Introduction to Computer Systems Cheat Sheet

Data representation

Byte = 8 bits

32-bit Machine

Char: 1 byte; Short: 2 bytes; Int: 4 bytes; Long: 4 bytes; Pointers: 4 bytes; Float: 4 bytes; Double: 8 bytes

64-bit Machine

Char: 1 byte; Short: 2 bytes; Int: 4 bytes; Long: 8 bytes; Pointers: 8 bytes; Float: 4 bytes; Double: 8 bytes

x86-64 systems

Char: 1 byte; Short: 2 bytes; Int: 4 bytes; Long: 8 bytes; Pointer: 8 bytes; Float: 4 bytes; Double: 8 bytes

- Decimal- Binary- Octal- Hexadecial

In C, numeric constants starting with "0x" or "0X" are interpreted as being in **hexadecimal**. The letters A to F is not Case-sensitive.

The most common computer representation of signed numbers is known as **two's-complement form**. This is defined by interpreting the most significant bit of the word to have negative weight. When the sign bit is set to 1, the represented value is negative, and when set to 0 the value is nonnegative. A useful equality: $-x = \sim x + 1$.

Consider the range of values that can be represented as a w-bit two's complement number. The least representable value is given by bit vector 10 ... 0 (set the bit with negative weight, but clear all others), having integer value **TMin(w bits)** $= -2^{w-1}$.

The greatest value is given by bit vector 01 ... 1 (clear the bit with negative weight, but set all others as "1"), having integer value **TMax(w bits)** $= 2^{w-1} - 1$.

So you can see: |TMin| = |TMax| + 1, UMax = 2TMax + 1, -1 has the same bit representation as UMax---a string of all ones.

$$U2T_w(u) = \begin{cases} u, & u < 2^{w-1} \\ u - 2^w, & u \geq 2^{w-1} \end{cases} \quad T2U_w(x) = \begin{cases} x + 2^w, & x < 0 \\ x, & x \geq 0 \end{cases}$$

When do calculation or comparison between unsigned and signed values, signed values implicitly cast to unsigned.

To convert an unsigned number to a larger data type, we can simply add leading zeros to the representation; this operation is known as **zero extension**.

For converting a two's- complement number to a larger data type, the rule is to perform a **sign extension**, adding copies of the most significant bit to the representation.

Shift operations

- Left shift
- **Logical right shift**: zero-extend
- **Arithmetic right shift**: sign-extend

Different effects for signed (2's complement) and unsigned numbers when simply throwing away the MSB in the overflow results

Array

- **Multi-Dimensional Arrays**
  - Row-major ordering in C
    - Each row is allocated contiguously, and all rows are allocated contiguously
- **Multi-Level Arrays**
  - The second-level arrays are not necessarily contiguous in memory

Satisfy alignment in structures with padding:

- Within structure: each field satisfies its own alignment requirement
- Overall structure: align to the largest alignment requirement of all fields

Big Endian: Least significant byte has highest address.

Little Endian: Least significant byte has lowest address.

Array not affected by byte ordering.

Big endian

	0x100	0x101	0x102	0x103	
...	01	23	45	67	...

Little endian

	0x100	0x101	0x102	0x103	
...	67	45	23	01	...

Floating-point

The IEEE floating-point standard represents a number in a form $V = (-1)^s \times M \times 2^E$:

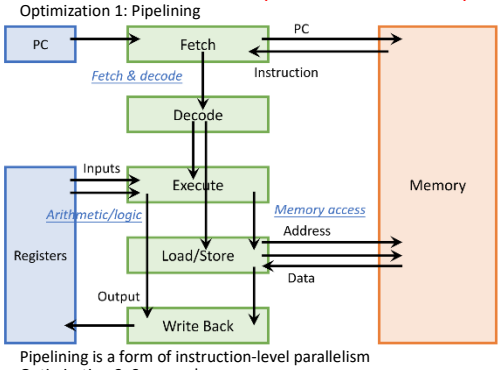
- The *sign* s determines whether the number is negative ($s = 1$) or positive ($s = 0$), where the interpretation of the sign bit for numeric value 0 is handled as a special case.
- The *significand* M is a fractional binary number that ranges either between 1 and $2 - \epsilon$ or between 0 and $1 - \epsilon$.
- The *exponent* E weights the value by a (possibly negative) power of 2.

The bit representation of a floating-point number is divided into three fields to encode these values:

- The single sign bit s directly encodes the sign s.
- The k-bit exponent field $\exp = e_{k-1} \cdots e_1 e_0$ encodes the exponent E.
- The n-bit fraction field $\text{frac} = f_{n-1} \cdots f_1 f_0$ encodes the significand M, but the value encoded also depends on whether or not the exponent field equals 0.

Hardware-software interface: ISA (instruction set architecture)

Optimization 1: Pipelining



Pipelining is a form of instruction-level parallelism

Optimization 2: Superscalar

Optimization 3: Multi-Core

PC and registers are private per-core; memory is shared

RISC-V ISA: States and Instructions

Program counter (PC): 32-bit

32 general-purpose registers (x0 to x31), each 32-bit

x0 is always fixed at 0; writing to x0 has no effect

Memory: 1 byte address

Word address is 4-byte aligned

Arithmetic/Logic Instructions

<op> rd, rs1, rs2 # inputs are 2 registers

<op>i rd, rs1, imm # inputs are 1 register and 1 constant

xor, add, sub, addi, subi

slli instruction: shift logical left by an immediate value

Load a 32-bit word (memory to register): lw rd, offset(rs1)

Store a 32-bit word (register to memory): sw rs2, offset(rs1)

Displacement addressing

- Address is given by a base register and a signed offset immediate
- Effective address is a byte address = register value + offset

Conditional branch: based on a condition

- Mark the target instruction of branch/jump with a label
- **beq rs1, rs2, L # branch to L if rs1 == rs2**
- **bne rs1, rs2, L # branch to L if rs1 != rs2**
- **blt rs, rt, label # branch to label if rs<rt**
- **bge rs, rt, label # branch to label if rs>=rt**

Sometimes we can invert the condition for convenience. if (a == d) => bne

Unconditional jump: always change flow

- **j L # jump to L**

Passing control: jump from caller to callee; return back

- **jal L**: jump-and-link, jump to L and save return position to register ra (x1)
- **jr ra**: jump to the address stored in ra

