Section 11: File Systems, Reliability, and Queueing Theory

CS162

April 12, 2019

Contents

1 Warmup **2**

2 Vocabulary **3**

3 Problems **5**

3.1 Extending an inode 5

3.2 Queuing Theory 7

3.3 Tying it all together 8

1 Warmup

What are the ACID properties? Explain each one and discuss the implications of a system without that property.

Atomicity: all work done or none of it. | intermediate data state
Consistency: consistent state → consistent state | invalid state – e.g. wrong metadata
Isolation: independent processes | transactions interfere w/ each other
Durability: commits should persist after crash | crash → lose committed data

Name 2 different RAID levels that offer redundancy. For each level, explain how a recovery program could recover data from a degraded array.

fail → read copy
Raid 1: data copied
Raid 2: bit-level striping*
Raid 3: byte-level striping*
Raid 4: Block level striping*
Raid 5: Block level striping + distributed parity
2-5 fail → read other disks + parity → correct
* = dedicated parity disk
* = important

Explain the difference between a hard link and a soft link (symbolic link).

hard link: regular directory entry, file points to same inode as another
soft link: reference to another directory/file
← map one name to another

How could you implement hard links for the FAT file system? What problem would you encounter?

two dir entries could have same block
⇒ share same data
– no ref count → one delete ⇒ other file loses it too

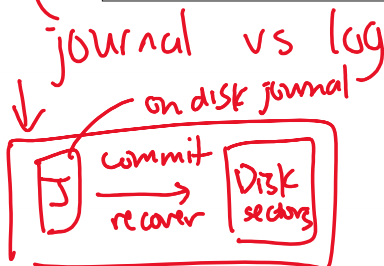
What is a journaled file system? Explain the purpose of the file system's "journal".

journal stores updates to filesys metadata.
Journal used to recover data if lost

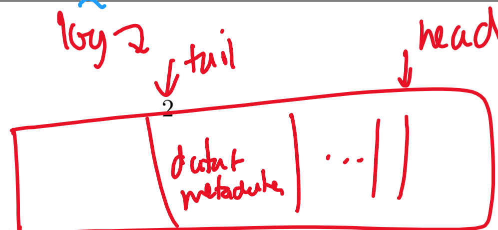
Discuss the advantages and drawbacks of memory mapped file accesses compared to traditional disk accesses for small random file reads and writes to many files of varying size.

map files in memory - good for updating w/o going to disk
page in & out these files → essentially just like going to disk !!

