Input and Output

Reference: Operating Systems, *Fourth Edition, William Stallings, Chapters 11, 12*

Nick Urbanik <nicku@nicku.org>

A computing department

Input/Output

What is involved in input/output?

How does the operating system manage I/O?

I/O — Enormous Variety

- Three main categories:
  - **Human readable:** video displays, printers, keyboard, mouse
  - **Machine readable:** disk, CDROM, tape drives, sensors, controllers, actuators
  - **Communications:** network cards, modems

Differences in I/O Devices

Data rate: Speeds of I/O devices vary greatly: Fast: Gigabit Ethernet — 10^9 bps, Slow: keyboard — 10^2 bps

Application: OS Policies and supporting utilities required depend on application. Disks holding files need file management support. Disks for swap require virtual memory hardware and software.

Complexity of control: A printer has a simple control interface; a disk has a complex control interface.

Unit of transfer: Can transfer a stream of bytes (e.g., keyboard), or large blocks of data (e.g., disk I/O)

Data representation: Different data encoding schemes

Error conditions: Type of errors, way they are reported vary greatly

Techniques of I/O

Programmed I/O: CPU gives I/O command; CPU polls in a loop waiting for I/O to complete (busy waiting)

Interrupt-driven I/O: CPU gives I/O command, continues with some other work, then I/O module sends interrupt to CPU when it is finished.

Direct memory access (DMA): A DMA controller moves data between memory and I/O device *independently of CPU*. CPU simply tells DMA controller *where* to move data, and *how much*, DMA controller does the rest. DMA controller interrupts CPU when finished.

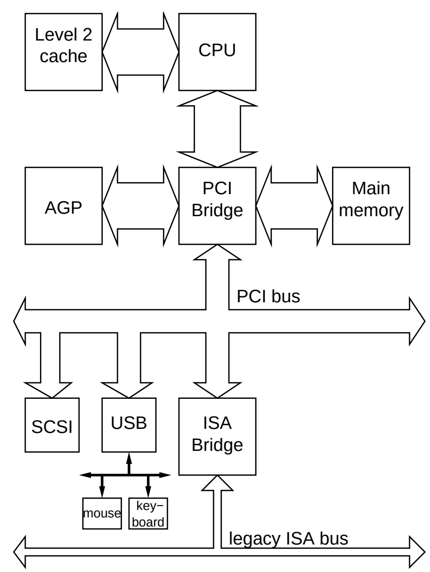
What is I/O?

- Involves transferring data to/from hardware, e.g.
  - Printers, hard disks, mice, displays, cameras, networks,. . .
- Hardware often has support for data transfer independent of main CPU, e.g., DMA, CPU on I/O device.
- CPU will have some involvement
- *Disk I/O is very important*, and most effort in improving I/O in OS goes into improving disk I/O

I/O (usually) limits execution speed

- Most processes perform a burst of:
  - CPU activity
  - I/O activity
  - CPU activity
  - I/O activity . . .
- Even with maximum priority, most processes are limited by the I/O they perform.
- We say that most processes are *I/O bound*
  - If a process performs much computation, but little I/O, it is said to be *CPU bound*

PC architecture



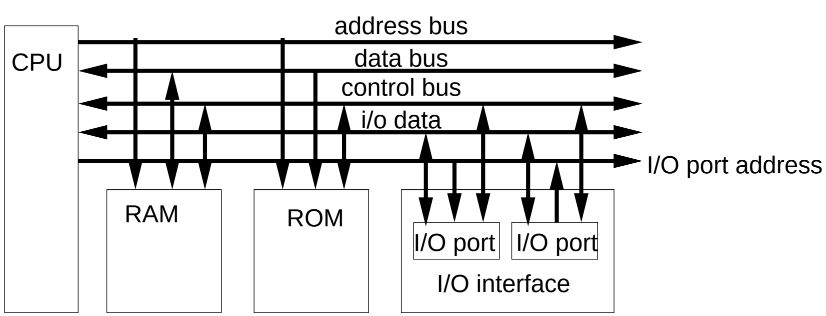
I/O ports

- Registers in I/O interface, often grouped into:
  - Control registers
  - Status registers
  - Input register
  - Output register
- Originally mostly connected to “I/O bus”
- Use only `in`, `out` machine instructions
- Memory mapped I/O ports more common now
- PCI cards use high addresses

