File Systems: Caching and Recovery

Azza Abouzied

From Last Class: File Descriptors

What happens when you open a file?

```c
#include <stdio.h>
#include <errno.h>
int main (void) {
  FILE *fp;
  fp = fopen ("test.txt","w");
  if (fp == NULL) {
    printf ("File not created, errno = %d\n", errno);
    return 1;
  }
  ...
  fclose (fp);
  return 0;
}
```

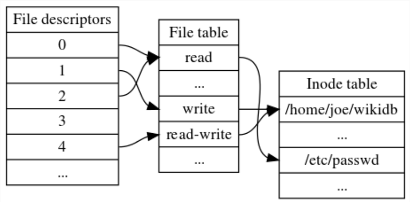
You can check the file descriptors with ls -l /proc/PID/fd

File Descriptors

1. Each PCB has a pointer to a **file descriptor table.**

2. These index into a system-wide **file table**: a table of all files. You can see all opened files with lsof -u user. The table records the access mode for each file.

3. The **file table** indexes into a third table: the inode table that describes the actual underlying files.



The Disk Cache aka The Page Cache

The Page Cache

What is it?

Physical pages in RAM that correspond to physical blocks on a disk.

How big is it?

It is dynamic: it can grow to consume any free memory and shrink to relieve memory pressure.

How is it used?

On a read() system call, kernel checks cache first, if a cache hit, it reads directly from RAM, otherwise cache miss, it schedules one or more block I/O operations to read from disk.

What is cached?

Depends on what is accessed!

What about writes to the cache?

Three policies:

1. No-write: A write operation is written directly to disk and the cache is invalidated.

2. Write-through: Write operations immediately go through the cache to the disk. Caches are *coherent* i.e. synchronized and valid for the backing store.

3. Write-back: Writes occur directly on the page cache without immediately updating the backing store. Written-to pages are marked as dirty and are added to a dirty list. Periodically, pages in the dirty list are written back to disk in a process called *writeback*.

Cache Eviction

Least Recently Used (LRU):
Ideally, evict the pages least likely to be used in the future.

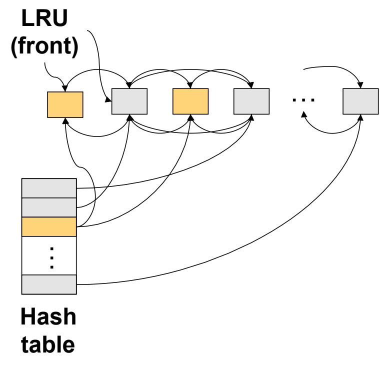
Keep track of when each page is accessed (or at least sort a list of pages by access time) and evict the pages with the oldest timestamp (or at the start of the sorted list).

This strategy works well because the longer a piece of cached data sits idle, the less likely it is to be accessed in the near future.

Problem: many files are accessed once and then never again.

Linux implements a two-list strategy: active (hot) and inactive lists.
Pages on the inactive list are available for cache eviction.

How to determine a hit?



Linux doesn't use a hash-table:

1. A single global lock protected the hash → high lock contention.

2. Large hash: all pages in the cache (other solutions had a smaller memory footprint).

3. Poor performance on collisions.

It uses a (radix)-tree per file.

The Buffer Cache

Not all block device access is through a file. Inode updates for example are through bread().

A separate block cache: buffer cache.

Most operating systems unify the buffer and the page cache.

Why unify?

Crashing

The crux of Write-back

Cache more $\rightarrow$ Write faster

Cache more $\rightarrow$ Worse crash

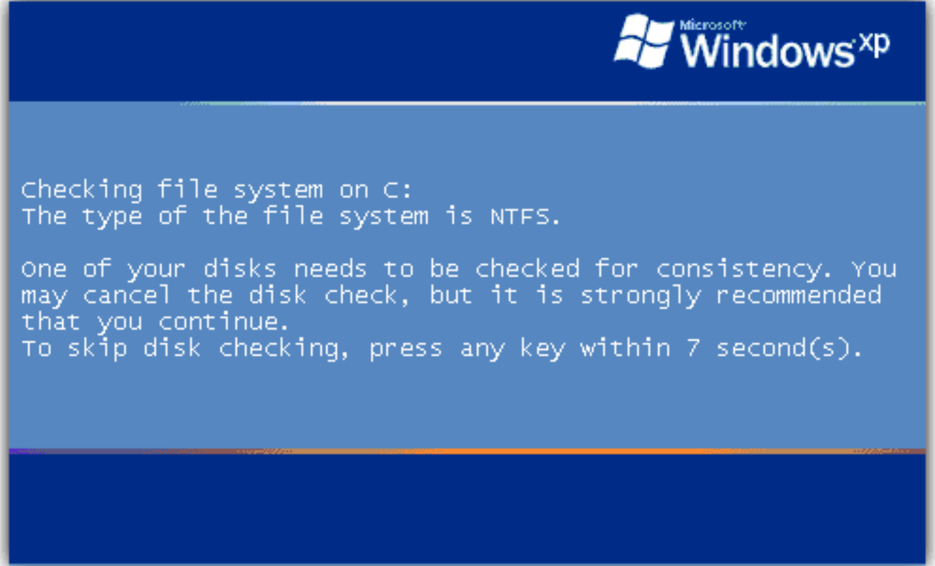
When do we write back?

On...

1. Block eviction
2. File Close
3. Device Eviction
4. Explicit flush (sync() command in unix)
5. Fixed interval (flusher threads run frequently)

Issues

No guarantees! If I lose data, how much will I lose? What state will I be in? Can I recover?


Microsoft Windows XP
Checking file system on C:
The type of the file system is NTFS.
One of your disks needs to be checked for consistency. You may cancel the disk check, but it is strongly recommended that you continue.
To skip disk checking, press any key within 7 second(s).

How consistent is the file system?

Inconsistencies

1. Same block in multiple files
2. Free blocks not in free list
3. Directory pointers to nowhere
4. Orphans
5. Funny attributes: bizarre modification times

A consistent file system might still have corrupt data

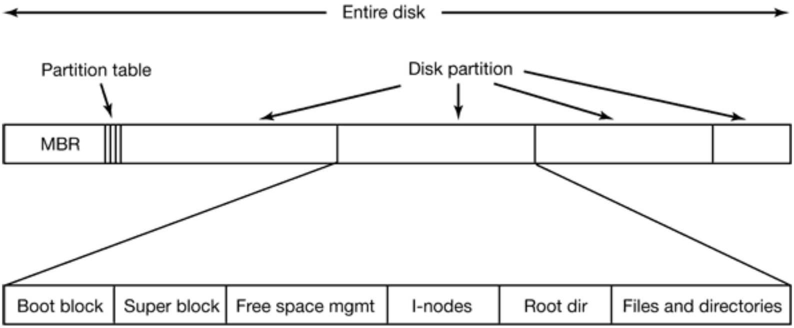
How do consistency checkers work?

Checkers:
Unix: fsck; Windows: chkdisk, scandisk

1. Start from root (/) inode
2. Traverse the entire directory tree and mark reachable files/blocks
3. Verify logical structure
4. Figure out which blocks are free

Do garbage collection: put free blocks in free list
Put orphaned files in /lost+found

How to recover?



The havoc of a crash

Example 1: Move a file

1. Place it in directory

2. Delete from old

Crash happens both directories have problems

Example 2: Delete a file:

1. Remove directory entry

2. Add blocks to free list

3. Update statistics in superblock

Crash happens file system logical structure hurt

Questions?