Design Document for:

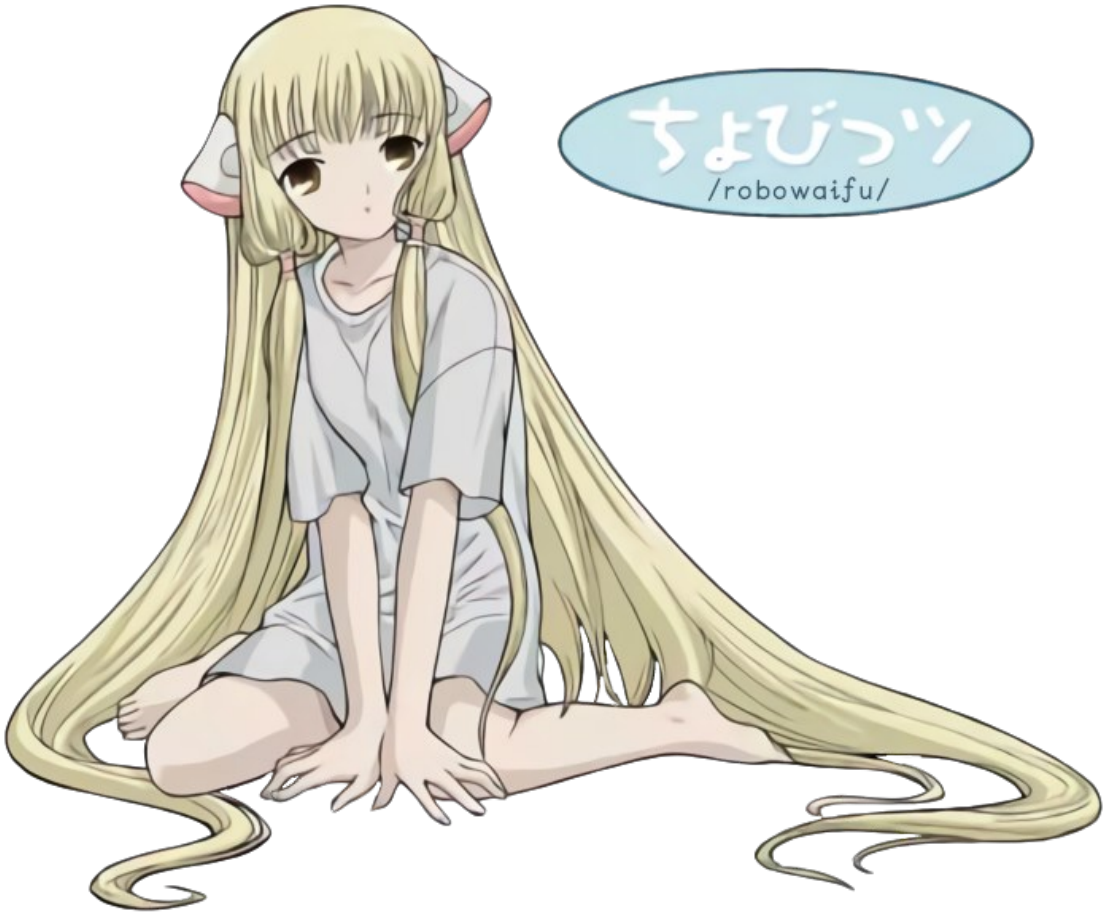
IPCNet

Inter-Process Communication Network Library

“In processes we trust.”™

Written by robowaifu engineers on /robowaifu/

Version 0.1
Thursday, April 16, 2020



Contents

"Are you the one just for me?" – Chii

DESIGN HISTORY **4**

VERSION 1.0 4

PROJECT OVERVIEW **5**

COMMON QUESTIONS 5

What is the project? *5*

Why create this project? *5*

What do processes control? *5*

What is it intended for? *5*

What is the main focus? *5*

What's different? *5*

DEVELOPMENT PHILOSOPHY 6

Simple design *6*

Robust and fail-safe *6*

More design goals to be added here *6*

PROJECT REQUIREMENTS **7**

DATA TRANSFER 7

MODULARITY 7

FAULT TOLERANCE 8

SECURITY 9

Permissions *9*

Process Groups *9*

Denial of Service mitigation *9*

Compromised Systems *9*

FEATURE SET **10**

GENERAL FEATURES 10

IMPLEMENTATION **11**

Booting....11
Event-driven Interface....11
Networking....11
IPC Sockets....11
Network Sockets....11
Security....11
Status Codes....11
Success....11
Client Errors....12
Service Errors....12
Custom Status Codes....12
System Shutdown....12
Memory Management....13
Language Bindings....13
DISCUSSION....14
Fail-safety....14
QUICK-RUN-DOWN APPENDIX....15
Modularity....15
Messaging I....15
Messaging II....15
Component libraries....16
Adaptability and Collaboration....16
Not need to be a one-man army....17

Design History

This is a brief summary of the history of this document and a space for explaining changes and their motivations in later versions as the design develops.

Version 1.0

Version 1.0 is just a rough initial draft. Details may be missing, erroneous or need to be filled in. The project is not ready for development yet and the details of its implementation may change dramatically in future versions.