Memory mapped I/O

- I/O ports can be mapped into the same address space as RAM, ROM.
- Full range of normal instructions can be used to access I/O ports
- Fuller address space available

I/O mapped I/O — 1



I/O mapped I/O — 2

- The diagram may be just conceptual
  - There is a *logically separate* data and address bus for I/O ports
  - Some hardware provides a separate bus dedicated to I/O, e.g. PCI
- Limited special instructions perform input or output to port.
- On Intel architecture, only 2^{16} different port addresses available for ISA devices

I/O interface

- An *I/O interface* is a hardware circuit between
- some I/O ports and their
- device controller
- Often connected to Programmable Interrupt controller through IRQ line
- Can have
- Custom I/O interface, or
- General purpose I/O interface

Examples of I/O interfaces

Custom interfaces

- Specialised to a particular function
- Examples:
  - Keyboard
  - Graphic display
  - Disk
  - Network

General interfaces

- Can plug in many types of device
- Examples:
  - Parallel port
  - Serial port
  - USB port
  - PCMCIA
  - SCSI

I/O controller

- Hardware used by complex devices, e.g., disk controller
- Interprets high level commands received by interface through I/O ports
- Converts these to complex operations on the hardware
- Simpler devices have no device controller, e.g., PIC and PI/T

Methods of I/O

- There are three basic methods of I/O:
  - DMA (Direct Memory Access)
  - Interrupt-driven I/O
  - polling

DMA

- Used on PCI bus and ATA hard disks
- A special processor called a *DMA controller* (DMAC) takes over from CPU
- Transfers data rapidly, no need to fetch instructions
- Called a *bus master* because it takes over the buses from CPU
- PC has five ISA DMA channels, and also special DMA controllers built into PCI controller

Interrupts

- PC supports 15 hardware interrupts
- Each interrupt is a hardware connection from I/O device to CPU
- Goes through a PI/T (Programmable interrupt controller) module
- Interrupts have different priorities
- Notice that I/O devices usually use interrupts

What is an interrupt? Example

- Use serial port (UART) as an example
- UART = Universal Asynchronous Receiver/Transmitter = an IC
- Receives/transmits characters one bit at a time.
- Frame each char with start/stop bit

Interrupt example: UART — 2

- When character is finally received by UART:
  1. UART sends hardware signal to CPU
  2. CPU determines whether priority higher than current task—if so,
  3. CPU stores program counter (PC), some other registers on *stack*
  4. Executes a subroutine called an ISR (interrupt Service Routine) that allows CPU to copy character(s) from UART to buffer in RAM
  5. CPU restores PC, registers, continues from where it left off.

Polling

- I/O device has a register
- CPU checks a flag in this register in a loop.
- CPU may be busy, unable to do other tasks, so waste time
- But simpler than other I/O methods

Input/Output: how?

- In the old days, (MS-DOS), application programs would write directly to input/output hardware
- Examples: hardware = registers in UART, or DMAC, other registers in I/O device
- Problems: if program has bug, whole system hangs: unreliable
- What if two processes want the same resource?

I/O: how now? — 2

- Each hardware device has a *device driver*.
- High level command $\rightarrow$ Device driver $\rightarrow$ low level actions
- Application uses same high level commands with different hardware
- Provides modularity

Advantages of device drivers

- Reliability
- OS controls access to the hardware
- Modular system possible, e.g. Linux has dynamically loading modules
- Separate kernel from low-level details of hardware.
- In Linux, device drivers part of the kernel distribution.

