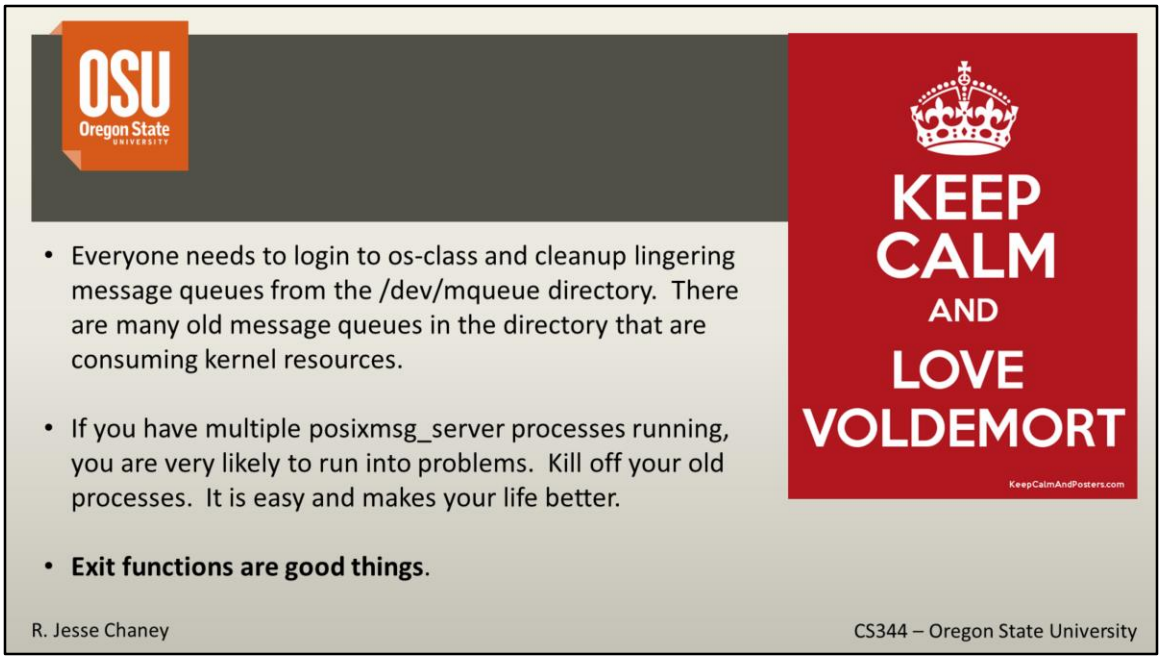
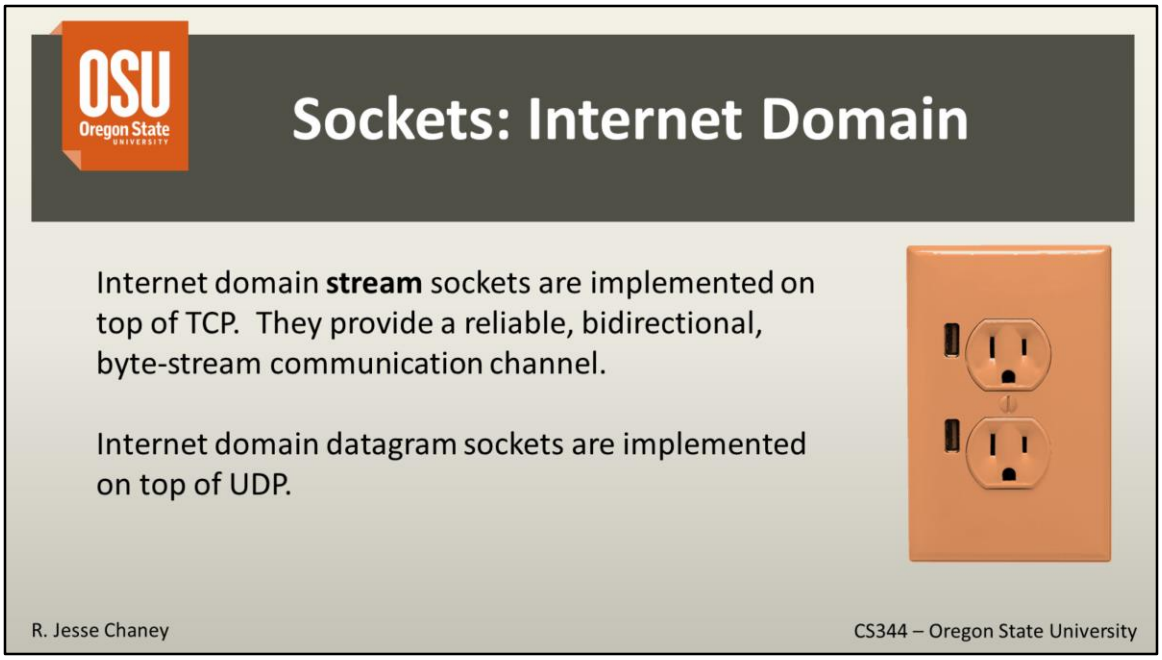
