File System Allocation and Performance Strategies

File System Operations

- Create, destroy or change the contents of files
- Some changes covert unused disk blocks
- Needs to be efficient

Creating a new file

- Allocate a free file control block
- UNIX
    - Search the super block for free Inode list
    - Take the first free Inode
- DOS
    - Search the parent directory for an unused directory entry
- Initialize the new file control block
    - File type, protection, ownership
- Give the file a name

Extending a File

- App requests new data to be assigned to a file
- Find free chunk of space
    - Traverse the free list to find and appropriate chunk
    - Remove the chosen chunk from the free list
- Associate the selected block with the address in the file
    - Go to the appropriate address in the file
    - Update it to point to the newly allocated chunk

Deleting a file

- Release the space allocated to the file
    - UNIX: return each clock to the free block list
    - DOS doesn't free space
        - Uses garbage collection
        - Searches for deallocated blocks and adds all of them to the free list at a future time
- Deallocate the file control block
    - UNIX: zero inode and return it the free list
    - DOS: zero the first byte of the name in the parent directory

Free List Maintenance

- File System manager manages free space
- Block that is chosen matters
    - Can't write fully written flash blocks
    - May want to do wear leveling and keep data continuous

Allocation and Transfer Size

- Per operation overheads
  - DMA startup, interrupts, device specific costs
- Large transfer units are more efficient
- Unit to use to allocate storage space
  - Small chunks reduce efficiency
  - Large chunks cause internal fragmentation
  - Variable sized chunks cause external fragmentation

Data Access Performance

Flash Drive Issues

- Faster than hard drives, slower than RAM
- Any location is equally fast to access
- Write once / read many access
- Can only erase large chunks of memory

Read Caching

- Persistent storage I/O takes a long time
- Want to store data in RAM
- Eliminate persistent storage I/O using an in memory cache
  - Depends on locality and reuse of the same blocks
  - Check cache before scheduling I/O

Read Ahead

- Request blocks from the device before any process asked for them
  - Reduces wait time
- Works when:
  - Client is requesting for sequential access
  - If it's not sequential access, then don't predict
- Risks
  - May waste device access time reading unwanted blocks
  - May waste buffer space on unneeded blocks

Write Caching

- Most device writes go to a write back cache
  - Flushed out to the device later
- Combines small writes into large writes
  - If app does less than full block writes
- Eliminates small, incremental writes
  - If the application rewrites the same data or deletes the file, then the operation doesn't need to be performed
- Accumulates a large batch of writes
  - Longer queue means better disk scheduling

- If the system crashes, data in cache is lost

Common Disk Caching Types

- General Block Caching
  - Contains popular files that are frequently read
  - Files that are written then promptly re read
  - Provides buffers for read ahead and deferred writes

- Special purpose caches
  - Directory caches speed up searches on same dirs
  - Inode caches speed up reuse of same file

- Special purpose cache
  - Caching inodes or directory entries
  - Ex. Using the same directory entry over and over again

Pinning File Data in Caches

- Caching usually controlled by LRU strategy
- Some file data pinned in memory
  - Pinned data cant be replaced in the cache
- Inodes of processes
  - Ensures quick access to a structure that will will be used again
- Contents of the current working directory could be pinned