CPSC 313 Course Review Notes
April 16, 2021

3 Memory and Cacheing

3.1 The Memory Hierarchy

- Memory is arranged hierarchically. An example memory hierarchy might be

  Registers $<$ L1 … L4 Caches $<$ Main Memory $<$ Flash drives $<$ Disk drives $<$ Network

- Moving down a well designed memory hierarchy, the following should be observed:
  - Latency increases/speed decreases
  - Size increases
  - Price per unit of storage decreases

3.2 Caching

- *Def*: Temporarily moving data up the hierarchy to reduce latency
- Obeying the memory hierarchy, caches will be smaller than their data source so must sometimes *miss*:
  - *Compulsory Miss*: Accessing an item for the first time which is not in the cache
  - *Capacity Miss*: Accessing an item which does not fit into the cache
  - *Conflict Miss*: Accessing an item whose spot in the cache is taken
- Caches are predicated on *spatial* and *temporal locality*, the principle that data is generally accessed near each other in space and in time.
- The *line size* of a cache is the minimum unit of data stored in a cache. Caches on disks are generally 4KB, which is the *block size* of most disks.
- Direct mapped caches hash address to cachelines by splitting addresses as *Tag/Index/Offset*.
  - The offset bits index into a cache line, namely there are log(line size) bits.
  - The index bits index cache lines. The size of a cache with i index bits and o offset bits is 2^{i+o} bytes.
- A direct mapped cache is a 1-way set associative cache, because each *cache set* (the set of positions in the cache a cacheline can occupy) is exactly one cache line. In general, for a 2^n way set associative cache move n bits from the index to the tag.

3.3 Cache Eviction Policies

- *Belady's Algorithm*: Evict the item used farthest in the future. This is impossible to do in practice, but is optimal. For any other algorithm, we can construct an adversarial access pattern.
- *LRU (Least Recently Used)*: Evict the item used furthest in the past. An adversarial access pattern for a size n fully associative LRU cache is

  $$1, 2, \cdots, n, n+1, 1, 2, \cdots$$

  which is fixed by MRU.
- *LFU (Least Frequently Used)*: Track uses of each tag, evict item with the fewest.
- *MRU (Most Frequently Used)*: Evict most recently used item.
- *FIFO (First In, First Out)*: Evict item loaded earliest into the cache (queue).
- *LIFO (Last In, First Out)*: Evict item loaded last into the cache (stack).

3.4 Amdahl's Law

- If a program spends the portion α of it's time in a task which is sped up by a factor of k, the total speedup to the system is

  $$\frac{1}{(1-\alpha)+\frac{\alpha}{k}}$$

  and if the old runtime (often called latency) is T, the new runtime will be $T((1-\alpha)+\frac{\alpha}{k})$.

3.5 Cache Write Policies

- On a hit, caches are *writeback* or *writethrough.*
  - Writeback caches keep dirty data in the cache, and write on eviction (or when requested). This is faster, and can avoid unnecessary writes.
  - Writethrough caches write all dirty data back to disk immediately. This can make synchronization easier.
- On a miss, caches are *write allocate* or *no write allocate.*
  - Write allocate caches allocate address they are attempting to write to in the cache (bringing the rest of the cacheline into cache from the disk). This is fast especially with many localized writes.
  - No write allocate caches just pass write misses along to the disk. This avoids filling the cache with write data.

- While the above policies are orthogonal, we generally only use writethrough/no write allocate and writeback/write allocate caches.
- Low in the memory hierarchy, we generally use writeback/write allocate.

3.6 Strided Access

- Increasing the stride of an access pattern can force misses on upper level caches, and give us information about the size and latencies of a memory hierarchy.
- The standard graph is as follows: size on the x axis, time on the y axis, and several lines plotting the total access time for a size x array with a given stride.
  - The graphs will be piecewise constant. The value at which it is constant is the latency of a member of the memory hierarchy, and the x value where the graph jumps to member M's latency is the total size of member $M - 1$.
- C arrays are *row major*, that is rows of arrays are contiguous in memory. Like arrays in mathematics, 2D arrays in C are indexed first by row, then by column (so in memory it looks like $A_{0,0}, A_{0,1}, A_{0,2}, \cdots$).

