Serialization and Bit Operations
CS 2022: Introduction to C

Instructor: Hussam Abu-Libdeh
Presented by: Renato Paes Leme

Cornell University
(based on slides by Saikat Guha)

Fall 2009, 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
AND-Mask (clear bits)

\[ a \& b \]
\[
\begin{array}{c}
1101111010101101 \\
\& \\
0000000000000000
\end{array}
\begin{array}{c}
11101110 \\
\& \\
11111111
\end{array}
\begin{array}{c}
0xdeadbeef \\
\& \\
0x0000FF00
\end{array}
\begin{array}{c}
= \\
= \\
=
\end{array}
\begin{array}{c}
0000000000000000 \\
1011111000000000
\end{array}
\begin{array}{c}
0x0000be00
\end{array}\]
OR-Mask (sets bits)

\[
a \mid b
\]

\[
\begin{align*}
1101111010101101 & \mid 1011111011101111 & 0\text{deadbeef} \\
0000000000000000 & \mid 0101010100000000 & 0\text{00005500} \\
= & & = \\
1101111010101101 & \mid 1111111111101111 & 0\text{deadFFef}
\end{align*}
\]
Bit-Operations

Left-Shift

\[ a \ll b \]
\[ \begin{array}{l}
0x\text{deadbeef} \\
\ll \\
8 \\
= \\
0x\text{adbeef00}
\end{array} \]

\[ \begin{array}{l}
11011110101011011011111011011111 \\
\ll \\
8 \\
= \\
101011011111011101111000000000
\end{array} \]
Bit-Operations

Right-Shift

\[ a \gg b \]

\[
\begin{array}{c}
1101111010110111110110111110 \\
\gg \\
8 \\
= \\
00000000110111010110110111110
\end{array}
\quad \quad
\begin{array}{c}
0x\text{deadbeef} \\
\gg \\
8 \\
= \\
0x00\text{deadbe}
\end{array}
\]

\footnote{for unsigned ints only. For signed ints, the instead of zero-padding, the top-most bit is repeated}
Bit-Operations

Compliment (flips bits)

\[ \sim a \sim 110111101010110111101111011 \sim 0x{\text{deadbeef}} \]

\[ \sim 00100001010100100001000100010000 = 0x21524110 \]

2’s compliment representation for negative numbers:

\[ -x = \sim x + 1 \]
Exercise 1

```c
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;
}
```
Exercise 2

```c
int htonl(int x) {
    int y;
    char *xs = &x, *ys = &y;

    __ = __;
    __ = __;
    __ = __;
    __ = __;
    __ = __;

    return y;
}
```
Use `uint8_t`, `uint16_t`, `uint32_t`
#pragma pack(push)
struct ip {
    #ifdef LITTLE_ENDIAN
        uint8_t ihl:4, ver:4;
    #else
        uint8_t ver:4, ihl:4;
    #endif
    uint8_t tos;
    uint16_t len;
    uint16_t iid;
    uint16_t off;
    uint8_t ttl;
    uint8_t prt;
    uint16_t csm;
    uint32_t src;
    uint32_t dst;
    char opt[40];
};
#pragma pack(pop)
void foo(void) {
    struct ip *ip1, *ip2;

    ip1 = (struct ip *)malloc(sizeof(struct ip));
    ip2 = (struct ip *)malloc(sizeof(struct ip));

    ip1->ver = 4;
    ip1->protocol = 6;
    ip1->len = htonl(40);

    memcpy(ip2, ip1, sizeof(struct ip));

    printf("%d %d", ip2->ver, ntohl(ip2->len));

    ...
}

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 $\sum_{i=0}^{k-1} a_i 2^i$. 

Serialization and Bit Operations

CS 2022, Fall 2009, Lecture 10
Tricks with bits

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

```c
int S;
for (S = 0; S < (1<<k); ++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\}:**
  \[ 1 \ll i \]

- **Testing $i \in A$:**
  \[ A \ & \ (1 \ll i) \neq 0 \]
Set Operations

Given two sets $A$ and $B$ represented as binary strings:

- Adding element $i$ to $A$:
  $$A = A \ | \ (1 \ll i)$$

- Removing element $i$ from $A$:
  $$A = A \ & \ \sim(1 \ll i)$$

- Toggle element $i$ in $A$:
  $$A = A \ \sim (1 \ll i)$$
Given $S$ a string representation of a set, how to iterate over all its subsets $T \subseteq S$:

```c
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 \( k \).

```c
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 \( t_1 \leq t_2 \leq \ldots \leq t_k \).

3. Write a code that iterates over all the permutations of \( \{1, \ldots, n\} \) and writes them on the screen.