Project Overview

Common Questions

What is the project?

IPCNet is a proposed parallel computing library that provides an easy-to-use interface for processes implemented in different languages across many platforms to share their data and provide data services to each other.

Why create this project?

Developers require a way to focus on building projects they find interesting and affordable to make, while others can quickly drop these components into their own robowaifu project.

What do processes control?

Processes have control over their data and subroutines and define the permissions and rules which they may be accessed, modified and executed. Processes may implement their own whitelists and blacklists, rate limits and usage limits. When processes query or request data it is just that, a request. Other processes do not have to fulfill it.

What is it intended for?

IPCNet is intended for parallel processes working together in collaboration, such as the components of a robot plugged into each other. IPCNet is not intended for programs with tightly coupled dependencies. When a process requests data there may be multiple processes available to fulfill the request.

What is the main focus?

The main focus is to connect developers' projects together and unite them with a single interface.

What's different?

IPC libraries that exist are specific to one or two programming languages and are not easy for developers to quickly include into their projects and interface processes across different languages or even different platforms. Parallel computing and grid computing software is often tooled for special purposes and are not intended for general use either. IPCNet provides a way for all kinds of processes and systems to interact, safely and quickly.

Development Philosophy

Simple design

IPCNet should be simple in its implementation and its interface. The interface should work the same across all platforms and programming languages and parse communicated data into a proper format that the receiving process can readily use. The library and processes that use it should not obfuscate their subroutines beyond what is sensible for security and fail-safety.

> “ Relying on complex tools to manage and build your system is going to hurt the end users. [...] "If you try to hide the complexity of the system, you'll end up with a more complex system". Layers of abstraction that serve to hide internals are never a good thing. Instead, the internals should be designed in a way such that they NEED no hiding.
>
> — Aaron Griffin ”

Robust and fail-safe

IPCNet needs to be robust, fault tolerant and fail-safe. Data coming in through various input methods may contain errors from interference or malice and need to be corrected or discarded. Errors and failures should be handled gracefully and degrade the system in a way that will cause minimal harm to the equipment, other equipment, to the environment and to people. The library and processes implementing the library should consider the following questions:

1. How critical is the component/process?
2. How likely is the component/process to fail?
3. How can the component/process be made fault tolerant?
4. How can the component/process be designed to reduce impact and damage to the rest of the system in the event of failure?

More design goals to be added here

Project Requirements

Data Transfer

All forms of data should be transmittable over the network with low-latency and high-throughput. Some forms of data that need to be carefully considered:

1. Variables of various data types passed to subroutines and returned
2. Data structures
3. AI tensor data (different libraries need to be able to exchange tensor data)
4. Streaming video, raw and encoded
5. Streaming audio, raw and encoded
6. Other sensor data, such as haptic sensors and LIDAR
7. Files

Beyond subroutine parameters, data types will need to be properly annotated with further metadata, describing the encoding of the data so it can be properly decoded and read by other processes.

All data transmitted over the network needs to be checked for integrity, both of malicious intent and errors caused by possible interference. Data should be traceable to its source and correctly marked indicating which processes modified it and in which order, so data pipelines can be quickly inspected and debugged. Historical changes to the data should be available in debug mode.

Modularity

Processes connected together through IPCNet should be easily separated and recombined, with the benefit of flexibility and a variety of choice in data and service providers. However, in the spirit of simple design these processes should not hide their complexity behind the interface beyond what is reasonable for security and safety to ensure the re-usability and versatility of these programs.

Fault Tolerance

1. **No single point of failure** – If a system experiences a failure, it must continue to operate without interruption during the repair process.
2. **Fault isolation to the failing component** – When a failure occurs, the system must be able to isolate the failure to the offending component.
3. **Fault containment to prevent propagation of the failure** – Some failure mechanisms can cause a system to fail by propagating the failure to the rest of the system. Mechanisms that can isolate a rogue component or failing component when it is unable to contain a fault are required to protect the system.
4. **Availability of reversion modes** – the abandonment of one or more recent changes in favor of a return to a previous version. If a component update breaks the system, it should revert back to the last working version.

Security

If someone wants to intercept data of an IPC socket, they're effectively tampering with a low-level mechanism part of the core operating system. If an attacker is able to do this, then the device has already been compromised.

Some processes may have access to the internet and potentially be exploited to create malicious requests within the IPC network or become fully compromised by arbitrary code execution. Additionally, any process could connect to the IPCNet and send malicious requests not created through the IPCNet library. Potentially rogue processes need to be identified, mitigated and isolated. Inputs from requests need to be sanitized and bad requests refused.

Permissions

Process routines registered on the network should have explicitly defined read, write and execution permissions for processes and process groups.

Process Groups

Processes can register processes into groups and give them their own permissions.

Denial of Service mitigation

1. Identify normal conditions for network traffic
2. Filter malicious traffic (connection tracking, component reputation lists, black/whitelisting, or rate limiting)

Compromised Systems

If a system is compromised by an attacker there is no way to verify the trustworthiness of processes. Data may be modified or processes abruptly shutdown by an attacker with intent to cause damage or harm. Checks should be in place to detect compromised states and shutdown safely.

