A JOURNEY OF A BYTE AND BUFFERING

Contents of today's lecture:

- A Journey of a Byte
- Buffer Management

Reference: Folk, Zoellick and Riccardi, File Structures, 1998. Section 3.8 and 3.9.

Complementary reading: Section 3.10 "I/O in Unix", if interested.

A Journey of a Byte

Suppose in our program we wrote :

```
...
outfile << c;
...
```

This causes a call to the file manager, the part of the operating system responsible for input/output operations.

The O/S (File Manager) makes sure the byte is written to the disk.

Pieces of software/hardware involved in I/O operations :

- **Application program**
  - requests the I/O operation (`outfile << c;`)
- **Operating system/file manager**
  - keeps tables for all opened files (types of accesses allowed, FAT with each file's corresponding clusters, etc)
  - brings appropriate sector to buffer
  - writes **byte** to buffer
  - gives instruction to I/O processor to write data from this buffer into correct place in disk.

  Note: the operating system is a program running in CPU and working on RAM (it copied the content of variable `c` into the appropriate buffer). The **buffer** is an exact image of a cluster in disk.
- **I/O Processor** (a separate chip in the computer; it runs independently of CPU so that it frees up CPU to other tasks - I/O and internal computing can overlap)
  - Finds a time when drive is available to receive data, and puts data in proper format for the disk.
  - Sends data to the disk controller
- **Disk Controller** (a separate chip on the disk circuit board)
  - Controller instructs the drive to move the read/write head to the proper track, waits for proper sector to come under the read/write head, then sends the **byte** to be deposited on the surface of the disk.

Refer to Figure 3.21 at page 90 of the textbook.

Buffer Management

Buffering means working with large chunks of data in main memory so that the number of accesses to secondary storage is reduced.

Today we will discuss the **System I/O Buffers**.

Note that the application program may have its own "buffer" - i.e. a place in memory (variable, object) that accumulates large chunks of data to be later written to disk as a chunk.

Recommended Reading :
Chapter 4.2 - using classes to manipulate buffers. This has nothing to do with the system I/O buffer which is beyond the control of the program and is manipulated by the operating system.

System I/O Buffers

Buffer Bottlenecks What if the O/S used only one I/O buffer ?

Consider a piece of program that reads from a file and writes into another, character by character:

```
      while(1) {
(1)      infile >> ch;
(2)      if (file.fail()) break;
(3)      outfile << ch;
      }
```

Suppose that the next character to be read from `infile` is physically stored in sector `X` of the disk. Suppose that the place to write the next character to `outfile` is sector `Y`.

With a single buffer :

- When line (1) is executed, sector `X` is brought to the buffer and `ch` receives the correct character.
- When line (3) is executed, sector `Y` must be brought to the buffer. Sector `Y` is brought and `ch` is deposited to the right position.
- Now line (1) is executed again. Suppose we did not reach the end of sector `X` yet. Then, sector `X` must be brought again to the buffer, so the current content of the buffer must be written to sector `Y` before this is done.

And so on.

This could be solved if there were more buffers available!

Most operating systems have an input buffer and an output buffer.
One buffer could be used for `infile` and one for `outfile`.
A new trip to get a sector of `infile` would only be done after all the bytes in the previous sector had been read.

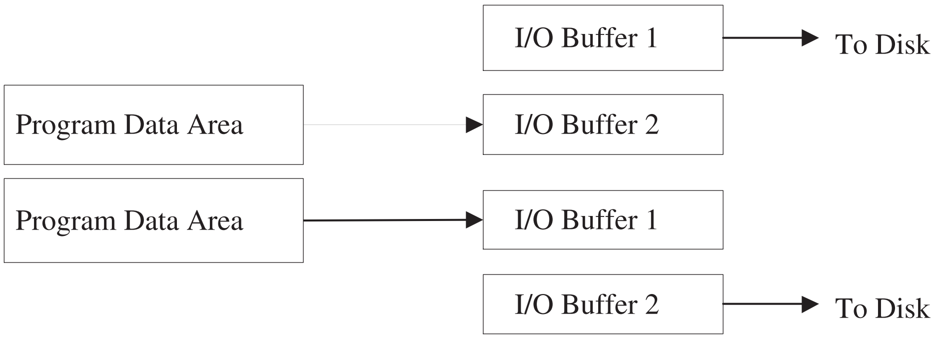
Similarly, the buffer for `output` would be written to the file only when full (or when file was closed).

Question : If the sector size is 512 bytes. How many extra trips to the disk we have to do if we have only 1 buffer in comparison to two or more, **in our previous program** ?

Buffering Strategies

1. Multiple Buffering

Double buffering: Two buffers + I/O-CPU overlapping



Several buffers may be employed in this way (multiple buffering).

Some operating systems use a buffering scheme called **buffer pooling** :

- There is a pool of buffers.
- When a request for a sector is received by the O/S, it first looks to see if that sector is in some of the buffers.
- If not there, then it brings the sector to some free buffer. If no free buffer exists then it must choose an occupied buffer, write its current contents to the disk, and then bring the requested sector to this buffer.

Various schemes may be used to decide which buffer to choose from the buffer pool. One effective strategy is the **Least Recently Used (LRU)** Strategy: when there are no free buffers, the least recently used buffer is chosen.

2. Move Mode and Locate Mode

Move Mode

Situation in which data must be always moved from system buffer to program buffer (and vice-versa).

Locate Mode

This refers to one of the following two techniques, in order to avoid unnecessary moves.
The file manager uses system buffers to perform all I/O, but provides its location to the program, using a pointer variable.
The file manager performs I/O directly between the disk and the program's data area.

3. Scatter/Gather I/O

We may want to have data separated into more than one buffer (for example: a block consisting of header followed by data).

Scatter Input: a single read call identifies a collection of buffers into which data should be scattered.

Similarly, we may want to have several buffers gathered for output.

Gather Output: a single write call gathers data from several buffers and writes it to output.