Announcements:

- Anonymous feedback: cs61.seas.harvard.edu/feedback
    - request lecture to go slower or faster - Live results
- Midpoint Evaluation
    - Do it if you haven't already.

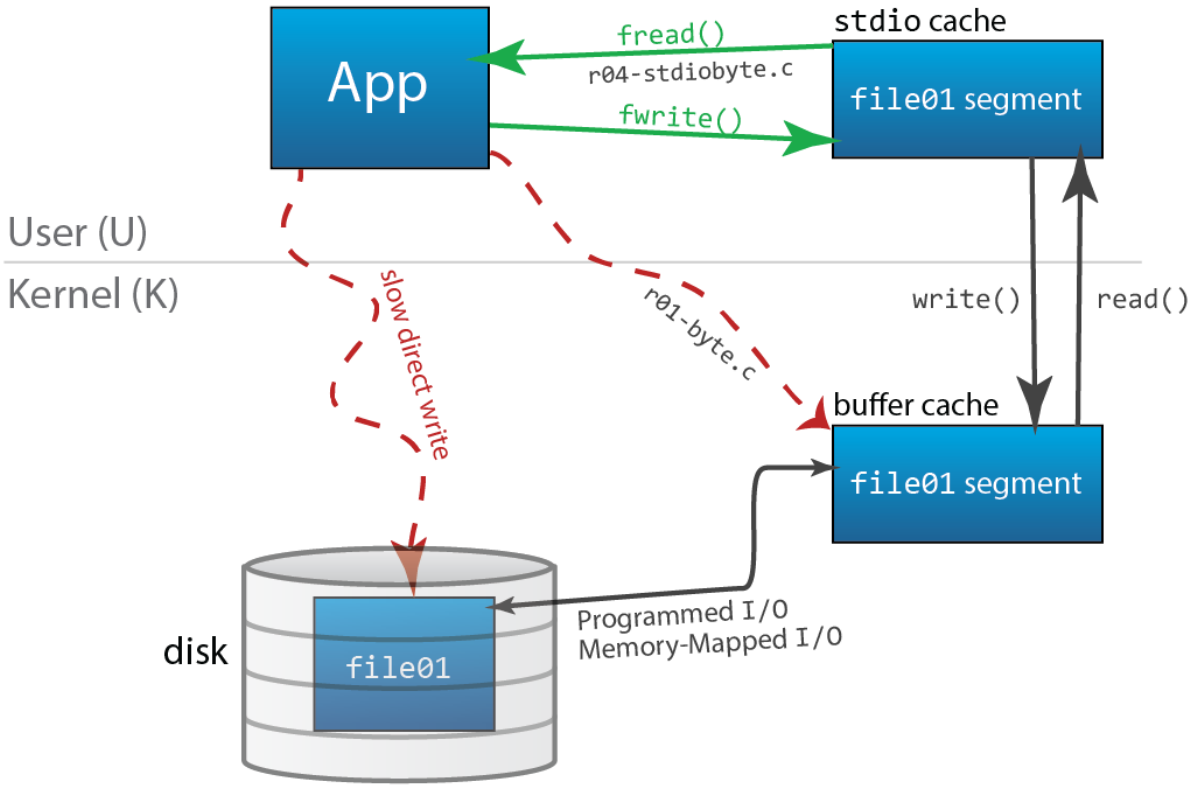
Keywords:

- cache organization
- c = r + nu
- block
- slot
- direct mapped
- fully associative
- hit vs. miss

Outline

- **Buffer Caching**
- **Buffer Organization**
- **Processor Caches**

Recap from last time:
There are two caches on the *application side* there is a **stdio cache** and on the *kernel side* there is a **buffer cache**.



Synchronous read/writes are orders of magnitudes slower than using caches.