Installing a driver (e.g. for NIC)

- To install the driver for a 3Com 3c59x card in Linux, simply do:
- `modprobe 3c59x`
- To install it permanently for `eth0`,
  - Edit `/etc/modules.conf`
  - Add the line:
  - `alias eth0 3c59x`
  - Restart networking on that NIC with `ifup eth0`
  - Do not reboot!
- Command `lsmod` lists all installed drivers; drivers are usually loadable modules.

Block and character devices

- *Block devices* include disks
  - Provide random access
  - Data transferred in *fixed-sized* blocks
- *Character devices* include serial devices, printers
  - Transfer data in *variable sized* chunks
  - Provide sequential access

I/O Buffering — 1

- Buffering provides two benefits:
  - Improves *performance*
  - Allows process to be *swapped out* of RAM while OS performs I/O for the process

I/O Buffering — 2

- If transfer I/O data directly to or from a slow device to a process, the memory page getting or providing the data must stay in RAM
- So OS could not completely swap out this process
- More *efficient* to use *buffering*:
  - perform *input* transfer *before* the data is required
  - perform *output* sometime *later* than when request is made.

Single Buffer

- OS requests one block of data before it is needed — read ahead.
- I/O device fills block
- Application finds data is there immediately it is required

Double Buffer

- Use two memory buffers instead of one.
- While OS fills one buffer, user process empties the other.
- Very important for process that continuously provides data, such as *data acquisition*, since data may be lost with a single buffer, as the input cannot be paused while emptying the buffer ready for new data

Case Study: DMA and Double Buffering — 1

- I wrote software some years ago to collect data, analyse the data, graph the data, and write the data to disk
- Used DMA on the ISA bus (A Data Translation DT2821 data acquisition card)
- The DT2821 used an interrupt and two DMA channels
- The device driver initialised the two DMA channels with this information:
  - The *direction* of data transfer (read or write)
  - The *address* of the DMA buffer
  - The *size* of the DMA buffer

Case Study: DMA and Double Buffering — 2

- The device driver enables the DMA controller
  - The DMA controller transfers data from DT2821 to the DMA buffer
- When DMA buffer filled, DMAC raises an interrupt
- The Interrupt Service Routine (ISR) starts the other DMA channel, then re-programs the DMA controller that has just finished.

Buffer in Process or Device Driver?

- Should we put the buffers in the process that uses the data or should we put it in the device driver?
- For most applications, should *put buffers in device driver*
  - Allows process to be swapped out during I/O
  - Avoids problems with special memory needs of special I/O devices:
    - High speed DMA requires the buffer to be continuous memory, not pages scattered about in RAM

Main Effects of Buffering

- Smooth out peaks of I/O demand
- Allow processes to be swapped in and out, regardless of state of I/O for the process

Device files — 1

- In Linux and Unix, most I/O devices are accessed through a *device file*
- These are in the `/dev` directory.
- Why?
  - Allows device access to be managed using file access permissions
  - Provides a uniform access method to all I/O devices, similar to those used to access files
- Example: a temperature controller device file may give the current temperature when read it
- Example: the same `write()` system call can write data to an ordinary file or send it to a printer by writing to the `/dev/lp0` device.

Device files — 2

- There are two types of device files as mentioned in slide 26: block and character
- Each device file has two numbers:
  - Major number
  - Minor number
- These two numbers, and the type, uniquely identify a device and its device file.

```
$ ls -l /dev/hd[a-d]
brw-rw----    1 root     disk       3,   0 Oct 22 22:53 /dev/hda
brw-rw----    1 root     disk       3,  64 Aug 31  2002 /dev/hdb
brw-rw----    1 root     disk      22,   0 Aug 31  2002 /dev/hdc
brw-rw----    1 root     disk      22,  64 Aug 31  2002 /dev/hdd
```

- Note: major number is on the left, minor number is on the right; for `/dev/hdd`, major is 22, minor is 64

Device files — 3

- How do I know what device numbers my hardware should have?
  - see the file `devices.txt` that is part of the kernel source code:

```
$ rpm -ql kernel-doc | grep '/devices.txt'
/usr/share/doc/kernel-doc-2.4.18/devices.txt
```

- How do I create a device file if it is missing (for eaxample, on a rescue disk?)
  - Use the `mknod` command.
  - See also the command `/dev/MAKEDEV`, which also has its manual page.

