Storage Systems

NPTEL Course
Jan 2012
(Lecture 19)

K. Gopinath
Indian Institute of Science

“old” block layer

I/O path and associated routines

- ll_rw_block: routine for reading or writing buffers corresponding to block devices
- request structure: list of buffers adjacent on disk
- make_request: attempts clustering with existing request structures or creates one
  - makes I/O requests corresponding to the buffers
- add_request applies elevator alg using insertion sort
  - if device Q empty calls request_fn of driver
  - Otherwise, end_request will invoke it from interrupt context
- request_fn: strategy routine of driver
- queue: function to return device queue

Device plugging

- Clustering of requests

I/O Handling in Linux

- Linux uses request structures to pass the I/O requests to the devices
  - Each block device maintains a list of request structures
- When a buffer is to be read or written, kernel calls ll_rw_block() routine and passes it an array of pointers to buffer heads
- ll_rw_block calls make_request() routine for each buffer
- make_request() first tries to cluster the buffer with the existing buffers in any of the request structures present in the device queue
  - A request structure consists of a list of buffers which are adjacent on the disk
  - If clustering is possible, no new request structure is created
  - Otherwise, a new request taken from global pool of structures and initialized with buffer and passed to add_request()
- add_request applies elevator alg using insertion sort based on minor number of device and block number of buffer.
- If device idle, kernel calls strategy routine request_fn() of driver
- otherwise, responsibility of driver to reinvoke it from interrupt context
  - request_fn() should return if there are no requests in the device queue
  - request_fn() cannot block as it needs to be called from interrupt context

Plugging

- To allow accumulation of requests in device queue, a plug used.
- When request comes in and the device queue is empty
  - A plug is put at the head of the device queue
  - An unplug function registered in the disk task queue
- Requests keep accumulating for some time and then
- A thread executes the unplug routine
  - Removes the plug
  - Calls request_fn() to service the requests

ll_rw_block (deprecated!)

void ll_rw_block (int rw, int nr, struct buffer_head * bhs[]);

- rw: whether to READ or WRITE or SWRITE or maybe READA (readahead)
  - SWRITE is like WRITE: current data in buffers sent to disk
- nr: number of struct buffer_heads in the array
- bhs[]: array of pointers to struct buffer_head

Drops any buffer

- cannot get a lock on (with the BH_Lock state bit) unless SWRITE
- appears to be clean when doing a write request
- appears to be up-to-date when doing read request

- marks as clean buffers that are processed for writing
  - buffer cache won't assume that they are actually clean until the buffer gets unlocked
- sets b_end_io to a simple completion handler that marks the buffer up-to-date (if approriate)
- unlocks the buffer and wakes any waiters.

All buffers must be

- for the same device
- a multiple of the current approved size for the device

Consider more complex block devices

- Redundant Array of Independent Disks (RAID)
  - Mirroring (RAID1)
  - Block interleaved parity (RAID5)
  - Declustered RAID1
- Consider a pseudo device driver layered over a regular one
  - RAID 1 driver for a “virtual” device, say, /dev/raid1
    - Issues read/write requests to disks
      - Waits for both the requests to finish (“synch”), or
      - Waits for first response synch and completes the 2nd asynch
    - Allows configuration
      - Can “drop mirror” to allow “point-in-time” backup
  - Std Disk driver actually handles R/W to each disk (say, /dev/disk1 and /dev/disk2)
  - Called a “volume manager” in industry

How layering can be problematic

- Blocking in Interrupt Context
  - strategy routine called from interrupt handler but cannot block
  - (upper) strategy calls (lower) ll_rw_block in layered dd
  - can block as the global array of request structures can become exhausted
  - One solution: return imm and Q task in schedule Q for later execution
    - need to change ll_rw_block
  - Another solution: consume all requests to device Q in one single invocation of strategy routine
    - kernel calls request_fn from process context only if device Q empty
    - problem: one process can get delayed due to others but only if Q full
- Fixed Size Buffer Problem
  - RAID5: need to distinguish between full and partial stripes for efficiency
  - Linux fixed buffer size: logical buffer already split into multiple buffers
  - Have to rediscover logical buffer
  - Similar problem with reporting errors: cannot report errors at stripe level; only at fixed buffer level

“New” block layer

- Avoids “segmentation and reassembly” problem
- Generic block layer uses per-Q parameters rather than fixed values for all
- I/O scheduler algorithm itself can be replaced/set as appropriate
  - 2.4 allows alternate schedulers (anticipatory, null, CFQ)
  - 2.6 allows improved modularization of i/o scheduler with more pluggable callbacks
- I/O barriers
- Pass info from higher layers (fs/db) eg. readahead request

Current block layer designs designed for throughput, not latency

- May need careful rethinking with newer flash/PCM type of devices
  - eg. avoid plugging/queuing to reduce latency

Conclusions

- Designing “lower level” layers for use by multiple upper level consumers complex
  - Need lots of experience!
  - Concurrency or unusual hardware capabilities may present difficulties