journal vs log



log →

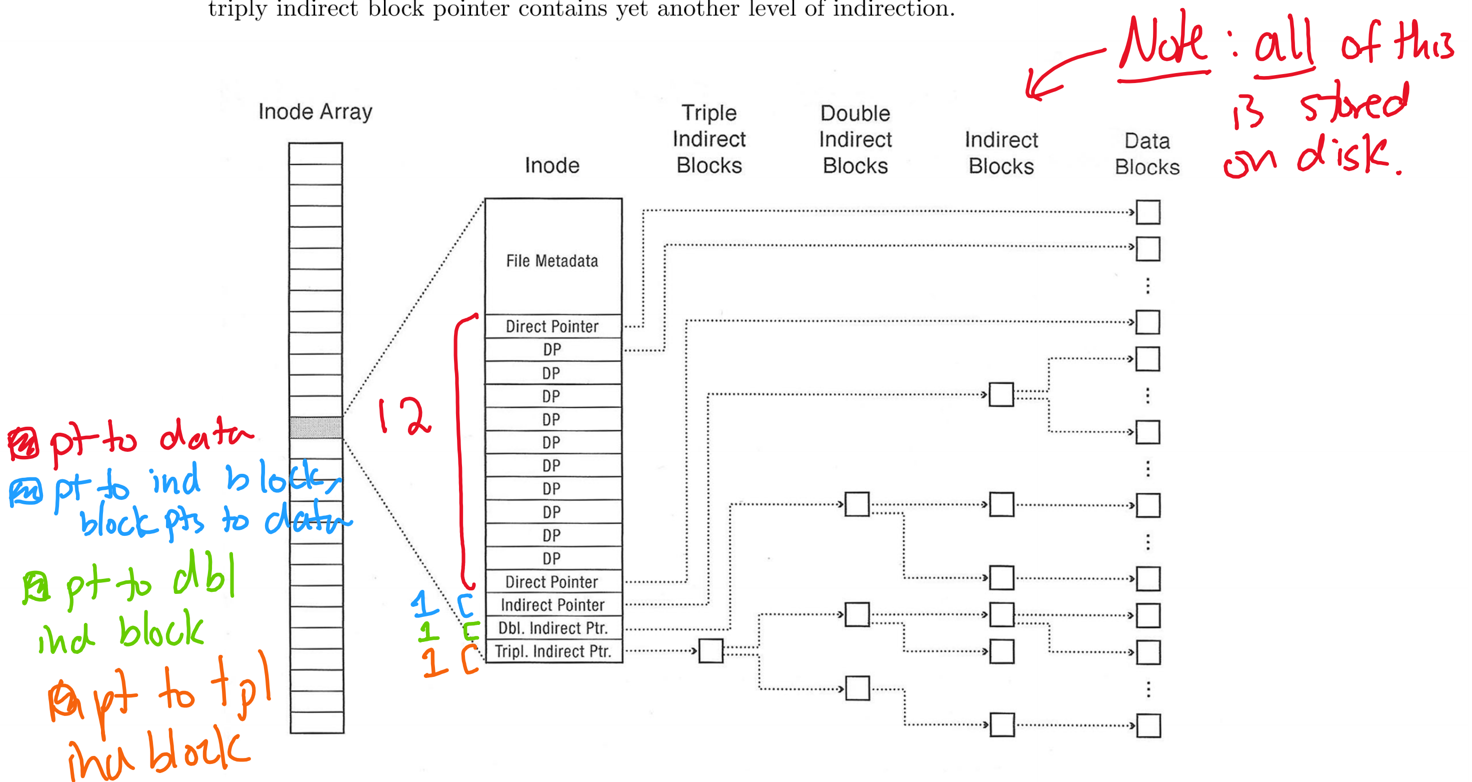


2 Vocabulary

- **Unix File System (Fast File System)** - The Unix File System is a file system used by many Unix and Unix-like operating systems. Many modern operating systems use file systems that are based off of the Unix File System.

- **inode** - An inode is the data structure that describes the metadata of a file or directory. Each inode contains several metadata fields, including the owner, file size, modification time, file mode, and reference count. Each inode also contains several data block pointers, which help the file system locate the file's data blocks.

  Each inode typically has 12 direct block pointers, 1 singly indirect block pointer, 1 doubly indirect block pointer, and 1 triply indirect block pointer. Every direct block pointer directly points to a data block. The singly indirect block pointer points to a block of pointers, each of which points to a data block. The doubly indirect block pointer contains another level of indirection, and the triply indirect block pointer contains yet another level of indirection.



- **Transaction** - A transaction is a unit of work within a database management system. Each transaction is treated as an indivisible unit which executes independently from other transactions. The ACID properties are usually used to describe reliable transactions.

- **ACID** - An acronym standing for the four key properties of a reliable transaction.

  Atomicity - the transaction must either occur in its entirety, or not at all.

  Consistency - transactions must take data from one consistent state to another, and cannot compromise data integrity or leave data in an intermediate state.

Isolation - concurrent transactions should not interfere with each other; it should appear as if all transactions are serialized.

Durability - the effect of a committed transaction should persist despite crashes.

- **Idempotent** - An idempotent operation is an operation that can be repeated without effect after the first iteration.
- **Logging file system** - A logging file system (or journaling file system) is a file system in which all updates are performed via a transaction log ("journal") to ensure consistency, in case the system crashes or loses power. Each file system transaction is first written to an append-only redo log. Then, the transaction can be committed to disk. In the event of a crash, a file system recovery program can scan the journal and re-apply any transactions that may not have completed successfully. Each transaction must be idempotent, so the recovery program can safely re-apply them.
- **Queuing Theory** Here are some useful symbols: (both the symbols used in lecture and in the book are listed)
  - μ is the average service rate (jobs per second)
  - T_{ser} or S is the average service time, so $T_{ser} = \frac{1}{\mu}$
  - λ is the average arrival rate (jobs per second)
  - U or u or ρ is the utilization (fraction from 0 to 1), so $U = \frac{\lambda}{\mu} = \lambda S$
  - T_q or W is the average queuing time (aka waiting time) which is how much time a task needs to wait before getting serviced (it does not include the time needed to actually perform the task)
  - T_{sys} or R is the response time, and it's equal to $T_q + T_{ser}$ or $W + S$
  - L_q or Q is the average length of the queue, and it's equal to λT_q (this is Little's law)

3 Problems

3.1 Extending an inode

Consider the following `inode_disk` struct, which is used on a disk with a 512 byte block size.

```
/* Definition of block_sector_t */
typedef uint32_t block_sector_t;

/* Contents of on-disk inode. Must be exactly 512 bytes long. */
struct inode_disk
  {
    off_t length;                       /* File size in bytes. */
    block_sector_t direct[12];          /* 12 direct pointers */
    block_sector_t indirect;            /* a singly indirect pointer */
    uint32_t unused[114];               /* Not used. */
  };
```

4B

$\frac{2^9}{2^2} = 2^7$ dps in one ind block

Why isn't the file name stored inside the `inode_disk` struct?

Doesn't need to be (redundant). Directory entry stores it already.

What is the maximum file size supported by this inode design?