Monitoring I/O performance

- Both `vmstat` and `procinfo` provide information about I/O.
- Also `sar -b`
- `iostat` gives details of data transfers to/from each hard disk
- The command `hdparm -tT` gives some basic performance parameters for your hard disks.
- `hdparm` and `hdparm -i` gives other info about your hard disks.
- `netstat -i`, `ifconfig` and `ip -statistics -statistics link` give details of network performance

Types of hard disk

- Currently have two choices:
- ATA133, or SCSI.
- ATA is cheaper, now reasonable performance compared with SCSI for a small number of disks.
- A cost effective method is to:
  - add a number of IDE controllers to a server,
  - Add one disk to each IDE channel
  - Use software RAID for reliability & speed
- SCSI controller has processor to reduce load on main CPU, ATA adds load to main CPU
- Coming soon: serial ATA and serial SCSI, reducing size of cables, increasing data rate (early models are already on the market)

Getting info about I/O devices

- Can list all PCI devices with `lspci`
- Can list all devices with `lsdev`
- List all modules that are currently loaded into kernel with `lsmod`
- The `/proc` file system contains all you ever wanted to know about your hardware and OS
- See which hard disks are the busiest with `iostat`

Virtual Filesystem

- Linux supports many file systems:
  - Filesystems for Linux: Ext2, Ext3, ReiserFS
  - Filesystems for other Unixs: SYSV, UFS, MINIX, Veritas VxFS
  - Microsoft filesystems: MS-DOS, VFAT, NTFS
  - Filesystems for CDs and DVDs: ISO9660, Universal Disk Format (UDF)
  - HPFS (OS2), HFS (Apple), AFFS (Amiga), ADFS (Acorn)
  - Journalling filesystems from elsewhere: JFS from IBM, XFS from SGI
- Same commands and system calls work with all!
- How does it all work?

Virtual Filesystem (VFS) — 2

- VFS is a kernel software layer
- handles all system calls to a file system
- Provides a *common interface* to all filesystems
- Also handles network filesystems, including NFS, SMB (used in Microsoft networking), NCP (Novell)
- Also handles special or "virtual" file systems, such as the `/proc` filesystem, and supports block devices such as `/dev/loop0` which allow mounting a file as if it were a disk partition.

Volume Managers

- A *Volume Manager* is a software layer built into an OS that allows logical disk partitions to be created from physical disks.
- Allows logical partitions to be *dynamically resized* (without rebooting the computer)
- Very useful for large systems with rapidly changing data
- Examples: *Logical Volume Manager* (LVM) from Sistina on Linux, a standard part of the kernel
  - Can be built on top of software RAID, a good choice
- Microsoft Dynamic Disks (see `http://www.winnetmag.com/WindowsServer2003/Index.cfm?ArticleID=27085`)

