Distributed and Secure Systems

Stefan Schindler (@dns2utf8)

15 June 2019

Rust Zürichsee, Schweiz, CH - hosted by Cosin - Chaos Singularity Biel/Bienne, CH

Index

1. About me
2. Multi-Processing
    - Local aka. Multi-Core
    - Distributed
3. Computer Networks
    - TCP/IP
4. Example Project
    - A distributed Gallery
    - Multi-Processing
5. Questions

About me

About:me

Hello my name is Stefan and I work on and with computers.

I organize

- RustFest.eu Barcelona: probably at 9th - 12th November 2019 with "impl days" after
- Meetups in and around Zürich, CH
- ErnstEisprung.ch

Some of my side projects

- rust threadpool (maintainer)
- Son of Grid Engine (SGE) interface
- run your own infrastructure - DNS, VPN, Web, ...

Latest T-Shirt idea: Aufklären statt Aufregen => uplift instead of upset

Timetable

- concurring => IPv6 workshop
- now => Talk Stefan: Distributed and Secure Systems
- after => Discussions
- after => p===p Sync
- 20:00 => Food
- tomorrow => CCC-CH GV
- the day after => secure the distributed World!

What will we learn tonight?

- Basics of computer networks
- What common assumptions are inside our technology
- TCP & TLS
- Encrypted communication is not hard
- The actor model
- Basic protocol design steps
- RPC vs. MessagePassing

Multi-Processing

Why?

For costs (not just money)

- Expensive Hardware
- Better utilisation of energy
- One big vs. many cheap

For organisaiton

- Remote work
- Redundancy

What kind of hardware can we use?

On one board

- Multi Socket
- SMP
- Hyper Threading

Multi-Processing

Local aka. Multi-Core

Independent Units

Today we have a network in many levels …

- inside every SoC
- between components (I^2C, UART, GPIO, …)
- on our boards (PCIe, SATA, USB, …)
- outside of our boxes (Ethernet, WiFi, USB, …)

Each component is a Actor with own memory and communication channels.

Common abstractions are: ThreadPool, OpenCL.

Multi-Processing

Distributed

Was means distributed?

Geographically seperated

- inside every SoC
- between components (I^2C, UART, GPIO, ...)
- on our boards (PCIe, SATA, USB, ...)
- outside of our boxes (Ethernet, WiFi, USB, ...)

Each component is a Actor with own memory and communication channels.

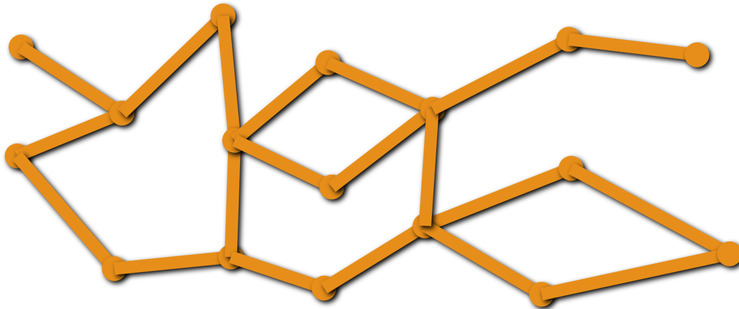
Common abstractions are: ThreadPool, OpenCL.

Computer Networks

How far can our computers see?

Topologies:

- Point 2 Point
- Bus
- Star
- Tree (Ethernet)
- Ring (TokenRing)
- Mesh (shared medium)


[1]

[1] A Graph by Stefan Schindler

What is the OSI model?

Application Process I — Data — Application Process n

System I	Logical connection	System 2
7: Application Layer	Data	7: Application Layer
6: Presentation Layer	Data	6: Presentation Layer
5: Session Layer	Data	5: Session Layer
4: Transport Layer	Data	4: TransportLayer
3: Network Layer	Data	3: Network Layer
2: Datalink Layer	Data	2: Datalink Layer
I: Physical Layer	Data	I: Physical Layer