Cache - Fast storage that stores temporary copies of data from slower storage to speed up future requests

- Buffer cache is faster storage than the disk
- Standard I/O is faster than buffer cache because it reduces the amount of system calls (which are slow)

Lecture code in l17 folder of repository

Comparison of reading patterns

block size	stride	source of read calls?	rate (bytes/second)
1	0	sys	
1	1<<20	sys	1.04 e 6
1	0	stdio	
1	1<<20	stdio	6.76 e 5 (bad caching) 2.03 e 7
512	0	sys	

512	1 << 20	sys	
512	0	stdio	
512	1 << 20	stdio	

Standard I/O is faster for sequential calls because it reduces system call overhead

c = r + n*u where:

- c = cost of a request
- r = overhead of a request
- n = size of unit
- u = cost per unit

Caching can be slower for stride access pattern because we read in a bunch of data to the cache but then stride past the block size and have to throw away all of this data

- very low utilization

Cache Organization
Units of cache organization block and slot.
block: unit of data transfer to and from the cache

- Standard I/O: 1 page (4096 bytes)
- Buffer Cache: 1 page (4096 bytes)

slot: space in cache for at most 1 block

- Standard I/O: 1 slot
- Buffer Cache: size of memory

When first opening a file with standard I/O the slot is empty
Besides the memory used to store kernel, the rest is used as a cache for the disk

direct mapped cache - every block has exactly one slot into which it can fit

- Each student is assigned to a certain seat
- if multiple students were assigned to the same seat, only one could come to any given lecture
- Standard I/O is direct mapped because there is only one slot

fully associative cache - any block can fit into any slot

- Lecture hall is fully associative anyone can sit in any seat

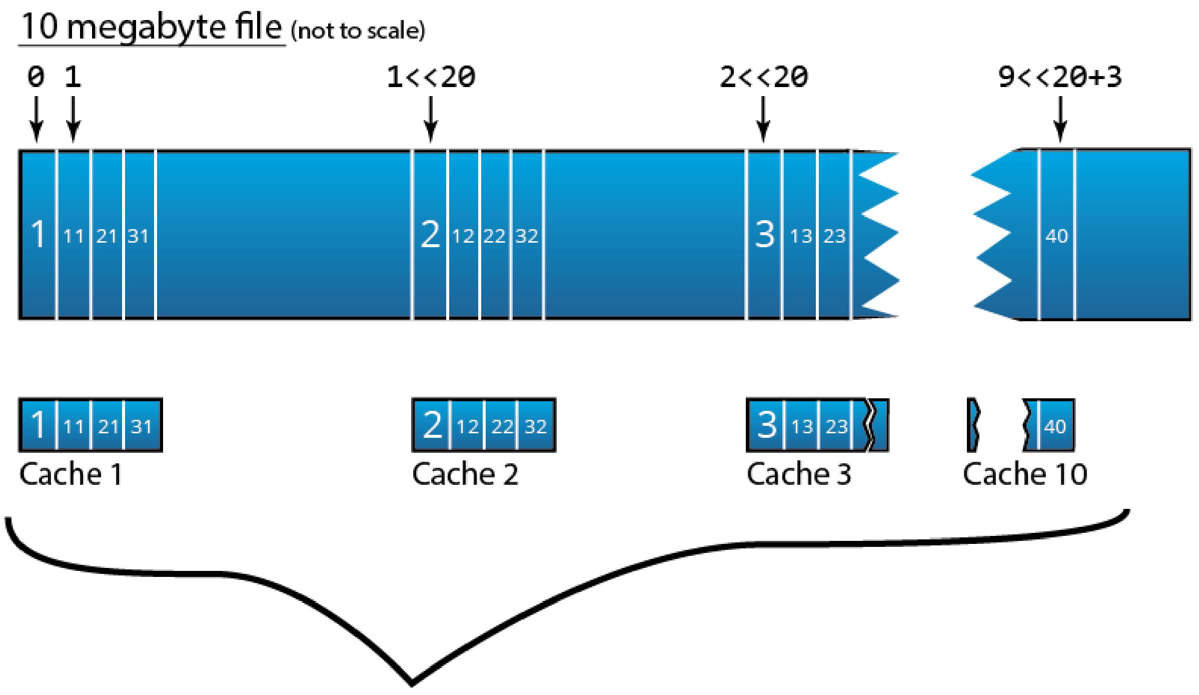
hit - Access that is satisfied by the cache.
miss - Access that is not satisfied by the cache. (Requires a read from slower storage that the cache is optimizing)
hit rate - Fraction of accesses that 'hit'. High hit rate is good.

- sequential access pattern: Hit rate = 4095/4096.

- strided access pattern: Hit rate = 0. Every access is a miss.

