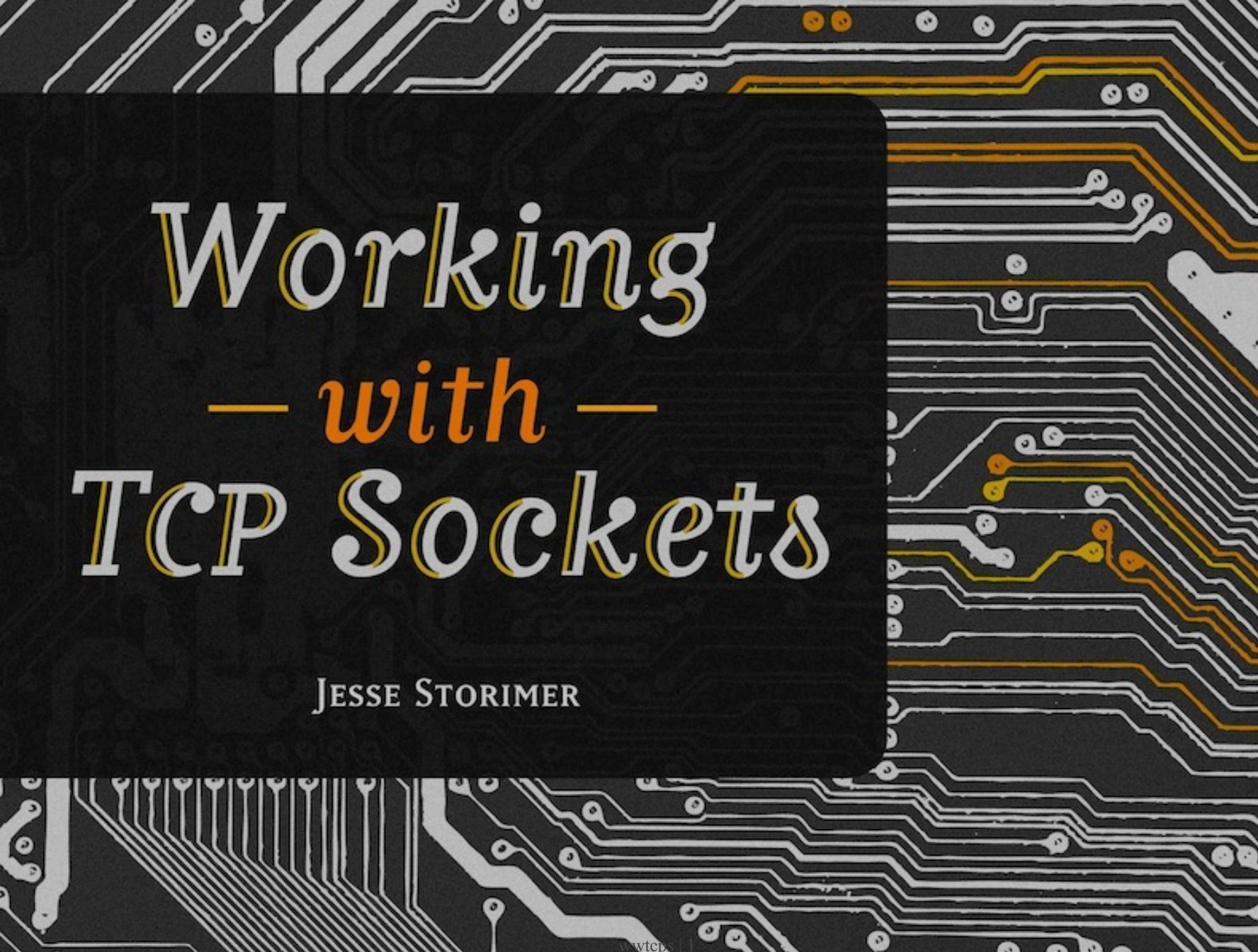
Working
— with —
TCP Sockets
Jesse Storimer

Copyright (C) 2012 Jesse Storimer.

Chapter 3

# Server Lifecycle

A server socket listens for connections rather than initiating them. The typical lifecycle looks something like this:

1. create
2. bind
3. listen
4. accept
5. close

We covered #1 already; now we'll continue on with the rest of the list.

## Servers Bind

The second step in the lifecycle of a server socket is to **bind** to a port where it will listen for connections.

```ruby
# ./code/snippets/bind.rb
require 'socket'

# First, create a new TCP socket.
socket = Socket.new(:INET, :STREAM)

# Create a C struct to hold the address for listening.
addr = Socket.pack_sockaddr_in(4481, '0.0.0.0')

# Bind to it.
socket.bind(addr)
```

This is a low-level implementation that shows how to bind a TCP socket to a local port. In fact, it's almost identical to the C code you would write to accomplish the same thing.

