Chapter 13  I/O Systems
Contents

- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Streams
- Performance
Objectives

- Explore the structure of an operating system’s I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide details of the performance aspects of I/O hardware and software
I/O Hardware

- Incredible variety of I/O devices
  - Function
  - speed
- Device driver
  - Present a uniform device-access interface to the I/O subsystem, much as systems calls
I/O Hardware

- Common concepts
  - Port
  - Bus (daisy chain or shared direct access)
  - Controller (host adapter)

- I/O instructions control devices
- Devices have addresses, used by
  - Direct I/O instructions
  - Memory-mapped I/O
A Typical PC Bus Structure

- Peripheral component interconnection

Diagram showing the typical PC bus structure with components such as monitor, processor, graphics controller, bridge/memory controller, cache, memory, IDE disk controller, expansion bus interface, keyboard, expansion bus, parallel port, and serial port.
## Device I/O Port Locations on PCs (partial)

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

- PCs use I/O instructions to control some devices.
- An I/O port typically consists of four registers
  - The data-in register is read by the host to get input
  - The data-out register is written by the host to send output
  - The status register contains bits that be read by the host.
  - The control register can be written by the host to start a command or to change the mode of a device.
Polling

- Determines state of device
  - command-ready
  - busy
  - Error

- Busy-wait cycle to wait for I/O from device
Polling

1. The host repeatedly reads the *busy* bit until that bit becomes clear.

2. The host sets the *write* bit in the *command* register and writes a byte into the *data-out* register.

3. The host sets the *command-ready* bit.

4. When the controller notices that the *command-ready* bit is set, it sets the *busy* bit.

5. The controller reads the command register and sees the write command. It reads the *data-out* register to get the byte, and does the I/O to the device.

6. The controller clears the *command-ready* bit, clears the *error* bit in the status register to indicate that the device I/O succeeded, and clears the *busy* bit to indicate that it is finished.
由操作系统的“服务程序”负责将用户数据传送至打印机端口
服务程序顺序传送打印数据，填满接口缓冲区后就等待（空循环）
每次循环中都检查接口缓冲区是否可用，一旦可用就继续传送数据
数据传送完成后“服务程序”结束，用户进程继续运行
缺点：靠CPU以“忙等待”的形式与打印机进行通信，浪费CPU资源
Interrupts

- CPU Interrupt-request line triggered by I/O device
- Interrupt handler receives interrupts

- We need more sophisticated interrupt-handling features
  - we need ability to defer interrupt handling during critical processing
  - we need an efficient way to dispatch to the proper interrupt handler for a device without first polling all the devices to see which one raised the interrupt.
  - we need multilevel interrupts, so that the OS can distinguish between high-and low- priority interrupts and can respond with the appropriate degree of urgency.

- Two interrupt request lines:
  - Nonmaskable interrupt
  - Maskable to ignore or delay some interrupts

- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some nonmaskable

- Interrupt mechanism also used for exceptions
Interrupts

- Interrupt Vector Table
  - Each interrupt service procedure has a unique entry address. We collect all the interrupt vectors in a specific area of memory. This area is called the interrupt vector table. In other words, each interrupt service procedure establishes a one-to-one correspondence with an entry in the table.

- Convert the interrupt number N × 4, get the first byte of the interrupt vector (i.e., the low 8 bits of IP) as a pointer, which is the vector address. = 0000: N × 4.
例如，软盘"INT 13H"，它的中断向量为"0070H（CS）：0FC9H（IP）"，当处理中断时，CPU根据类型号（13H）乘4后得到中断向量的第一个字节的指针，即：13H × 4 = 004CH。从它开始连续4个字节单元中用来存放"INT 13H"的中断向量（即入口地址）
Interrupt-Driven I/O Cycle

1. CPU receiving interrupt, transfers control to interrupt handler
2. CPU executes checks for interrupts between instructions
3. CPU resumes processing of interrupted task
4. Interrupt handler processes data, returns from interrupt
5. Input ready, output complete, or error generates interrupt signal
6. Initiates I/O
# Intel Pentium Processor Event-Vector Table