We can make a cache with more slots in order to capture all of the data in the strided pattern:

- Number of slots = Size of file / size of stride
- The problem is we need to know the size of the stride in order to create a cache that is the correct size
- `r10-stridestdiomulti.c`
    - Reads from multiple copies of the same file
    - One copy reading each new strided segment



Caches' states just before they need to be refilled.
(Assume 4 byte caches for the sake of the drawing.)

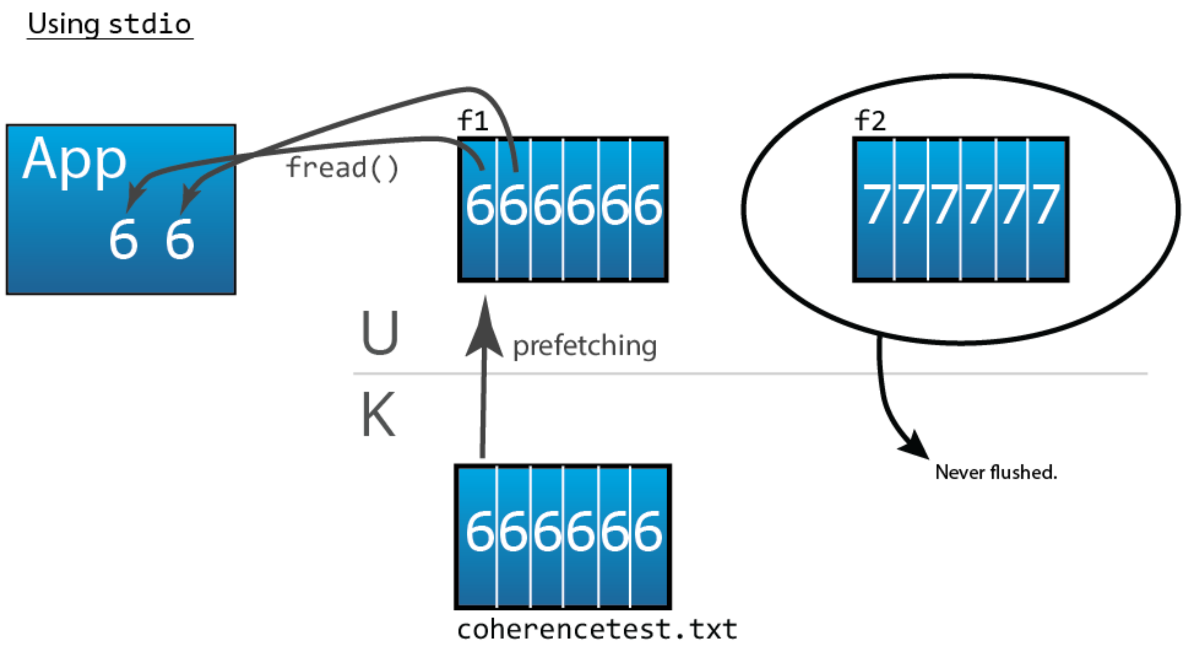
Note:
Accessing bytes 1 through 10 is all misses. Then, we start getting hits at byte 11.
After the caches are full, byte 41 will be a miss again and will be loaded into Cache 1.

`readandwrite.c`

- `Reads in coherencetest.txt contains` "666"

1. open up file
2. read one byte from file
3. open same file for writing
4. write over contents of file with "777777"
5. read one byte from original file
6. print two bytes that were read

7. This will print: “`67`” if using system calls “`66`” with Standard I/O
The reason Standard I/O does not print the correct result is because it’s cache is not coherent.



cache coherence - a file is coherent if its contents are always equal to the corresponding blocks in slow storage

- write back caching - writes can sit in cache for a while before being sent to backing storage
    - Opposite of write through cache where every write is immediately sent to the backing storage
- no invalidates - when data changes cache is not notified

Standard I/O does not have a coherent cache

- will fail if we overwrite file as we are reading
- Standard I/O cache will read in a block from the file
- Then if we overwrite data within this block in the file when we go to read it will read from cache which is old data
- We could solve the write back policy by flushing after every call
- We could solve the issue of invalidation by alerting to other file cache’s when the data was changed

processor caches are coherent

Analogy of bank account and coherent cache

- would want a bank account cache to be coherent
- only one person could modify the account at any time

- Can be done using a system with a lock

Buffer cache is coherent with the disk and has a lock on the file on disk. All write-throughs go through the buffer cache.