Test and set: **t&s reg, addr**

- reg = *addr; /* test */ if (reg == 0) *addr = 1; /* set */

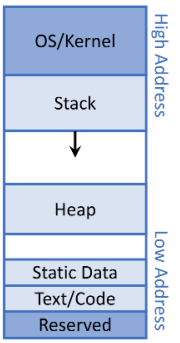
SIMD: Single-Instruction, Multiple-Data

Memory Layout of a Program

Heap: dynamic data allocation, managed by OS

Stack: to support (recursive) procedure calls in hardware

- Arguments, **local** variables, temporary variables, ...
- last in, first out (LIFO), to match nested procedure call
- Stack grows downwards (low address) and shrinks upwards



Primary storage, a.k.a., main memory, or simply memory

- Fast but volatile (lose contents after powered off)

Secondary storage, a.k.a., external storage (treated as I/O devices)

- Slow but non-volatile

Tradeoff: large memory capacity vs. fast memory access speed

Locality

Temporal locality (locality in time)

- If an item has been referenced, it will tend to be referenced again soon

Spatial locality (locality in space)

- If an item has been referenced, nearby items will tend to be referenced soon

Exploiting Locality: Memory Hierarchy

Caches

Data are always copied back and forth between level k and level k + 1 in block-size transfer units

- **Cache hit**: data found in the cache; serve with short latency
- **Cache miss**: data not found in the cache; need to fetch the block from memory, and may replace a block in the cache

An empty cache is sometimes referred to as a cold cache, and misses of this kind are called compulsory misses or **cold misses**.

Restrictive placement policies of this kind lead to a type of miss known as a **conflict miss**, in which the cache is large enough to hold the referenced data objects, but because they map to the same cache block, the cache keeps missing.

When the size of the working set exceeds the size of the cache, the cache will experience what are known as **capacity misses**.

Hit rate: percentage of accesses that hit

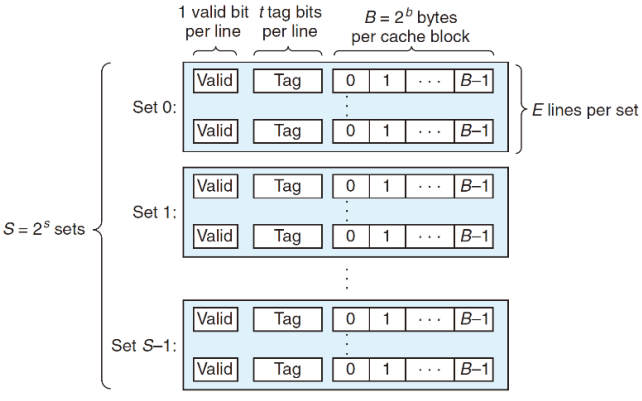
Miss rate = 1 – hit rate

Hit latency = data access latency on a hit = time to access cache

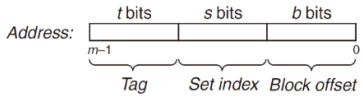
Miss penalty = overhead of fetching data from memory on a miss = time to access memory + time to deliver to cache (+ time to replace in cache)

AMAT (average memory access time) = hit latency + miss rate × miss penalty

At each level, Miss rate = local miss rate = # misses in this level / # accesses to this level = # misses in this level / # misses from the previous level and Miss penalty = AMAT of the next level, so AMAT formula is recursively applied



In general, a cache's organization can be characterized by the tuple (S, E, B, m). The size (or capacity) of a cache, C, is stated in terms of the aggregate size of all the blocks. The tag bits and valid bit are not



included. Thus, **C = S × E × B**.

The s set index bits in A form an index into the array of S sets. The first set is set 0, the second set is set 1, and so on. When interpreted as an unsigned integer, the **set index bits** tell us which set the word must be stored in. Once we know which set the word must be contained in, the **tag bits** in A tell us which line (if any) in the set contains the word. A line in the set contains the word if and only if the valid bit is set and the tag bits in the line match the tag bits in the address A. Once we have located the line identified by the tag in the set identified by the set index, then the **b block offset bits** give us the offset of the word in the B-byte data block.

The process that a cache goes through of determining whether a request is a hit or a miss and then extracting the requested word consists of three steps: **(1) set selection, (2) line matching, and (3) word extraction.**

Fully Associative Caches

A fully associative cache consists of a single set (i.e., E = C/B) that contains all of the cache lines.

Notice that there are no set index bits.

Direct-Mapped Caches

A cache with exactly one line per set (E = 1) is known as a direct-mapped cache.

Set-Associative Caches

A cache with 1<E<C/B is often called an **E-way set associative cache**.

A **least frequently used (LFU)** policy will replace the line that has been referenced the fewest times over some past time window. A **least recently used (LRU)** policy will replace the line that was last accessed the furthest in the past.

write-through, is to immediately write w's cache block to the next lower level.

write-back, defers the update as long as possible by writing the updated block to the next lower level only when it is evicted from the cache by the replacement algorithm. The cache must maintain an additional dirty bit for each cache line that indicates whether or not the cache block has been modified.

write allocate, loads the corresponding block from the next lower level into the cache and then updates the cache block.

Associativity->conflict misses; Blocksize->compulsory misses; Capacity->capacity misses

Processes and Threads

A process is an instance of a executing program

A thread is a single unique execution context including registers, PC, stack pointer, memory, ...

a process = one or multiple threads + an address space

Process executions are time-interleaved on the processor

Usually one process has one address space, which is shared by all threads of that process

Base and bound (B&B)

OS kernel represents each process as a process control block (PCB), an in-memory data structure that contains process states

hardware provides at least two modes

- **Kernel mode** (or system mode, supervisor mode, protected mode): high privilege
- **User mode**: limited privilege

Three Ways of User To Kernel Transition:

- **System call (syscall)**: user process purposely requests a system service; proactive
- **Interrupt**: an **external** event triggers a context switch to kernel; reactive
- **Exception and trap**: an **internal** event triggers a context switch to kernel; reactive; internal reasons

Context switch could be triggered by

- An interrupt of a timer, e.g., every 100 microseconds
- A syscall to voluntarily yield the processor, e.g., sleep

Syscalls for Process Management

- **exit** – terminate a process

exit is called but never returns (to the program)

- **fork** – copy the current process

Parent process creates a new running child process by calling fork. The newly created child process is almost, but not quite, identical to the parent. The child gets an identical (but separate) copy of the parent's user-level virtual address space, including the code and data segments, heap, shared libraries, and user stack. The child also gets identical copies of any of the parent's open file descriptors, which means the child can read and write any files that were open in the parent when it called fork. The most significant difference between the parent and the newly created child is that they have different PIDs.

