ELISA
Enabling Linux in Safety Applications

WORKSHOP

SafetyGurad – Watchdog Software

Elisa Systems WG

Presented by
Sathishkumar Duraisamy, SDuraiEngineer
Philipp Ahmann, ETAS GmbH

SafetyGuard

A reusable, safety-oriented Linux supervision middleware.

ELISA
Enabling Linux in Safety Applications

WORKSHOP

SafetyGuard Goals

- Zero-copy IPC
- Deterministic timing
- Fault detection + watchdog
- Restart / recovery
- Rust-native core and C based API using FFI for language-agnostic for application
- Scalable toward industrial / automotive-style safety

Architectural Principles

- **Control Plane**

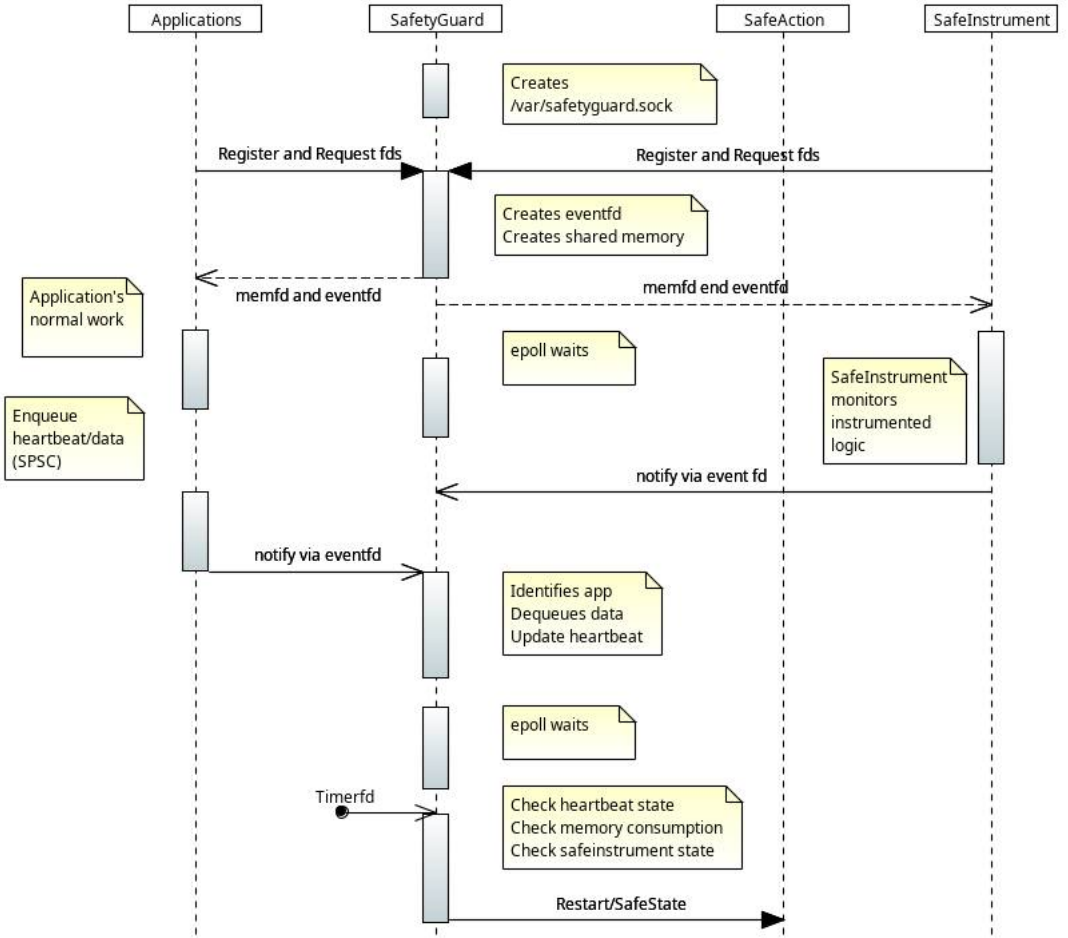
  UNIX socket registration + FD handoff
- **Data Plane**