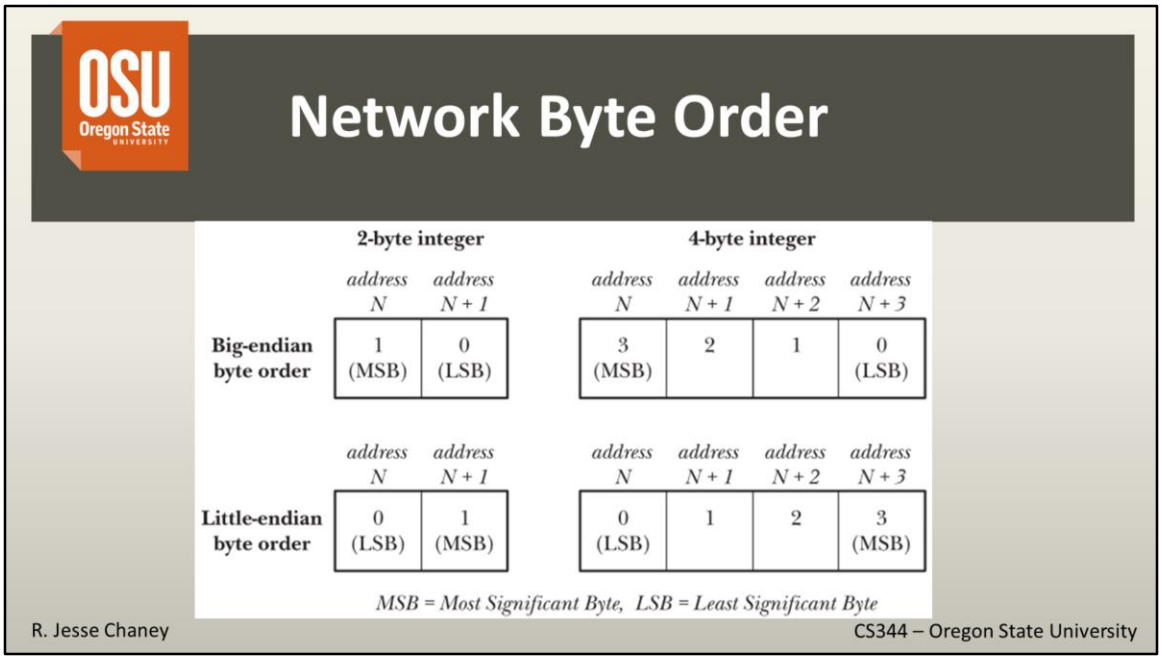