Call once, return twice. The fork function is interesting (and often confusing) because it is called once but it returns twice: once in the calling process (the parent), and once in the newly created child process. **In the parent, fork returns the PID of the child. In the child, fork returns a value of 0.**

Concurrent execution. The parent and the child are separate processes that run concurrently. The instructions in their logical control flows can be interleaved by the kernel in an arbitrary way. In general, as programmers we can never make assumptions about the interleaving of the instructions in different processes.

- **exec** – change the program being executed by the current process

One call, no return if succeed.

- **wait** – wait for a process to finish

A terminated process that has not yet been reaped is called a **zombie**.

A process waits for its children to terminate or stop by calling the wait or waitpid function.

pid_t wait(int* wstatus)

- Suspends current process until one of its children terminates
- **Return value is the PID of the child process that terminated**

pid_t waitpid(pid_t pid, int* wstatus, int options)

- Suspends current process until the specific child process pid terminates

If a parent process terminates without reaping its zombie children, then the kernel arranges for the init process to reap them.

- **Signals** – small message to notify a process that some event has occurred

Sending/receiving signals: kill, sigaction

Scheduling

Waiting time: time when the job waits in the ready queue

Response time = waiting time + execution time

Throughput: number of jobs completed per unit of time **Fairness**: all jobs use resource in some equal way

Convoy effect: short jobs are blocked behind long jobs

Two classes

- **Non-preemptive**: once giving a job some period of time to execute

(not necessarily to completion), cannot take processor back before the period ends
- **Preemptive**: we can take processor back at any time, and use it for other jobs; preempted job goes back to ready queue and can resume later
FCFS (First-Come, First-Served)
Round Robin (RR): each job gets a small unit of time (time quantum, q) to execute, and then gets preempted and added back to end of queue
SJF (shortest job first): run the job with the shortest execution time first
SRTF (shortest remaining time first): preemptive version of SJF
SJF/SRTF is the optimal policy to minimize average response time

Basic shell command

1. Basic Commands
ls: Lists directory contents.
cd: Change directory.
mkdir: Make directory.
pwd: Print working directory.
cat: Concatenate and display files.
less / more: Pager programs to view long text files.
2. Managing Files and Directories
cp: Copy files and directories.
mv: Move or rename files and directories.
rm: Remove files or directories.
3. Redirection and Pipes
Redirection (>, >>, <): Redirects output to a file or reads input from a file.
Example: echo "Hello, World!" > hello.txt (writes "Hello, World!" to hello.txt).
Pipe (|): Sends the output of one command to another command as input.
Example: ls -l | grep "Jan" (lists files modified in January).
4. File Content and Size Management
wc (word count): Prints newline, word, and byte counts for each file.
Example: wc -l file.txt (counts the number of lines in file.txt).
df (disk free): Reports file system disk space usage.
Example: df -kh (shows disk space in human-readable form, including GB, MB).
5. Sorting and Manipulating Text
sort: Sorts lines of text files.
uniq: Reports or omits repeated lines.
6. Detailed File and Directory Manipulation
find: Searches for files in a directory hierarchy.
Example: find /home -name "*.txt" (finds all .txt files in the /home directory).
grep: Searches for patterns in files.
Example: grep "pattern" file.txt(Searches for pattern in file.txt and prints lines containing the pattern).
7. System Monitoring and Configuration
ps (process status): Reports a snapshot of current processes.
Example: ps -ef (shows every process running on the system).
top: Displays an up-to-date list of running processes.
kill: Sends a signal to a process, usually related to stopping the process.
nohup: Allows a command to continue running after logging out.
8. Shell globbing
?: Matches any single character.
Example: ls file?.txt would match files named file1.txt, file2.txt, etc., but not file10.txt.
*: Matches zero or more characters.
Example: ls *.txt would match any file ending in .txt, such as file.txt, document.txt, etc.
Brace Expansion Syntax: {item1,item2,...}
Expands into multiple items that can be used to generate strings for file names or command arguments.

Regex

[] The string of characters inside the braces specifies a disjunction of characters to match.
- In cases where there is a well-defined sequence associated with a set of characters, the brackets can be used with the dash (-) to specify any one character in a range.
^ If the caret ^ is the first symbol after the open square brace [, the resulting pattern is negated.
? This means "the preceding character or nothing".
***** The Kleene star means "zero or more occurrences of the immediately previous character or regular expression".
+ This is the Kleene +, which means "one or more occurrences of the immediately preceding character or regular expression".
. A wildcard expression that matches any single character (except a carriage return).
| Means disjunction
Anchors:

Regex	Match
^	start of line
$	end of line
\b	word boundary
\B	non-word boundary

Regex	Expansion	Match
\d	[0-9]	any digit
\D	[^0-9]	any non-digit
\w	[a-zA-Z0-9_]	any alphanumeric/underscore
\W	[^\w]	a non-alphanumeric
\s	[\r\t\n\f]	whitespace (space, tab)
\S	[^\s]	Non-whitespace

{n} exactly n occurrences of the previous char or expression
{n,m} from n to m occurrences of the previous char or expression
{n,} at least n occurrences of the previous char or expression
{,m} up to m occurrences of the previous char or expression