  Shared Memory (mmap / memfd) + SPSC Ring Buffer
- **Signal Plane**

  eventfd + epoll (Like mosquitto broker, where mosquitto works over tcp socket)
- **Safety Plane**

  Monitor + timerfd watchdog + SafeInstrument + SafeAction (Heartbeat + Memory)

High-Level Architecture Diagram



ELISA Enabling Linux in Safety Applications

WORKSHOP

SAFE State and Safe Instrument

- Safe State and Safe Instrument are application specific.
- API will be provided.
- One can develop required domain specific safe state application and safe instrument application.
- As part of AGL, we will develop reference safe state and safe instrument apps.

Health Monitoring

Heartbeat

Each thread in app periodically sends: Timestamp and Status

Watchdog

SafeMonitor uses timerfd to check: now - last_heartbeat > timeout

Response

- Restart the app or container
- Enter degraded mode (When one or more apps timeout and restarted. Back to normal, when all apps does sends normal heartbeat)
- Enter safe state (Goal not restart the system, but to move to safesate, as determined by domain)
- Restart system if required (If safe state app sends response to restart)

Why SPSC?

Advantages

- Single Producer / Single Consumer simplicity
- Lock-free
- No mutex overhead
- Fixed-size deterministic memory
- Strong fit for safety arguments

Talk: https://www.youtube.com/watch?v=K3P_Lmq6pw0

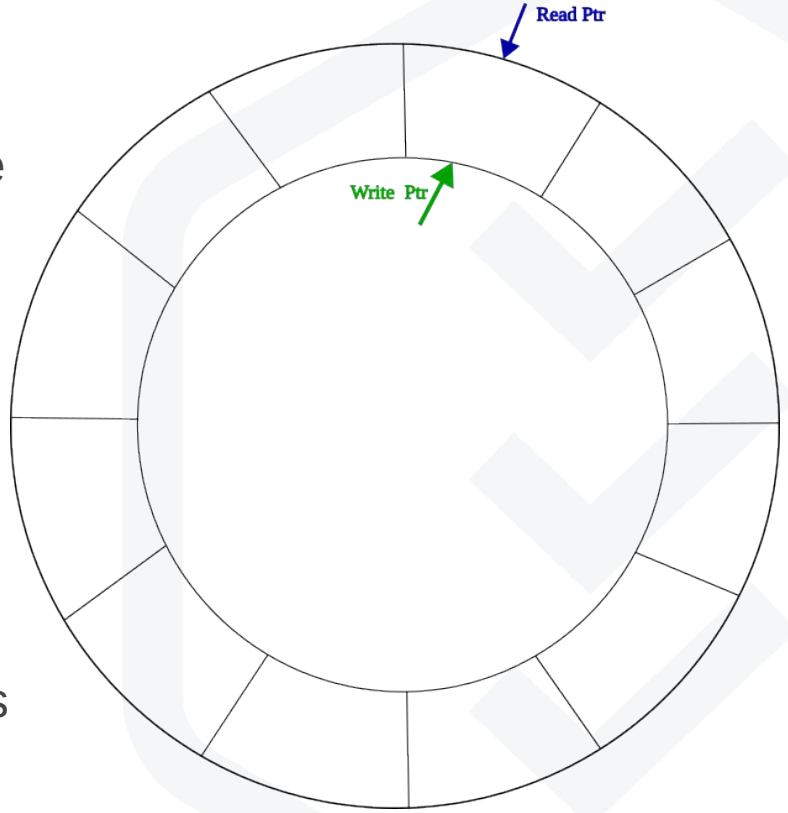
Runtime Data Flow (Fast Path)

Producer App

- ➔ Write Message to SPSC Queue
- ➔ Notify eventfd

SafetyGuard

- ➔ epoll wakeup
- ➔ Identify source app
- ➔ Dequeue all messages
- ➔ Process health / metrics / errors



