IO Management and Disk Scheduling

Arnav Gupta

April 9, 2024

Contents

1 IO Devices

IO device categories are:

1. **human readable**: for communicating with the computer user (printers, terminals, display, etc)
2. **machine readable**: for communicating with electronic equipment (disk drives, USB keys, etc)
3. **communication**: for communicating with remove devices (digital line drivers, modems)

Difference between classes are:

- **data rate**: can have several orders of magnitude difference in data transfer rate
- **application**: device use influences software, OS policy, and utilities
- **complexity of control**: differences in control interface are filtered by the complexity of the IO module that the controls the device
- **unit of transfer**: data may be transferred as a stream of bytes or characters, or in larger blocks
- **data representation**: different data encoding schemes are used by different devices including differences in character code and parity conventions
- **error conditions**: the nature of errors, how they are reported, consequences, and available response range differs between devices

  Uniform approach to IO is difficult.

2 Organization of the IO Function

Techniques for performing IO:

1. **programmed IO**: processor issues IO command to a module (on behalf of a process), the process busy waits for the operation to complete before proceeding
2. **interrupt-driven IO**: processor issues an IO command (on behalf of a process)
   - if the IO instruction from the process is nonblocking, the process continues
   - if the IO instruction is blocking, the next instruction the processor executes is from the OS, which puts the process in a blocking state and schedules another process
3. **direct memory access (DMA)**: a DMA module controls the exchange of data between main memory and an IO module
   - the processor sends a request for block data transfer to the DMA module, and is interrupted after the entire block has been transferred
   - dominant in most computer systems

2.1 Evolution of the IO Function

Evolutionary steps of the IO function have been:

1. processor directly controls device (microprocessor-controlled devices)
2. controller or IO module added, so processor uses programmed IO without interrupts, so the processor becomes less connected from the specific details of external device interfaces
3. same configuration as 2, but with interrupts, so the processor does not spend time waiting for an IO operation, increasing efficiency
4. IO module given direct control of memory with DMA, so it can move a block of data to/from memory without the processor, except at the beginning and end of transfer
5. IO module is a separate processor with specialized instructions for IO

    (a) CPU directs the IO processor to execute to execute an IO program in main memory

    (b) the IO processor fetches and executes instructions without processor intervention, allows the processor to specify a sequence of IO activities and to be interrupted only when the entire sequence has been performed

6. IO module has local memory, so a large set of IO devices can be controlled with minimal processor involvement

    (a) IO processor takes care of most tasks involved in controlling terminals

With evolution, CPU is less involved in IO tasks, improving performance.

IO module in step 5 is an **IO channel**, IO module in step 6 is an **IO processor**.

2.2 Direct Memory Access

DMA unit can mimic processor and take control of system bus like a processor, to transfer data to and from memory.

When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending the following info:

- whether a read or write is requested, using the read or write control line between the processor and the DMA module
- the address of the IO device involved, communicated on the data lines
- the starting location in memory to read from or write to, communicated on the data lines and stored by the DMA module in its address register
- the number of words to be read or written, communicated via the data lines and stored in the data count register

Processor does other work while the DMA module transfers the data block. When the transfer finishes, the DMA module sends the processor and interrupt.

DMA configurations

- **single-bus, detached DMA**: all modules share the same system bus, where the DMA module is a processor that uses programmed IO to exchange data between memory and IO modules, which is inefficient as each transfer consumes 2 bus cycles (transfer request, then transfer)

- **single-bus, integrated DMA/IO**: DMA logic can be part of an IO module or separate that controls 1+ IO modules
- **IO bus**: connect IO modules to the DMA module using an IO bus, which reduces the number of IO interfaces in the DMA module to 1 and is more expandable

For last 2 DMA configurations, the system bus is used by the DMA module only to exchange data with memory and exchange control signals with the processor.

3 Operating System Design Issues

3.1 Design Objectives

Objectives in designing IO:

- **efficiency** is important since IO operations are a bottleneck
  - with multiprogramming, some processes are waiting on IO operations while others execute
  - swapping can bring in ready processes, but this is an IO operation
- **generality** allows handling all devices in a uniform manner, in how processes view IO devices and how the OS manages IO devices and operations
  - can use a hierarchical, modular approach to the design of the IO function so devices only see read, write, open, close, lock, unlock

3.2 Logical Structure of the IO Function

OS functions should be separated according to complexity, characteristic time scale, and abstraction level.