Concurrency and Synchronization

Concurrent threads are a very useful abstraction
- Allow transparent overlapping of computation and I/O
- Allow use of parallel processing when available
Concurrent threads introduce problems when accessing shared data
- Programs must be insensitive to arbitrary interleavings
- Without careful design, shared variables can become completely inconsistent
Atomic Operation: an operation that always runs to completion or not at all
Synchronization: using atomic operations to ensure cooperation between threads
Critical Section: piece of code that only one thread can execute at once
Mutual Exclusion: ensuring that only one thread executes critical section
Semaphores are a kind of generalized locks
P(s): If s is nonzero, then P decrements s and returns immediately. If s is zero, then suspend the thread until s becomes nonzero and the thread is restarted by a V operation. After restarting, the P operation decrements s and returns control to the caller.
V (s): The V operation increments s by 1. If there are any threads blocked at a P operation waiting for s to become nonzero, then the V operation restarts exactly one of these threads, which then completes its P operation by decrementing s.
The basic idea is to associate a semaphore s, initially 1, with each shared variable (or related set of shared variables) and then surround the corresponding critical section with P(s) and V (s) operations.
A semaphore that is used in this way to protect shared variables is called a **binary semaphore** because its value is always 0 or 1. Binary semaphores whose purpose is to provide mutual exclusion are often called **mutexes**. Performing a P operation on a mutex is called **locking the mutex**. Similarly, performing the V operation is called **unlocking the mutex**. A thread that has locked but not yet unlocked a mutex is said to be **holding the mutex**. A semaphore that is used as a counter for a set of available resources is called a **counting semaphore**.
Mutual Exclusion (initial value = 1)
Scheduling Constraints (initial value = 0)
- Allow thread 1 to wait for a signal from thread 2, i.e., thread 2 schedules thread 1 when a given constrained is satisfied
Producer-Consumer Problem
A producer and consumer thread share a bounded buffer with n slots. The producer thread repeatedly produces new items and inserts them in the buffer. The consumer thread repeatedly removes items from the buffer and then consumes (uses) them.
Since inserting and removing items involves updating shared variables, we must guarantee mutually exclusive access to the buffer. But guaranteeing mutual exclusion is not sufficient. We also need to schedule accesses to the buffer. If the buffer is full (there are no empty slots), then the producer must wait until a slot becomes available. Similarly, if the buffer is empty (there are no available items), then the consumer must wait until an item becomes available.

```
Semaphore fullSlots = 0;   // Initially, no coke
Semaphore emptySlots = bufSize;   // Initially, num empty slots
Semaphore mutex = 1;  // No one using machine
Producer(item) {
      emptySlots.P();  // Wait until space
      mutex.P();  // Wait until machine free
      Enqueue(item);
      mutex.V();
      fullSlots.V();  // Tell consumers there is more coke
}
Consumer() {
      fullSlots.P();  // Check if there's a coke
      mutex.P();  // Wait until machine free
      item = Dequeue();
      mutex.V();
      emptySlots.V();  // tell producer need more
      return item;
}
```

The order of P's is important, the order of V's is not important.
Monitor: a lock and zero or more condition variables for managing concurrent access to shared data
Condition Variable: a queue of threads waiting for something inside a critical section
Operations:
- **Wait(&lock)**: Atomically release lock and go to sleep. Re-acquire lock later, before returning.
- **Signal()**: Wake up one waiter, if any
- **Broadcast()**: Wake up all waiters

```
Lock lock;
Condition dataready;
Queue queue;
AddToQueue(item) {
      lock.Acquire();  // Get Lock
      queue.enqueue(item);  // Add item
      dataready.signal();  // Signal any waiters
      lock.Release();  // Release Lock
}
RemoveFromQueue() {
      lock.Acquire();  // Get Lock
      while (queue.isEmpty()) {
            dataready.wait(&lock);   // If nothing, sleep
      }
      item = queue.dequeue();  // Get next item
      lock.Release();  // Release Lock
      return(item);
}
```

Hoare monitors
Signaler gives up lock and CPU to waiter; waiter runs immediately.
Waiter gives up lock, processor back to signaler when it exits critical section or if it waits again.
In Hoare Monitors, when a process signals a condition, it immediately transfers control to a waiting process. This means the signaling process stops its execution and the waiting process resumes, using the resource or condition that has just been signaled.
Mesa monitors
Signaler keeps lock and processor
Waiter placed on a local "e" queue for the monitor
In Mesa Monitors, when a process signals a condition, it does not immediately transfer control to a waiting process. Instead, the signaling process continues, and the waiting process is simply moved to the ready queue.
The waiting process in Mesa Monitors must recheck the condition upon waking up because there's no guarantee that the condition is still true by the time it gets processor control. This is known as "spurious wake-ups" and requires extra handling in the code. So we should use "while()" instead of "if() with Mesa monitors.
Readers-Writers Problem
Writers must have exclusive access to the object, but readers may share the object with an unlimited number of other readers. In general, there are an unbounded number of concurrent readers and writers.
The second readers-writers problem, which favors writers, requires that once a writer is ready to write, it performs its write as soon as possible.

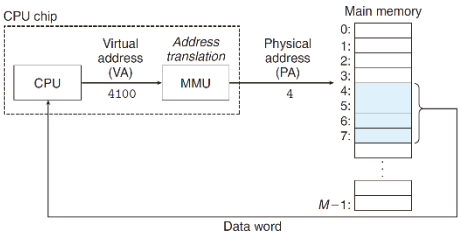
```
Reader() {
      // check into system
      lock.Acquire();
      while ((AW + WW) > 0) {
            WR++;
            okToRead.wait(&lock);
            WR--;
      }
      AR++;
      lock.release();
      // read-only access
      AccessDbase(ReadOnly);
      // check out of system
      lock.Acquire();
      AR--;
      if (AR == 0 && WW > 0)    okToWrite.signal();
      lock.release();
}
Writer() {
      // check into system
      lock.Acquire();
      while ((AW + AR) > 0) {
            WW++;
            okToWrite.wait(&lock);
            WW--;
      }
      AW++;
      lock.release();
      // read/write access
      AccessDbase(ReadWrite);
      // check out of system
      lock.Acquire();
      AW--;
      if (WW > 0){
            okToWrite.signal();
      } else if (WR > 0) {
            okToRead.broadcast();
      }
      lock.release();
}
int readcnt; /* Initially = 0 */
sem_t mutex, w; /* Both initially = 1 */
void reader(void){
   while (1) {
      P(&mutex);
      readcnt++;
      if (readcnt == 1) /* First in */
            P(&w);
      V(&mutex);
      /* Critical section *//* Reading happens */
      P(&mutex);
      readcnt--;
      if (readcnt == 0) /* Last out */
            V(&w);
      V(&mutex);
   }
}
void writer(void){
   while (1) {
      P(&w);
      /* Critical section *//* Writing happens */
      V(&w);
   }
}
```

starvation, where a thread blocks indefinitely and fails to make progress.
deadlock, where a collection of threads is blocked, waiting for a condition that will never be true.
Four requirements for Deadlock
- **Mutual exclusion**
Only one thread at a time can use a resource
- **Hold and wait**
Thread holding at least one resource is waiting to acquire additional resources held by other threads
- **No preemption**
Resources are released only voluntarily by the thread holding the resource, after thread is finished with it
- **Circular wait**
There exists a set {T1, …, Tn} of waiting threads
T1 is waiting for a resource that is held by T2; T2 is waiting for a resource that is held by T3⋯Tn is waiting for a resource that is held by T1
Methods for Handling Deadlocks
- Allow system to enter deadlock and then recover
- Deadlock prevention: ensure that system will never enter a deadlock
- Ignore the problem and pretend that deadlocks never occur in the system
Techniques for Preventing Deadlock
- Infinite resources
  - Include enough resources so that no one ever runs out of resources. Doesn't have to be infinite, just large
- No Sharing of resources (totally independent threads)
- Don't allow waiting
Mutex lock ordering rule: Given a total ordering of all mutexes, a program is deadlock-free if each thread acquires its mutexes in order and releases them in reverse order.

