Computer Networking

CS2 Recitation Series
Friday, February 21, 2014

Last time...

- Networking: basic concepts
- The CS2Net socket wrapper
  - Connecting to other servers
  - Sending and receiving data
  - Polling sockets

Last time...

CS2Net::Socket sock;

```
std::string hostname("host1.example.net");
uint16_t port = 9001;
int con_ret = sock.Connect(&hostname, port);

// Send some data.
std::string data("Hello, world!\n");
int send_ret = sock.Send(&data);

// Receive some data.
std::string * got_data;
got_data = sock.Recv(1024, false);
```

Last time...

```
std::vector<CS2Net::PollFD> to_poll(1);
to_poll[0].sock = &sock;
to_poll[0].SetRead(true);
// poll with 10ms timeout
int poll_err = CS2Net::Poll(&to_poll, 10);
// ... check for various errors ...
if(to_poll[0].CanRead())
{
    // ... do stuff ...
}
```

Today

- Some assignment-related details
- Some light GUI work
- The CS2Net socket wrapper
  - Listener sockets
  - Accepting incoming connections
  - Polling on many sockets at a time

How to GUI?

First, a demonstration of the finished product...

How to GUI?

- We use the GTK+ 2.x toolkit
  - lets you create GUI applications
  - well-supported by most Linux distributions
- Basic ideas:
  - bunch of UI elements
  - whenever user interacts, a signal is emitted
    - clicking buttons, typing, etc.
  - GTK+ catches the signal and calls a callback function
  - our callbacks do useful things

Callbacks in client2

client2 has callbacks for:

- pressing [ENTER] at the chat input
- clicking the Connect/Close buttons
- submitting the Connect form
- idle time
  - run whenever nothing better to do

Some are provided; others you'll have to fill in.

Callbacks aren't everything

There are some support functions you should fill in:

- send message given type and payload
- connect to a server

Some other support functions are given:

- update the user list on the right
- add a line of text to the chat buffer

String tokenization in C++

- std::string::find_first_of(char A, size_t start)
  - returns the index of the first occurrence of A
- std::string::substr(size_t start, size_t len)
  - returns a len-length substring starting from start
- Basic idea:
  - take the substring from wherever we left off up to the next delimiter
  - do something with the substring
- You'll need to tokenize strings to populate the user list
  - newline-delimited list of users in a MSG_USERLIST message

Listener sockets

- Sockets put into "listening mode"
- Can't themselves send or receive data
- Can accept connections
  - each connection accepted spawns a new "normal" socket
  - send and recv on the newly created socket to talk to the host on the other side

Listener sockets in CS2Net

We can bind a socket to a given port and tell it to start listening with one call:

```
uint16_t port = 9001;
int ret = sock.Bind(port, 3);
```

The 3 means up to 3 incoming connections can be put on backlog at a time.

Listener sockets in CS2Net

Assuming this worked, our socket is now bound and listening on port 9001.

We can try to accept a connection:

```
CS2Net::Socket * incoming;
incoming = sock.Accept();
```

Listener sockets in CS2Net

- The Accept() call blocks until a connection can be accepted.
- What if we can't wait all day?
- Poll()!
  - check the listener socket for data to read
  - (no data is really available, but this is set if there's a pending connection we can grab)

Listener sockets in CS2Net

- How to store the sockets that get spawned?
- You tell me.
  - could make a list of sockets
  - could create a wrapper data structure around each socket, then make a list of those
  - could do something completely different
- The only requirement is that you can keep track of all the sockets.
  - needs to be an arbitrarily extensible data structure

Polling on multiple sockets

- Your repeater server will need to potentially poll on lots of sockets.
- Create a vector of CS2Net::PollFD objects
  - each PollFD object has three members:
    - sock
      - pointer to a CS2Net::Socket
    - requested_events
      - bitmask denoting events of interest for this socket
    - returned_events
      - bitmask denoting what events are available
      - changed in place by CS2Net::Poll()
- Pass the entire vector to Poll()
- Check the returned_events for return states for each socket of interest.

Polling on multiple sockets

Bitfields are sometimes inconvenient to work with. PollFD has some helper methods.

- SetRead, SetWrite
  - manipulate requested_events
- CanRead, CanWrite, HasHangup, HasError
  - query returned_events

Special topic: tcpdump

- tcpdump allows you to see packets that are coming across the network interface.
- This can be quite useful for debugging purposes.
- Will show some short demos.