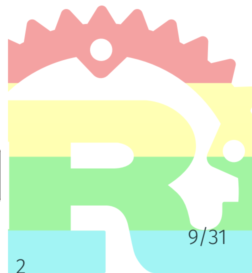
Data Transmission

System I

System 2

-------- Logical Connection in each Layer

▬▬▬ Realisation of the communication



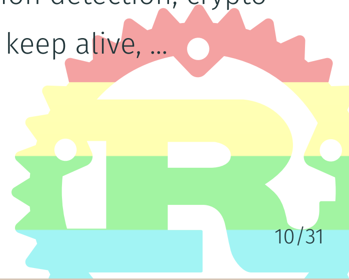
Computer Networks

TCP/IP

The assumptions of TCP/IP

Per Layer

- 1. The medium has noise -> Error correction, Collision Domains
- 2. There are different hosts -> MAC Address, direct logical access
- 3. Hosts are segmented -> IP, Routing, multi cast indirect access
- 4. Transmissions are unreliable -> TCP, ordering, retransmit, Streams
- 5. Untrusted transport -> TLS, confidentiallity, manipulation detection, crypto
- 6. Common problems -> Multi Plexing, HTTP, WebSocket, keep alive, ...
- 7. Business needs -> Our protocol



Example Project

Example Project

A distributed Gallery

Demo

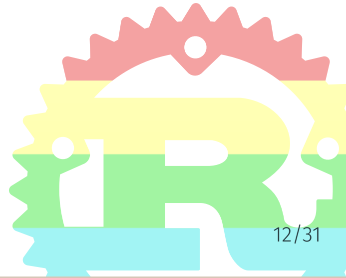
IPv4 https://94.45.228.24:3000

IPv6 https://[2a0a:e5c1:122:f00d:20d7:ec6f:86dd:1bf5]:3000

IPv6 local https://[fe80::ebaf:745:bd5a:814b]:3000

User Goals

1. Share your images with friends
2. Keep control of your data
3. Have an easy to setup central hub



User Stories

A. View Pictures:

- Open Applicaion with Browser
- Then browse Galleries

B. Share Images:

- Open Applicaion with Browser
- Optional: Choose a custom Name
- Select Images from your System
- Click the ”Share Button”

Features

Clients

- Subscribe to Hub Meta Data Service
- Dynamically load Images
- Non-Blocking
- Responsive

Central Hub

- Fully transport encrypted (TLSv1.2 & TLSv1.3)
- Do not store data
- Fast
- Async
- Multiple Clients at the same time

Storage Pods

- Easy to setup
- Have a way to update published Pictures
- Optional: reduce amount of data by resizing
- Optional: Remove Pictures without destroying the Pod

Example Project

Multi-Processing

The crates - Cargo.toml

```
[package]
name = "distributed_gallery"    version = "0.2.0"
authors = ["Stefan Schindler <dns2utf8@estada.ch>"]
edition = "2018"

[dependencies]
actix = "0.8"
actix-web = { version="1.0", features=["rust-tls"] }
chrono = { version = "0.4", features = ["serde"] }
actix-files = "0.1"                actix-web-actors = "1.0"
env_logger = "0.6"                 serde_derive = "1.0"
futures = "0.1"                    serde_json = "1.0"
serde = "1.0"                      rustls = "0.15"
```

The project layout

```
├── Cargo.lock
├── Cargo.toml
├── cert.pem
├── identity.pfx
├── key.pem
├── Makefile
├── README.md
├── src
│   ├── actors.rs
│   ├── main.rs
│   └── protocols.rs
│
...
└── static
    ├── DebugAll.js
    ├── gallery.js
    ├── index.html
    ├── main.js
    ├── self_host.js
    └── style.css
```

Preparing the actor system - main.rs

```
let sys = actix::System::new("master process");

let incrementor = Arc::new(Mutex::new(
    Incrementor { i: 0 }
));

// Start only one instance of our central Hub
let hub = SyncArbiter::start(1, || {
    Hub::default()
});
```

Loading the keys - main.rs

```
fn get_rustls_acceptor() -> Result<ServerConfig, rustls::TLSError
  use rustls::internal::pemfile::{certs, rsa_private_keys};
  use std::io::BufReader;
  let mut config = ServerConfig::new(NoClientAuth::new());

  let cert_file = &mut BufReader::new(File::open("cert.pem").expe
  let key_file = &mut BufReader::new(File::open("key_decrypted.pe
  let cert_chain = certs(cert_file).expect("unable to construct c
  let mut keys = rsa_private_keys(key_file).expect("unable to con

  config.set_single_cert(cert_chain, keys.pop().expect("no privat
  Ok(config)
}
```

