 - introduce the same logical block at the user-space IO level and do the mapping between the logical block size and the underlying several sectors in the device independent IO software.

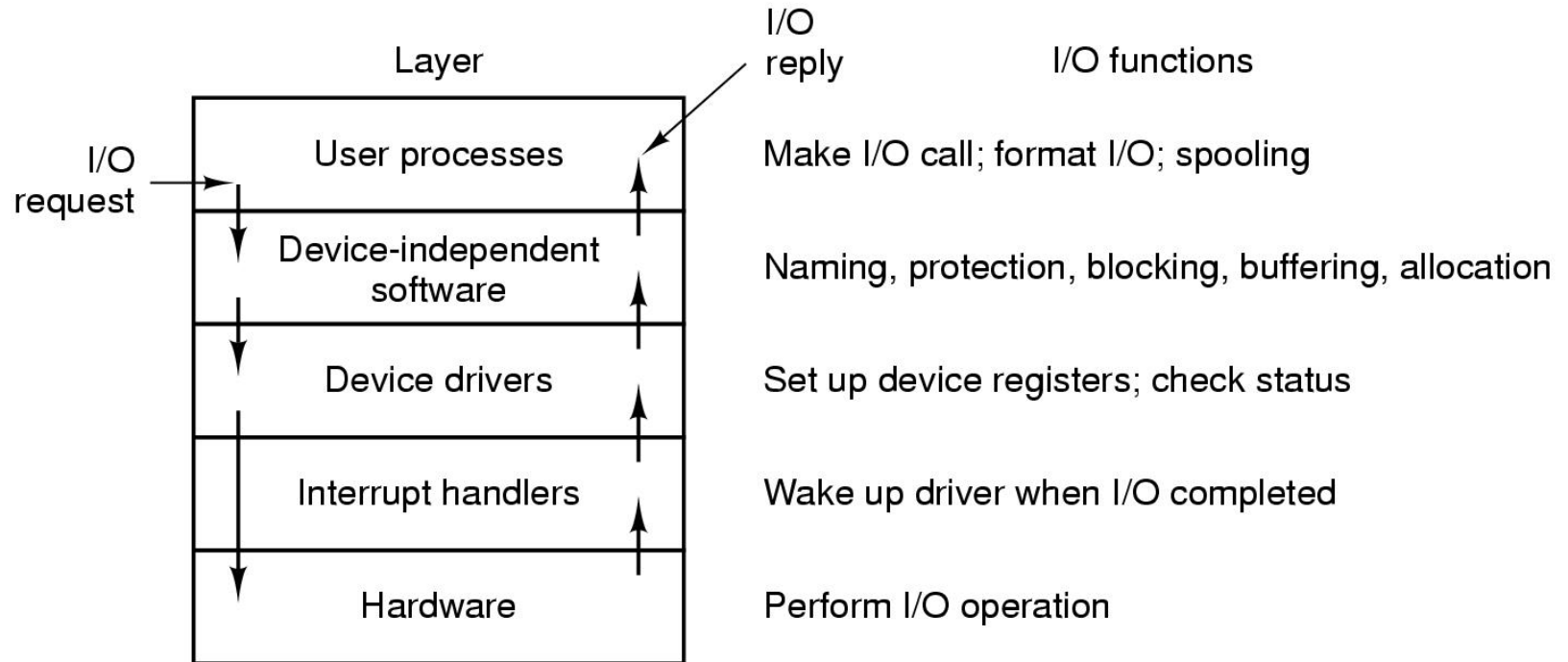
User-Space I/O Software

- Applications in user space call IO system calls to access IO.
- IO system calls are normally made by library procedures.
- Steps to perform an IO system call:
 - User-space library calls the device-independent IO software
 - Device-independent I/O software calls the corresponding device driver
 - Device driver accesses the device controller and through it the corresponding device.

Spooling

- Another way to do I/O is the **spooling system**.
- Spooling is an approach to deal with IO devices in a multiprogramming system
 - Example: printer (spooled device)
- Spooling includes a special process, called *daemon*, and special directory, called *spooling directory*.
- Example: To print a file, the printing process creates the entire file, puts it in the spooling directory, and the daemon then takes out the file and prints it.

I/O Software



Layers of the I/O system and the main functions of each layer

Summary

- Consider carefully the I/O software principles
 - Interrupt handlers device drivers are low layers of I/O software
 - Difficult to program, a lot of synchronization is needed
 - Device-specific
 - Device-independent I/O software makes life easier
 - Exact boundary between device drivers and device independent I/O software is system dependent
 - Functions in device-independent I/O software are
 - Buffering, error reporting, uniform interfaces, etc.