Special topic: tcpdump

- Usually requires root access.
- Usually want to specify interface to listen on (especially if using ssh).
- Can find a list of interfaces on linux using ifconfig.
  - Not all may be available. Use tcpdump -D to get list of interfaces that tcpdump can use.
- Example: sudo tcpdump -i eth0

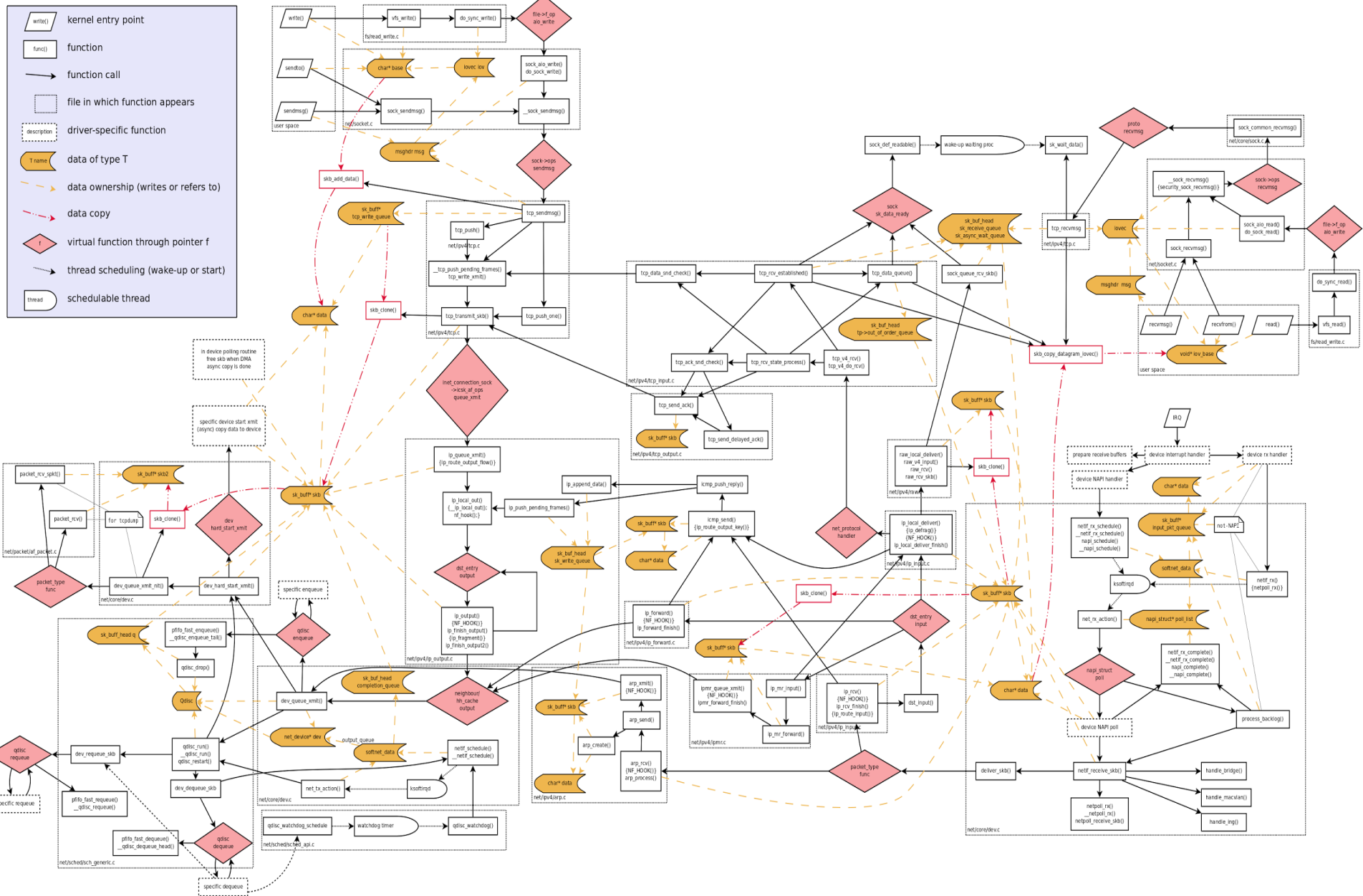
Special topic: tcpdump

- By default, will print to stdin. Can send to capture file instead.
  - Writing: tcpdump -w filename.log
  - Reading: tcpdump -r filename.log
- Can have tcpdump capture N files.
  - #tcpdump -c N
- Can show contents of packet.
  - tcpdump -X

Special topic: Userspace networking

- Everything in CS2 is done using the networking stack provided by the Operating System.
- This provides a lot of flexibility.
  - Sockets are nice and easy to use, interface is common among all application.
- This does come at a cost.
  - Speed. OS provided networking provides poor performance for some loads.
  - High Performance Computing and High Frequency Trading are both places where latency is important.

Scary Diagram You Don't Need to



Special topic: Userspace networking

- Last slide shows complexity of network stack, and number of copies that are performed.
- The use of interrupts also greatly increases latency and decreases throughput.
  - How interrupts work are not important, but you will see them in some detail in CS24.