Starting the server - main.rs

```
HttpServer::new(move || {
    App::new()    .data((incrementor.clone(), hub.clone()))
    .service(web::resource("/ws/")
.to(|req: HttpRequest, stream: web::Payload, data: Data<(Arc<Mute
    let (incrementor, hub) = &*data;
    ws::start(
        Ws { id: incrementor.lock().unwrap().increment(),
             hub: hub.clone(), is_pod: false, },
        &req, stream )  }))
    .service(fs::Files::new("/", "./static/").show_files_listing(
})
.bind_rustls(bind_addr, rustls_acceptor)
.expect("unable to construct HttpServer")  .start();
```

The WebSocket - actors.rs

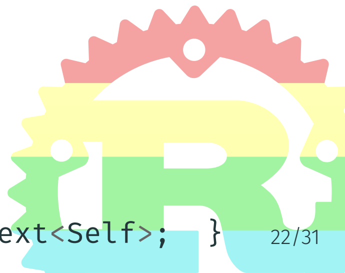
```
/// Define http actor
pub struct Ws {
    pub id: PodId, pub hub: Addr<Hub>, pub is_pod: bool,
}
impl Actor for Ws {
    type Context = ws::WebsocketContext<Self>;

    fn started(&mut self, ctx: &mut Self::Context) {
        self.hub.do_send(SubscribeClient {
            id: self.id, addr: ctx.address(),
        });    }
    fn stopped(&mut self, _ctx: &mut Self::Context) {
        self.hub.do_send(UnsubscribeClient(self.id));    }
}
```

The WebSocket handling rpc style messages - actors.rs

```
impl StreamHandler<ws::Message, ws::ProtocolError> for Ws {
  fn handle(&mut self, msg: ws::Message, ctx: &mut Self::Context)
    match msg {
      ws::Message::Text(text) => {
let message: Result<JsonProtocol, _> = serde_json::from_str(&text
match message {
    Ok(JsonProtocol::ClientRequest(message)) => {
        self.hub.send(message)    .into_actor(self)
            .then(|response, this, ctx| {
                let response: ClientResponse = response.expect("e
                actix::Handler::handle(this, response, ctx);
                fut::ok( () )
            }).spawn(ctx);
        }
```

```rust
#[derive(Default)]
pub struct Hub {
    pods: HashMap<PodId, PodInfo>,
    clients: HashMap<PodId, Addr<Ws>>,
}
impl Hub {
    fn broadcast_client_response(&self, message: ClientResponse)
        for addr in self.clients.values() {
            addr.do_send(message.clone())
        }
    }
}
impl Actor for Hub {  type Context = SyncContext<Self>;  }
```



The Hub handling rpc style messages - actors.rs

```
impl Handler<SubscribePod> for Hub {
    type Result = ();

    fn handle(&mut self, msg: SubscribePod,
              _ctx: &mut Self::Context) -> Self::Result {
        self.pods.insert(msg.id, PodInfo {
            addr: msg.addr,
            name: msg.name.clone(),
            image_paths: vec![], last_modified: Utc::now(),
        });
        self.broadcast_client_response(ClientResponse::NewPod {
              id: msg.id, name: msg.name, });
    }
}
```

The Hub handling async style messages - actors.rs

```
impl Handler<IdedPodRequest> for Hub {
    type Result = ();
    fn handle(&mut self, msg: IdedPodRequest,
                    _ctx: &mut Self::Context) -> Self::Result {
        use PodRequest::*;
        match msg.message {
            RegisterSelf { .. } =>
                unreachable!("must be handled by Ws"),
            UpdateTitle { name } => {
                self.pods.get_mut(&msg.id).expect("unable to find
                    .name = name.clone();
                self.broadcast_client_response(ClientResponse::Po
            }
```

