Serialization and Bit Operations

CS 2022: Introduction to C

Instructor: Hussam Abu-Libdeh

Cornell University
(based on slides by Renato Paes Leme)

Fall 2011, Lecture 10

Serialization

- Sending data between programs
  - Disk
  - Network
  - Pipes
- Between programs on multiple hosts
  - Different endianness
  - Different architectures

Binary vs. Text

Binary...

- Compact
- Easy to encode/decode
- Faster

e.g. IP, TCP, AIM, ...

Text...

- Easily debugged
- (Can be) self-documenting
- Arch/Endian independent

e.g. HTTP, SMTP, MSN

Ok, but how?

What serialization solution to use?

- tpl library
- c11n library
- Google protocol buffers
- Customized solution

Which standard to use?

- XML, XDR, protocol buffer, ...
- Network protocol standards

Handling Endianness

Decimal: 3735928559
Binary: 11011110101011011011111011101111
Hex: 0xdeadbeef
Big Endian: 0xde 0xad 0xbe 0xef
Little Endian: 0xef 0xbe 0xad 0xde

Always in big-endian form when loaded into the CPU

Bit-Operations

AND-Mask (clear bits)

a & b

11011110101011011011111011101111	0xdeadbeef
&	&
00000000000000001111111100000000	0x0000FF00
=	=
00000000000000001011111000000000	0x0000be00

Bit-Operations

OR-Mask (sets bits)

a | b

11011110101011011011111011101111	**0xdeadbeef**
|	|
00000000000000000101010100000000	**0x00005500**
=	=
11011110101011011111111111101111	**0xdeadFFef**

Bit-Operations

Left-Shift

a << b

11011110101011011011111011101111	**0xdeadbeef**
<<	<<
8	**8**
=	=
10101101101111101110111100000000	**0xadbeef00**

Bit-Operations

Right-Shift

a >> b

11011110101011011011111011101111	**0xdeadbeef**
>>	>>
8	**8**
=	=
00000000110111101010110110111110	**0x00deadbe**[1]

[1] for unsigned ints only. For signed ints, the instead of zero-padding, the top-most bit is repeated

Bit-Operations

Compliment (flips bits)

$\sim$a

$\sim$**11011110101011011011111011101111**	$\sim$**0xdeadbeef**
=	=
00100001010100100100000100010000	**0x21524110**

2's compliment representation for negative numbers:

`-x = ~x + 1`

Exercise 1

```
int htonl(int x) {
    int b1, b2, b3, b4, y;

    b1 = (x _____) ___;
    b2 = (x _____) ___;
    b3 = (x _____) ___;
    b4 = (x _____) ___;

    y = (b1 _____) __ (b2 ____)
        __ (b3 ____) __ (b4 ____);

    return y;
}
```

Serialization

- Use structures for data-types
- Copy data in one-go
  `memcpy(dst, src, numbytes)`
- Use standard (big) endianness for multi-byte variables
- NEVER serialize pointer values. Why?

Tricks with bits

- How to iterate over all sets $S \subseteq \{0, 1, 2, \ldots, k-1\}$?
- There are 2^k such sets. I just need one **for** to do that.
- Think of a number $0 \leq 0 < 2^k$ in binary. It represents a subset of S.
- Given a subset S, let $a_i = \begin{cases} 1, i \in S \\ 0, i \notin S \end{cases}$, then we represent S by $\sum_{i=0}^{k-1} a_i 2^i$.

Tricks with bits

How to iterate over all sets $S \subseteq \{1, \ldots, n\}$?

```
int S;
for (S = 0; S < (1<<n); ++S) {
    // process subset S
}
```

Set Operations

Given two sets A and B represented as binary strings:

- Union:

```
A | B
```

- Intersection:

```
A & B
```

- Single element set $\{1\}$:

```
1 << i
```

- Testing $i \in A$:

```
A & (1 << i) != 0
```

Set Operations

Given two sets A and B represented as binary strings:

- Adding element i to A:

```
A = A | (1 << i)
```

- Removing element i from A:

```
A = A & ~(1 << i)
```

- Toggle element i in A:

```
A = A ^ (1 << i)
```

Set Operations

Given S a string representation of a set, how to iterate over all its subsets $T \subseteq S$:

```
int T;
for (T = S; T>=0; T = (T-1)&S) {
    // process subset T
}
```

Set Operations

More complicated exercise: how to iterate over all subsets of $\{1, \ldots, n\}$ of size k.

```
int s = (1 << k) - 1;
while (!(s & 1 << N))
{
    // do stuff with s
    int lo = s & ~(s - 1);          // lowest one bit
    int lz = (s + lo) & ~s;         // lowest zero bit above lo
    s |= lz;                        // add lz to the set
    s &= ~(lz - 1);                 // reset bits below lz
    s |= (lz / lo / 2) - 1;         // put back right number of bits at end
}
```

Source of bit tricks

A bit of fun, fun with bits:
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=bitManipulation

Other iteration exercises

1. Write a code that iterates over $\{0, \ldots, n-1\}^k$, i.e., all k-uples $(t_1, \ldots, t_k)$ where $0 \leq t_i < n$.
2. Write a code that iterates over $\{0, \ldots, n-1\}^k$, i.e., all k-uples $(t_1, \ldots, t_k)$ where $0 \leq t_i < n$ and $t_1 \geq t_2 \geq \ldots \geq t_k$.
3. Write a code that iterates over all the permutations of $\{1, \ldots, n\}$ and writes them on the screen.