3.7 Multicore Caches

- Two solutions are snoopy caches, and directory based coherence. We will use the former, that is there is some bus between caches which allow cores to send messages between each other.
- *MSI* protocol: each cacheline can be in the state *(m)odified*, *(s)hared*, and *(i)nvalid*. Data which is just read is shared. Writing data to a cacheline puts it into the modified state, and invalidates shared data in other cores (this data cannot be dirty). If another core attempts to read or write data which this core has in the modified state, this core must writeback and invalidate.
- *MESI* protocol: more efficient when only one core has data. This protocol introduces the *(e)xclusive* state, which is for clean data cached only by one core.

4 Filesystems

4.1 FS API and Syscalls

- *System calls* are a method whereby unprivileged processes can request the operating system to perform potentially harmful actions on their behalf. See page 2 of the manpages.
- Filesystem syscalls in `libc` (a thin library which performs the syscalls)
  - `open`: Takes path, returns file descriptor (integer). Takes oflags (read/write, create, append, exclusive, disable caching) and a mode for file creation.
  - `close`: Take file descriptor and deallocates everything associated to it. Returns 0 on success, -1 on error.
  - `read/write`: Reads/writes files to or from a buffer. Returns `ssize_t` (signed `long`) of bytes read or written, 0 on eof and -1 on error.

4.2 Disks

- Disks are a collection of spinning platters, which store magnetic data on both sides.
- Platters are organized into tracks. Tracks are divided into sectors (equiv. blocks), which used to be 512B and are now at least 4KB.
- Data is contiguous if laid out on the same cylinder (not the same platter).
- Disk access time is measured in a couple parts:
  - Seek time: Time for disk arm to move to correct cylinder
  - Rotation time: Time for head to get to correct block in the track. Data laid out over an entire track does not pay this. Average rotation time is half maximum rotation time.
  - Transfer time: Time once data starts reading.

4.3 File Descriptors

- Every process has some common data structures
  - File descriptors for stdin, stout, and stderr (0,1,2 respectively).
  - A *file descriptor table*, containing pointers from file descriptors to an open file table entry.
- The *open file table* has an entry for every unique call to `open`, containing file position, reference count, and reference to a vnode table entry.

- It is possible to get a reference count greater than zero by i. forking a process with an open file, or ii. using the `dup` or `dup2` syscalls. The latter helps support pipes:
  - Redirection `>` can be done by opening a file descriptor `fd` (with wronly, create, trunc), and calling `dup2(fd, STDOUT_FILNO)`. This also closes the old stdout.
  - The way the shell implements redirection using fork: in the child it performs the `dup2` redirection and `execv` of the command. This way the parent still has the normal stdout.
- A *vnode* is an in-memory representation of a file, which has a copy of file metadata and a way of locating actual disk blocks.
- The file system implementation is motivated by file descriptor management:
  - `open` needs a persistent mapping between file names and physical file metadata.
  - `open`, `close`, `read` and `write` all need to manage the file descriptor table.
  - `read` and `write` will need to follow the file descriptor table to an open file table entry and then a vnode, and also need a map between file offsets and disk blocks.
  - We call the mapping of file names to metadata and file offsets to blocks an *inode* and is stored on disk and in memory.

4.4 File System Implementation

- On disk metadata is stored in an *inode*.
- A filesystem has *internal fragmentation* when it only uses part of a block.
- A filesystem has *external fragmentation* when small pieces of contiguous memory are unused.
- There are several methods of allocating space for files to address these issues
  - *Single Extent*: Metadata points directly to one allocation unit. It is inflexible, and leads to external fragmentation.
  - *Fixed Size Blocks*: Metadata points to a list of blocks. Flexible and minimizes both fragmentations but uses large amount of metadata.

  We use fixed size blocks because it handles sparse files far better, and is better matched to the POISX paradigm of files as streams of size that is not fixed. There are several strategies:

  - *Flat index*: Inode points to fixed size index, index points to disk blocks. Can represent sparse files, but also wastes space for small files.

