Computer Science 61
Scribe Notes

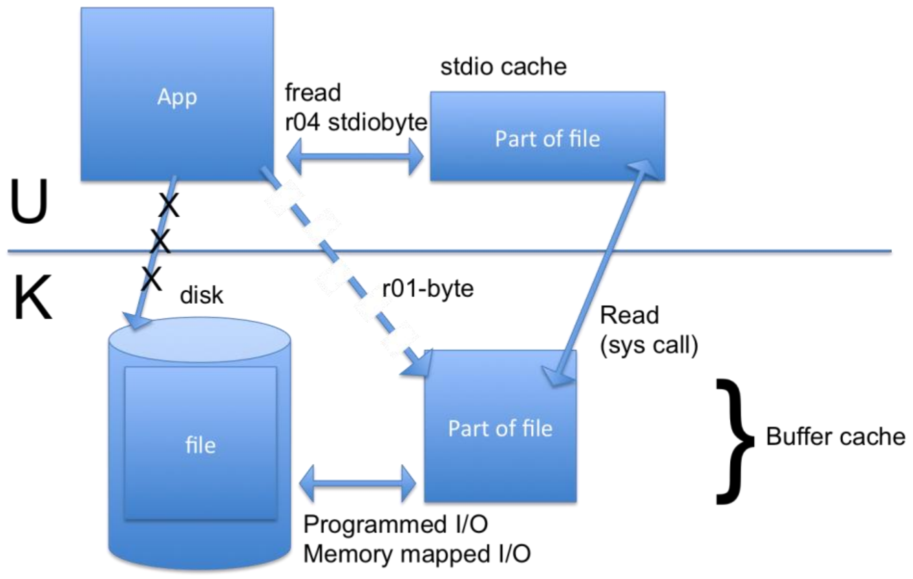
Tuesday, November 6, 2012.

There is an anonymous feedback form which can be found at
http://cs61.seas.harvard.edu/feedback/

Buffer Caching and Cache organization and Processor caches:

There are two primary caches:

- In the application level, there is a stdio cache for the application
- In the kernel, there is a buffer cache, which acts as a cache for the disk.



At the beginning, our applications had direct write and read access to the disk without using cache, but that was very slow.

Cache: Fast storages that store temporary data from slower storage to speed up future requests.

stdio cache: has a faster storage than the buffer cache. It reduces per request overhead (over head of system calls, exceptional control flow).

-System call overhead is the difference between r01-stdiobyte and r01-byte rates.

-The difference between stdiobyte and stridestdiobyte is interesting because caching is slowing down the performance of stridestdiobyte.

Why:

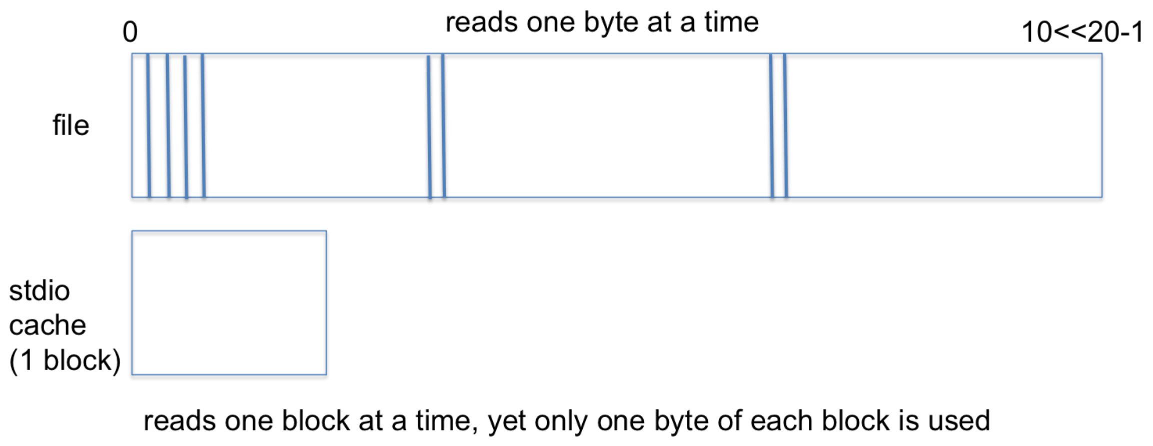
Reading a bunch of useless things in and moving on to the next, increased R

Cost of Request:
C = R + NU

Where N is size in units. U is cost of units. R is cost per request.

-In the Stdio version, the cache is 4095/4096 useless because in striding, we only use 1 byte out of the block.

-Utilization is 1/4096 which causes bad performance. In the seek version, the data filling the stdiocache block is meaningless because you only need one byte.


0
reads one byte at a time
10<<20-1
file
stdio cache (1 block)
reads one block at a time, yet only one byte of each block is used

Cache organization:

Cache is divided into units:

Blocks - unit of data transfer to and from the cache
(stdio: 1 page (4096 B))
(buffer cache: 1 page)

Slots - space in the cache for at most one block, can be empty
(stdio: 1 slot)
(buf cache: size of memory)

Two types of Cache:

Direct Mapped:
A cache in which every block has exactly one slot into which it can fit.
Ex. every student has an assigned seat in a lecture hall.
If there's more students in the lecture hall that can possibly fit, and each seat can fit specifically one or another student, the lecture hall is a cache.

