OPERATING SYSTEMS

File Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

File Management – Efficiency and Performance

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Efficiency

Efficiency dependent on:

- Disk allocation and directory algorithms
- Types of data kept in file's directory entry
- Pre-allocation or as-needed allocation of metadata structures
- Fixed-size or varying-size data structures

Performance

Performance dependent on:

- Keeping data and metadata close together
- Buffer cache – separate section of main memory for frequently used blocks
- Synchronous writes sometimes requested by apps or needed by OS
  - No buffering / caching – writes must hit disk before acknowledgement
  - Asynchronous writes more common, buffer-able, faster
- Free-behind and read-ahead – techniques to optimize sequential access
- Reads frequently slower than writes

OPERATING SYSTEMS

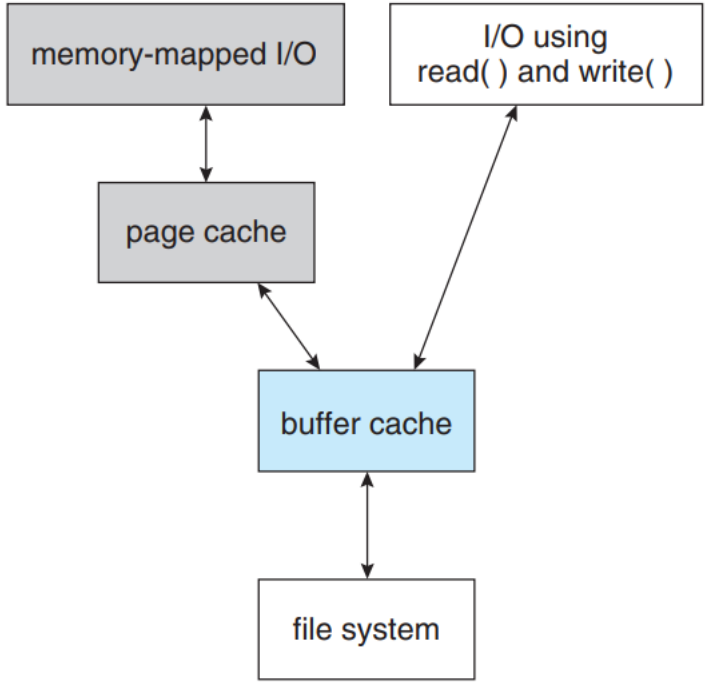
Page cache

- Some systems maintain a separate section of main memory for a **buffer cache**, where blocks are kept under the assumption that they will be used again shortly.
- Other systems **cache file data using a page cache**. The page cache uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks.
- Caching file data using virtual addresses is far more efficient than caching through physical disk blocks, as accesses interface with virtual memory rather than the file system.
- Several systems—including Solaris, Linux, and Windows —use page caching to cache both process pages and file data. This is known as **unified virtual memory**.

OPERATING SYSTEMS

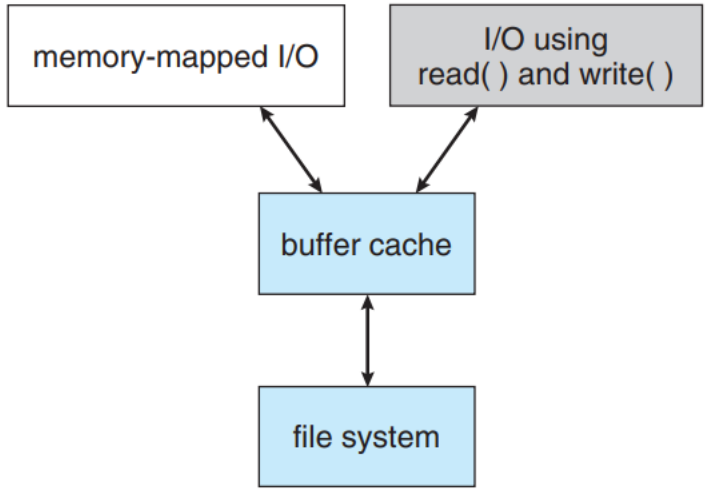
I/O without a unified buffer cache

- Consider standard system calls read() and write()
- Without a unified buffer cache, the read() and write() system calls go through the buffer cache.
- The memory-mapping call, requires two caches—the page cache and the buffer cache.
- A memory mapping proceeds by reading in disk blocks from the file system and storing them in the buffer cache. Because the virtual memory system does not interface with the buffer cache, the contents of the file in the buffer cache must be copied into the page cache.
- This situation, known as **double caching**, requires caching file-system data twice. Not only does it waste memory but it also wastes significant CPU and I/O cycles due to the extra data movement within system memory.



I/O using a unified buffer cache

- When a unified buffer cache is provided, both memory mapping and the read() and write() system calls use the same page cache.
- This has the benefit of avoiding double caching, and it allows the virtual memory system to manage file-system data



Synchronous and Asynchronous writes

- Another issue that can affect the performance of I/O is whether writes to the file system occur synchronously or asynchronously.
- Synchronous writes occur in the order in which the disk subsystem receives them, and the writes are not buffered. Thus, the calling routine must wait for the data to reach the disk drive before it can proceed.
- In an asynchronous write, the data are stored in the cache, and control returns to the caller.
- Most writes are asynchronous, metadata writes can be synchronous

OPERATING SYSTEMS

Page replacement

- Sequential access can be optimized by techniques known as free-behind and read-ahead.
- **Free-behind** removes a page from the buffer as soon as the next page is requested. The previous pages are not likely to be used again and waste buffer space.
- With **read-ahead**, a requested page and several subsequent pages are read and cached. These pages are likely to be requested after the current page is processed.
- Retrieving these data from the disk in one transfer and caching them saves a considerable amount of time

OPERATING SYSTEMS

Disk cache

- When data is written to a disk file, the pages are buffered in the disk cache, and the disk driver sorts its output queue according to disk address to minimize disk-head seeks
- Unless synchronous writes are required, a process writing to disk simply writes into the cache, and the system asynchronously writes the data to disk when convenient
- The user process sees very fast writes.
- When data are read from a disk file, the block I/O system does some read-ahead
- Writes are much more nearly asynchronous than are reads. Thus, output to the disk through the file system is often faster than is input for large transfers, counter to intuition

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu