COP 4225 Advanced Unix Programming

I/O Systems

Chi Zhang
czhang@cs.fiu.edu
I/O Hardware

- The kernel is structured to use device-driver modules.

Common concepts

- Port (for one device)
- Bus (shared direct access)
- Controller (host adapter) accepts commands from the processor through buses
  - The controller has one or more registers for data and control signals.
A Typical PC Bus Structure

- Monitor
- Graphics controller
- Processor
- Bridge/memory controller
- Cache
- Memory
- SCSI controller
- IDE disk controller
- Expansion bus interface
- Expansion bus
- Keyboard
- Parallel port
- Serial port

Bus connections:
- PCI bus
- SCSI bus
I/O Hardware

- Expansion bus connects relatively slow devices
- Devices have addresses, used by
  - Direct I/O instructions (in, out)
    - Slower
    - Space limited
  - Memory-mapped I/O (mov, add, or, …)
    - Faster
    - Prone to software faults
- An I/O port typically consists of four registers
  - status, control, data-in, data-out
## 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>
Polling

- Producer-consumer handshake
  - command-ready bit in the command register
  - busy bit in the status register
- Busy-wait cycle to wait for I/O from device
  - The processor polls the busy bit until it becomes clear
  - The processor sets the write bit in the command register and writes a byte into the data-out register before setting the command-ready bit

- Wastes CPU time
Interrupts

- CPU Interrupt request line triggered by I/O device
- Interrupt vector to dispatch interrupt to correct handler
  - Registered at boot time.
  - Based on priority (Some unmaskable)
- CPU saves a small amount of state, and jumps to the interrupt handler
- Interrupt handler processes interrupts
- Interrupt handler then return to the execution state prior to the interrupt.
Interrupts

- Interrupt handler
  - Transfer data from the controller to memory (if not DMA)
  - Wake up the process waiting for the I/O completion
- Interrupt mechanism also used for exceptions
  - Page Fault in virtual memory paging
- Interrupt mechanism also used for Systems Calls
  - Trap
  - Switch to kernel mode
Interrupt-Driven I/O Cycle

1. CPU
   - device driver initiates I/O
2. I/O controller
   - initiates I/O
3. input ready, output complete, or error
   - generates interrupt signal
4. CPU receiving interrupt, transfers control to interrupt handler
5. interrupt handler processes data, returns from interrupt
6. CPU resumes processing of interrupted task
7. CPU executing checks for interrupts between instructions
Direct Memory Access

- Used to avoid programmed I/O (PIO) for large data movement
  - Bypasses CPU to transfer data directly between I/O device and memory
- Requires DMA controller
  - DMA-request from the device controller to the DMA controller
  - DMA-ack from the DMA controller to the device controller.
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
Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
  - Device-driver layer hides differences among I/O controllers from kernel
- Back-door to transparently pass arbitrary commands from an application to a device driver
  - Unix: ioctl
    - An integer argument to select one of the commands
I/O system calls encapsulate device behaviors in generic classes
Block and Character Devices

- Block devices include disk drives
  - Commands include read, write, seek
  - Memory-mapped file access possible

- Character devices include keyboards, mice, serial ports
  - Commands include `get`, `put`
  - Produce data input at unpredictable time.
STREAMS

- **STREAM** – a full-duplex communication channel between a user-level process and a device
  - Character devices only
- Message passing is used to communicate between queues (e.g. putmsg vs. write)
  - Message boundaries and control information between modules
- Modules providing processing functionality can be pushed into Stream by ioctl().
  - Modular and incremental development
The STREAMS Structure
Clocks and Timers

- Provide current time, elapsed time, timer

- If programmable interval time used for timings, periodic interrupts
  - Virtual clocks

- `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
  - Efficiencies can be improved via multi-threading

- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)

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

- Scheduling
  - Some I/O request ordering via per-device queue
  - Minimize disk arm seeks and improve fairness

- 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”
    - Application might change the buffer after system calls
Kernel I/O Subsystem

- Spooling - hold output for a device
  - If device can serve only one request at a time
  - Each application’s output is spooled to a separate disk file
  - E.g. a daemon process for printing

- Error handling
  - Most return one bit information about the status (success / failure)
  - An error number or code indicating the error nature (Unix: errno)
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**
  - User-process memory
  - Per-process open-file table

- **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

- **Active-_inode Table**
- **Networking Information Table**
  - 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
I/O Requests to Hardware Operations

Consider reading a file from disk for a process:

- Determine device holding file
  - Longest match prefix in the mount table
  - `<major, minor>` device number
  - Minor passed to the driver selected by major.
- 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**
   - User process
   - System call

2. **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

3. **Process request, issue commands to controller, configure controller to block until interrupted**
   - Device controller commands

4. **Monitor device, interrupt when I/O completed**
   - Receive interrupt, store data in device-driver buffer if input, signal to unblock device driver

5. **I/O completed, generate interrupt**

6. **I/O completed, input data available, or output completed**
   - Return from system call

**Flow of Time:**
Performance

I/O a major factor in system performance:
- Demands CPU to execute device driver, kernel I/O code
- Context switches due to interrupts
  - Sometimes Programmed I/O is more efficient, if the number of busy-waiting cycles is not excessive.
- Data copying
- Network traffic especially stressful
Intercomputer Communications
Improving Performance

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

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

new algorithm

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

increased flexibility