<table>
<thead>
<tr>
<th>vector number</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>divide error</td>
</tr>
<tr>
<td>1</td>
<td>debug exception</td>
</tr>
<tr>
<td>2</td>
<td>null interrupt</td>
</tr>
<tr>
<td>3</td>
<td>breakpoint</td>
</tr>
<tr>
<td>4</td>
<td>INTO-detected overflow</td>
</tr>
<tr>
<td>5</td>
<td>bound range exception</td>
</tr>
<tr>
<td>6</td>
<td>invalid opcode</td>
</tr>
<tr>
<td>7</td>
<td>device not available</td>
</tr>
<tr>
<td>8</td>
<td>double fault</td>
</tr>
<tr>
<td>9</td>
<td>coprocessor segment overrun (reserved)</td>
</tr>
<tr>
<td>10</td>
<td>invalid task state segment</td>
</tr>
<tr>
<td>11</td>
<td>segment not present</td>
</tr>
<tr>
<td>12</td>
<td>stack fault</td>
</tr>
<tr>
<td>13</td>
<td>general protection</td>
</tr>
<tr>
<td>14</td>
<td>page fault</td>
</tr>
<tr>
<td>15</td>
<td>(Intel reserved, do not use)</td>
</tr>
<tr>
<td>16</td>
<td>floating-point error</td>
</tr>
<tr>
<td>17</td>
<td>alignment check</td>
</tr>
<tr>
<td>18</td>
<td>machine check</td>
</tr>
<tr>
<td>19–31</td>
<td>(Intel reserved, do not use)</td>
</tr>
<tr>
<td>32–255</td>
<td>maskable interrupts</td>
</tr>
</tbody>
</table>
“打印服务程序”只将最开始的数据传送至打印机端口，然后阻塞
CPU可继续调度其他进程运行，不浪费CPU时间
一旦打印缓冲区空后，打印机端口发出硬件中断
CPU响应中断，恢复“打印服务程序”运行，继续传送数据
缺点：虽然节省了CPU资源，但是中断响应也消耗较大的系统资源
Direct Memory Access

- Used to avoid **programmed I/O** for large data movement

- Requires **DMA** controller

- Bypasses CPU to transfer data directly between I/O device and memory
Six Step Process to Perform DMA Transfer

1. Device driver is told to transfer disk data to buffer at address X.

2. Device driver tells disk controller to transfer C bytes from disk to buffer at address X.

3. Disk controller initiates DMA transfer.

4. Disk controller sends each byte to DMA controller.

5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0.

6. When C = 0, DMA interrupts CPU to signal transfer completion.
用户进程发出系统调用后进入阻塞态，CPU直接设置DMA端口

CPU与DMA并行工作，DMA负责将用户数据传送给打印机

当DMA完成所有工作后，向CPU发出中断，CPU响应后唤醒用户进程

优点：只有一次中断，DMA与CPU并行提高了系统运行效率

缺点：DMA速度较慢，如果CPU并不繁忙，那么DMA机制并无太大意义
13.3 Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes.
- Devices vary in many dimensions:
  - Character-stream or block
  - Sequential or random-access
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only
- Device-driver layer hides differences among I/O controllers from kernel.
A Kernel I/O Structure

- kernel
- kernel I/O subsystem
  - SCSI device driver
  - keyboard device driver
  - mouse device driver
  - •••
  - PCI bus device driver
  - floppy device driver
  - ATAPI device driver

- Hardware
  - SCSI device controller
  - keyboard device controller
  - mouse device controller
  - •••
  - PCI bus device controller
  - floppy device controller
  - ATAPI device controller

- Software
  - SCSI devices
  - keyboard
  - mouse
  - •••
  - PCI bus
  - floppy-disk drives
  - ATAPI devices (disks, tapes, drives)
## Characteristics of I/O Devices

<table>
<thead>
<tr>
<th>aspect</th>
<th>variation</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-transfer mode</td>
<td>character block</td>
<td>terminal disk</td>
</tr>
<tr>
<td>access method</td>
<td>sequential random</td>
<td>modem</td>
</tr>
<tr>
<td></td>
<td>asynchronous</td>
<td>CD-ROM</td>
</tr>
<tr>
<td>transfer schedule</td>
<td>synchronous</td>
<td>tape</td>
</tr>
<tr>
<td></td>
<td>asynchronous</td>
<td>keyboard</td>
</tr>
<tr>
<td>sharing</td>
<td>dedicated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sharable</td>
<td></td>
</tr>
<tr>
<td>device speed</td>
<td>latency</td>
<td></td>
</tr>
<tr>
<td></td>
<td>seek time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>transfer rate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delay between operations</td>
<td></td>
</tr>
<tr>
<td>I/O direction</td>
<td>read only</td>
<td>CD-ROM</td>
</tr>
<tr>
<td></td>
<td>write only</td>
<td>graphics controller disk</td>
</tr>
<tr>
<td></td>
<td>read–write</td>
<td></td>
</tr>
</tbody>
</table>
Block and Character Devices

- **Block devices include disk drives**
  - Commands include read, write, seek
  - Raw I/O or file-system access
  - Memory-mapped file access possible

- **Character devices include keyboards, mice, serial ports**
  - Commands include `get`, `put`
  - Libraries layered on top allow line editing
Network Devices

- Varying enough from block and character to have own interface

- Unix and Windows NT/9x/2000 include socket interface
  - Separates network protocol from network operation
  - Includes `select` functionality

- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)
Clocks and Timers

- Provide current time, elapsed time, timer

- **Programmable interval timer** used for timings, periodic interrupts

- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers
Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs

- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written

- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed
Two I/O Methods

(a) kernel user
   \[\text{requesting process} \rightarrow \text{waiting} \rightarrow \text{device driver} \rightarrow \text{interrupt handler} \rightarrow \text{hardware} \rightarrow \text{data transfer} \rightarrow \text{time} \rightarrow \text{user}\]

(b) user
   \[\text{requesting process} \rightarrow \text{device driver} \rightarrow \text{interrupt handler} \rightarrow \text{hardware} \rightarrow \text{data transfer} \rightarrow \text{time} \rightarrow \text{kernel}\]
13.4 Kernel I/O Subsystem

- Kernels provide many services related to I/O:
  - scheduling,
  - buffering,
  - caching,
  - spooling,
  - device reservation,
  - and error handling.