For a local peripheral device with simple communication, the layers are:

- **logical IO**: deals with the device as a logical resource, managing general IO functions on behalf of user processes using a device ID and simple commands (open, close, read, write)

- **device IO**: requested operations and data are converted to IO instructions, channel commands, and controller orders, buffering could be used to improve utilization
- **scheduling and control**: queueing, scheduling, and control of IO operations, including interrupts and IO status, interacts with IO module and device hardware

Communications device is the same, but with communication architecture rather than logical IO.

Structure for IO on a secondary storage device that supports a file system has device IO and scheduling and control, but before these it also has:

- **directory management**: file names converted to IDs that reference files directly or through file descriptor/index table, also concerned with user operations on directories (add, delete, reorganize)
- **file system**: deals with logical structure of files, access rights, and user operations (open, close, read, write)
- **physical organization**: logical references to files and records must be converted to physical secondary storage addresses, accounting for device structure, includes allocation of secondary storage space and main storage buffers

4 IO Buffering

A process reading data waits on the IO operation to complete, which can interfere with swapping decisions by the OS (otherwise data lost). If paging used, the page containing the target locations must be locked into main memory. Otherwise, could cause deadlock if process swapped out before IO operation begins, so process waits on IO event and IO operation waits on process to be swapped in.

Same considerations for output operations.

Can avoid overheads and inefficiencies by performing input transfers in advance of requests and output transfers after request, which is **buffering**.

Block-oriented device stores info in blocks of fixed size, transfers made one block at a time, so data can be referenced by block number. (disks, USBs)

Stream-oriented device transfers data in and out as a stream of bytes. (terminals, printers, etc.)

4.1 Single Buffer

When a user process issues an IO request, the OS assigns a buffer in the system portion of main memory to the operation.

For block-oriented devices:

- input transfers made to the system buffer, which is moved into user space when the transfer completes
- another block is requested (reading ahead/anticipated input), since expected that block will eventually be needed
- unnecessary block only read at end of sequence
- speedup since process can process one block while next is read in
- process can be swapped out since input is in system memory
- OS must keep track of assignment of system buffers to user processes
- if IO and swapping are on same disk, makes no sense to queue disk writes to the same device

For stream-oriented devices:

- single buffering can be used line-at-a-time or byte-at-a-time
  - line-at-a-time for scroll-mode terminals, where buffer can hold a line
    - user process is suspended during input
    - user process places lines in the buffer and continue processing, so it is not suspended unless another line of output must be sent before the buffer is emptied
  - byte-at-a-time for forms-mode terminals
    - follows producer/consumer model

4.2 Double Buffer

Double buffering/buffer swapping: assigns 2 system buffers to operations, where a process transfers data to/from one buffer while the OS empties/fills the other

For block-oriented transfer, processor does not have to wait on IO, always an improvement over single buffering.

For stream-oriented transfer, could again use line-at-a-time IO (no suspension for input or output, unless process ahead of double buffers) or byte-at-a-time IO (no advantage over single buffer of twice the length)

4.3 Circular Buffer

Use more than 2 buffers, with each individual buffer being 1 unit in the circular buffer.

Better for bursty IO.

4.4 Utility of Buffering

Smoothes peaks in IO demand, but does not allow indefinite high pace of IO demand. All buffers eventually fill up and processor will have to wait after processing each data chunk.

5 Disk Scheduling

5.1 Disk Performance Parameters

Disk IO operation depends on the computer system, the OS, and the nature of the IO channel and disk controller hardware.

Disk rotates at constant speed, where for reads and writes, the head must be positioned at the desired track and at the beginning of the desired sector on that track.

Track selection moves the head (moveable-head system) or electronically selects one head (fixed-head system).

Seek time: time to position the head at the track (moveable-head system)

Once track is selected, disk controller waits until appropriate sector rotates to line up with head.

Rotational delay/latency: time for the beginning of the sector to reach the head

Access time: sum of seek time and rotational delay, time to get into position to read/write

Transfer time: time required for data transfer, time to move head over sector

Device waits in queue for device to be available, which could include other drives if device shares IO channel(s). Seek starts after device and channel available.

Rotational positional sensing works by attempting to connect to the channel when the head reaches the sector, if the control unit or channel is busy with another IO, then the head must rotate another revolution (RPS miss)..

5.1.1 Seek Time

Consists of initial startup time and time taken to traverse tracks that must be crossed once access arm is up to speed.

Traversal time includes settling time (time to confirm track ID).