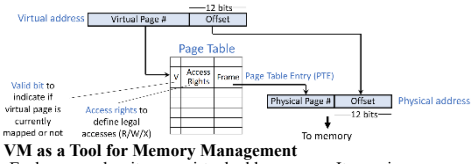
Virtual Memory & Memory Management

Virtual memory mechanism provides fine-grained and dynamic management of address spaces
Physical and Virtual Addressing
The main memory of a computer system is organized as an array of M contiguous byte-size cells. Each byte has a unique **physical address (PA)**.
In a system with virtual memory, the CPU generates virtual addresses from an address space of N = 2^n addresses called the **virtual address space**: {0, 1, 2, . . . , N − 1}
-Contiguous and linear
A system also has a **physical address space** that corresponds to the M bytes of physical memory in the system: {0, 1, 2, . . . , M − 1}
All virtual address spaces are mapped to the physical address space in a flexible way, in the unit of **pages**. (Each page is typically 4 kB, but sometimes can also be 2 MB or 1 GB)
Unallocated. Pages that have not yet been allocated (or created) by the VM system. Unallocated blocks do not have any data associated with them, and thus do not occupy any space on disk.

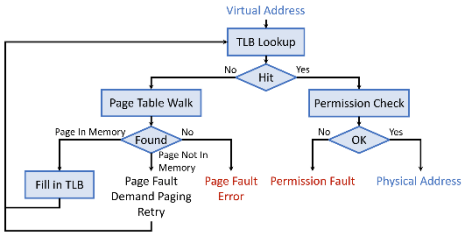
Cached. Allocated pages that are currently cached in physical memory.
Uncached. Allocated pages that are not cached in physical memory. Contiguous virtual pages may not be contiguous in physical address space.



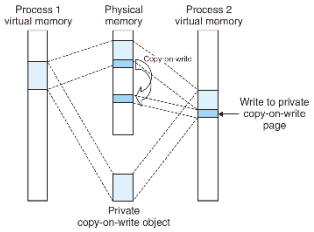
Page table: translate virtual address to physical address
-One page table entry (PTE) per virtual page (lower address bits are not translated)
-Each PTE includes: valid bit, physical page number (a.k.a., frame number), metadata (e.g., R/W/X permissions)



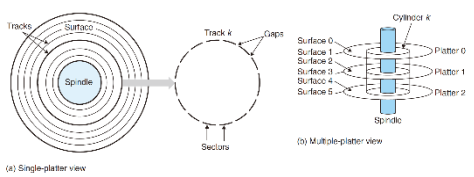
VM as a Tool for Memory Management
-Each process has its own virtual address space. It can view memory as the same simple linear array as before.
VM as a Tool for Memory Protection
-One process cannot directly access other's data through its virtual address space
-User processes cannot access kernel if not mapped to its virtual address space
-Each page has permission information stored in page table
--PTE has permission bits: Read, Write, eXecute
Virtual memory as a tool for data caching
-Use DRAM as a cache for part of a virtual address space; only the data that need to be accessed are mapped to physical DRAM
-Allocated but unmapped pages are swapped out to external storage
Page fault is a type of exception, raised when hardware finds a page miss in DRAM.
Demand paging: if a page fault occurs, the page is paged in on demand, by the page fault handler in OS kernel.
Invoke OS page fault handler to move data from storage into memory
The process suspends until resolved, and retries the faulted load/store
If it is a valid page (V == 1)
-Check access permissions (R, W, X) against access type
-If allowed, translate to physical address and access memory
-Otherwise, generate a permission fault, i.e., an exception, handled by OS
Otherwise, an invalid page (V == 0), i.e., page is not currently in memory
-Generate a page fault, i.e., an exception, handled by OS
-If it is due to program errors, e.g., unallocated: terminate process
-If page is on external storage: refill & retry, i.e., demand paging



At any time, the program tend to access a limited set of active pages, called its **working set**.
Speeding Up Translation With TLB
Translation Look-aside Buffer (TLB) = a hardware cache just for address translation entries, i.e., PTE
If we miss in TLB, access PTE in memory; this is called **page table walk**.
-If page is in memory, copy PTE into TLB and retry
-If page is not in memory, resolve the page fault first, then fill in TLB
Multi-Level Page Tables
-Only top level must be resident in memory
-Remaining levels can be in memory or on disk
-Or unallocated if corresponding ranges of the virtual address space are not used
Page sharing: different processes can share a page by setting their virtual addresses to point to the same physical address
Copy-on-Write (CoW)
-When copying, just make destination virtual address point to the same physical page as source virtual address, and mark the page as write-protected in PTE
-When writing to either virtual address, trigger a permission fault and do actual copy, i.e., allocate a new physical page



