I/O Software Layers

Chapter 11 of [OS4e],
Chapters 2, 3 and 13 of [OSC6e]:

- Device Drivers
- Buffering and Spooling
- I/O System Call Interfaces
- Java I/O

Device Drivers
Recall from first lecture
- OS must provide abstraction from detailed operation of I/O devices
  - device-driver module
    - encapsulates routines that handle low-level device operations
      - routines communicate with device controller (I/O module) using memory-mapped I/O or dedicated I/O instructions, programmed I/O or interrupt-driven I/O, and possibly DMA
    - part of kernel of OS
      - adding devices originally meant recompiling OS with new drivers
      - a reconfigurable OS registers those drivers that are required when it boots up
      - Linux and Windows XP allow plug 'n play in that some device drivers can be loaded on demand, i.e., upon detecting device
    - N.B. device-driver modules are software, whereas I/O modules are hardware

device-status table
- maintained by device drivers
- contains one entry per device that includes
  - device status
  - list of pending requests

![Device Status Table Diagram]
Device drivers that use interrupt-driven I/O are often divided into two halves

- **“Top half”**
  - initiates operation and updates device’s entry in table
    e.g. disk driver that
  1. is supplied with linear block number on invocation
  2. converts to cylinder, head and sector numbers
  3. checks status of device in table
  4. if device is idle, issues command to disk controller and changes device status to busy;
     otherwise, queues request as pending
  - blocks calling process
- **“Bottom half”**
  - routine to which interrupt service routine branches
  - issues next pending request for device according to scheduling policy
  - unblocks original caller

---

**device-driver layer**

- collection of device-driver modules
  e.g., SCSI device driver, keyboard device driver, mouse device driver, PCI bus device driver, floppy device driver
- calls to this layer are made from within the kernel
  e.g. in Unix systems a device class (block or character), a major device number, a minor device number and a routine (function) are specified
  - each class has an array of entry points into device drivers
  - major number indexes into array
  - minor number is passed as argument to driver, which would normally use it to identify a specific device

---

- block device drivers (such as disk drivers) read into and write from a buffer cache (essentially a disk cache)
- classically, character device drivers read into C-lists filtering the raw character stream into a cooked (canonical line-edited) stream
- UNIX System V introduced the STREAMS mechanism for stacking modules on top of device drivers, with bidirectional communication at interfaces, to allow more complex processing
- Windows similarly allows such driver stacks
Above the device-driver layer, kernel is independent of hardware
- simplifies job of OS developer
- benefits hardware manufacturers
  - new device can be used by OS if someone writes device driver for it (unfortunately, has to be rewritten for each OS)
  - otherwise, new device’s controller would have to be compatible with one that OS could already handle

*A free book about writing Linux device drivers can found at* http://www.xml.com/ldd/chapter/book/

---

**Buffering and Spooling**

*buffer*
- temporary storage area for data being transferred between two devices or between device and application
- device-independent software in OS (i.e., above device-driver layer) maintains buffers in main memory (as described below)
- device controllers employ their own buffers/caches (as mentioned in the last lecture)
- application software also employs buffers to reduce number of system calls, e.g., writing N bytes in one go, instead of N writes one byte at a time

Buffers are maintained by OS for three reasons:
- to cope with speed mismatch between producer and consumer of data stream
  - e.g., file being received from modem for storage on hard disk; modem 1000x slower than disk, so modem should fill up buffer before disk write is requested; modem can continue to fill second buffer (double buffering)
• to adapt between devices that have different data-transfer sizes
e.g., sequence of packets received over network are reassembled in buffer to form original message
• to support copy semantics
  – if an application makes `write` system call with pointer to N bytes as argument, then modifies those bytes, OS must ensure that this does not affect what is written
  i.e., OS should copy the bytes into buffer within kernel before returning from call or give them "copy-on-write" protection

N.B. in handling disk read requests, buffers used for writing to disk should be examined (along with the disk cache) in case they are able to satisfy them

---

**spooling**

• Simultaneous Peripheral Operation On Line
• form of buffering in secondary storage
  – files are held in a spooling directory for a device
  e.g., typically, files are held in a print queue, before being output to a printer
  e.g., in the 1970s, jobs were held in a batch queue, after being input from a card reader
• a single, dedicated process (daemon) accesses the device
  – avoids problems of mutual exclusion and deadlock

---

![Diagram of I/O system layers](image)

Fig. 5-16. Layers of the I/O system and the main functions of each layer.

(Tanenbaum’s Modern Operating Systems, 2nd ed.)
I/O System Call Interfaces

OS hides differences between devices from applications

The major access conventions are

- file-system interface
- network sockets-interface

Each provides standard interface of system calls

- actual calls may differ from one OS to another
- escape or backdoor system call (e.g., ioctl in UNIX) allows access to any additional functionality of driver

Java I/O

java.io package contains abstract classes

- InputStream (read bytes)
  - e.g., System.in
- OutputStream (write bytes)
  - e.g., System.out, System.err
    - writing an integer reduces it modulo 256
    - flushing will empty any buffers used by the OS or the Java runtime environment (automatic upon closing)
- Reader (read characters)
- Writer (write characters)
  - size of char in Java is 2 bytes (Unicode), whereas in C it is only 1 byte (ASCII)

N.B. Untrusted applets can’t do much in the way of I/O

Buffering for better performance

BufferedInputStream

- adds buffer (internal byte-array) to underlying input stream
  - reading is from buffer, rather than stream
  - buffer is filled from stream initially and refilled upon becoming empty

BufferedOutputStream

- adds buffer to underlying output stream
  - writing is to buffer, rather than stream
  - buffer empties out to stream whenever full or flushed

BufferedReader and BufferedWriter are similar
Encoding
OutputStreamWriter
• characters are converted to bytes according to specified character set and written to underlying output stream
InputStreamReader
• bytes are read from underlying input stream and converted to characters according to specified character set

Reading and Writing Files
FileInputStream
• provides an input stream connected to a particular file
FileOutputStream
• provides an output stream connected to a particular file

Message-Passing between Threads
PipedOutputStream
PipedInputStream
• output stream written by one thread can be connected to input stream read by another

Communicating over Networks
java.net package contains classes
• URL
  - openStream
• URLConnection and Socket
  - getInputStream and getOutputStream

The following tutorials are highly recommended:
http://java.sun.com/docs/books/tutorial/essential/io/
http://java.sun.com/docs/books/tutorial/networking/