Project 4: Reliable Networking

Slide heritage: Previous TAs
Announcements

- Project 4 has been released
- I assume you’ve read the project description
- Due  **Tue, Apr 24th**
- This is a pretty complex project ⇒ Start early!
## Our network stack vs. the real world

<table>
<thead>
<tr>
<th>Layer</th>
<th>Our Network Stack</th>
<th>PortOS Network Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transport Layer</strong></td>
<td>UDP, TCP</td>
<td>minimsg, minisocket</td>
</tr>
<tr>
<td><strong>Network Layer</strong></td>
<td>Internet Protocol</td>
<td>network.h</td>
</tr>
<tr>
<td><strong>Link Layer</strong></td>
<td>Ethernet (or similar)</td>
<td>Host OS UDP Stack</td>
</tr>
</tbody>
</table>

TCP/IP Stack vs. PortOS Network Stack
Minisocket is a simplified TCP

- Protocol is connection oriented
  - You must find a way to establish a connection between two endpoints
- Data is sent as a continuous stream of bytes
  - Messages are an application level concept
  - Minisocket must maintain correct ordering
- No limit on message sizes
  - You must fragment and reassemble the data
State Machine

- **Listening**
  - minisocket_server_create
  - received connection
  - connection accepted
  - connection failed

- **Connect**
  - minisocket_client_create
  - connection failed

- **Sending**
  - message sent
  - minisocket_send
  - either side calls minisocket_close

- **Connected**
  - message received
  - Received

- **Receiving**
  - message received
  - Other party sends FIN

- **Closed**
  - other party doesn't reply
Of course, it’s much more complicated...
What can go wrong?

- Any party can die
- Messages can get lost
- Data might be reordered
- Network might be partitioned

Welcome to the fun world of distributed systems!
Connecting: Three-Way Handshake

Non-blocking protocol
- Any packet might be lost
- Will be resent up to seven times
- Timeout doubles every time

Initial Timeout: 100ms
* Give up after 12.7s
Messages can get lost

Timeout

Client

MSG_SYN

MSG_SYN

MSG_SYNACK

MSG_ACK

Server

Lost
Messages can get lost

**Note:** In this case both parties might retransmit
Messages can get lost
Messages can get lost multiple times.
Sending Data: SEQ and ACK Numbers

Sender

MSG_ACK with
seq_number=34 and "hodor"

MSG_ACK with
ack_number=34

Receiver

seq_number represents how many packets have been sent
✓ is used to order messages

ack_number shows total received packets
✓ is used to resend lost messages

Note: This is a symmetric channel. Both parties can send and receive.
Again, messages can get lost

```
Sender

seq_number=34 with "hodor"

Receiver

Lost

Timeout

seq_number=34 with "hodor"

ack_number=34
```
Again, messages can get lost

Sender

seq_number=34 with "hodor"

Receiver

Lost

ack_number=34

seq_number=34 with "hodor"

ack_number=34

Timeout
Either side can send and receive!

Participant 1

seq=34, ack=12 with “hodor”

seq=12, ack=34

seq=13, ack=34 with “arya”

seq=34, ack=13

Participant 2
Closing connections

Again, this is a symmetric protocol. Both sides can close the connection.
### Minisocket Header

<table>
<thead>
<tr>
<th>Bytes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>protocol</td>
<td>source_address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>source_port</td>
<td>destination_address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td>destination_port</td>
<td>type</td>
<td></td>
<td>seq_number</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>ack_number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first 21 bytes are identical to minimsg_header!

Use protocol field to multiplex protocols.
**Tricky Part: How to implement timeout?**

Remember that:

- Parties might never respond
- Multiple threads can call `minisocket_receive()` on the same port
- At most one thread can call `minisocket_send()` on a port

Things you must avoid:

- Putting threads on the run queue more than once
- Thread keeps waiting after message is received
- Thread blocks infinitely
Tricky Part: How to implement timeout?

- Setup alarm & Put thread on wait queue for port
- Waiting
  - Alarm Fires: Remove thread from port’s wait queue & wake up thread
  - ACK received
  - Deregister alarm & wake up thread
To make it a little easier

- You don’t have to implement congestion control
- Sending one packet at a time is sufficient
- minimsg_send can block until corresponding ACK is received

But you can implement window sizes > 1 if you want to!
(and have the time…)
Where to start

- Think about the state machine from earlier!
- Try to make connection setup and termination work first.
- Test with no loss and single-thread access
Test all the code!

- What happens if you send very large messages?
- Can you handle a lot of messages?
- What if there is loss?
- If one party crashes the other one shouldn’t.
- What if multiple threads are sending/receiving from the same port?
Test all the code!

In network.c:

double loss_rate = 0.0;
double duplication_rate = 0.0;
bool synthetic_network = false;

These change the behavior of the network

You have to set this to true for the other values to have any effect!
Good Luck

Questions?

As always, if you need help, come to office hours or post your questions on Piazza!