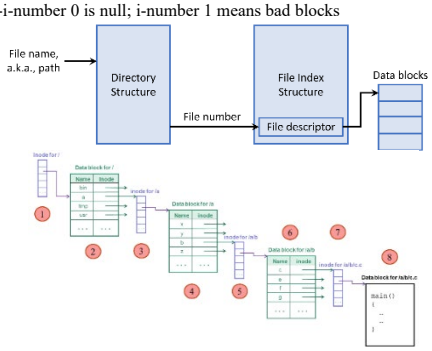
Memory mapping: associate a virtual memory area with a file object
-Get initial data from this file into memory: regular file on disk
Because of demand paging, none of these virtual pages is actually swapped into physical memory until the CPU first touches the page (i.e., issues a virtual address that falls within that page's region of the address space).
-Swap the physical memory content from/to this file: special swap file
-For newly allocated space initialized to all zero values: special **anonymous file**
The first time the CPU touches a virtual page in such an area, the kernel finds an appropriate victim page in physical memory, swaps out the victim page if it is dirty, overwrites the victim page with binary zeros, and updates the page table to mark the page as resident.
Dynamic Memory Allocation
External Fragmentation
-Naïve allocator policy like first-fit may cause fragmented memory space
Segregated Free Lists
-Organize free spaces into multiple linked lists for different sizes
Internal Fragmentation
-Internal fragmentation happens when the requested size is smaller than the allocated free space, and the remaining space is wasted
Garbage Collection(GC)
automatic reclamation of heap-allocated storage
Classical GC Algorithm: Mark-and-Sweep
Root nodes: objects that are not dynamically allocated in the heap. E.g., registers, locations on the stack, global variables, ...
Reachable nodes: heap objects that are reachable from any root node
Non-reachable nodes are garbage to be collected
Mark: start from root nodes, traverse graph and mark a bit in each reachable object; non-reachable objects are not visited and not marked
Sweep: scan all objects in the memory, and free unmarked objects
Classical GC Algorithm: Reference Counting
Each object maintains a reference counter, which keeps the number of other pointers that point to this object. When the reference counter becomes zero, we free the object
Problem: does not work for circular cases
-Mark-and-sweep delays memory reclamation, and may temporarily pause program execution
-Reference counting is simple, frees objects more timely, has small impact on normal program execution; but may leak memory
I/O Devices
Memory-Mapped I/O
-How to access I/O devices?
--Each I/O device may have interface registers and data buffers
-Memory-mapped I/O: assign an unused range of physical address to each I/O device, which is called its I/O address
I/O address space is protected by the virtual memory mechanism
-Method 1: use syscalls to access I/O addresses, e.g., read, write
-Method 2: ask OS to map I/O addresses to virtual addresses, e.g., mmap
--After mapping, can access I/O devices through normal load and store instructions, just like accessing normal virtual memory addresses
I/O Types By Access Granularity/Pattern
-Block device (e.g., disks, SSDs)
-Stream device
--Character/byte device (e.g., keyboards, mice, serial ports)
--Network device (e.g., Ethernet, wireless, Bluetooth)
I/O Types By Timing
-Blocking interface, i.e., "wait"
-Non-blocking interface, i.e., "do not wait"
-Asynchronous interface, i.e., "tell me later"\
I/O Notification with Polling
-The OS periodically checks the status register (may be memory-mapped)
I/O Notification with Interrupts
-When I/O device needs attention, it interrupts the processor
Avoid waste CPU time: interrupts is better than polling
Minimize extra overheads: polling is better than interrupts
Direct Memory Access (DMA)
A custom hardware engine for data movement
-Transfer blocks of data to or from memory without CPU intervention
-Processor sets up DMA by supplying
--Identity of the device and the operation (read/write)
--Memory address for source/destination
--Number of bytes to transfer
-DMA engine starts transfer when data is ready
-Notify processor when complete or on error, typically with interrupts
I/O Devices: Disks and SSDs
Magnetic Hard Disks
-Long-term, non-volatile
-Large, inexpensive
-Slow
Disks are constructed from **platters**. Each platter consists of two sides, or **surfaces**, that are coated with magnetic recording material. A rotating spindle in the center
of the platter spins the platter at a fixed rotational rate, typically between 5,400 and 15,000 revolutions per minute (RPM). A disk will typically contain one or more of
these platters encased in a sealed container.
Each surface consists of a collection of concentric rings called **tracks**. Each track is partitioned
into a collection of **sectors**. Each sector contains an equal number of data bits (typically 512 bytes) encoded in the magnetic material on the sector. Sectors are separated by gaps where no data bits are stored. Gaps store formatting bits that identify sectors.
Disk manufacturers describe the geometry of multiple-platter drives in terms of **cylinders**, where a cylinder is the collection of tracks on all the surfaces that are equidistant from the center of the spindle.
By moving the arm back and forth along its radial axis, the drive can position the head over any track on the surface. This mechanical motion is known as a **seek**.
Disk Access Time Breakdown
Controller overheads
Seek time. To read the contents of some target sector, the arm first positions the head over the track that contains the target sector. The time required to move the arm is called the seek time.
Rotational latency. Once the head is in position over the track, the



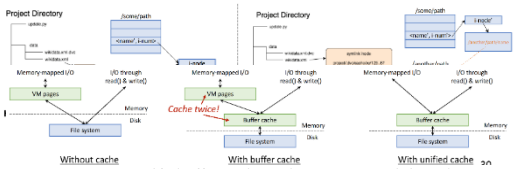
drive waits for the first bit of the target sector to pass under the head.
Transfer time. When the first bit of the target sector is under the head, the drive can begin to read or write the contents of the sector. We can roughly estimate the average transfer time for one sector in seconds as

$$T_{\text{avg transfer}} = \frac{1}{\text{RPM}} \times \frac{1}{(\text{average \# sectors/track})} \times \frac{60 \text{ secs}}{1 \text{ min}}$$

Disk Scheduling Policies
-FIFO: in the order of arrival
-Shortest seek time first (SSTF): pick the request that is closest to head
-SCAN: elevator algorithm, take the closest request in the travel direction
-C-SCAN (circular SCAN): like SCAN but only serves in one direction
Solid State Drive (SSD), a.k.a., Flash
SSDs are usually made by NAND/NOR multi-level cell Flash memory
4 kB per page; 32 to 128 pages per block
Reads are in units of pages
Flash Translation Layer (FTL)
No in-place modification in SSD/Flash. Must write data to a new page in an erased block; then let the address point to this page. Such management is handled by the Flash Translation Layer (FTL).
Address mapping: logical to physical
Wear Out and Wear Leveling
File Systems; Protection
File system: a component in OS that transforms block interface into high-level interface like files and directories
Files Are Named
When a file is named, it becomes independent of the process, the user, and even the system that created it
From a user's perspective, a file is the smallest granularity of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file
A file system mainly consists of two parts
-A collection of files, each storing relevant data in a group of disk blocks
-A directory structure, organizing files and mapping names to files
Disk management: organize disk blocks into files
-Track which blocks contain data for which file: where to access a file on disk
-Track free blocks on disk: where to put newly written data
Naming: be able to find files by names, not by blocks
-Track what files are in a directory: find a file given its name
Protection: keep different users' data isolated
Reliability/durability: keep data consistent despite crashes, failures, etc.
File descriptor: OS data structure about a file's metadata information. Called **inode** in Unix-style systems
Metadata in a file descriptor, i.e., file attributes
-File size, File location, Access timestamps, e.g., create, last read, last write, Protection info: e.g., access permissions (R/W/X), owner ID, group ID, etc.
File Operations
Open: use name to search a file in the directory, i.e., resolve file name
-Translate file name into a file number (i-number in Unix), which is used to locate the file descriptor (inode in Unix) on disk
Return a file handle (usually an integer) to user process
Two levels of open-file tables: **per-process** and **system-wide**
File Access Patterns
Sequential: data are processed in order, one byte after another
Random: can address any byte in the middle of a file directly
Keyed (or indexed): search for blocks with particular contents
Data Block Organization
file data are always in full blocks
most files are small, growing numbers of files
most of the space is occupied by the rare big files
Data Block Organization 1: Contiguous
Simplest approach: contiguous allocation
Data Block Organization 2: Linked Files
Linked files: keep a linked list of all blocks of a file
-Maintain a separate FAT (File Allocation Table)
Data Block Organization 3: Indexed Files
Indexed files: keep the set of pointers to data blocks
-Example: 4.3 BSD Unix – Multi-Level Index
File descriptor has 14 block pointers
-First 12 point to data blocks (direct blocks)
-Next one points to an indirect block, which contains 1024 pointers to data blocks
--Indirect block is not allocated until needed
-Next one points to a double-indirect block
BSD Unix approach: bitmap
Directory
Directory is used to map names to file numbers
Directories are just like regular file
"Data" of a directory is an unordered list of <name, pointer> pairs
A special directory called the root has no name, and has i-number of 2
-Fixed location on disk
-i-number 0 is null; i-number 1 means bad blocks



