Device Drivers

U.Va.
Department of Computer Science

Overview

- Device drivers implement a uniform interface to varied hardware devices
  - This allows the rest of the kernel to be hardware-independent
- Device drivers convert high-level function calls, like read(), into specific hardware commands
  - These commands are typically reads/writes to specific addresses in the physical address space that are assigned to each device
  - Long-latency commands signal completion via interrupts
- Note that device drivers may be layered
  - eg, a command may first be handled by a device driver that manages bus transactions, then a device driver that controls the specific device

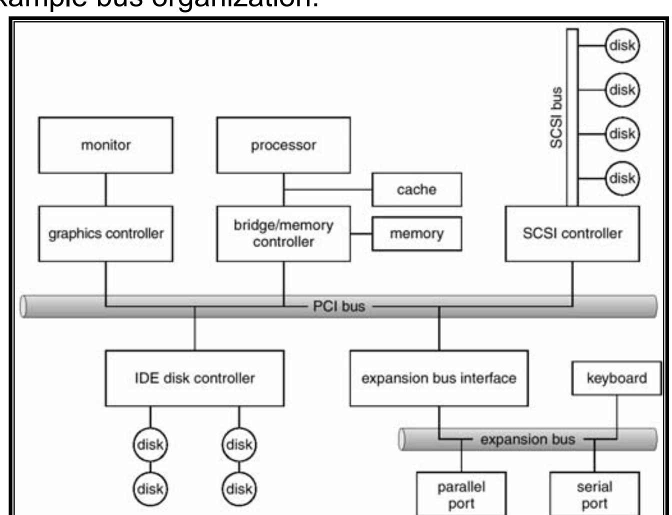
U.Va.
Department of Computer Science

Memory-Mapped I/O

- Regions of the physical address space are assigned to each device on the bus
  - These addresses are usually only accessible by the kernel
  - Multiple devices may share a single controller

Typical Bus Organization

- Example bus organization:



Source: Silberschatz, Galvin and Gagne ©2002,
http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

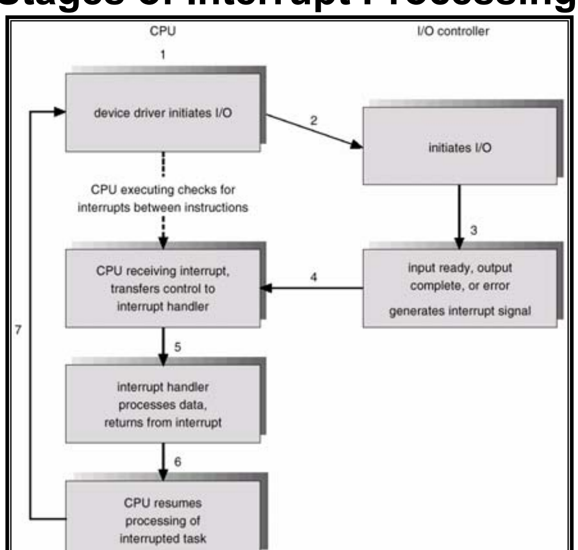
Memory-Mapped I/O, cont.

I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

Source: Silberschatz, Galvin and Gagne ©2002, http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

U.Va.
Department of Computer Science

Stages of Interrupt Processing



Source: Silberschatz, Galvin and Gagne ©2002, http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

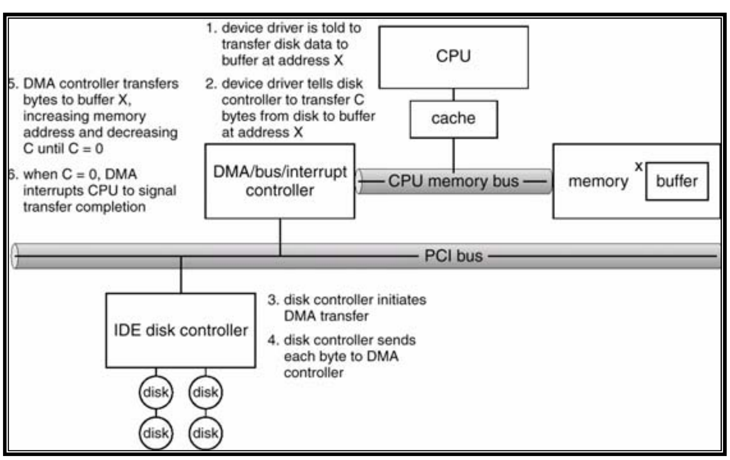
U.Va.
Department of Computer Science

Sample Interrupts

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19Ð31	(Intel reserved, do not use)
32Ð255	maskable interrupts