Special topic: Userspace networking

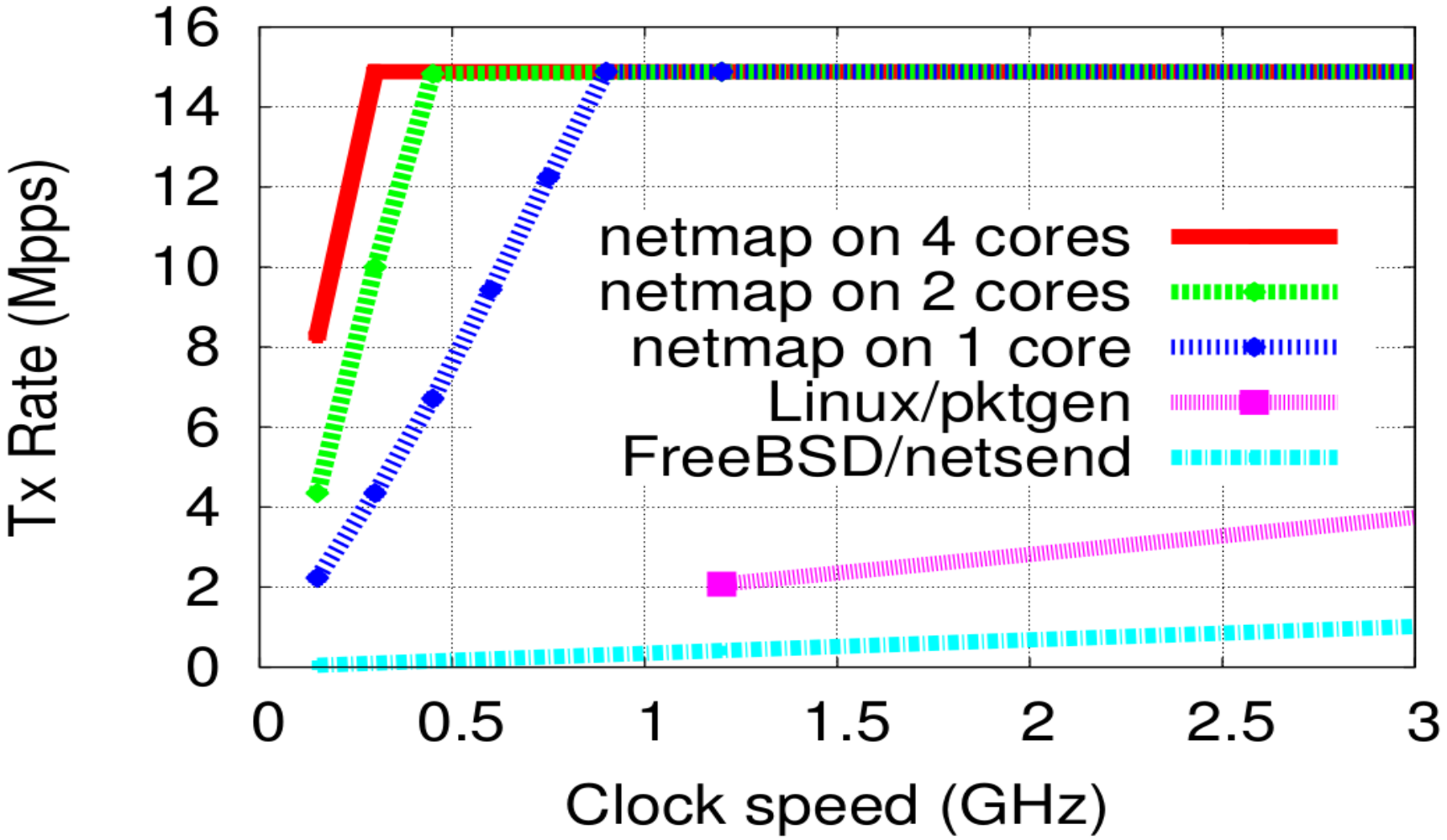
- Userspace networking allows for higher throughput and lower latency.
  - It comes at a cost of pricier hardware in some cases.
  - The interface is harder to use.
  - Can do networking with no memory copying.
- There are a few open source userspace networking stacks (including netmap).
- There are many more proprietary stacks.

Special topic: Userspace networking

- netmap vs. FreeBSD network stack
  - CPU time to send small UDP packet:
    - 8ns vs. 104 ns
  - Netmap can process using zero copies, while FreeBSD uses at least 1 data copy, usually more.

From netmap: a novel framework for fast packet I/O, Proceedings of the 2012 USENIX Annual Technical Conference, June 2012

Special topic: Userspace networking



From netmap: a novel framework for fast packet I/O, Proceedings of the 2012 USENIX Annual Technical Conference, June 2012

Questions?

- This is a challenging assignment; we expect you to have lots of questions.
- Let us know if something appears broken
  - some students have discovered server bugs already...