- Everyone needs to login to os-class and cleanup lingering message queues from the /dev/mqueue directory. There are many old message queues in the directory that are consuming kernel resources.
- If you have multiple posixmsg_server processes running, you are very likely to run into problems. Kill off your old processes. It is easy and makes your life better.
- **Exit functions are good things**.



TCP: Transmission Control Protocol. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network.

UDP: User Datagram Protocol. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.



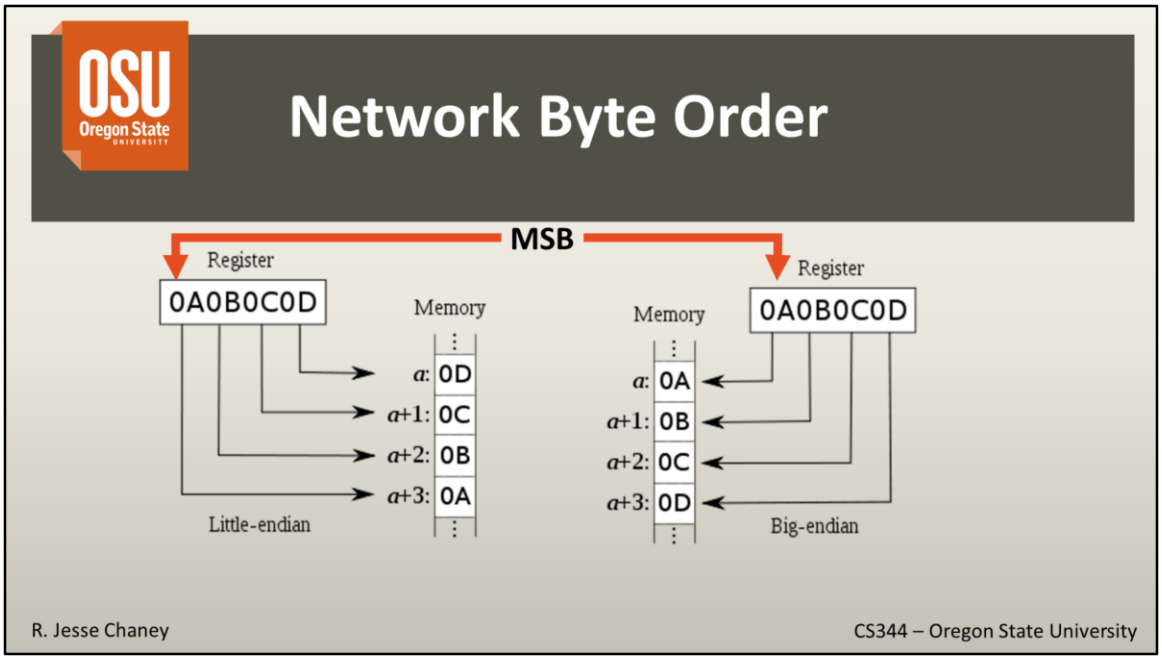
One problem we encounter when passing these data across a network is that different hardware architectures store the bytes of a multi-byte integer in different orders.

I'm sure you've talked about big-endian and little-endian in other classes.

The Internet Protocol defines big-endian as the standard *network byte order* used for all numeric values in the packet headers and by many higher level protocols and file formats that are designed for use over IP. The Berkeley sockets API defines a set of functions to convert 16-bit and 32-bit integers to and from network byte order: the htons (host-to-network-short) and htonl (host-to-network-long) functions convert 16-bit and 32-bit values respectively from machine (*host*) to network order; the ntohs and ntohl functions convert from network to host order. These functions may be a no-op on a big-endian system.

In 1726, Jonathan Swift described in his satirical novel *Gulliver's Travels* tensions in Lilliput and Blefuscu: whereas royal edict in Lilliput requires cracking open one's soft-boiled egg at the small end, inhabitants of the rival kingdom of Blefuscu crack theirs at the big end (giving them the moniker *Big-endians*).[6][7] The terms *little-endian* and

endianness have a similar intent.



Big-endian is the most common convention in data networking (including IPv6), hence its pseudo-synonym ***network byte order***, and little-endian is popular (though not universal) among microprocessors in part due to **Intel's** significant historical influence on microprocessor designs.