- Stdio is directly mapped; one block can fit in one slot.

Fully Associative:

A cache in which any block can fit into any slot.
Ex. our lecture hall, where any student can fit in any seat

Hit:
Access that is satisfied by the cache.

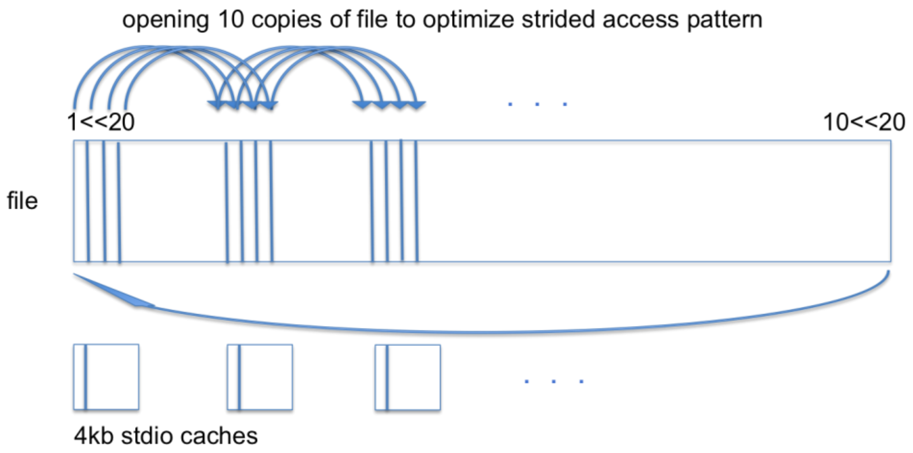
Hit rate -> fraction of accesses that hit
If we read sequentially from stdio, the hit rate is 4095/4096.
If we read in a strided pattern, the hit rate is 0. Every single one is a miss.

Miss:
Access that is not satisfied by the cache.

Question: How big does a stdiocache need to be to achieve a 4095/4096 hit rate?

Goal: 4095/4096 stdio cache hit rate

Example:


opening 10 copies of file to optimize strided access pattern
1<<20
10<<20
file
. . .
4kb stdio caches

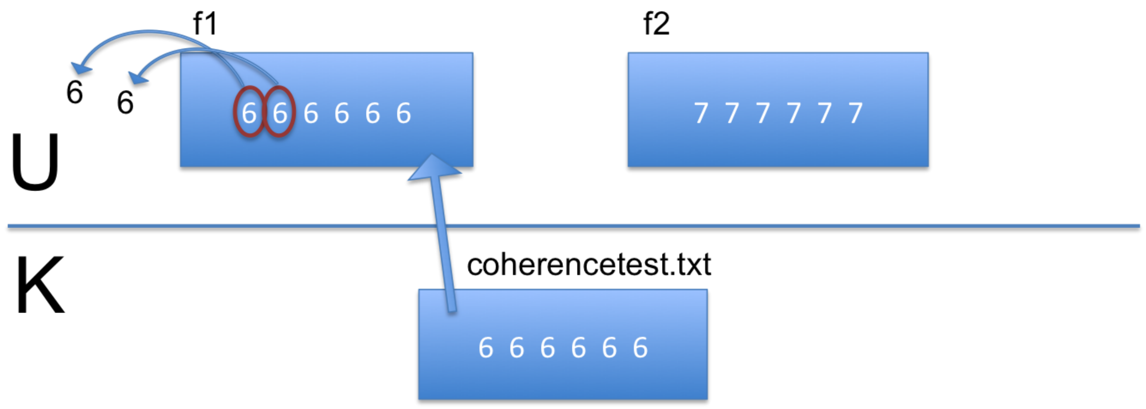
readandwrite.c:

What it does:
1. ct starts out containing 666666
2. open ct
3. read 1B
4. open ct for writing

5. write into ct 777777
6. read 1B from the original ct
at the end, prints two characters: 67

Steps 2,3,6 are the same file descriptor.

readandwrite using stdio prints 66



Cache coherence:
A cache is COHERENT if its contents are always equal to the corresponding blocks in slow storage.

-Stdio cache is not coherent. (One has a value of 67 and another has a value of 66)

Why:

1) Batches writes: (WRITE-BACK policy vs. WRITE-THROUGH policy)
write-back: a write can sit in the cache for a while before it's sent to the slower data storage.
Write-through: every write is sent immediately to the backend storage (more system calls).

2) No invalidates (when data changes, cache is not notified.)

-Buffer cache is kept coherent with the disk

-There is no difference between program reading/writing buffer cache and program reading/writing memory of the buffer cache

memory mapped version is 10 times faster than stdio in this example
we have eliminated overheads: system calls (now, only one sys call to mmap), not copying the data out of the buffer cache into the stdio cache (directly accessing memory that makes up bufcache)

Accessing primary memory is roughly the same no matter where that memory belongs. Kernel memory is not slower than application memory. The overhead we are avoiding is in system calls and we’re avoiding copying bytes from one region to another. The version that uses stdio copies each byte at least twice (once using system call and once using stdio cache to application). In the improved version, we’re copying the data only once, skipping stdio cache.

- The stdio cache is NOT useless
- It handles things that are not disk files
  and things that are too big to fit in memory