Math Review: Counting and Addressing

ICS332 Operating Systems

Henri Casanova (henric@hawaii.edu)

Disclaimer

- The content in these slides will be obvious to many of you
 This is a good thing!
- But when I teach this course, this material often causes problems
 - Although it's not technically OS material
- And we need it to be solid for this second part of the semester
 - □ And for the rest of your life!
- So I am presenting is here now, so that students who have difficulties with this have time to practice before it becomes critical for this course

Units of Storage

The smallest unit of information is the bit
 Anybody knows why it's called a bit?

The basic unit of memory is a byte

I Byte = 8 bits

□ 1 KiB = 2¹⁰ Byte = 1,024 bytes

 \square 1 MiB = 2¹⁰ KiB = 2²⁰ bytes (1 Million) (mega)

 \square 1 GiB = 2¹⁰ MiB = 2³⁰ bytes (1 Billion) (giga)

$$\square$$
 1 TiB = 2¹⁰ GiB = 2⁴⁰ bytes (1 Trillion) (tera)

 \square 1 EiB = 2¹⁰ PiB = 2⁶⁰ bytes (1 Million Trillion) (exa)

Often the "i" above is missing, which is not great
 1GB = 10⁹ bytes, but 1GiB = 2³⁰ bytes!

You have to know the units and order above!!!

Exponents, Logarithms

- We'll use Exponents:
 - $\Box \ \alpha_{x} \cdot \alpha_{\lambda} = \alpha_{x+\lambda}$
 - $\Box \alpha^{-x} = 1 / \alpha^{x}$
 - $\Box \alpha^{x} / \alpha^{y} = \alpha^{x-y}$
- But we'll do only powers of 2:
 - $\Box \ 2^{x} \cdot 2^{y} = 2^{x+y}$
 - □ 2⁻x = 1 / 2^x
 - $\Box 2^{x}/2^{y} = 2^{x-y}$
- We'll use Logs:
 - $\Box \log_{\alpha} \alpha^{n} = n$
- But only for base 2:
 - $\Box \log_2 2^n = n$
 - Not to be confused with the natural logarithm, In, which is really log_e (In e^x = x), and which you've seen in Calculus
 - In computer science: log₂ is often just written as log, especially when we deal with asymptotic computational complexities (e.g., O(log₂ n) = O(log₁₀ n))
- I am going to assume the above is solid for everyone, but if it's not, you know what you have to do...

Counting Bytes

- When studying operating systems, we often need to count "chunks" of bytes in some memory space
- Example #1: how many 1MiB chunks are there in a 8MiB file?
 - □ easy: 8
- Example #2: how many 4KiB chunks are there in a 8GiB file?
 - not so easy perhaps?
- The way to do this: use powers of 2
 - We want results in powers of 2 anyway because numbers are typically too large to just write them out in decimal conveniently

Examples

How many groups of 12 parking spots are there in 252-spot parking lot?

answer: 252 / 12 = 21

- How many groups of x thingies are there in a set of y thingies?
 answer: y / x
- How many 2KiB chunks are there in 1 GiB?
 - \Box 1 GiB = 2³⁰ bytes
 - \square 2 KiB = 2×2¹⁰ = 2¹¹ bytes
 - answer: 2³⁰ / 2¹¹ = 2¹⁹ chunks

How many 8 KiB chunks are there in 128 MiB?

- \square 128 MiB = 2⁷ × 2²⁰ = 2²⁷ bytes
- \square 8 KiB = 2³ × 2¹⁰ = 2¹³ bytes

answer: 2²⁷ / 2¹³ = 2¹⁴ chunks

Addressing

We often partition thingies into chunks

- Partition a pie into slices
- Partition a computer's memory into bytes
- Partition a file into "blocks"
- Partition an address-space into "pages"
- Partition a disk into "sectors"
- After partitioning we need to address the chunks
- Addressing means: "refer to something using a name/number"
- We already know what addresses are:
 - Each byte in RAM is addressed by a number (called "the address")
 - An address is stored in binary form in the computer (like all numbers)
 - We can then use these addresses, for instance in instructions ("store value 00110011 at address 1101001")

How Many Address Bits?

- Key question: what is the range of addresses that we need to address all chunks (uniquely)?
- We also want the smallest range not to waste address bits by having large addresses that are not used
 - □ For saving on storage (we store addresses as data to do indirection)

Example:

- We have 7 houses
- We want to "address" them via binary addresses
- We should use 3 address bits 000, 001, 010, 011, 100, 101, 110
- □ With 3 bits we can address $2^3 = 8$ houses, so we're "wasting" one slot
- With 2 bits we can address only 2² = 4 houses (00, 01, 10, 11), so that's not enough
- We don't want to use 4-bit addresses because when we need to store house addresses as data, then we're wasting 1 bit of storage per house
 - i.e., all addresses would have the same leftmost bit

How Many Address Bits?

- If you have 2ⁿ thingies, then you need n-bit addresses to address the thingies
 - fewer, and you can't address them all
 - more, and you're wasting address bits
- More generally, if you have n thingies, then you need [log₂ n]-bit addresses
 - Example with 7 houses: log₂ 7 ~ 2.8074, therefore we should use [log₂7] = 3 bits
- In this course we'll almost always have a number of thingies that's a power of 2
 - After all we "build" the system and choose what we use
 - And as you can see above in red, powers of 2 are convenient when using binary addresses

Some More Discrete Math

- Say you have a parking lot that consists of a long row of N parking spots, numbered 0 to N 1
- We structure this long row into blocks of n parking spots (assume n divides N)
- Here are two simple discrete math "results":
- The x-th spot in the parking lot (0≤x<N) is the (x mod b)-th spot in the ([x/b])-th block
- 2. The y-th spot in the z-th block is the (z×b+y)-th spot in the parking lot
- Let's see this on an example...

Parking Lot Example

- Say we have a parking lot with 3000 spots, and we structure them in blocks of 100 spots
- What is the index of spot 2212 in its block?
 2212 mod 100 = 12
- In what block is spot 2212?
 - 2212 / 100 = 22 (integer division!)
- What is the global index of spot 5 in block 20?
 - □ 20×100+5 = 2005

(because the first block is block 0)

The End

- In an upcoming lecture we'll just do a few simple inclass exercises
- You must be absolutely comfortable with all this since we'll soon be doing counting/addressing all the time
- Besides, being a computer scientist implies that you can count and address things, and that you're not fazed by powers of 2!
 - Sometimes a first interview question: what's 2 to the 8?