  - *Multi-level index*: Inode points to tree of indirect blocks (blocks of disk addresses). Has the advantage of not wasting many disk blocks for small files, through has a minimum of several IO's per block access.
  - *Hybrid index*: Inode points to several trees of various depths. Allows fast access to small files, and the potential for large files as well. Filesystems generally use this.

- Filesystem wide metadata is can be accessed by `statfs` and `fstatfs`, giving a `struct statfs` (see `man statfs`). It is stored on disk in the *superblock*, typically the first block, and the exact information varies by filesystem.
- Individual file metadata is accessed via `stat` and `fstat` giving a `struct stat` (see `man stat`). It includes the inode number.
- Directories are files, and can be accessed by library calls `opendir` and `readdir`. See `man readdir` for information about the `struct dirent` structure. It includes inode numbers, dirent size, dirent type, namelen, and name. We round record length up to 4 byte boundaries.
- Information about traversing directories:
  - In most POISX systems, the root inode always has number 2.
  - Some blocks contain metadata (indodes) and others contain data (dirents or files)
  - Example: Read the block containing indode 2, indode 2 contains the disk address of the dirent for `/`. Pick a folder in the dirent (say `usr`), it has an associated inode number. Read the block containing the inode of `usr`, it contains a disk address of the dirent for `usr`. Continue in this fashion until you find your file.
  - This is an expensive operation. To open a file $/1/2/3/\cdots/n$ requires $2n+1$ IO's (two for each directory including `/` up to $n-1$, and one to get the inode of n. To actually read the file requires an additional IO, making $2(n+1)$.
  - A *hard link* is when you have multiple names for the same inumber. We maintain a reference count for each file, and a hard link increments this. At a minimum, directories have a reference count of 2. Moving hard links will always work.
  - A *soft link* has it's own inode number, but just contains the name of another filein the current directory (no data or dirent). It has a different type. Moving soft links might not always work.
  - Almost all systems cache the current working directory.

I will not include notes on the V6/EXT2 filesystems.

5 Virtual Memory

5.1 Processes

- Programs that are running are *processes* whose heap, stack, and text live in this proceces's *address space*.
- Process isolation is the illusion that each process has access to all of, and sometimes even more than, the entire physical memory.
- Processes use *virtual addresses*, which a *memory management unit (MMU)* converts into physical addresses. The MMU is a hardware construct, so must be simple and fast, but the precise mapping it uses is complex (involving allocation, policies about sharing memory, etc.) and controlled by the operating system.
- Syscalls involving processes:
  - `fork`: replicates parent. Returns 0 if in child, child PID if in parent.
  - `execve` (one of many `exec*`): Replace current process from file.
  - `wait` and `waitpid`: stop parent until a child completes, return child exit code. This can be blocking or non-blocking, which allows us to wait via *blocking* or *polling* which have different advantages.

5.2 The Operating System and Control Transfer

- We restrict processes from being able to do anything with *privilege levels*. All computers have at least two (user/kernel). Intel has 4 which are called *rings*.
- Data also has permissions. We do not want data to be executable, or to allow self-modifying programs.
- Combined, every piece of virtual memory must have metadata related to the privilege level required to access it, and permissions about what to use it for. Given this information, the MMU will either produce data or a *fault*.
- The current privilege level is maintained by the processor and can be modified by *traps*:
  - *System calls*: A process asks the operating system to perform some task.
  - *Exceptions*: A process requires the operating system to resolve an invalid state.
  - *Interrupts*: An asychronous event completes.
  - *Timers*: Generated by an inbuilt timer to give the operating system control periodically so no single process can hijack the entire system.

  There are several other examples in real life. Note the differing synchronicity and intentionality of each.

- Each type of trap is assigned an index in a *trap handler table*, which contains pointers to functions which are executed on each type of trap.
- The x86 has some more details:
  - *Interrupt Descriptor Registers* which constitute the trap handler table. The first 32 are for hardware defined traps.
  - *Gates* which allow processes to increase or decrease their current privilege level.
  - *PICs* (programmable interrupt controller), or *APICs* (advanced PICs), or most modernly *LAPICs* (local APICs), which are sets of gates that hook up to at most 16 devices and send the correct interrupt to the processor.
  - Syscalls evolved from the `INT` instruction (synchronous software interrupt with a specified syscall value) which returns via `IRET`, to modern systems with `SYSENTER` or `SYSCALL` and `SYSEXIT` or `SYSRET`.