- I/O Scheduling
  - To schedule a set of I/O requests means to determine a good order in which to execute. Is it important?

- Implementation
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness
## Device-status Table

<table>
<thead>
<tr>
<th>Device</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyboard</td>
<td>idle</td>
</tr>
<tr>
<td>laser printer</td>
<td>busy</td>
</tr>
<tr>
<td>mouse</td>
<td>idle</td>
</tr>
<tr>
<td>disk unit 1</td>
<td>idle</td>
</tr>
<tr>
<td>disk unit 2</td>
<td>busy</td>
</tr>
</tbody>
</table>

- **Request for Laser Printer**
  - Address: 38546
  - Length: 1372

- **Request for Disk Unit 2**
  - File: xxx
  - Operation: read
  - Address: 43046
  - Length: 20000

- **Request for Disk Unit 2**
  - File: yyy
  - Operation: write
  - Address: 03458
  - Length: 500
Buffering (缓冲)

- A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application.

- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”
Sun Enterprise 6000 Device-Transfer Rates

- Gigaplane bus
- SBUS
- SCSI bus
- Fast Ethernet
- Hard disk
- Ethernet
- Laser printer
- Modem
- Mouse
- Keyboard

SHANDONG UNIVERSITY 32
Caching (高速缓存)

- **Caching** - fast memory holding copy of data

- **Reasons to use it**
  - Speed
  - Data size

- **Differences between caching and buffering**
  - A buffer may hold the only existing copy of a data item.
  - A cache is just holds a copy on faster storage of an item.
Disk cache in RAM

Peripheral bus (ISA, EISA, PCI, etc.)

Local bus

CPU
Spooling

- Spooling - hold output for a device
  - Simultaneous peripheral Operations On-Line
  - If device can serve only one request at a time
  - Its speed is very slow
  - i.e., Printing
Device reservation

- Device reservation - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Watch out for deadlock
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
I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
  - Memory-mapped and I/O port memory locations must be protected too
Use of a System Call to Perform I/O

1. trap to monitor
2. perform I/O
3. return to user
Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state.

- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks.

- Some use object-oriented methods and message passing to implement I/O.
UNIX I/O Kernel Structure

- File descriptor
- Per-process open-file table
- User-process memory
- System-wide open-file table
  - File-system record
    - Inode pointer
    - Pointer to read and write functions
    - Pointer to select function
    - Pointer to ioctl function
    - Pointer to close function
  - Networking (socket) record
    - Pointer to network info
    - Pointer to read and write functions
    - Pointer to select function
    - Pointer to ioctl function
    - Pointer to close function
- Active-inode table
- Network-information table
- Kernel memory
The I/O subsystem supervises the following procedures:

- Management of the name space for files and devices
- Access control to files and devices
- Operation control
- File-system space allocation
- Devices allocation
- I/O scheduling
- Device-status monitoring, error handling, and failure recovery
- Device-driver configuration and initialization
Consider reading a file from disk for a process:

- 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
Life Cycle of An I/O Request

1. request I/O
2. system call
3. kernel I/O subsystem:
   - can already satisfy request?
     - yes: transfer data (if appropriate) to process, return completion or error code
     - no: send request to device driver, block process if appropriate
4. process request, issue commands to controller, configure controller to block until interrupted
5. device-controller commands
6. monitor device, interrupt when I/O completed
7. time
8. I/O completed, generate interrupt
9. device controller:
   - interrupt handler
   - receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
10. device driver
11. determine which I/O completed, indicate state change to I/O subsystem
12. return from system call
STREAMS

- STREAM – a full-duplex communication channel between a user-level process and a device in Unix System V and beyond

- 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

Diagram showing the structure of the STREAMS system, including user process, stream head, read and write queues, and driver end.
I/O a major factor in system performance:

- Demands CPU to execute device driver, kernel I/O code
- Context switches due to interrupts
- Data copying
- Network traffic especially stressful
Intercomputer Communications

![Diagram of intercomputer communications process]

- Character typed by hardware
- Interrupt generated
- Interrupt handled
- Device driver
- Kernel
- Context switch
- User process

- System call completes
- Interrupt handled
- Network adapter
- Device driver
- Kernel
- Context switch
- Sending system

- Network packet received
- Network adapter
- Device driver
- Kernel
- Network subdaemon
- Context switch
- Receiving system
Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput
Device-Functionality Progression

- new algorithm
  - application code
    - kernel code
    - device-driver code
    - device-controller code (hardware)
    - device code (hardware)

- increased time (generations)
- increased efficiency
- increased development cost
- increased abstraction

increase flexibility
Assignment

☐ 3
End of Chapter 13

Any Question?