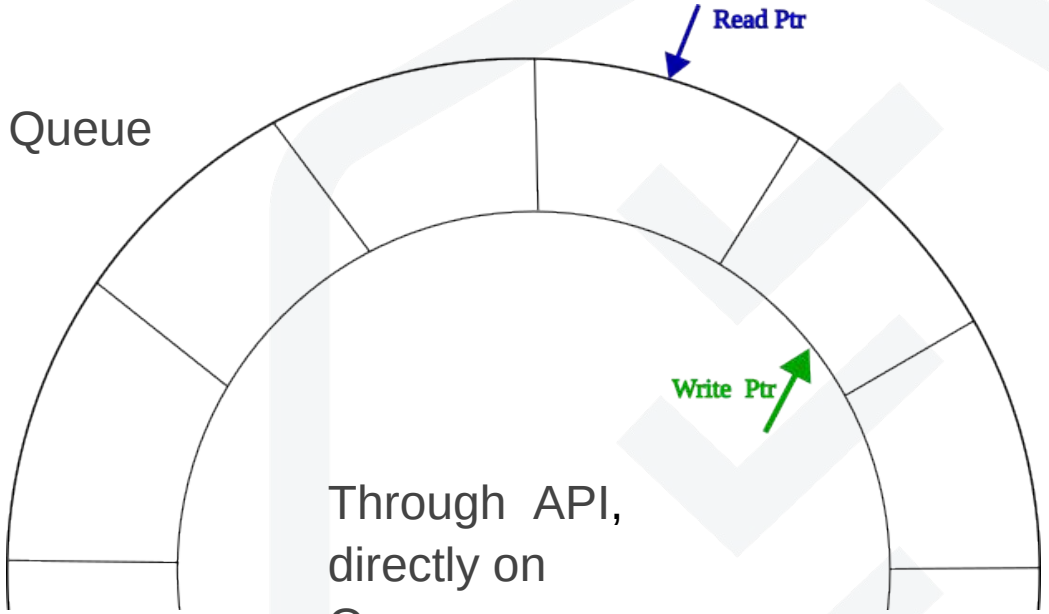
Runtime Data Flow (Fast Path)

Producer App

- ➔ Write Message to SPSC Queue
- ➔ Notify eventfd

SafetyGuard

- ➔ epoll wakeup
- ➔ Identify source app
- ➔ Dequeue all messages
- ➔ Process health / metrics / errors



Through API, directly on Queue Message Area

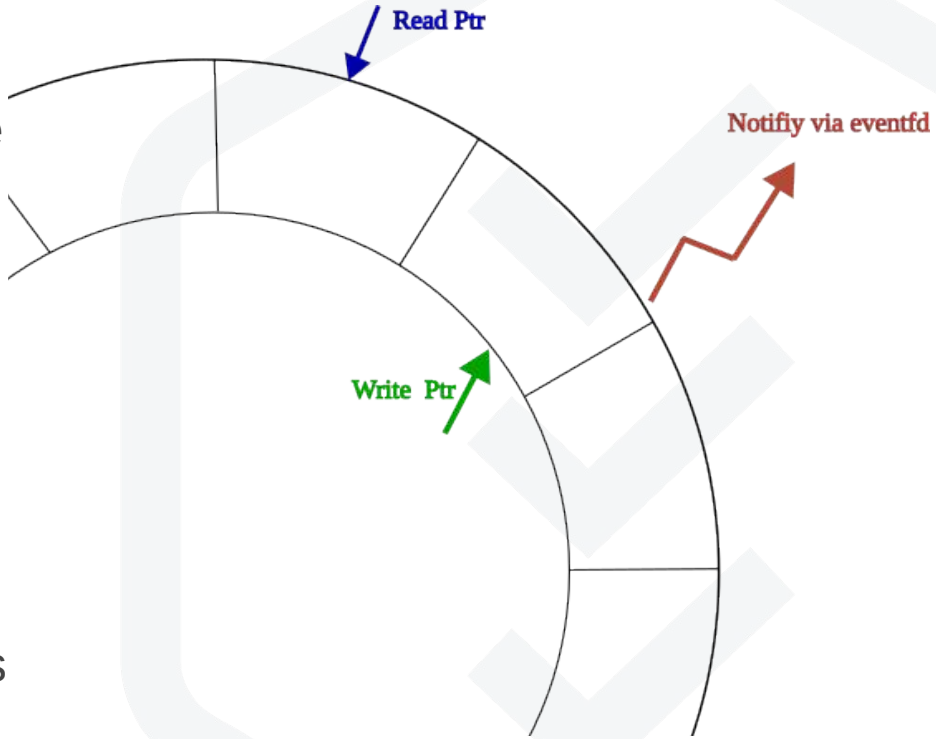
Runtime Data Flow (Fast Path)

