

## Caching

A **cache** is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. For instance, the instructions of the currently running process are stored on disk, cached in physical memory, and copied again in the CPU's secondary and primary caches. The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, whereas a cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

Caching and buffering are distinct functions, but sometimes a region of memory can be used for both purposes. For instance, to preserve copy semantics and to enable efficient scheduling of disk I/O, the operating system uses buffers in main memory to hold disk data. These buffers are also used as a cache, to improve the I/O efficiency for files that are shared by applications or that are being written and reread rapidly. When the kernel receives a file I/O request, the kernel first accesses the buffer cache to see whether that region of the file is already available in main memory. If it is, a physical disk I/O can be avoided or deferred. Also, disk writes are accumulated in the buffer cache for several seconds, so that large transfers are gathered to allow efficient write schedules.

## Spooling and Device Reservation

A **spool** is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together. The operating system solves this problem by intercepting all output to the printer. Each application's output is spooled to a separate disk file. When an application finishes printing, the spooling system queues the corresponding spool file for output to the printer.

The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In others, it is handled by an in-kernel thread. In either case, the operating system provides a control interface that enables users and system administrators to display the queue, remove unwanted jobs before those jobs print, suspend printing while the printer is serviced, and so on.

## Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors, so that a complete system failure is not the usual

result of each minor mechanical malfunction. Devices and I/O transfers can fail in many ways, either for transient reasons, as when a network becomes overloaded, or for “permanent” reasons, as when a disk controller becomes defective. Operating systems can often compensate effectively for transient failures. For instance, a disk read() failure results in a read() retry, and a network send() error results in a resend(), if the protocol so specifies.

Unfortunately, if an important component experiences a permanent failure, the operating system is unlikely to recover.

### **I/O Protection**

Errors are closely related to the issue of protection. A user process may accidentally or purposely attempt to disrupt the normal operation of a system by attempting to issue illegal I/O instructions. We can use various mechanisms to ensure that such disruptions cannot take place in the system.

To prevent users from performing illegal I/O, we define all I/O instructions to be privileged instructions. Thus, users cannot issue I/O instructions directly; they must do it through the operating system. To do I/O, a user program executes a system call to request that the operating system perform I/O on its behalf. The operating system, executing in monitor mode, checks that the request is valid and, if it is, does the I/O requested. The operating system then returns to the user.

In addition, any memory-mapped and I/O port memory locations must be protected from user access by the memory-protection system. Note that a kernel cannot simply deny all user access. Most graphics games and video editing and playback software need direct access to memory-mapped graphics controller memory to speed the performance of the graphics, for example. The kernel might in this case provide a locking mechanism to allow a section of graphics memory (representing a window on screen) to be allocated to one process at a time.

### **Kernel I/O Subsystem**

the I/O subsystem coordinates an extensive collection of services that are available to applications and to other parts of the kernel. The I/O subsystem supervises these procedures:

- Management of the name space for files and devices
- Access control to files and devices
- Operation control (for example, a modem cannot seek())
- File-system space allocation
- Device allocation

- Buffering, caching, and spooling
- I/O scheduling
- Device-status monitoring, error handling, and failure recovery
- Device-driver configuration and initialization

The upper levels of the I/O subsystem access devices via the uniform interface provided by the device drivers.