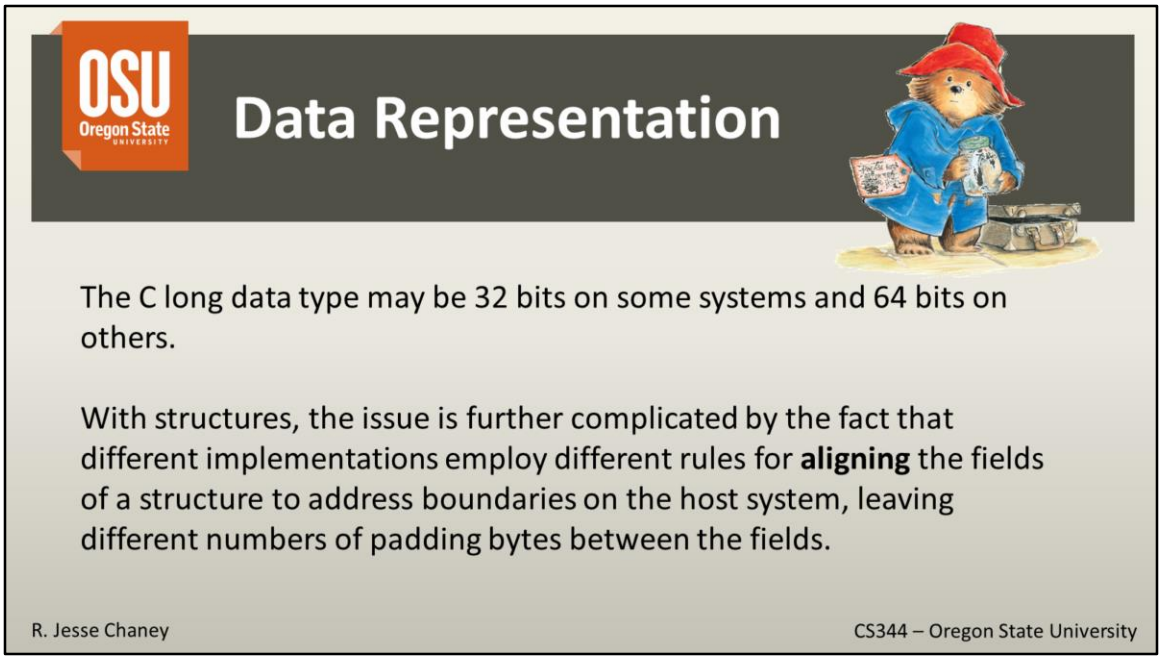
Failure to account for varying endianness across architectures when writing software code for mixed platforms and when exchanging certain types of data might lead to failures and bugs, though these issues have been understood and properly handled for many decades.

Ordering Your Bytes

```
#include <arpa/inet.h>

uint16_t htons(uint16_t host_uint16 );
```
Returns host_uint16 converted to network byte order
```
uint32_t htonl(uint32_t host_uint32 );
```
Returns host_uint32 converted to network byte order
```
uint16_t ntohs(uint16_t net_uint16 );
```
Returns net_uint16 converted to host byte order
```
uint32_t ntohl(uint32_t net_uint32 );
```
Returns net_uint32 converted to host byte order

Notice that a short is 16 bits and a long is 32 bits.