Source: Silberschatz, Galvin and Gagne ©2002,
http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

Stages of DMA

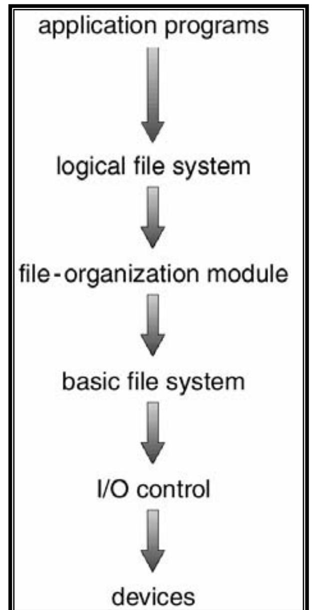


Source: Silberschatz, Galvin and Gagne ©2002,
http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

U.Va.
Department of Computer Science

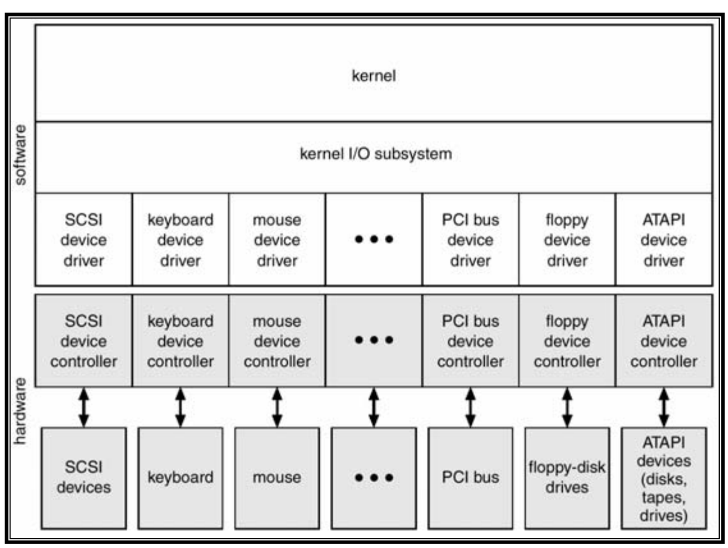
Device Infrastructure

- Hierarchy of abstraction
  - Device-dependence is confined to I/O control and device driver layers



Source: Silberschatz, Galvin and Gagne ©2002, http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

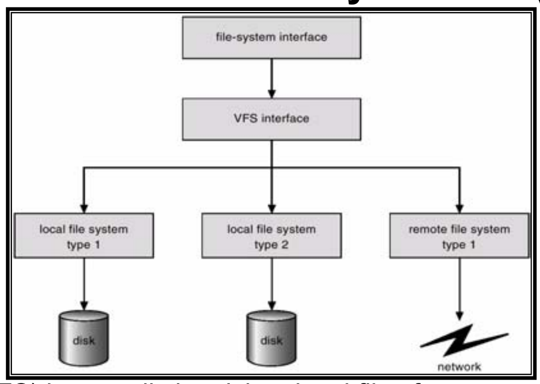
Kernel I/O Structure



Source: Silberschatz, Galvin and Gagne ©2002, http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

More abstraction: Virtual File System

- Even the file system has several layers

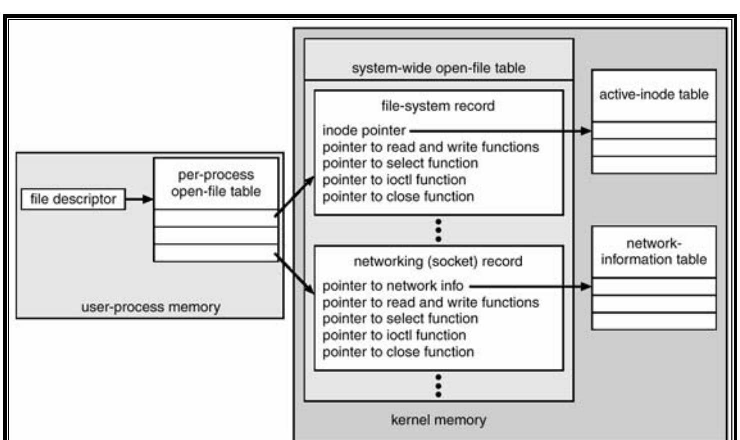


Source: Silberschatz, Galvin and Gagne ©2002, http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types.
  - The VFS activates file-system-specific operations to handle local requests according to their file-system types.
  - Calls the NFS protocol procedures for remote requests.

