Operating Systems

Christopher Kruegel
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~chris/
Input and Output
Input/Output Devices

- The OS is responsible for managing I/O devices
  - Issue requests
  - Manage corresponding interrupts
- The OS provides a high-level, easy-to-use interface to processes
- The interface, in principle, should be as uniform as possible
- The I/O subsystem is the part of the kernel responsible for managing I/O
- Composed of a number of *device drivers* that deal directly with the hardware
I/O Devices

- Two categories:
  - Block devices
    - Store information in blocks of a specified size
    - Block can be accessed (read or written) independently
    - Example: disk
  - Character devices
    - Deal with a stream of characters without a predefined structure
    - Characters cannot be addressed independently
    - Example: mouse, printer, keyboard

- Classification not perfect
  - Example: Clocks
## Device Data Rates

<table>
<thead>
<tr>
<th>Device</th>
<th>Data rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>10 bytes/sec</td>
</tr>
<tr>
<td>Mouse</td>
<td>100 bytes/sec</td>
</tr>
<tr>
<td>56K modem</td>
<td>7 KB/sec</td>
</tr>
<tr>
<td>Telephone channel</td>
<td>8 KB/sec</td>
</tr>
<tr>
<td>Dual ISDN lines</td>
<td>16 KB/sec</td>
</tr>
<tr>
<td>Laser printer</td>
<td>100 KB/sec</td>
</tr>
<tr>
<td>Scanner</td>
<td>400 KB/sec</td>
</tr>
<tr>
<td>Classic Ethernet</td>
<td>1.25 MB/sec</td>
</tr>
<tr>
<td>USB (Universal Serial Bus)</td>
<td>1.5 MB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>4 MB/sec</td>
</tr>
<tr>
<td>IDE disk</td>
<td>5 MB/sec</td>
</tr>
<tr>
<td>40x CD-ROM</td>
<td>6 MB/sec</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>12.5 MB/sec</td>
</tr>
<tr>
<td>ISA bus</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>EIDE (ATA-2) disk</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>FireWire (IEEE 1394)</td>
<td>50 MB/sec</td>
</tr>
<tr>
<td>XGA Monitor</td>
<td>60 MB/sec</td>
</tr>
<tr>
<td>SONET OC-12 network</td>
<td>78 MB/sec</td>
</tr>
<tr>
<td>SCSI Ultra 2 disk</td>
<td>80 MB/sec</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>125 MB/sec</td>
</tr>
<tr>
<td>Ultrium tape</td>
<td>320 MB/sec</td>
</tr>
<tr>
<td>PCI bus</td>
<td>528 MB/sec</td>
</tr>
<tr>
<td>Sun Gigaplane XB backplane</td>
<td>20 GB/sec</td>
</tr>
</tbody>
</table>
Device Controllers

- I/O devices typically have two components
  - Mechanical component
  - Electronic component (e.g., connected to the mechanical component through a cable)

- The electronic component is the device controller
  - Often a PCI/ISA card installed on the motherboard (host adapter)
  - May be able to handle multiple devices (e.g., daisy chained)
  - May implement a standard interface (SCSI/EIDE/USB)

- Controller's tasks
  - Convert serial bit stream to block(s) of bytes (e.g., by internal buffering)
  - Perform error correction as necessary
  - Make data available to CPU/memory system
Accessing the Controller

• The OS interacts with a controller
  – By writing/reading registers (command/status)
  – By writing/reading memory buffers (actual data)

• Registers can be accessed through dedicate CPU instructions
  – Registers mapped to I/O ports
  – IN REG, PORT and OUT REG, PORT
    transfer data from CPU’s registers to a controller’s registers

• Registers can be mapped onto memory (Memory-Mapped)

• Hybrid approach
  – Registers are accessed as I/O ports
  – Buffers are memory mapped
  – Used by the Pentium (640K-1M mem-mapped buffer, 0-64K ports)
Accessing the Controller