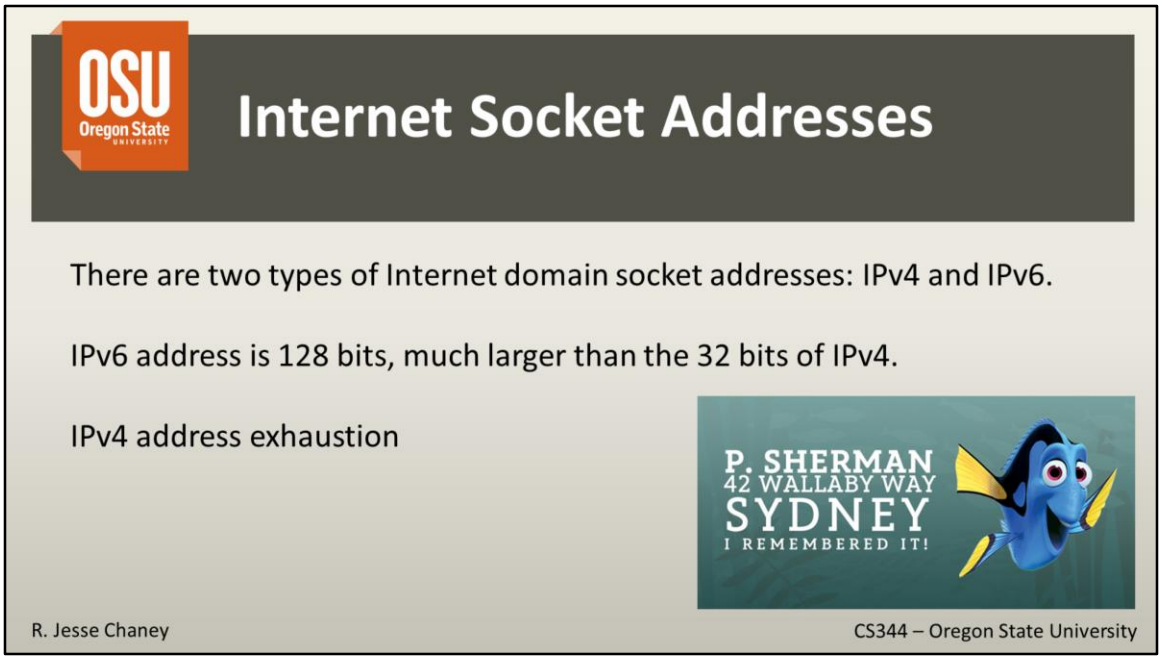


Just when you thought it would be easy to switch back and forth between host-order and network order, a whole new set of complications come up.

Because of these differences in data representation, applications that exchange data between heterogeneous systems over a network must adopt some common convention for encoding that data. The sender must encode data according to this convention, while the receiver decodes following the same convention. The process of putting data into a standard format for transmission across a network is referred to as marshalling.

A simpler approach than marshalling is often employed: encode all transmitted data in text form, with separate data items delimited by a designated character, typically a newline character. One advantage of this approach is that we can use telnet to debug an application.

In the book, the author goes into some detail about how to write a newline() function that will return a NULL terminated string from a socket.



If IPv4 and IPv6 coexist on a host, they share the same port-number space. This means that if, for example, an application binds an IPv6 socket to TCP port 2000 (using the IPv6 wildcard address), then an IPv4 TCP socket can't be bound to the same port.

In this class, we'll mostly deal with the IPv4 addresses. However, the world is moving to the IPv6 standard, so the more you know about that, the better off you are.

IPv4 address exhaustion is the depletion of the pool of unallocated Internet Protocol Version 4 (IPv4) addresses, which has been anticipated since the late 1980s. This depletion is the reason for the development and deployment of its successor protocol, IPv6.

IPv4 provides approximately 4.3 billion addresses.

On 31 January 2011, the last two unreserved IANA /8 address blocks were allocated to APNIC according to RIR request procedures. This left five reserved but unallocated /8 blocks.

