Internet Sockets: Challenges

Lecture 08.02

nc --recv-only 127.0.0.1 30000

advise_server.c

New challenge: Inter-operability

- How to ensure proper network communication between heterogeneous machines and operating systems?

1. Number representation
2. End of message – new line
3. TCP stream boundaries

Challenge 1: Endianness

Intra-Lilliputian quarrel over the practice of breaking eggs

- Traditionally, Lilliputians broke boiled eggs on the larger end
- A few generations ago, an Emperor of Lilliput, the Present Emperor's great-grandfather, had decreed that all eggs be broken on the smaller end after his son cut himself breaking the egg on the larger end
- The differences between Big-Endians (those who broke their eggs at the larger end) and Little-Endians had given rise to "six rebellions... wherein one Emperor lost his life, and another his crown“
- The Lilliputian religion says an egg should be broken on the *convenient* end, which is now interpreted by the Lilliputians as the smaller end

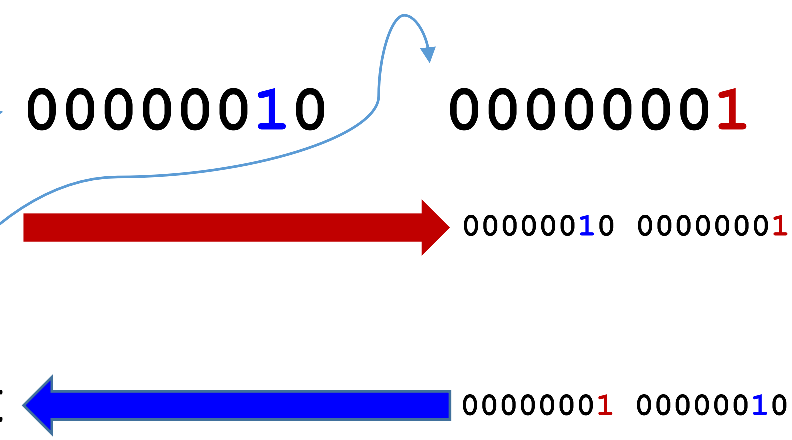
Gulliver's Travels by Jonathan Swift

Numbers can be big-endian or little-endian

- Each byte consists of 8 bits
- Bytes are the same for all architectures:

00000010 is number 2, 00000001 is number 1

- For multi-byte numbers:



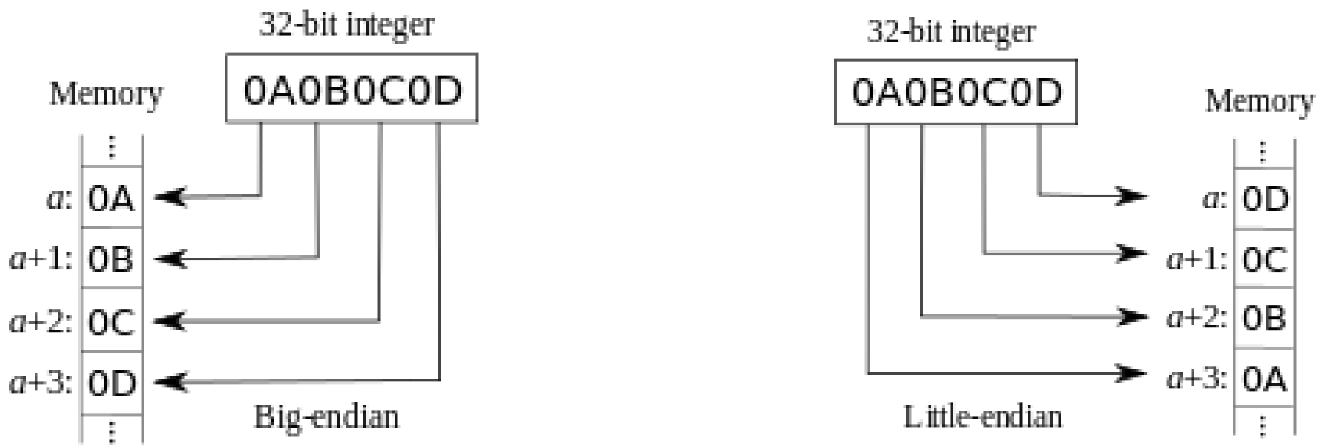
  - Big endian: left-to-right

    $2^9 + 2^0 = 513$

  - Little-endian: right-to-left

    $2^8 + 2^1 = 258$

Byte order for multi-byte numbers



- Intel is little-endian, and Sparc is big-endian
- The standard network byte order is **big-endian**; a "little-endian" machine must swap bytes in integers when copying them to and from network transmission buffers