UC Santa Barbara
Accessing the Controller

• When a controller register has to be accessed
  – CPU puts address on the bus
  – CPU sets a line that tells if this address is a memory address or an I/O port
  – In case the register/buffer is memory-mapped, the corresponding controller is responsible for checking the address and service the request if the address is in its range
Memory-Mapped I/O

• Advantages
  – Does not require special instructions to access the controllers
  – Protection mechanisms can be achieved by not mapping processes’ virtual memory space onto I/O memory

• Disadvantages
  – Caching would prevent correct interaction (hardware must provide a way to disable caching)
  – If the bus connecting the CPU to the main memory is not accessible to the device controllers, the hardware has find a way to let controllers know which addresses have been requested
Memory-Mapped I/O

CPU
Memory
I/O

All addresses (memory and I/O) go here

Bus

CPU reads and writes of memory go over this high-bandwidth bus

CPU
Memory
I/O

This memory port is to allow I/O devices access to memory
Direct Memory Access (DMA)

• Reading/writing one word at a time may waste CPU time

• A DMA controller supports “automatic” transfer between controllers and main memory

• A DMA controller can be associated with each device or can be one for all the devices

• The DMA controller
  – Has access to the device bus and to the memory
  – Has a memory address register, a count register, and one or more control register (I/O port to use, direction of transfer, etc.)
Reading with Direct Memory Access

- The CPU
  - Loads the correct values in the DMA controller
  - Sends a read operation to the device controller

- The DMA
  - Waits for the operation to complete
  - Sets the destination memory address on the bus
  - Sends a transfer request to the controller

- The controller
  - Transfers the data to memory
  - Sends an ACK signal when the operation is completed

- When the DMA has finished it sends an interrupt to the CPU
Direct Memory Access (DMA)
DMA Schema Variations

• Cycle stealing
  – DMA acquires the bus competing with the CPU for each word transfer

• Burst mode
  – DMA tells the controller to acquire the bus and issue a number of transfers

• The DMA may ask the controller to transfer data to a buffer on the DMA controller and then perform the actual transfer to memory
  – supports device-to-device direct transfer
Interrupts

- When a device has completed a task it sends out a signal
- The signal is detected by the interrupt controller
- The interrupt controller puts the device address on the bus and sends a signal to the CPU
- The CPU
  - Stops
  - Saves PC and PSW and uses the address on the bus to look up the interrupt vector
- The interrupt vector contains the address of the handling routine which is loaded in the program counter
- After the processing of the interrupt has started an ack is sent to the interrupt controller
Interrupts
Saving the CPU State

• The interrupt handler should save the current CPU state

• If registers are used, nested interrupts would overwrite the data and, therefore, the acknowledgment to the interrupt controller must be delayed

• If a stack is used, the information should be stored in a portion of memory that will not generate page faults
Restoring the CPU State

• Restoring is easier said then done when instructions may end up... half-baked (in case of pipelining)

• A *precise* interrupt leaves the machine in a well-defined state
  – The PC is saved in a known place
  – All instructions before the one pointed by the PC have been fully executed
  – No instruction beyond the one pointed by the PC has been executed
  – The execution state of the instruction pointed by the PC is known

• Restoring in case of imprecise interrupts requires a lot of information to be saved
Goals of I/O Software

• Device independence
  – Programs can access any I/O device without specifying device in advance (reading from floppy, hard drive, or CD-ROM should not be different)

• Uniform naming
  – Name of a file or device should not depending on the device

• Error handling
  – Errors should be handled as close to the hardware as possible

• Synchronous vs. asynchronous transfers
  – User program should see blocking operations even though the actual transfer is implemented asynchronously