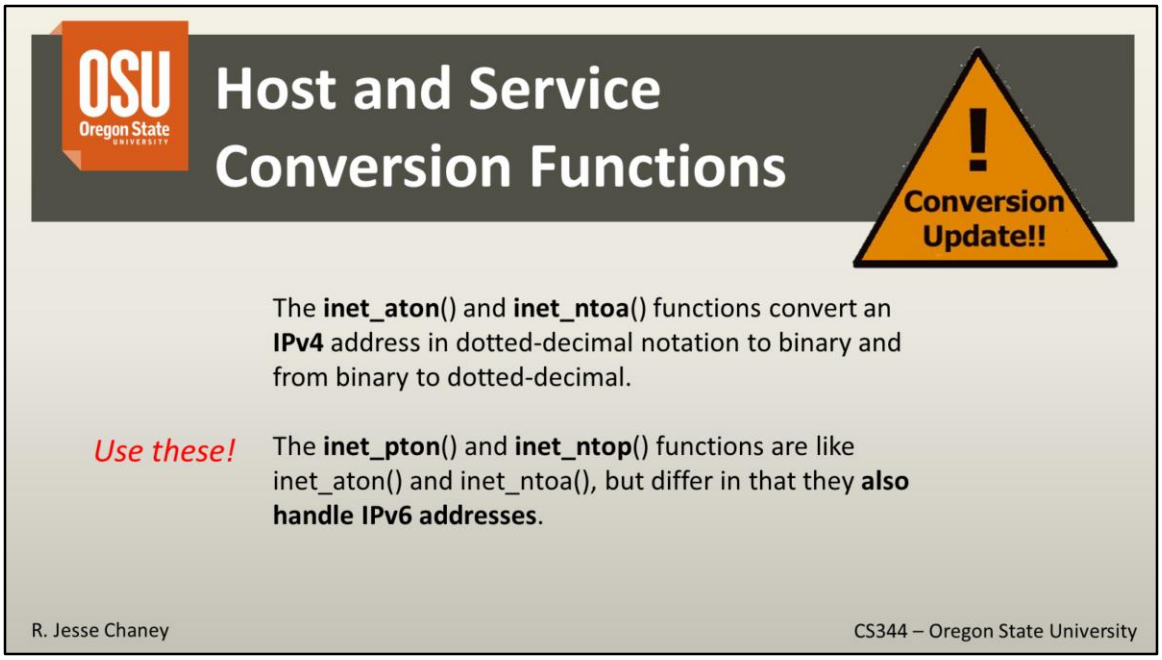
IPv6 uses a 128-bit address, allowing 2^{128}, or approximately 3.4×10^{38} addresses, or more than 7.9×10^{28} times as many as IPv4. IPv6 addresses are represented as eight groups of four hexadecimal digits separated by colons, for example

2001:0db8:85a3:0042:1000:8a2e:0370:7334

, but methods to abbreviate this full notation exist.

How Big is IPv6?

