The Page Cache

COS 316 Lecture 11

Amit Levy

Last Time: CPU Cache

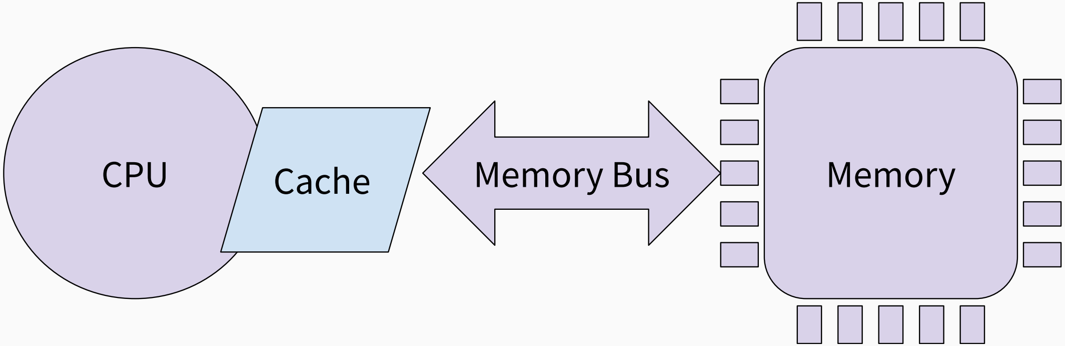

Figure 1: CPU Connected Directly to Memory

This Time: Caching Disk Blocks

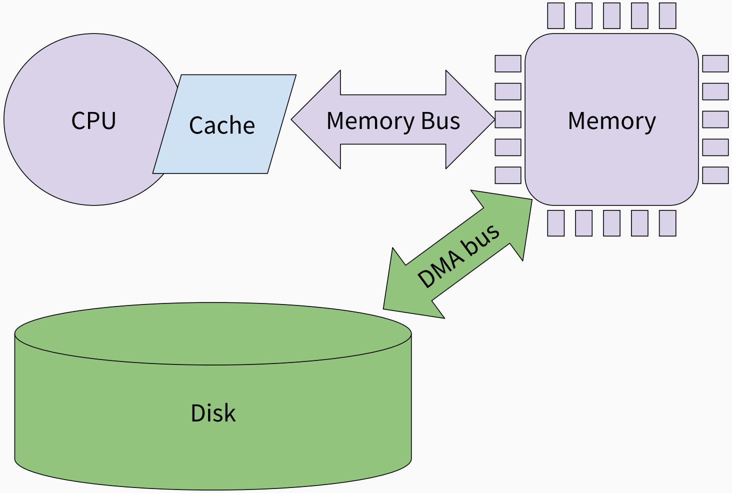

Figure 2: Disk blocks have to go through memory

Why cache disk blocks? Why have to

CPU can’t operate directly on data in disk.

- There are no `load-from-disk` or `store-to-disk` instructions
- Our only option is to initiate a copy of data from disk to memory or from memory to disk.

Why cache disk blocks? It’s faster!

Type	Access Time	Typical Size	\$/MB
Registers	$< 0.5ns$	~256 bytes	$\$1000$
SRAM/”Cache”	$5ns$	1-4MB	$\$100$
DRAM/”Memory”	$50ns$	GBs	$\$0.01$
Magnetic Disk	$5ms$	TBs	$\$0.000001$

Typically operating on a small subset of data from disk:

- Updating a few hot rows in a very large database. What kind of locality is this?
- Scanning through all files on disk sequentially for viruses. What kind of locality is this?

Why cache disk blocks? We can share resources!

Often, multiple processes need to access the *same* disk blocks.

Examples:

- The root diretory inode
- The system log file
- IDE and compiler accessing the same source code file
- Multiple browser windows accessing the same (locally cached) image

Why cache disk blocks? We can share resources!

Often, multiple processes need to access the *same* disk blocks.

Examples:

- The root diretory inode
- The system log file
- IDE and compiler accessing the same source code file
- Multiple browser windows accessing the same (locally cached) image

Caching disk blocks in memory is an *opportunity* to save avoid duplicating memory

The Linux Page Cache

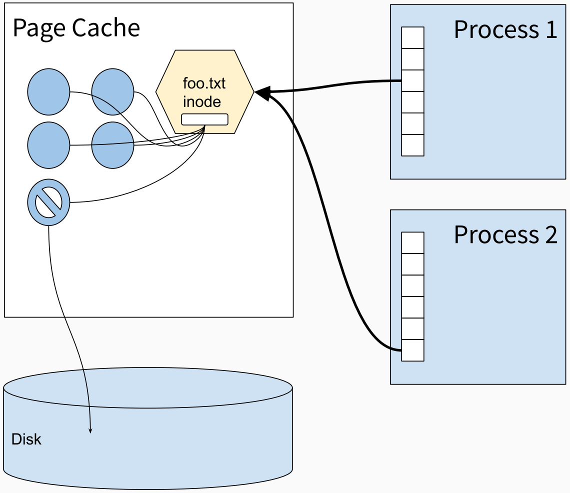

Figure 3: The Page Cache

Linux Page Cache: Look-through

Access to the cache implicity accesses the disk.

```
int c = read(fd, buf, 512);
```

1. Kernel translates local file descriptor to `(inode, file offset)`
2. Is inode in page cache?
    - No? *read inode from disk and put in page cache*
3. Find data block number for offset from inode
4. Is block in page cache?
    - No? *read block from disk and put in page cache*
5. Copy 512 bytes of cached disk block to user supplied buffer

Linux Page Cache: Write-through or Write-back?

By default, kernel marks written pages dirty and flushes after a delay:

```
write(fd, ”hello world”, 11);
```

1. Kernel writes “hello world” to page for cached disk
2. Kernel adds the page to the “dirty list”—pages that have been modified but not yet written to disk
3. Periodically, kernel writes all pages in dirty list to disk

Linux Page Cache: Write-through or Write-back?

`O_SYNC` flag converts file descriptor to write-through

```
int fd = open("myfile", O_SYNC);
write(fd, "hello world", 11);
```

This affects *all* accesses to the same disk blocks

Linux Page Cache: Write-allocate or write-no-allocate?

By default, kernel adds writes of new blocks to the cache. E.g., extending a file...

```
write(fd, "hello world", 11);
```

1. Kernel writes “hello world” to a new block allocated in a cache page (no write to disk yet)
2. Kernel modifies inode as necessary (e.g. increase file size field in inode)
3. Kernel adds both page containing new data and inode’s page to dirty list

Linux Page Cache: Write-through or Write-back?

O_DIRECT flag converts file descriptor to write-no-allocate

```
char buf[512] = "...";
int fd = open("myfile", O_DIRECT);
write(fd, buf, 512);
```

Writes do not go through the page cache, but directly transfer process memory to disk.

Writes-back all pages from all processes related to same pages.

Summary

The Linux page cache solves two problems:

1. How to access disk blocks from the CPU
2. How to avoid allocating multiple memory pages for shared file system data

API provides a flexible cache:

- Write-through *or* Write-back
- Write-allocate *or* Write-no-allocate

Coming up

- Precept tomorrow on SQL
    - You will need this for assignment 3
- Prof. Freedman on Web caching on Monday
- Assignment 2 due on Tuesday