• Buffering
I/O Software

• System call in user-space

• Data is copied from user space to kernel space

• I/O software can operate in several modes
  – Programmed I/O
    • Polling/Busy waiting for the device
  – Interrupt-Driven I/O
    • Operation is completed by interrupt routine
  – DMA-based I/O
    • Set up controller and let it deal with the transfer
Programmed I/O

copy_from_user(buffer, p, count);
for (i = 0; i < count; i++) {
    while (*printer_status_reg != READY);
    *printer_data_register = p[i];
}
return_to_user();
/* p is the kernel buffer */
/* loop on every character */
/* loop until ready */
/* output one character */
Interrupt-Driven I/O

```c
copy_from_user(buffer, p, count);
enable_interrupts();
while (*printer_status_reg != READY) {
    *printer_data_register = p[0];
scheduler();
}
if (count == 0) {
    unblock_user();
} else {
    *printer_data_register = p[i];
    count = count - 1;
    i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```
I/O Using DMA

copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler();

acknowledge_interrupt();
unlock_user();
return_from_interrupt();
I/O Software Layers

- User-level I/O software
- Device-independent operating system software
- Device drivers
- Interrupt handlers
- Hardware
Interrupt Handlers

• Device driver starts I/O and then blocks (e.g., p->down)

• Interrupt handler does the actual work and then unblocks driver that started it (e.g., p->up)

• Mechanism works best if device drivers are threads in the kernel
Interrupt Handlers

• Save registers not already saved by interrupt hardware
• Set up context for interrupt service procedure (TLB, MMU)
• Set up stack for interrupt service procedure
• Acknowledge interrupt controller, re-enable interrupts
• Copy registers from where saved to process table
• Run service procedure
• Decide which process to run next
• Set up MMU context for process to run next
• Load new process' registers
• Start running the new process
Device Drivers

- A device driver is a specific module that manages the interaction between the device controller and the OS
- Device drivers are usually provided by the device manufacturer (or by frustrated Linux users!)
- Device are usually part of the kernel
  - compiled and linked in
  - loadable modules
- Usually provide a standard API depending on the type of device
  - Character
  - Block
- Device drivers are usually the source of kernel problems
Device Drivers

UC Santa Barbara
Device Driver’s Tasks

- Device initialization
- Accept read-write request from the OS
- Check input parameters
- Start the device if necessary (e.g., start spinning the CD-ROM)
- Check if device is available: if not, wait
- Issue command(s)
- Wait for results
  - Busy wait (awakened by interrupt)
  - Block
- Check for possible errors
- Return results
Device-Independent I/O Software

- Some I/O related functionalities are independent of the particular device and may be carried out outside the device driver
  - Uniform interfacing for device drivers: make all the devices look more or less the same
    - Uniform API
    - Uniform naming
  - Buffering: maintain a copy of the data to read/write in the kernel and transfer to user-space only when needed
  - Error reporting
  - Allocating and releasing dedicate devices
  - Providing a device-independent block size: hide logical/physical differences
User-Space I/O Software

- User processes
  - Make I/O call; format I/O; spooling
- Device-independent software
  - Naming, protection, blocking, buffering, allocation
- Device drivers
  - Set up device registers; check status
- Interrupt handlers
  - Wake up driver when I/O completed
- Hardware
  - Perform I/O operation
Disk

• Most important and commonly used device

• Used for secondary memory (swap space, file system)

• Different types:
  – Magnetic (floppy, hard disk)
  – Optical (CD-ROM, DVD)
Magnetic Disks

- Disk “geometry” specified in terms of
  - Cylinders composed of tracks (one per head)
  - Tracks composed of sectors