The super protocol - protocols.rs

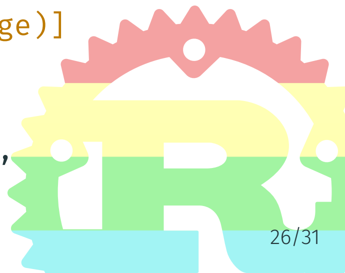
```rust
use serde_json as json;

pub type PodId = u64; // <- danger zone

#[derive(Serialize, Deserialize, Debug, Message)]
/// Communicate with everything
pub enum JsonProtocol {
    ClientRequest(ClientRequest),
    ClientRequestAsync(ClientRequestAsync),
    ClientResponse(ClientResponse),
    PodRequest(PodRequest),
    PodResponse(PodResponse),
}
```

The client client - protocols.rs

```
/// Browser -> Master rpc style
#[derive(Serialize, Deserialize, Debug, Message)]
#[rtype(result = "ClientResponse")]
pub enum ClientRequest {
    ListAllPods,     ListPodStructure(PodId),
}
/// Browser -> Master
#[derive(Serialize, Deserialize, Debug, Message)]
pub enum ClientRequestAsync {
    RequestImage {
        gallery_id: PodId,        path: String,
        #[serde(skip)]
        client_id: PodId,
    }
```



Working with the client 1/3 - protocols.rs

```
pub(crate) fn print_all_messages() {
    let t = |t| { println!("\n==== {} ====", t); };
    let p = |obj| {
        let s = json::to_string(&obj).unwrap();
        println!("  {}", s);
    };
    t("ClientRequest");
    p(JsonProtocol::ClientRequest(ClientRequest::ListAllPods));
    p(JsonProtocol::ClientRequest(
                ClientRequest::ListPodStructure(42)));

    ...
```

Working with the client 2/3 - protocols.rs

```
==== ClientRequest ====
  {"ClientRequest":"ListAllPods"}
  {"ClientRequest":{"ListPodStructure":42}}

==== ClientRequestAsync ====
  {"ClientRequestAsync":{"RequestImage":{"gallery_id":42,"path":"

==== ClientResponse ====
  {"ClientResponse":{"Pods":[{"id":42,"name":"bla","paths":[],"la
  {"ClientResponse":{"NewPod":{"id":23,"name":"blubb"}}}
  {"ClientResponse":{"UnknownPod":123}}
  {"ClientResponse":{"PodGone":1234}}
  {"ClientResponse":{"PodUpdateName":{"id":42,"name":"String"}}}
  {"ClientResponse":{"PodUpdatePaths":{"id":42,"paths":["String"]
```

Working with the client 3/3 - protocols.rs

```
==== PodRequest ====
  {"PodRequest":{"RegisterSelf":{"proposed_id":42,"name":"bla"}}}
  {"PodRequest":{"RegisterSelf":{"proposed_id":null,"name":"bla"}
  {"PodRequest":{"UpdateTitle":{"name":"bli"}}}
  {"PodRequest":{"UpdatePaths":{"paths":["bli"],"replace_images":
  {"PodRequest":{"DeliverImage":{"client_id":23,"path":"String","

==== PodResponse ====
  {"PodResponse":{"Registered":{"global_id":42}}}
  {"PodResponse":{"AlreadyRegistered":{"global_id":42}}}
  {"PodResponse":{"RequestImage":{"client_id":42,"path":"bli"}}}
```

Questions

Thank you for your attention!

Stefan Schindler @dns2utf8

Happy hacking! Please ask questions!

Slides & Examples: `https://gitlab.com/dns2utf8/distributed-and-secure-systems`

- It is hard to write safe and correct code.
- Even harder to write correct parallel code.

```
char *pi = "3.1415926f32";
while(1) {
    printf("Nth number? ");  err = scanf("%d", &nth);

    if (err == 0 || errno != 0) {
      printf("invalid entry\n");    while (getchar() != '\n');
      continue;
    }

    printf("Input: %d\n", nth);
    printf("Gewünschte Stelle: '%c'\n", pi[nth]);
}
```

```rust
let pi = "3.1415926f32";
loop {
    print!("Nth number? ");
    io::stdout().flush().unwrap(); // force display on terminal
    let mut input = String::new();
    match io::stdin().read_line(&mut input) {
        Ok(_bytes_read) => {
            let nth: usize = input.trim().parse()
                                        .expect("invalid selection");
            println!("{}-th: '{:?}'", nth, pi.chars().nth(nth));
        }
        Err(error) => println!("error: {}", error),
    }
}
```