Finding endianness of your machine

Sample code in *my_endian.c*

Converting to network byte order

- To communicate between machines with unknown or different "endian-ness" we need to convert numbers to network byte order (big-endian) before we send them.
- There are functions provided to do this:

```
unsigned long htonl(unsigned long)
unsigned short htons(unsigned short)
unsigned long ntohl(unsigned long)
unsigned short ntohs(unsigned short)
```

Differences in data representation

- Different computer architectures use different conventions to represent data formats (byte order, size of integer and long, padding structures)
- To exchange data between heterogeneous systems over network – need to put data into agreed-upon format (marshalling protocols)
- A simpler approach: send data as text, as a sequence of bytes

Streaming bytes

- TCP sockets (streaming sockets) transmit data in packets
- If sender stops before typing the next character its previous bytes are already sent
- The message arrives in chunks
- How to signal the end of message in a streaming scenario?

With a new line!

Challenge 2. new line

- Different operating systems have different newline "conventions":
  - The ASCII standard: use single byte number 10 ("control-J", or "line feed" or "LF“)
  - Unix: byte 10 as a "newline character", and we get it in C in Unix by typing "**\n**“
  - MS-DOS and successors: a two-byte sequence to separate lines: byte 13 and byte 10 ("control-M" and "control-J“) "Control-M" is also known as "carriage return" or "CR". Together, this two byte sequence is called "CRLF " (" **\r\n**")
  - Some other operating systems have other newline conventions

Newline problem for sending data over the network

- In the case of transmitting text, the ASCII standard gives us standard byte values for just about everything except newlines
- So we need to adopt a newline standard for network text transmission

Network new line convention

- The network newline convention is CRLF. That is, a newline is represented by the two bytes (in order) which we could call CR and LF, or control-M and control-J, or 13 and 10, or \015 and \012.

Network new line:

\r\n rather than just \n

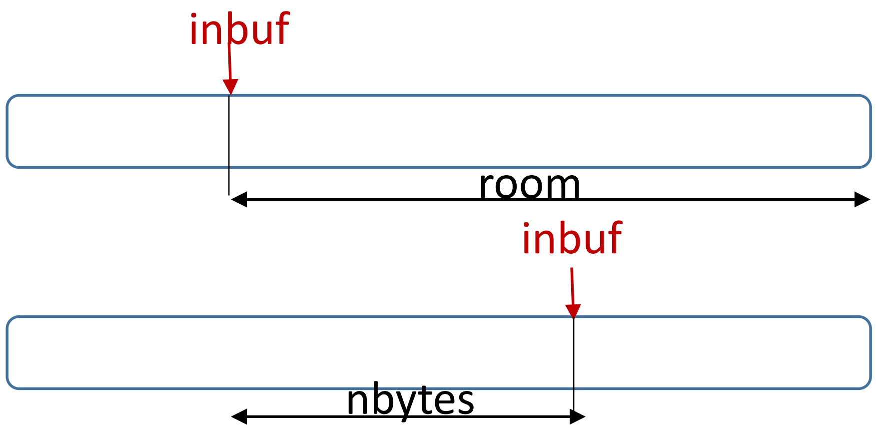
Challenge 3. Partial reads

- In TCP protocol, a single message arrives as a sequence of packets
- If we want to reconstruct the original message lines, we need to parse one line of a message, and keep the beginning of the next line in buffer
- For this, we keep one pointer for each buffer, to keep track of data length

```
char buf [BUFFER_SIZE];

int inbuf;
```