5.1.2 Transfer Time

Depends on rotation speed of the disk:

$$T = \frac{b}{rN}$$

where T is transfer time, b is the number of bytes to be transferred, N is the number of bytes on a track, and r is the rotation speed (revs/s).

Total average access time is

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

where T_s is the average seek time.

5.1.3 Timing Comparison

Can have huge difference between sequential read access and random read access.

If requests are randomly selected, will get poor performance.

5.2 Disk Scheduling Policies

In a multiprogramming environment, the OS maintains a queue of requests for each IO device.

Random scheduling: select items from queue in random order, gives poor performance, a benchmark against which to evaluate other techniques

5.2.1 First-In-First-Out

Process queue items in sequential order.

More fair and works well if there are only a few processes with requests to clustered file sectors.

Open approximates random scheduling if many processes compete for disk.

5.2.2 Priority

Can use priority to meet other objectives within the OS, rather than optimize disk utilization.

Short batch jobs prioritized over longer computations allowing shorter jobs to be flushed through system quickly.

Users might split jobs into small pieces to beat the system.

5.2.3 Last-In-First-Out

Tasking most recent request means less arm movement for moving through a sequential file by taking advantage of locality, improving throughput and reducing queue lengths.

Could allow for starvation if it never gets to the head of the line again.

5.2.4 Shortest-Service-Time-First

Select the disk IO request that requires least movement of the disk arm. Always choose minimum seek time (with tiebreaker for equal distances).

Does not guarantee minimum average seek time.

5.2.5 SCAN

Arm can only move in one direction and satisfy all requests in that route until it reaches the last track or no more requests in that direction. Direction is reversed and scan proceeds in opposite direction.

Prevents starvation. Favours jobs whose requests are for tracks nearest to innermost and outermost tracks. Favours latest-arriving jobs.

For usual request pattern, SCAN similar to SSTF.

Biased against area most recently traversed, so less locality.

5.2.6 C-SCAN

Scanning only in 1 direction, so when last track visited, arm returns to opposite end of disk and scan begins again.

Reduced max delay for new requests.

5.2.7 N-step-SCAN and FSCAN

If 1+ processes have high access rate to 1 track, they can monopolize the device by repeated requests. Can be avoided by segmenting the disk request queue, with 1 segment at a time being processed completely.

N-step-SCAN segments disk request queue into subqueues of length N. Subqueues are processed using SCAN. New requests added to some other queue when queue is processed. If $< N$ requests are available at the end of a scan, they are processed with the next scan. For large N, N-step-SCAN is SCAN, for $N = 1$, N-step-SCAN is FIFO.

FSCAN uses 2 subqueues. When a scan begins, all requests are in 1 queue, with the other empty. During the scan, all new requests put into other queue so new requests are deferred until old are processed.

6 RAID

7 Disk Cache

A buffer in main memory for disk sectors.

7.1 Design Considerations

When an IO request is satisfied from the disk cache, the data in the disk must be delivered to the requesting process. Done by transferring block within main memory from disk cache to user memory or passing a pointer to the slot in disk cache (saves time).

When a new sector is brought into disk cache, an existing block must be replaced. Most commonly used replacement algorithm is LRU, which can be implemented using a stack of blocks or pointers.

Can also use **least frequently used** (LFU), where the block that has experienced the fewest references is replaced. Implemented by associating a counter with each block, and the block with smallest count is replaced.

LFU less effective due to locality (many references to one block in burst, but none afterwards).

Can be fixed with frequency-based replacement. Based on LRU, where a certain portion of the top of the stack is the new section. On cache hit, the referenced block is moved to the top of the stack. If the block was already in the new section, its reference count is not incremented, otherwise incremented by 1. This results in reference counts for blocks that are repeatedly re-referenced in a short interval remaining unchanged. On a miss, a block from the old section is replaced, using count and LRU.

Problem with this strategy is that there is not a sufficiently long interval for blocks aging out of the new section to build up reference count.

Can be fixed by having new, middle, and old sections. Reference counts not incremented in new section, replacement only from old section. Allows blocks to build up reference count before coming eligible for replacement.

Replacement can be:

- **on-demand**: sector replaced only when slot needed
- **preplanned**: many slots released at a time, due to write back, can cluster writes and minimize seek time

7.2 Performance Considerations

Miss ratio is a function of the size of disk cache.

LRU can outperform frequency-based replacement, though frequency-based replacement is also superior. Sequence of references and block size affect performance achieved.