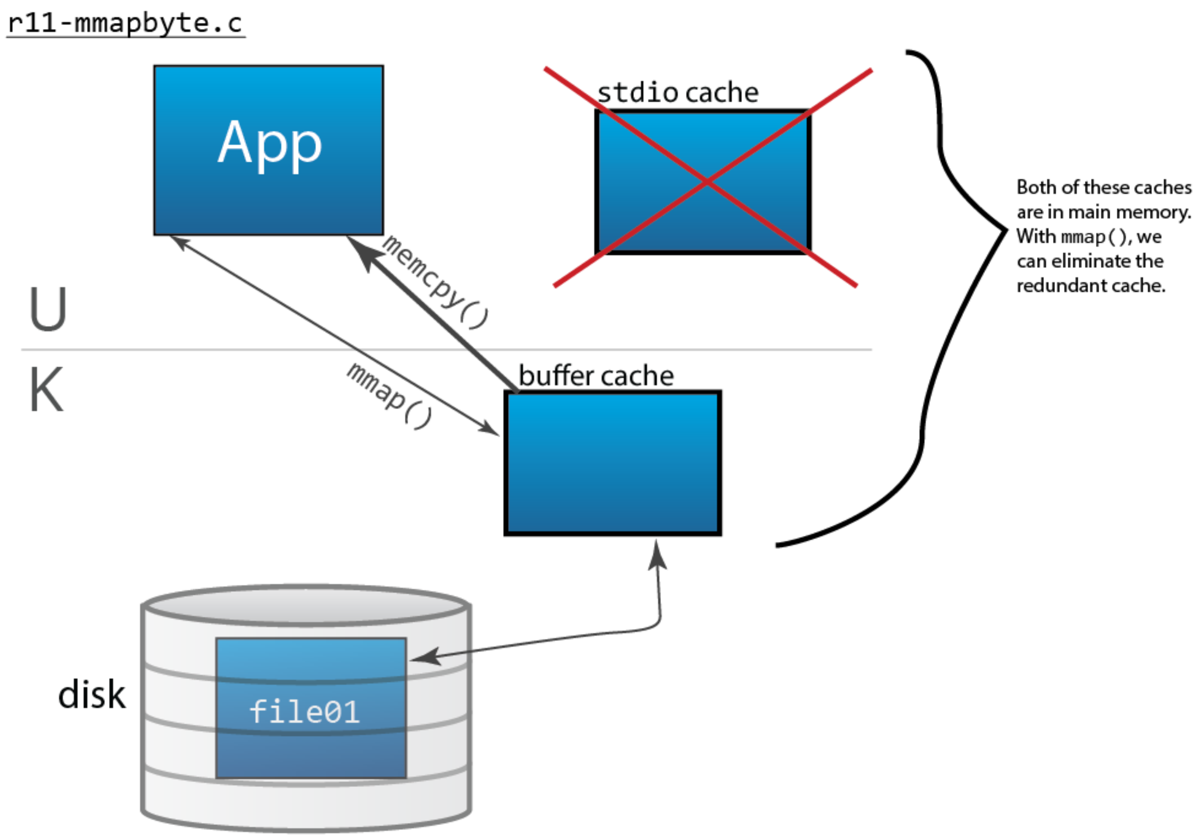
- It still has a write back policy, which can be observed if the computer is powered down before it is actually written to disk

We can use memory mapping to speed up caching

- buffer cache has aligned pages of memory
- Take the pagetable
- Insert a new pagetable entry in an empty slot and point to a page in the buffer cache

`r11-mmapbyte.c`

- We will use the linux system call `mmap`
- Takes a region of the buffer cache and maps it to applications virtual memory
- then a read is just a call to `memcpy`


r11-mmapbyte.c
App
stdio cache
memcpy()
U
K
mmap()
buffer cache
Both of these caches are in main memory. With mmap(), we can eliminate the redundant cache.
disk
file01

memory mapped version is *ten times faster* than standard I/O

- eliminate system calls overhead
- eliminate need to copy from standard I/O cache to buffer cache (buffer cache and standard I/O are both in primary memory)

    - We map standard i/o cache to buffer cache so we are directly manipulating the buffer cache

Benefits of standard io cache

- better errors
    - Memory mapping gives segmentation faults
    - Standard I/O gives errors based on system call return values
- can handle pipes
- can handle files that are too large to fit into memory