Producer App

- ➔ Write Message to SPSC Queue
- ➔ Notify eventfd

SafetyGuard

- ➔ epoll wakeup
- ➔ Identify source app
- ➔ Dequeue all messages
- ➔ Process health / metrics / errors



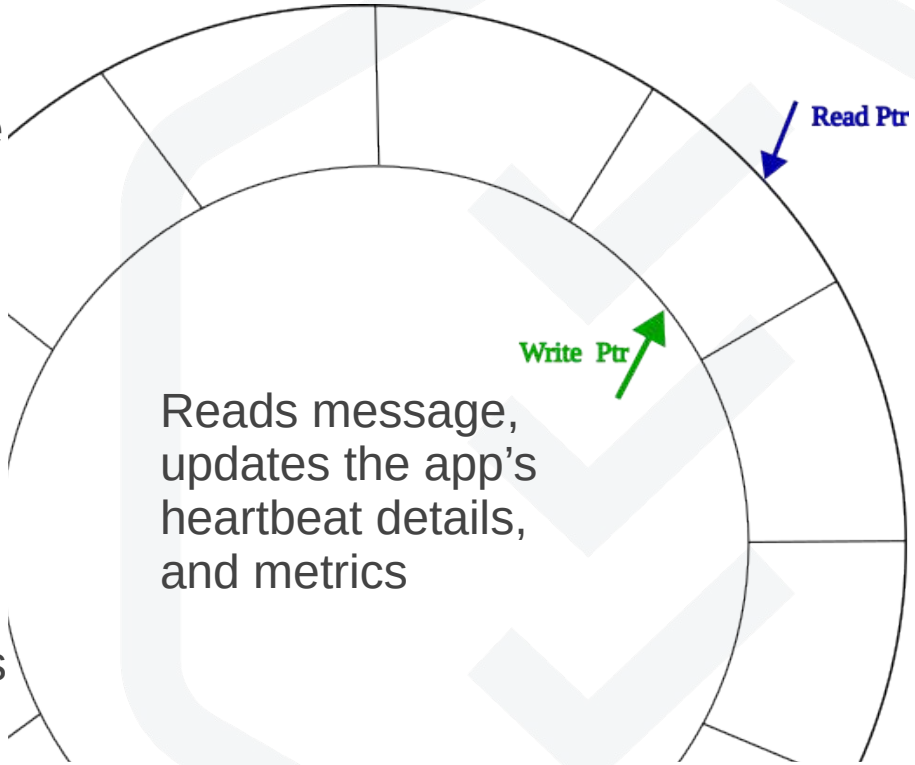
Runtime Data Flow (Fast Path)

Producer App

- ➔ Write Message to SPSC Queue
- ➔ Notify eventfd

SafetyGuard

- ➔ epoll wakeup
- ➔ Identify source app
- ➔ Dequeue all messages
- ➔ Process health / metrics / errors



Runtime Data Flow (Fast Path)

Producer App

- ➔ Write Message to SPSC Queue
- ➔ Notify eventfd

SafetyGuard

- ➔ epoll wakeup
- ➔ Identify source app
- ➔ Dequeue all messages
- ➔ Process health / metrics / errors

Benifits

- ✔ Zero-copy
- ✔ No-polling
- ✔ Per-app(thread) isolation
- ✔ With epoll single thread based design in safey-plane of Safetygurad.

Additional Components

Configuration Parser:

- Yaml based configuration parser to user defined parameters.

Configuration Parameter generator:

- Every system is different.
- During development this helps to generate parameters such as heartbeat rate, memory consumption and test safestates

AppManager:

- Generic API for managing the application.
- Act as interface to existing infrastructure to start, restart or stop Application.
- For example:
  - Systemd based AppManager
  - Container based AppManager
  - Limited native AppManager.