Feature Set

General Features

P2P networking for processes – connected processes form a mesh network and can communicate with each other even if they’re not directly connected.

Announce data – processes can announce to the network their data and data services.

Find data and data services – processes can search the network for data and other processes providing data services.

Query data – processes can query data that other processes already have in memory or on disk and have made available.

Subscribe to data and data services – processes can subscribe to data and receive new updates whenever it is changed, or subscribe to services that provide a stream of data, continuously or at regular or irregular intervals.

Offer service – processes can provide data services to other processes to execute and produce new data on request. These services may be public or private.

Request service – processes can request data providers to execute subroutines that produce new data.

Implementation

To be written at a later date.

Booting

To be written at a later date.

Event-driven Interface

1. How can we guarantee low-latency and high-throughput?
2. How can we minimize metadata, network chatter and bandwidth?

Subroutine identifier

Subroutine return type

Arguments

Argument type

Metadata

To be written at a later date.

Networking

IPC Sockets

Essential.

Network Sockets

Non-essential. To be implemented at a later date.

Security

To be written at a later date.

Status Codes

Incomplete list, to be written at a later date.

Success

100 OK

101 Created

102 Accepted

103 No Data

104 Partial Data

Client Errors

200 Bad Request

201 Unauthorized

202 Forbidden

203 Not Found

204 Method Not Allowed

205 Too Many Requests

206 Upgrade Required

Service Errors

300 Internal Process Error

301 Not Implemented

302 Bad Gateway

303 Service Unavailable

304 Gateway Timeout

305 IPCNet Version Not Supported

306 Insufficient Storage

307 Insufficient Memory

308 Loop Detected

309 Network Authentication Required

Custom Status Codes

To be written at a later date.

System Shutdown

1. **How can we gracefully shutdown the system?** A robot may need to move to a safe location and shutdown in a way that does not cause damage to the components, other equipment near it, the environment or nearby people.
2. **How can we gracefully shutdown a compromised system?** An attacker who has compromised a device in the network may modify data or shut it down with intent to cause damage or harm.

Memory Management

To be written at a later date.

Language Bindings

1. C/C++
2. Python
    i. Pytorch
    ii. Tensorflow

Discussion

A summary of questions asked in the design document.

Fail-safety

1. How critical is the component/process?
2. How likely is the component/process to fail?
3. How can the component/process be made fault tolerant?
4. How can the component/process be designed to reduce impact and damage to the rest of the system in the event of failure?
5. How can we gracefully shutdown the system?
6. How can we gracefully shutdown a compromised system?

Quick-run-down Appendix

In case anything was missed in the design document.

Modularity

- Things must be swappable and interface with each other
- Modularity will be really important.
- We need a library and protocol or at least a guideline for networking components together, both hardware and software.
- That way people can work on building different parts they find interesting and can afford to make, while others can quickly drop these components into their own robowaifu project.
- Good modularity will help ensure different components work well together.

Messaging I

- Components need to communicate together properly
- Low latency, high throughput systems are needed.
- Will need to be able to pass large amounts of data around, such as AI tensor data and video data.
- Communications needs to be seamless across hardware and software boundaries.
- Similar to an IoT, but not on the Internet.
- Local data is stored within components, and shared with the rest of the internal robowaifu ’cloud’ in a standardized way.
- Discoverable interfaces, so other components can easily find, query, subscribe to, and request what they need from each other.

Messaging II

- Communications channels need to ensure integrity

- Where messages came from.
- Who modified messages.
- Where messages are going.
- Workaround communications bottlenecks, like the Internet does.
- Workaround rogue/misbehaving components that have been haxxored/damaged in some way.
- Workaround RF or other forms of interference.

Component libraries

- Uniform software interfaces between components
- C extern ABIs for consistency and provision of bindings to other languages.
- This will enable loosely-coupled collaboration to proceed more smoothly.
- For example;

    1. One anon could develop a chatbot.
    2. Another could develop their own in another language.
    3. They could quickly interface the two programs to each other in a few lines of code and have them banter for fun.
    4. Another dev could focus on making a visual waifu program.
    5. Then another anon could combine everything together through the component libraries to create two visual waifus bantering with each other. All without the devs having to directly collaborate with each other on each other's project development.

Adaptability and Collaboration

- The system must adapt to changing conditions and share workloads
- Components will need to be able to collaborate with other components.
- For example, a left-hand component will need a way to communicate with the right-hand component and coordinate their efforts.

- If a component becomes damaged or otherwise inhibited, the other components–and the system overall–should contain enough intelligence to adapt to this loss of component functionality.
- Even if all the pertinent data isn't immediately available, a component will still have to collaborate with other components. This means a component must be able to accept queries by remote components.

Not need to be a one-man army

- Anons being able to specialize in their areas of interest will be the key to progress and success.
- Having a reliable, modular system will open up ways for anons to collaborate with each other's work and allow each to contribute from their own interests.
- Let's have some fun with this!