CS 3204
Operating Systems

Lecture 23

Godmar Back

Announcements

- Project 4 Help Sessions
  - Th (tonight), Fr: 5:30-7:30 in McB 223
- Reading assignment: Ch 10, 11, 12

Disks & Filesystems

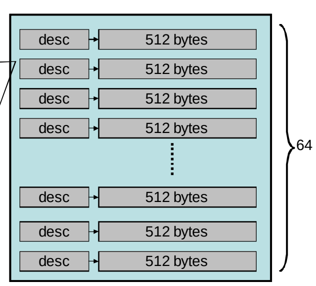
Buffer Cache

Disk Caching – Buffer Cache

- How much memory should be dedicated for it?
  - In older systems (& Pintos), set aside a portion of physical memory
  - In newer systems, integrated into virtual memory system: *e.g.*, page cache in Linux
- How should eviction be handled?
- How should prefetching be done?
- How should concurrent access be mediated (multiple processes may be attempting to write/read to same sector)?
  - How is consistency guaranteed? (All accesses must go through buffer cache!)
- What write-back strategy should be used?

Buffer Cache in Pintos

Cache Block Descriptor
- disk_sector_id, if in use
- dirty bit
- valid bit
- # of readers
- # of writers
- # of pending read/write requests
- lock to protect above variables
- signaling variables to signal availability changes
- usage information for eviction policy
- data (pointer or embedded)



A Buffer Cache Interface

```
// cache.h
struct cache_block;                    // opaque type
// reserve a block in buffer cache dedicated to hold this sector
// possibly evicting some other unused buffer
// either grant exclusive or shared access
struct cache_block * cache_get_block (disk_sector_t sector, bool exclusive);
// release access to cache block
void cache_put_block(struct cache_block *b);
// read cache block from disk, returns pointer to data
void *cache_read_block(struct cache_block *b);
// fill cache block with zeros, returns pointer to data
void *cache_zero_block(struct cache_block *b);
// mark cache block dirty (must be written back)
void cache_mark_block_dirty(struct cache_block *b);
// not shown: initialization, readahead, shutdown
```

Buffer Cache Rationale

Compare to buffer pool assignment in CS2604

```
class BufferPool { // (2) Buffer Passing
public:
  virtual void* getblock(int block) = 0;
  virtual void dirtyblock(int block) = 0;
  virtual int blocksize() = 0;
};
```

Differences:

- Do not combine allocating a buffer (a resource management decision) with loading the data into the buffer from file (which is not always necessary)
- Provide a way for buffer user to say they're done with the buffer
- Provide a way to share buffer between multiple users
- More efficient interface (opaque type instead of block idx saves lookup, constant size buffers)

Buffer Cache Sizing

- Simple approach
  - Set aside part of physical memory for buffer cache/use rest for virtual memory pages as page cache – evict buffer/page from same pool
- Disadvantage: can't use idle memory of other pool - usually use unified cache subject to shared eviction policy
- Windows allows user to limit buffer cache size
- Problem:
  - Bad prediction of buffer caches accesses can result in poor VM performance (and vice versa)

Buffer Cache Replacement

- Similar to VM Page Replacement, differences:
  - Can do exact LRU (because user must call cache_get_block()!)
  - But LRU hurts when long sequential accesses – should use MRU (most recently used) instead.
- Example reference string: ABCDABCDABCD, can cache 3:
  - LRU causes 12 misses, 0 hits, 9 evictions
  - How many misses/hits/evictions with MRU?
- Also: not all blocks are equally important, benefit from some hits more than from others

Buffer Cache Writeback Strategies

- Write-Through:
  - Good for floppy drive, USB stick
  - Poor performance – every write causes disk access
- (Delayed) Write-Back:
  - Makes individual writes faster – just copy & set bit
  - Absorbs multiple writes
  - Allows write-back in batches
- Problem: what if system crashes before you've written data back?
  - Trade-off: performance in no-fault case vs. damage control in fault case
  - If crash occurs, order of write-back can matter

Writeback Strategies (2)

- Must write-back on eviction (naturally)
- Periodically (every 30 seconds or so)
- When user demands:
  - fsync(2) writes back all modified data belonging to one file – database implementations use this
  - sync(1) writes back entire cache
- Some systems guarantee write-back on file close

Buffer Cache Prefetching

```
b = cache_get_block(n, _);
cache_read_block(b);
cache_readahead(next(n));
```

```
queue q;
cache_readahead(sector s) {
  q.lock();
  q.add(request(s));
  signal qcond;
  q.unlock();
}
cache_readahead_daemon() {
  while (true) {
    q.lock();
    while (q.empty())
      qcond.wait();
    s = q.pop();
    q.unlock();
    read sector(s);
  }
}
```

- Would like to bring next block to be accessed into cache before it's accessed
  - Exploit "Spatial locality"
- Must be done in parallel
  - use daemon thread and producer/consumer pattern
- Note: next(n) not always equal to n+1
  - although we try for it – via clustering to minimize seek times
- Don't initiate read_ahead if next(n) is unknown or would require another disk access to find out

Filesystems

Files vs Disks

File Abstraction

- Byte oriented
- Names
- Access protection
- Consistency guarantees

Disk Abstraction

- Block oriented
- Block #s
- No protection
- No guarantees beyond block write

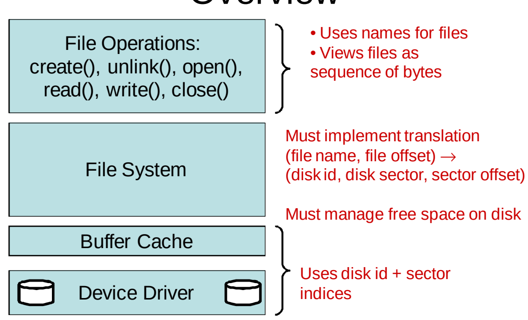
Filesystem Requirements

- Naming
  - Should be flexible, e.g., allow multiple names for same files
  - Support hierarchy for easy of use
- Persistence
  - Want to be sure data has been written to disk in case crash occurs
- Sharing/Protection
  - Want to restrict who has access to files
  - Want to share files with other users

FS Requirements (cont'd)

- Speed & Efficiency for different access patterns
  - Sequential access
  - Random access
  - Sequential is most common & Random next
  - Other pattern is Keyed access (not usually provided by OS)
- Minimum Space Overhead
  - Disk space needed to store metadata is lost for user data
- Twist: all metadata that is required to do translation must be stored on disk
  - Translation scheme should minimize number of additional accesses for a given access pattern
  - Harder than, say page tables where we assumed page tables themselves are not subject to paging!

Overview



The Big Picture