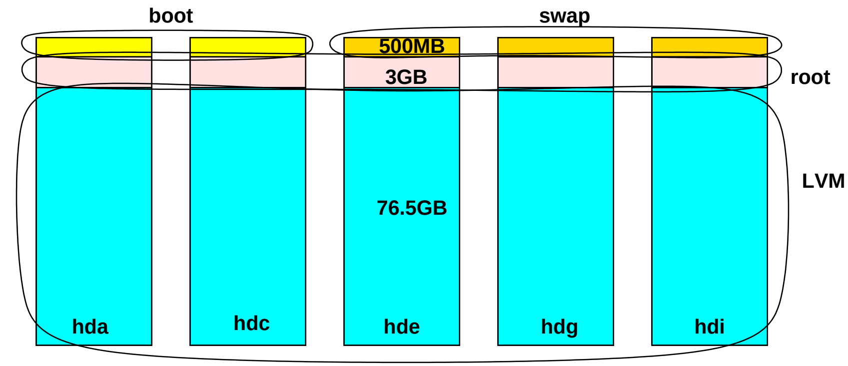
Redundant Array of Independent Disks

- RAID
- Allows disks or disk partitions to be combined to appear to be one disk
- Aims:
  - Performance Increase — because data can be transferred from many disks in parallel
  - Redundancy — RAID above "level 0" allows one or more disks to fail without losing any data: system continues to be available
- Can be implemented in *hardware* or *software*
  - Software RAID earned a bad name on Microsoft OS, but reliable and efficient on Linux

Example: RAID and LVM on Linux Server

- Have two IDE controller cards, two IDE interfaces on motherboard
- Five 80 GB hard disks, one on each of the six IDE interfaces, set as master,
- Partition all hard disks the same way:

Example Use of RAID and LVM— 2



Boot partitions, RAID 1, 500MB, 500MB total

Partitions for swap, RAID 5, 500 MB each, 1GB total

Partitions for root, RAID 5, 3GB each, 12GB total

Partitions for LVM, RAID 5, 76.5GB each, 306GB total

Example Use of RAID and LVM— 3

- Combine hda1, hdc1 into RAID 1, specify as the `/boot` partition: `md0`
  - install `grub` into Master Boot Record of both `hda` and `hdc` so can still boot if either fails
- Combine `hde1`, `hdg1`, `hdi1` into RAID 5 for the swap partition: `md1`
- Combine `hda2`, `hdc2`, `hde2`, `hdg2`, `hdi2` into RAID 5 for the root partition `/`: `md2`
- Combine `hda3`, `hdc3`, `hde3`, `hdg3`, `hdi3` into RAID 5 for large flexible LVM storage for `/home` and `/var`: `md3`

Example Use of RAID and LVM— 4

- Run `pvcreate` on `/dev/md3` to initialise the disks ready to hold LVM information
- Use `vgcreate` to create a logical volume group `main`
- Use `vgextend` to add the RAID device `md3` to the volume group
- Use `lvcreate` to create two logical volumes, one for `/var` and the other for `/home`.
- Format the two logical volumes and start using them.
- In the future, if `/var` is full and `/home` is not as full, then can dynamically resize both partitions, removing storage from the `/home` logical volume, and add to the `/var` logical volume, without needing to reboot or shut down the server

Summary of RAID, Volume Managers

- The use of RAID provides two benefits:
  - *Higher performance* (in example, because can read or write to five disks at once through five separate IDE interfaces)
  - *Redundancy*: if any disk fails, server can continue to operate normally, (with reduced disk performance), without data loss, until disk is replaced.
- The use of a volume manager, such as LVM on top of RAID, adds the benefit of *flexibility*, allowing system administrator to resize logical volumes, add new disks, re-arrange storage without disrupting service to the customers

References

- Daniel P. Bovet and Marco Cesati, *Understanding the Linux Kernel*, Second Edition, O'Reilly, Dec 2002, Chapter 13: *Managing I/O Devices*.
- William Stallings, *Operating Systems*, Fourth Edition, Prentice Hall, 2001, Chapters 11 and 12.
- Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, 2001, Chapter 5, *Input/Output*.
- Gary J. Nutt, *Operating Systems: A Modern Perspective*, 2nd Edition, Lab Update, Addison-Wesley, 2002, Chapter 5, *Device Management*.
- *LVM HOWTO*, `http://tldp.org/HOWTO/LVM-HOWTO/`