  - Sectors composed of bytes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IBM 360-KB floppy disk</th>
<th>WD 18300 hard disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cylinders</td>
<td>40</td>
<td>10601</td>
</tr>
<tr>
<td>Tracks per cylinder</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Sectors per track</td>
<td>9</td>
<td>281 (avg)</td>
</tr>
<tr>
<td>Sectors per disk</td>
<td>720</td>
<td>35742000</td>
</tr>
<tr>
<td>Bytes per sector</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Disk capacity</td>
<td>360 KB</td>
<td>18.3 GB</td>
</tr>
<tr>
<td>Seek time (adjacent cylinders)</td>
<td>6 msec</td>
<td>0.8 msec</td>
</tr>
<tr>
<td>Seek time (average case)</td>
<td>77 msec</td>
<td>6.9 msec</td>
</tr>
<tr>
<td>Rotation time</td>
<td>200 msec</td>
<td>8.33 msec</td>
</tr>
<tr>
<td>Motor stop/start time</td>
<td>250 msec</td>
<td>20 sec</td>
</tr>
<tr>
<td>Time to transfer 1 sector</td>
<td>22 msec</td>
<td>17 μsec</td>
</tr>
</tbody>
</table>
Disk Architecture

- Hard disk
  - several platters – disks (heads)
  - each platter has multiple tracks (start with 0)
  - each track has multiple sectors (start with 1)
Disk Architecture

Addressing sectors (blocks)

• CHS (cylinder, head, sector) triple
  – old disks use 10 bits for cylinder, 8 bits for head, 6 for sector
  – limits maximum disk size to ~ 8.4 GB

• Logical block address (LBA)
  – decouples logical and physical location
  – specifies 48 bit logical block numbers
  – allows controller to mask corrupt blocks
Disk Geometry

- Physical geometry could be different from the “logical” geometry
  - Mapping between the two performed by the controller
Disk Interfaces
between controller (motherboard) and disk

• ATA (AT Attachment)
  – 28 bit addresses (~128 GB maximum size)
  – 40 pin cables, 16 bit parallel transfer (single-ended signaling)
  – 2 devices (master and slave) can be attached to connection cable
  – ATA-3 introduced security features (passwords)

• Serial ATA (SATA)
  – 8 pin cables
  – higher data transfer (differential signaling)
Disk Architecture

• Hidden protected area (HPA)
  – introduced with ATA-4
  – disk can be set to report to OS less blocks than actually available
  – remaining blocks can be used for data that is not formatted utilities and diagnostic tools, but also malicious code or illegal material

• Device configuration overlay (DCO)
  – introduced with ATA-6
  – additional space (blocks) after HPA
  – used by manufacturers to shrink different disks to appear with exactly the same size
RAID

- Redundant Array of Inexpensive Disks vs. Single Large Expensive Disk (SLED)

- A set of disks is managed by a RAID controller
- Different RAID modes (called “levels”)

- RAID 0
  - Disks are divided into strips of k sector each
  - Strips are allocated to disks in a round-robin fashion
  - Request for consecutive strips can be carried out in parallel

- RAID 1
  - Striping + redundancy

- RAID 2
  - Striping at the word/byte level + ECC
RAID

• RAID 3
  – Parity word kept on a separate drive

  ![Diagram of RAID 3]

• RAID 4
  – Strip parity on extra drive (XOR of strip contents)

  ![Diagram of RAID 4]
RAID

• RAID 5
  – Parity strips are distributed over the disks
The initial sector for each track is skewed with respect to the previous one. This facilitates continuous reads across contiguous tracks by taking into account the rotation of the disk when the arm is moved.

- 7,200 rpm with 360 sectors
- Cycle in $60/7,200 = 8.3$ msec
- Sector rate $8.3$ msec/360 = 23usec
- Moving from track to track = 900usec
- Skew ~ 40 sectors
Interleaving

- A disk reads a sector and puts it in the controller’s buffer
- While the sector is being transferred to memory the next sector will pass under the disk head
- Solution: Interleaving (single, double, etc)
- Solution: Buffer a whole track at a time
Disk Arm Scheduling Algorithms

- Time required to read or write a disk block determined by 3 factors
  - Seek time
  - Rotational delay
  - Actual transfer time