12 dps, 1 ind $\Rightarrow (2^7 + 12) 2^9$ Bytes

How would you design the in-memory representation of the indirect block? (e.g. the disk sector that corresponds to an inode's `indirect` member)

packed block_sector_t's, 512B total $\Rightarrow$ fits $2^7 = 128$ entries

Implement the following function, which changes the size of an inode. If the resize operation fails, the inode should be unchanged and the function should return `false`. Use the value `0` for unallocated block pointers. You do not need to write the inode itself back to disk. You can use these functions:

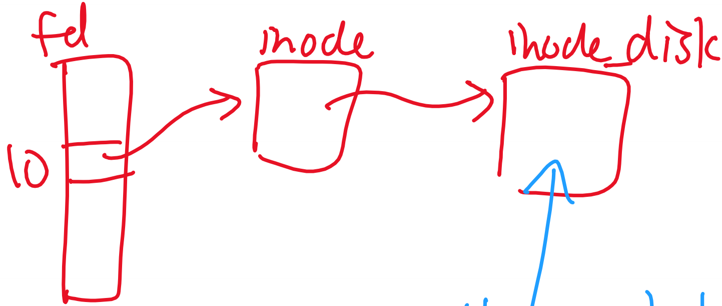
- "`block_sector_t block_allocate()`" – Allocates a disk block and returns the sector number. If the disk is full, then returns 0.
- "`void block_free(block_sector_t n)`" – Free a disk block.
- "`void block_read(block_sector_t n, uint8_t buffer[512])`" – Reads the contents of a disk sector into a buffer.
- "`void block_write(block_sector_t n, uint8_t buffer[512])`" – Writes the contents of a buffer into a disk sector.

```
bool inode_resize(struct inode_disk *id, off_t size) {
    block_sector_t sector;  // A variable that may be useful.
```

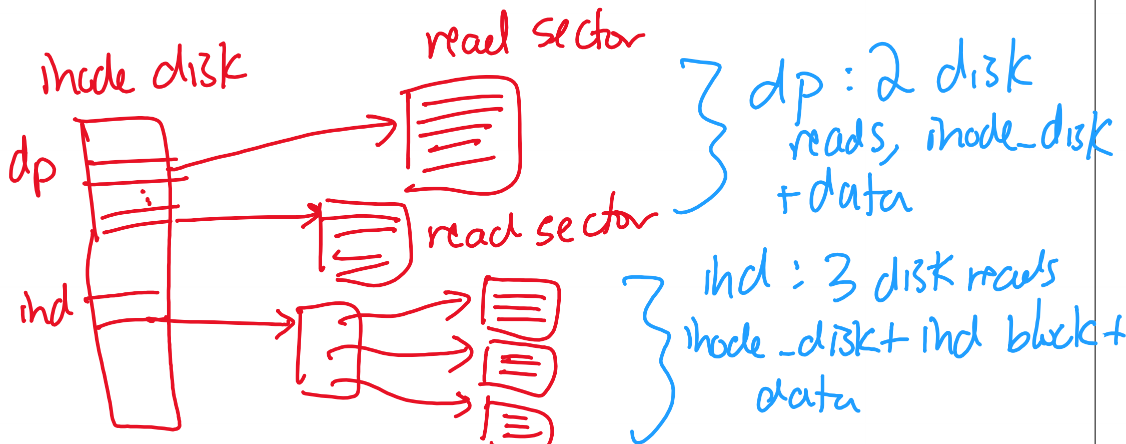
See soln - will be helpful for proj.

I'm going to cover a file traversal:

I want to read fd 10:



this is what we saw above as an inode



What about dbl? 4 reads disk + dbl + ind + data

Triple? 5

```
}
```

3.2 Queuing Theory

cover very lightly & high level

Explain intuitively why response time is nonlinear with utilization. Draw a plot of utilization (x axis) vs response time (y axis) and label the endpoints on the x axis.

See Kubi lecs for more info

If 50 jobs arrive at a system every second and the average response time for any particular job is 100ms, how many jobs are in the system (either queued or being serviced) on average at a particular moment? Which law describes this relationship?

Littles Law: $L_q = \lambda \cdot T_a = 50 \cdot 0.1 = 5$

λ ↑ arrival; T_a ↑ response time (time inside queue)

Is it better to have N queues, each of which is serviced at the rate of 1 job per second, or 1 queue that is serviced at the rate of N jobs per second? Give reasons to justify your answer.

1 server - less overhead w/ queuing delays/load balancing

What is the average queueing time for a work queue with 1 server, average arrival rate of λ, average service time S, and squared coefficient of variation of service time $\mathbf{C}$?

formulas

What does it mean if $\mathbf{C} = 0$? What does it mean if $\mathbf{C} = 1$?

See soln

3.3 Tying it all together

Assume that you have a disk with the following parameters:

- 1TB in size
- 6000RPM
- Data transfer rate of 4MB/s (4×10^6 bytes/sec)
- Average seek time of 3ms
- I/O controller with 1ms of controller delay
- Block size of 4000 bytes

What is the average rotational delay?

$1/2 \cdot \frac{60 s/min}{6000 rpm} = 5ms$

↑ halway around

What is the average time it takes to read 1 random block? Assume no queuing delay.

$\frac{4000 B}{4000000 B/s} = 1ms$

queue + controller + seek + rotate + transfer
0 + 1 + 3 + 5 + 1 = 10ms

Will the actual measured average time to read a block from disk (excluding queuing delay) tend to be lower, equal, or higher than this? Why?

lower - disk scheduling algs

Assume that the average I/O operations per second demanded is 50 IOPS. Assume a squared coefficient of variation of $\mathbf{C} = 1.5$. What is the average queuing time and the average queue length?

See solns