Serialization and Bit Operations

CS 113: Introduction to C

Instructor: Saikat Guha

Cornell University

Fall 2006, Lecture 9

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

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

11011110101011011011110111101111	0xdeadbeef
&	&
00000000000000001111111100000000	0x0000FF00
=	=
00000000000000001011111000000000	0x0000be00

Bit-Operations

OR-Mask (sets bits)

a | b

11011110101011011011110111101111	0xdeadbeef
|	|
00000000000000000101010100000000	0x00005500
=	=
11011110101011011111111111101111	0xdeadFFef

Bit-Operations

Left-Shift

a << b

11011110101011011011111011101111	0xdeadbeef
<<	<<
8	8
=	=
10101101101111011101111100000000	0xadbeef00

Bit-Operations

Right-Shift

a << b

11011110101011011011111011101111	0xdeadbeef
<<	<<
8	8
=	=
00000000110111101010110110111110	0x00adbeef[1]

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

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;
}
```

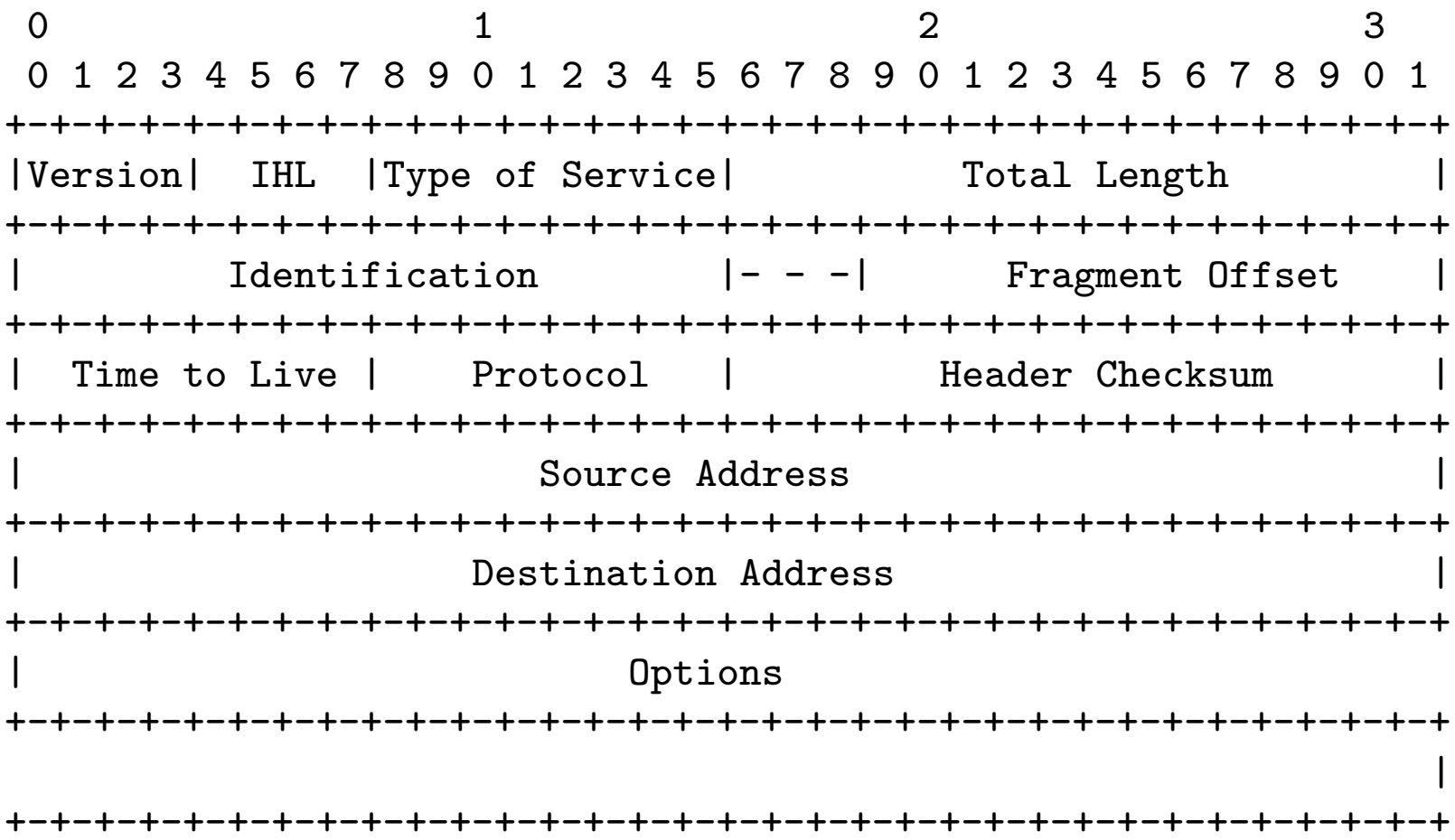
Exercise 2

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

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

    return y;
}
```

Exercise 3



Use `uint8_t`, `uint16_t`, `uint32_t`

Exercise 3

```
#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)
```

Serialization

```
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?

Bit-Operations

Compliment (flips bits)

$\sim$a

$\sim$**11011110101011011011111011101111** = **00100001010100100100000100010000**

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

2's compliment representation for negative numbers:

`-x = ~x + 1`