Hard link: set another directory entry to the file number for a file
-Create another name (path) for the same file
Soft or symbolic link: special file whose content is another name (path)
-Stored as regular files, but with a flag set in file descriptor
Buffer Cache
Use part of main memory to retain recently accessed disk blocks



Many OSes now unify buffer cache and VM page pool, in order to avoid double caching issue for memory-mapped files

Synchronous writes: immediately write through to disk

Delayed writes: do not immediately write to disk, e.g., wait for a while

Protection

Protection: prevent accidental and malicious misuse

-Authentication: identify a responsible party (principal)

Typically done with passwords

-Authorization: determine which principal can perform what actions on which objects

--Access Control List (ACL)

---Organized by columns: with each object, store info about which principals are allowed to perform what operations

--Capabilities

---Organized by rows: with each principal, indicate which objects may be accessed and in what ways

-Access enforcement: combine authentication and authorization to control access

Basic Network Functionalities

-Delivery: deliver packets between to any two hosts in the Internet without a large number of physical wires

-Reliability: tolerate packet losses

-Flow control and congestion control: avoid overflowing the receiver + avoid overflowing the routers

Multiplexing

Basic Building Block: Links

-Circuit Switching

-Packet Switching

Each endpoint is identified by a port number

MAC address(Media Access Control Address): 48-bit unique identifier assigned by card vendor

IP address(Internet Protocol Address): 32-bit (or 128-bit for IPv6) address assigned by network administrator or dynamically when computer connects to network

Port number: 16-bit identifier assigned by app or OS

Globally, an endpoint is identified by (IP address, port number)

Addressing: finding the right target

Addressing on a local area network (LANs)

-Broadcast search for IP address

--Hubs: All hosts in a LAN can share the same physical communication media.

--Switches: Hosts in same LAN can be also connected by switches. A switch forwards frames only to intended recipients

Wide Area Network (WAN): network that covers a broad area (e.g., city, state, country, entire world)

WAN connects multiple datalink layer networks (LANs)

Datalink layer networks are connected by **routers**

WANs: Routers

-Forward each packet received on an **incoming link** to an **outgoing link** based on packet's destination IP address (towards its destination)

-Store & forward: packets are buffered before being forwarded

-Forwarding table: mapping between IP address and the output link

Human-friendly Naming: DNS

Reliability and flow control

-Latency – how long does it take for the first bit to reach destination

-Capacity — how many bits/sec can we push through? (often termed "bandwidth")

-Jitter — how much variation in latency?

-Loss / reliability — can the channel drop packets?

-Packet reordering

Data may get Lost

Data may get Corrupted

Data may be delivered Out of Order

Avoid overflowing the receiver: Polite Networks: Stop & Wait Protocol

-Accomplished by using acknowledgements (ACK) and timeouts

Packet Delay

-Propagation delay on each link.

--Proportional to the length of the link

-Transmission delay on each link.

--Proportional to the packet size and 1/(link speed)

-Processing delay on each router.

--Depends on the speed of the router

-Queuing delay on each router.

--Depends on the traffic load and queue size

Layering

Introduce intermediate layers that provide set of abstractions for various network functionality & technologies

Properties of Layers

-Service: what a layer does (functionality)

-Service interface: how to access the service

Interface for layer above

-Protocol: how peers communicate to use the service

Set of rules and formats that specify the communication between network elements

Protocol Standardization

Ensure communicating hosts speak the same protocol

Layer 1: Physical layer.

At the physical layer, the communicating systems must agree on the electrical representation of a binary 0 and 1, so that when data are sent as a stream of electrical signals, the receiver is able to interpret the data properly as binary data. This layer is implemented in the hardware of the networking device. It is responsible for delivering bits.

Layer 2: Data-link layer.

The data-link layer is responsible for handling frames, or fixed-length parts of packets, including any error detection and recovery that occur in the physical layer. It sends frames between physical addresses.

Layer 3: Network layer.

The network layer is responsible for breaking messages

into packets, providing connections between logical addresses, and routing packets in the communication network, including handling the addresses of outgoing packets, decoding the addresses of incoming packets,

and maintaining routing information for proper response to changing load levels. Routers work at this layer.

Layer 4: Transport layer.

The transport layer is responsible for transfer of messages between nodes, maintaining packet order, and controlling flow to avoid congestion.

Layer 5: Session layer.

The session layer is responsible for implementing

sessions, or process-to-process communication protocols.

Layer 6: Presentation layer.

The presentation layer is responsible for resolving the differences in formats among the various sites in the network, including character conversions and half duplex–full duplexmodes(character echoing).

Layer 7: Application layer.

The application layer is responsible for interacting

directly with users. This layer deals with file transfer, remote-login protocols, and electronic mail, as well as with schemas for distributed databases.

Internet Transport Protocols

-User Datagram Protocol (UDP)

-consider packet loss, delay and jitter

--Live Streaming: For audio/video where timely delivery is more important than perfect accuracy.

--Online Gaming: Where real-time interaction is critical and minor data loss is acceptable.

--Network Discovery Protocols: For broadcasting information to all devices on a network.