Parsing partial reads into lines of text: 1/3

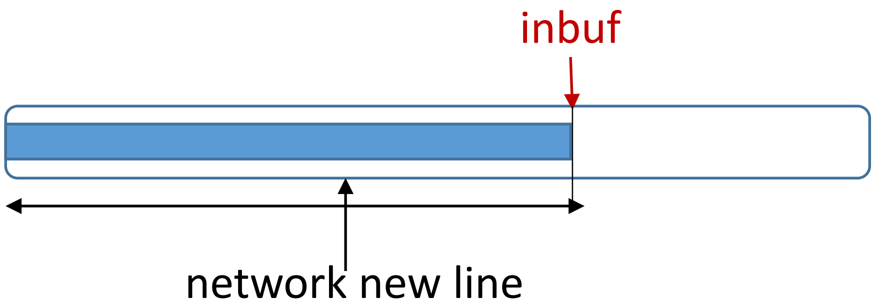



```
char *after = buf + inbuf;
int room = BUFFER_SIZE - inbuf;
int nbytes;
```

Read next piece of data (of size room) from fd into a computed place in buffer

```
if ((nbytes = read(fd, after, room)) > 0) {
    inbuf += nbytes;  //advance inbuf pointer
```

Parsing partial reads into lines of text: 2/3



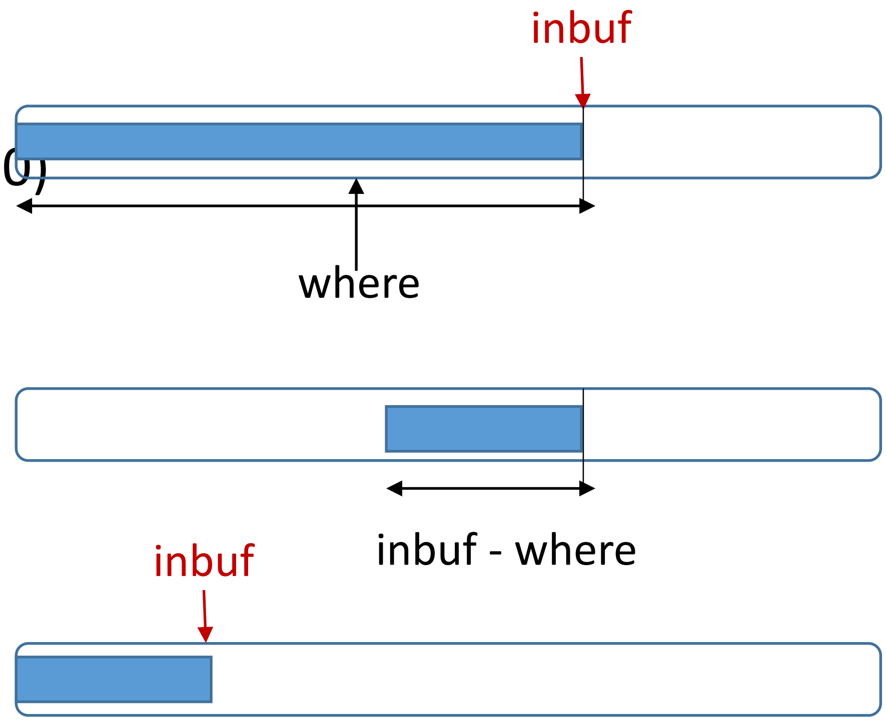

```
if ((nbytes = read(fd, after, room)) > 0)
{
    ...
    int where = find_network_newline (buf, inbuf);

    if (where >= 0) {
        buf[where] = '\0'; buf[where+1] = '\0';
        do_command(buf);
    }
}
```

Process data in buffer to find a new line

If data contains new line – make a C string and process it

Parsing partial reads into lines of text: 3/3




```
if ((nbytes = read(fd, after, room)) > 0)
{
    …



    if (where >= 0) {
        …
        where+=2;  // skip over \r\n
        inbuf -= where;
        memmove (buf, buf + where, inbuf);
    }
}
```

Move remaining data to the beginning of the buffer for next read