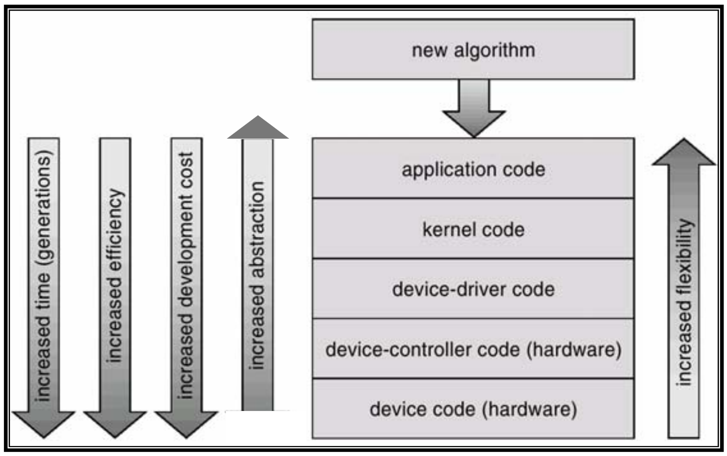
U.Va.
Department of Computer Science

Unix File-Access Control Structure



Source: Silberschatz, Galvin and Gagne ©2002, http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

U.Va.
Department of Computer Science



Modified from: Silberschatz, Galvin and Gagne ©2002, http://cs-www.cs.yale.edu/homes/avi/os-book/osc/index.html

Device-Independent System Call/Interrupt Handling

```
Func_i(…) ->  dev_func_i(devID, …) {
                 // Processing common to all devices
                 ….
                 switch(devID) {
                      case dev0: dev0_func_i(…);
                                 break;
                      case dev1: dev1_func_i(…);
                                 break;
                      …
                 }
                 // Processing common to all devices
              }
```

Adapted from G. Nutt, "Operating Systems", 3rd ed., ©2004

Example: Linux

```
struct file_operations{
   loff_t (*llseek) (struct file *, loff_t, int);
   ssize_t (*read) (struct file *, char *, size_t, loff_t *)
   …
}

foo_init() {
   …
  struct file_operations foo_fops = {
        NULL;         /* llseek – not used */
        foo_read;     /* read function */
        …
  }
}
```

U.Va.
Department of Computer Science

Example Device Driver Internals

- Interrupt handler for keyboard
  - (see http://www.ibiblio.org/navigator-bin/navigator.cgi?drivers/char/keyboard.c)
    - On screen

U.Va.
Department of Computer Science

Review: Stages in Processing an I/O Call

- Application makes call to C library function, eg fscanf()
- C library translates this into a system call, eg read()
- Execute syscall instruction
- Trap vector processes file name, determines which device is being accessed
  - This may entail additional disk accesses
- VFS consulted
- Appropriate local file system is consulted (or NFS access)
- File name -> inode -> block number
- Read(block num) call to device driver
  - Device driver should check input validity, avoid buffer overflow
  - Device driver checks device status; if busy, request is queued
- Read command issued over bus
  - This may actually entail a series of back-and-forth bus transactions
- Hardware controller performs read
- Interrupt generated when read completes
- Interrupt handler calls appropriate handler func in device driver

U.Va.
Department of Computer Science

Interrupt Processing

- Interrupt hardware saves some registers, indexes kernel's interrupt vector
- Kernel can ignore interrupt or proceed
- If proceeding, kernel typically temporarily suspends interrupts
- Kernel saves registers and sets up context for the interrupt service procedure
  - If a user process was running, this requires a context switch, TLB/MMU setup, and creation of a stack
- Run the "top half" of the interrupt service procedure
- Re-enable interrupts
- Finish interrupt servicing
- This may make a process runnable
- Choose which process to run
- Call kernel scheduler

U.Va.
Department of Computer Science

Re-entrant Code

- Note that an interrupt may be received at any time unless interrupts are disabled
  - While kernel is running
  - While device driver is handling a system call
  - While device driver is handling an interrupt
- Disabling interrupts briefly allows kernel/drivers to do minimal work necessary to maintain a coherent state, then interrupts can be re-enabled

U.Va.
Department of Computer Science

More Complexity: Hot-Swappable Devices

- Some devices can be installed or removed while computer runs, eg USB devices
- Either action generates an interrupt
- On an install, kernel must find, (install if necessary), and initialize the device driver
- On a removal, kernel must shut down the device driver and gracefully deal with pending requests
  - eg, Waiting processes shouldn't hang

U.Va.
Department of Computer Science