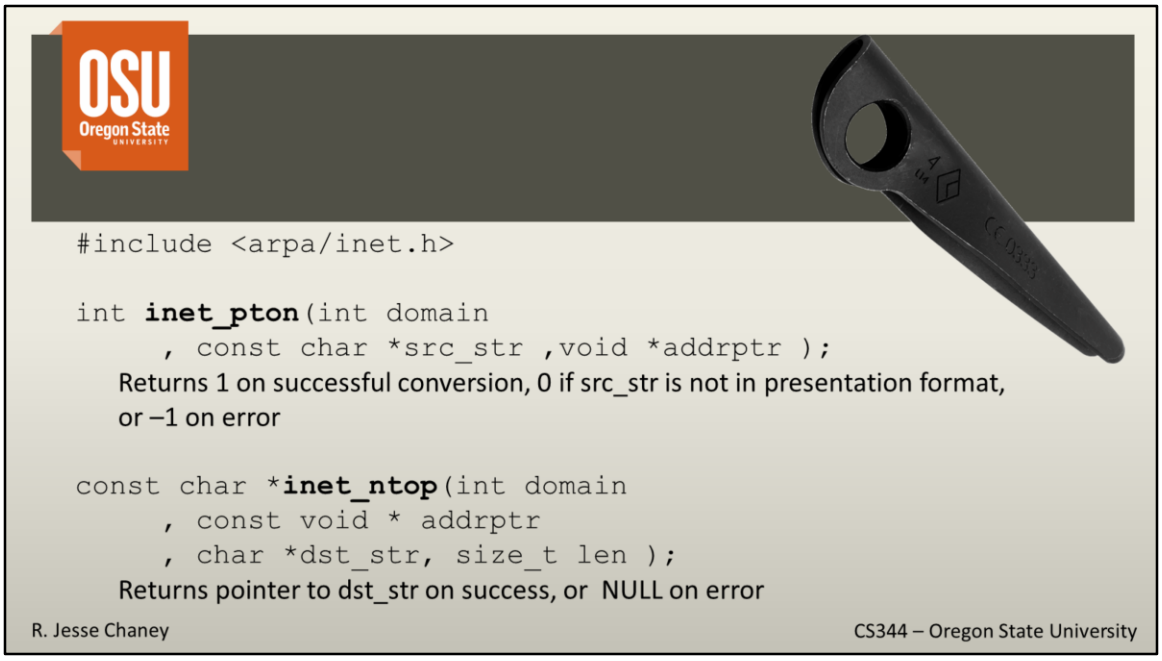
4.295×10^9	**IPv4** address space size (32-bit)
10^{22}	Estimated number of stars in the universe.
4.339×10^{26}	Nanoseconds that have passed since the Big Bang.
3.156×10^{31}	Unique IPv6 addresses that would be assigned after 1 trillion years if a new IPv6 address was assigned at every picosecond.
3.403×10^{38}	**IPv6** address space size (128-bit).
10^{81}	Estimated total number of atoms in the universe



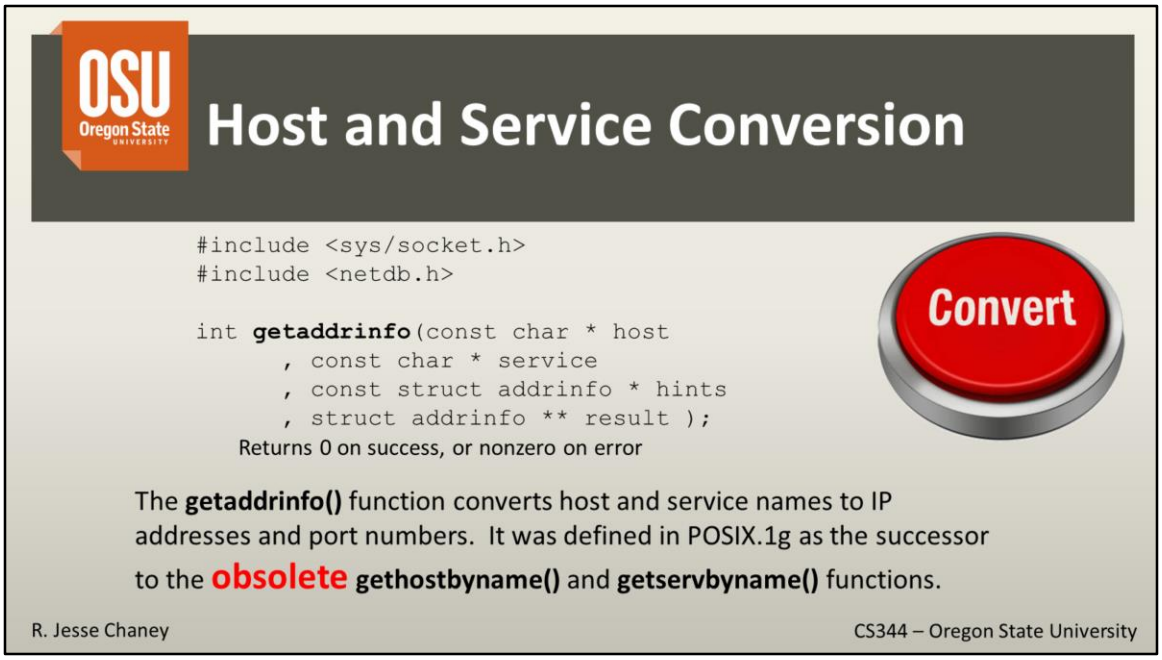
Computers represent IP addresses and port numbers in binary. However, humans find names easier to remember than numbers. Employing symbolic names also provides a useful level of indirection; users and programs can continue to use the same name even if the underlying numeric value changes.