This particular socket is now bound to port `4481` on the local host. Other sockets will not be able to bind to this port; doing so would result in an `Errno::EADDRINUSE` exception being raised. Client sockets will be able to connect to this socket using this port number, once a few more steps have been completed.

If you run that code block you'll notice that it exits immediately. The code works but doesn't yet do enough to actually listen for a connection. Keep reading to see how to put the server in `listen` mode.

To recap, a server binds to a specific, agreed-upon port number which a client socket can then connect to.

Of course, Ruby provides syntactic sugar so that you never have to actually use `Socket.pack_sockaddr_in` or `Socket#bind` directly. But before learning the syntactic sugar it's important that we see how to do things the hard way.

## What port should I bind to?

This is an important consideration for anyone writing a server. Should you pick a random port number? How can you tell if some other program has already 'claimed' a port as their own?

In terms of what's possible, any port from 1-65,535 *can* be used, but there are important conventions to consider before picking a port.

The first rule: **don't try to use a port in the 0-1024 range**. These are considered 'well-known' ports and are reserved for system use. A few examples: HTTP traffic defaults to port 80, SMTP traffic defaults to port 25, rsync defaults to port 873. Binding to these ports typically requires root access.

The second rule: **don't use a port in the 49,000-65,535 range**. These are the ephemeral ports. They're typically used by services that don't operate on a predefined port number but need ports for temporary purposes. They're also an integral part of the connection negotiation process we'll see in the next section. Picking a port in this range might cause issues for some of your users.

Besides that, **any port from 1025-48,999 is fair game for your uses**. If you're planning on claiming one of those ports as *the* port for your server then you should have a look at the IANA list of registered ports [2] and make sure that your choice doesn't conflict with some other popular server out there.

2. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

## What address should I bind to?

I bound to `0.0.0.0` in the above example, but what's the difference when I bind to `127.0.0.1`? Or `1.2.3.4`? The answer has to do with interfaces.

Earlier I mentioned that your system has a loopback interface represented with the IP address `127.0.0.1`. It also has a physical, hardware-backed interface represented by a different IP address (let's pretend it's `192.168.0.5`). When you `bind` to a specific interface, represented by its IP address, your socket is only listening on that interface. It will ignore the others.

If you bind to `127.0.0.1` then your socket will only be listening on the loopback interface. In this case, only connections made to `localhost` or `127.0.0.1` will be routed to your server socket. Since this interface is only available locally, no external connections will be allowed.

If you bind to `192.168.0.5`, in this example, then your socket will only be listening on that interface. Any clients that can address that interface will be listened for, but any connections made on `localhost` will not be routed to that server socket.

If you want to listen on *all* interfaces then you can use `0.0.0.0`. This will bind to any available interface, loopback or otherwise. Most of the time, this is what you want.

```ruby
# ./code/snippets/loopback_binding.rb
require 'socket'

# This socket will bind to the loopback interface and will
# only be listening for clients from localhost.
local_socket = Socket.new(:INET, :STREAM)
local_addr = Socket.pack_sockaddr_in(4481, '127.0.0.1')
local_socket.bind(local_addr)

# This socket will bind to any of the known interfaces and
# will be listening for any client that can route messages
# to it.
any_socket = Socket.new(:INET, :STREAM)
any_addr = Socket.pack_sockaddr_in(4481, '0.0.0.0')
any_socket.bind(any_addr)

# This socket attempts to bind to an unkown interface
# and raises Errno::EADDRNOTAVAIL.
error_socket = Socket.new(:INET, :STREAM)
error_addr = Socket.pack_sockaddr_in(4481, '1.2.3.4')
error_socket.bind(error_addr)
```

## Servers Listen

After creating a socket, and binding to a port, the socket needs to be told to listen for incoming connections.

```ruby
# ./code/snippets/listen.rb
require 'socket'

# Create a socket and bind it to port 4481.
socket = Socket.new(:INET, :STREAM)
addr = Socket.pack_sockaddr_in(4481, '0.0.0.0')
socket.bind(addr)

# Tell it to listen for incoming connections.
socket.listen(5)
```

The only addition to the code from the last chapter is a call to `listen` on the socket.

If you run that code snippet it still exits immediately. There's one more step in the lifecycle of a server socket required before it can process connections. That's covered in the next chapter. First, more about `listen`.

## The Listen Queue

You may have noticed that we passed an integer argument to the `listen` method. This number represents the maximum number of pending connections your server socket is willing to tolerate. This list of pending connections is called **the listen queue**.

Let's say that your server is busy processing a client connection, when any new client connections arrive they'll be put into the listen queue. If a new client connection arrives and the listen queue is full then the client will raise `Errno::ECONNREFUSED`.