ELISA Enabling Linux in Safety Applications

WORKSHOP

Roadmap

Multi-App Scaling Strategy

Rule:

One App = One Queue + One eventfd for each thread

Benefits:

- Isolation
- Fault containment
- Clear source ownership
- Easy scaling with epoll

Result:

Multiple apps manageable with low overhead (like mosquitto)

Deployment Strategy

meta-elisa:

- Bitbake-class for inheriting dependency
- Generic SafetyGuard application
- Common imporant system apps patches such as systemd, gRPC, Dbus, Wayland(weston)

meta-elisa-agl:

- AGL specific application patches
- Reference SafeState and SafeInstrument Apps.

Embedded Linux / Yocto:

- SatetyGuard being yet another application, it will get launched by systemd
- Journald logging.

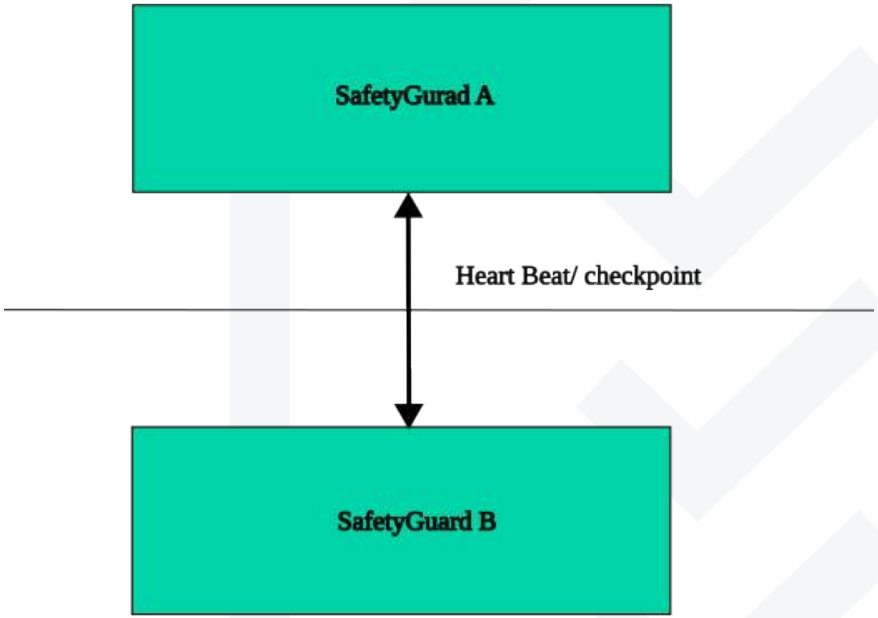
ELISA Enabling Linux in Safety Applications

WORKSHOP

Redundancy Roadmap (Future Safety Expansion)

- Dual Monitor (Primary + Secondary)
- Cross-monitor heartbeat/checkpoint
- Fail-safe state machine

Eg: Xen + dual VM running Linux



Security Considerations

Immediate

- Socket permission control
- App registration validation
- Version compatibility

Future

- Authentication (Require more inputs, can we do during provision of device?)
- Capability restrictions
- SELinux/AppArmor

Milestones

Phase 1 (Immediate)

- Single monitor
- Dynamic app registration
- SPSC + eventfd + epoll
- Heartbeat + memory monitoring + restart

Phase 2

- Logging + metrics
- Support for docker/container engine based applications
- C ABI stabilization
- Tools for system developer to debug and configure

Phase 3

- Redundancy
- Memory partition (Possible via cggroups?)

https://github.com/elisa-tech/safetyguard

ELISA
Enabling Linux in Safety Applications

Discussions

Discussions

The architecture technically allows to monitor the kernel threads as well via memfd+eventfd. Any use-cases?

Licensing of Workshop Results

All work created during the workshop is licensed under Creative Commons Attribution 4.0 International (CC-BY-4.0) [https://creativecommons.org/licenses/by/4.0/] by default, or under another suitable open-source license, e.g., GPL-2.0 for kernel code contributions.

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.