- Seek time is the most relevant and must be minimized

- Possible scheduling algorithms
  - First-Come First-Served: bad
  - Algorithms with request buffering in the driver
    - Shortest Seek First (SSF)
    - Elevator Algorithm

- Note that these algorithms imply that logical/physical geometry match or at least mapping is known
Shortest Seek First Algorithm

- SSF moves the arm towards the closest request
- If request are many the algorithm may be unfair towards request for sectors far from the arm’s current position
Elevator Algorithm

• The arms moves in one direction until there is no request left, then it changes direction
Clocks

- The clock is a fundamental device
- The counter is initialized with an OS-defined value
- The hardware decrements the counter with a certain frequency (e.g., 500 MHz)
- When the counter reaches 0, an interrupt is sent and the start value is restored
Clock Driver

- Maintains the time of day
- Checks processes’ CPU quantum usage
  - Calls the scheduler if quantum expired
- Does accounting of CPU usage and profiling of the system
- Handles alarms
  - Alarms are maintained in a list and fired whenever they expire
Character Oriented Terminals

- Simplest form of user-interaction
- A terminal is composed of a keyboard and a screen
- Characters typed from the keyboard are sent to the driver
- Characters sent by the driver are displayed on the screen
- Different modes of operation
  - Raw (non canonical): characters are passed by the driver to the user process as they are typed
  - Cooked, line-oriented (canonical): the drivers performs line-by-line processing before passing the line to the user process
- Drivers maintain buffered input/output and process special characters
RS-232 Terminal Hardware

- An RS-232 terminal communicates with computer 1 bit at a time (serial line)
- Bits are reassembled into characters by the UART (Universal Asynchronous Receiver/Transmitter)
- Windows uses COM1 and COM2 ports, UNIX uses /dev/tty
- Computer and terminal are completely independent
## Special Input Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>POSIX name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-H</td>
<td>ERASE</td>
<td>Backspace one character</td>
</tr>
<tr>
<td>CTRL-U</td>
<td>KILL</td>
<td>Erase entire line being typed</td>
</tr>
<tr>
<td>CTRL-V</td>
<td>LNEXT</td>
<td>Interpret next character literally</td>
</tr>
<tr>
<td>CTRL-S</td>
<td>STOP</td>
<td>Stop output</td>
</tr>
<tr>
<td>CTRL-Q</td>
<td>START</td>
<td>Start output</td>
</tr>
<tr>
<td>DEL</td>
<td>INTR</td>
<td>Interrupt process (SIGINT)</td>
</tr>
<tr>
<td>CTRL-\</td>
<td>QUIT</td>
<td>Force core dump (SIGQUIT)</td>
</tr>
<tr>
<td>CTRL-D</td>
<td>EOF</td>
<td>End of file</td>
</tr>
<tr>
<td>CTRL-M</td>
<td>CR</td>
<td>Carriage return (unchangeable)</td>
</tr>
<tr>
<td>CTRL-J</td>
<td>NL</td>
<td>Linefeed (unchangeable)</td>
</tr>
</tbody>
</table>
Special Output Characters

- The ANSI escape sequences are accepted by terminal driver on output and converted in specific terminal commands through the termcap mapping

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC [ nA</td>
<td>Move up n lines</td>
</tr>
<tr>
<td>ESC [ nB</td>
<td>Move down n lines</td>
</tr>
<tr>
<td>ESC [ nC</td>
<td>Move right n spaces</td>
</tr>
<tr>
<td>ESC [ nD</td>
<td>Move left n spaces</td>
</tr>
<tr>
<td>ESC [ m; nH</td>
<td>Move cursor to (m,n)</td>
</tr>
<tr>
<td>ESC [ sJ</td>
<td>Clear screen from cursor (0 to end, 1 from start, 2 all)</td>
</tr>
<tr>
<td>ESC [ sK</td>
<td>Clear line from cursor (0 to end, 1 from start, 2 all)</td>
</tr>
<tr>
<td>ESC [ nL</td>
<td>Insert n lines at cursor</td>
</tr>
<tr>
<td>ESC [ nM</td>
<td>Delete n lines at cursor</td>
</tr>
<tr>
<td>ESC [ nP</td>
<td>Delete n chars at cursor</td>
</tr>
<tr>
<td>ESC [ n@</td>
<td>Insert n chars at cursor</td>
</tr>
<tr>
<td>ESC [ nm</td>
<td>Enable rendition n (0=normal, 4=bold, 5=blinking, 7=reverse)</td>
</tr>
<tr>
<td>ESC M</td>
<td>Scroll the screen backward if the cursor is on the top line</td>
</tr>
</tbody>
</table>