--Simple Query/Response Protocols

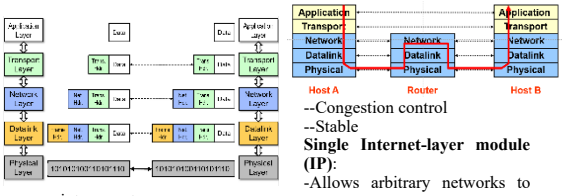
-Transmission Control Protocol (TCP)

--Connection set-up & tear-down

--Discarding corrupted packets (segments)

--Retransmission of lost packets (segments)

--Flow control



--Congestion control

--Stable

Single Internet-layer module (IP):

-Allows arbitrary networks to interoperate

-Allows applications to function on all networks

-Supports simultaneous innovations above and below IP

Drawbacks of Layering

-Layering can hurt performance

-Headers start to get really big

-Layer N may duplicate layer N-1 functionality

-Layers may need same information

If you have to implement a function end-to-end anyway, don't implement it inside the communication system

Remote procedure calls

Inter-process communication (IPC)

-fork() – return values

-Signal

-File

-including stdin / stdout

-Pipes

-Shared memory segments

-Networking - Socket

Communication mode 1: Clients and Servers

Client requests services and server provide services

Communication mode 2: Peer-to-Peer Communication

Each host is both server and client

Protocols

A protocol is an agreement on how to communicate

-Syntax: how a communication is specified & structured

-Semantics: what a communication means

Marshalling Arguments

Marshalling involves converting values to a canonical form, serializing objects, copying arguments passed by reference, etc.

Client and server use "stubs" to glue pieces together

Structured Data

A data model is a collection of entities and their relationships

Transactions

Databases and queries

A **data model** is a collection of entities and their relationships

A **schema** is an instance of a data model

E.g., describes the fields in the database; how the database is organized

A **relational data model** is the most used data model

-Relation, a table with rows and columns

-Every relation has a schema which describes the fields in the columns

A **Database Management System (DBMS)** is a software system designed to store, manage, and facilitate access to databases.

A DBMS provides:

-Data Definition Language (DDL)

--Define relations, schema

-Data Manipulation Language (DML)

--Queries – to retrieve, analyze and modify data.

-Guarantees about durability, concurrency, semantics, etc

A general way to reliability: transactions

-An atomic sequence of database actions (reads/writes)

-Takes DB from one consistent state to another

DBMS provides (limited) automatic enforcement, via integrity constraints (IC)

The ACID properties of Transactions

-Atomicity: all actions in the transaction happen, or none happen

-Consistency: transactions maintain data integrity, e.g., Balance cannot be negative, Cannot reschedule meeting on February 30

-Isolation: execution of one transaction is isolated from that of all others; no problems from concurrency

-Durability: if a transaction commits, its effects persist despite crashes

Implementing transactions: atomic writes to disk

Operations on persistent storage are not atomic !

Atomicity on persistent storage

-Even writing a single block is not atomic

-Many operations involves many reads/writes

-If broken, cannot recovery from a single reboot

Log

-One simple action is atomic – write/append a basic item

-Use that to seal the commitment to a whole series of actions

Implementing transactions: Locks

Transaction Scheduling

-Serial schedule: A schedule that does not interleave the operations of different transactions

--Transactions run serially (one at a time)

-Equivalent schedules: For any storage/database state, the effect (on storage/database) and output of executing the first schedule is identical to the effect of executing the second schedule

-Serializable schedule: A schedule that is equivalent to some serial execution of the transactions

--Intuitively: with a serializable schedule you only see things that could happen in situations where you were running transactions one-at-a-time

Anomalies with Interleaved Execution

-Read-Write conflict

-Write-Read Conflict

-Write-Write Conflict

Conflict Serializable Schedules

Two operations conflict if they

-Belong to different transactions

-Are on the same data

-At least one of them is a write

Two schedules are conflict equivalent iff:

-Involve same operations of same transactions

-Every pair of conflicting operations is ordered the same way

Schedule S is **conflict serializable** if S is conflict equivalent to some serial schedule

Conflict serializable => serializable

If you can transform an interleaved schedule by swapping consecutive non-conflicting operations of different transactions into a serial schedule, then the original schedule is conflict serializable

Dependency graph:

-Transactions represented as nodes

-Edge from Ti to Tj:

--an operation of Ti conflicts with an operation of Tj

--Ti appears earlier than Tj in the schedule

Theorem: Schedule is conflict serializable if and only if its dependency graph is acyclic

Serializability ⇏ Conflict Serializability

Two types of locks:

-shared (S) lock – multiple concurrent transactions allowed to operate on data

-exclusive (X) lock – only one transaction can operate on data at a time

Two-Phase Locking (2PL)

-Each transaction must obtain:

--S (shared) or X (exclusive) lock on data before reading,

--X (exclusive) lock on data before writing

-A transaction can not request additional locks once it releases any locks

2PL guarantees that the dependency graph of a schedule is acyclic.

=> Conflict Serializable

Distributed Systems

Distributed systems overview

A collection of independent computers that appears to its users as a single coherent system.

Remote procedure calls

Network file system (NFS) and Andrew file system (AFS)

NFS servers are stateless; each request provides all arguments require for execution

--E.g. reads include information for entire operation, such as ReadAt(inumber, position), not Read(openfile)

-No need to perform network open() or close() on file – each operation stands on its own

-Server keeps no state about client, except as hints to help improve performance (e.g. a cache)

Idempotent: Performing requests multiple times has same effect as performing it exactly once

-Example: Server crashes between disk I/O and message send, client resend read, server does operation again

-Example: Read and write file blocks: just re-read or re-write file block – no side effects

-Example: What about "remove"? NFS does operation twice and second time returns an advisory error

NFS Pros:

-Simple, Highly portable

NFS Cons:

-Sometimes inconsistent!

-Doesn't scale to large # clients

AFS Cell/Volume Architecture

Cells correspond to administrative groups

-/afs/andrew.cmu.edu is a cell

Cells are broken into volumes (miniature file systems)

-One user's files, project source tree, ...

-Typically stored on one server

-Unit of disk quota administration, backup

Client machine has cell-server database

-protection server handles authentication

-volume location server maps volumes to servers

More aggressive caching (AFS caches on disk in addition to just in memory)

-Cache the whole file

Prefetching (on open, AFS gets entire file from server, making later ops local & fast).

-Remember: with traditional hard drives, large sequential reads are much faster than small random writes.

-So easier to support (client A: read whole file; client B: read whole file) than having them alternate.

-Improves scalability, particularly if client is going to read whole file anyway eventually.

File access consistency

In UNIX local file system, concurrent file reads and writes have "sequential" consistency semantics

-Each file read/write from user-level app is an atomic operation

-The kernel locks the file vnode

-Each file write is immediately visible to all file readers

Ideal: One Copy Semantics

Neither NFS nor AFS provides such concurrency control

-NFS: "sometime within 30 seconds"

-AFS: session semantics for consistency

--A file write is visible to processes on the same box immediately, but not visible to processes on other machines until the file is closed