5.3 Virtual Memory Implementation

- Memory is divided into *pages*, x86 pages are 4KB.
- Physical and virtual page size are the same. If a page size is 2^i, virtual addresses are m bits and physical addresses are n bits, the MMU must map the first $m - i$ bits of a virtual address to the first $n - i$ bits of a physical address. The last i bits are identical.
- Address spaces can be larger or smaller than physical memory. If smaller, many entire processes can be in memory at once. If larger, the sparsity of address spaces allows a process to run without being completely in physical memory.
- x86-64 splits 64 bit addresses into 12 unused bits/36 page number bits/12 bits offset.
- The MMU caches mappings for speed in a *translation lookaside buffer (TLB)*. This is particularly helpful for instructions and local data.
- A simple TLB stores a tuple of virtual page number, physical page number, permissions (R/W/X) and user permissions (supervisor/user). The simplest kind of MMU is just a large TLB.
- A *page table* is a data structure of these mappings indexed by virtual page number for a given address space. The TLB is just a fast cache of page table entries, as page tables can be very large.
- Either the operating system (below) or the hardware (x86) handles page table lookups when data is not in the TLB.
- A page table entry is *invalid* (unallocated, accesses give segfaults), *valid and memory resident*, *valid and not memory resident*. Pages might not physically exist in memory, in which case the PTE contains information about how to acquire the data.

5.4 Software TLB Fault Handling

- If a user process generates an address not in the TLB, the MMU generates a trap for a TLB fault. Some processors (x86) can handle TLB faults in hardware.
- The operating system will look the page in the page table:
  - If invalid, kill the process.
  - If valid and memory resident, enter information into the TLB.
  - If valid and not memory resident, follow instructions in PTE to find the memory. This might be a device number and address, or an instruction to zero-fill a page (eg. `calloc`). Find a free page, place in contents, and restart instruction. It will now fault into case two and enter information into the TLB.

5.5 The x86 Virtual Memory System

- As per above, x86 addresses are split: 12 unused/36 page number/12 bits offset and have 4KB pages.
- There is a maximum of 2^{36} pages, which is too large for a flat page table.
- To solve this, the x86 uses a multi-level index:
  - The processor maintains a register `cr3` pointing to a physical page in memory.
  - There are three indirect blocks (pages of PTE's) called the L1 through L3 page tables.
  - Each L3 page table entry points to a L4 page table, which contain page table entries for actual virtual addresses.
  - Page table entries are 8 bytes; each page table has 2^9 page table entries. The 32 bits of virtual page number are split into L1 through L4 index.
  - Level 1,2, and 3 page table entries are arranged as follows:

Bit(s)	Name	Meaning
63	XD	Execute disable
12-51	-	Page table physical address
7	PS	Page size
5	A	Reference bit (see clock replacement)
4	CD	Allowed to cache
3	WT	Writethrough or Writeback policy
2	US	User or supervisor access
1	RW	Read only or Read/Write
0	P	Page present in memory

- Level 4 page table entries have the following additional information

Bit(s)	Name	Meaning
8	G	Global page (stays in TLB on process switch)
6	D	Dirty bit, set on write.

5.6 Page Replacement

- Memory is a cache for virtual memory pages, that might live on disk (swapfiles make everything live on disk).
- The operating system does not know the memory access pattern making many above eviction policies impossible.
- The solution is the *clock algorithm.* Pages track the use and dirty bits, the hardware maintains an index which iterates over all physical pages. When a new page needs to be allocated, iterate over memory clearing use bits and writing dirty data. Evict the first page with use bit 0.
- The clock algorithm approximates LRU.
- The *two handed clock algorithm* improves this by preemptively writing dirty data some fixed offset ahead of the clock index.
- Additionally, the operating system may make more complicated policies for large processes or to ensure fair sharing of memory.