A hostname is the symbolic identifier for a system that is connected to a network (possibly with multiple IP addresses). A service name is the symbolic representation of a port number.

inet_pton() and inet_ntop() convert **binary IPv4 and IPv6** addresses to and from presentation format—that is, either dotted-decimal or hexstring notation.



The p in the names of these functions stands for “presentation,” and the n stands for “network.”

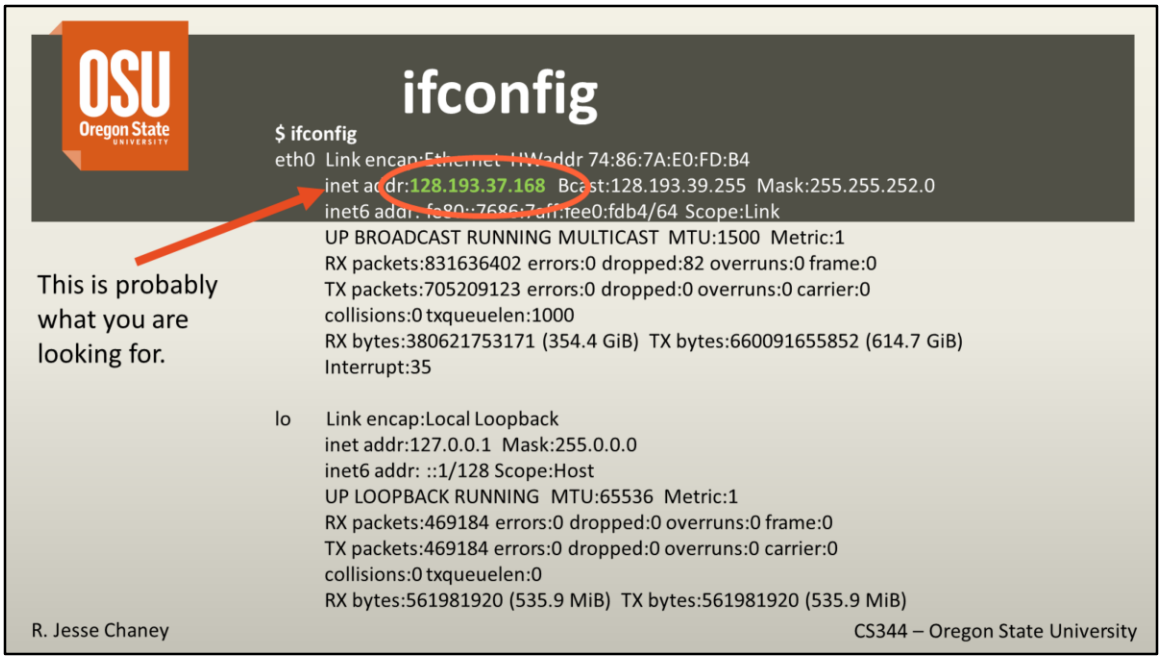


The getnameinfo() function is the converse of getaddrinfo(). It translates a socket address structure (either IPv4 or IPv6) to strings containing the corresponding host and service name.

The getaddrinfo() is a bit more awkward to use than the older functions, but provides greater flexibility.

As output, getaddrinfo() dynamically allocates a linked list of addrinfo structures and sets result pointing to the beginning of this list.

The getaddrinfo() function dynamically allocates memory for all of the structures referred to by result (Figure 59-3). Consequently, the caller must deallocate these structures when they are no longer needed. The freeaddrinfo() function is provided to conveniently perform this deallocation in a single step.



You can also try `ip addr`, but I don't find it easier to use.