Efficient Integration of GStreamer Pipelines into External S/W Components

VLADIMIR BELOBORODOV / IIT RTC 2023 PRESENTATION

Briefly About Myself

- Vladimir “Vlad” Beloborodov
- Senior Staff Engineer and S/W Architect at PASA
- Professional Experience
  - Telecom and UC industries
  - Embedded Systems and IoT
  - Automotive Digital Cockpit Designs
- Part of WebRTC community since 2012

Panasonic
AUTOMOTIVE

Media Processing with GStreamer (“GST”)

- Well-developed time-proven framework
  - Cross-platform, with many prog. lang. bindings
  - Very flexible **pipeline-based** approach
- Wide selection of pipeline elements
  - GST repo itself offers **~900 elements** already
  - Support for many h/w-accelerated video coders
- Great as standalone tool or part of apps

Integrating GStreamer with Your App

- As part of your app process itself
  - More straightforward and easier to implement
  - (*Potentially*) the most efficient / performant
- As external process communicating with the app
  - App and GST can use different runtime libs (for libc, *etc.*)
  - More modular and composable solution designs
  - (*Potentially*) improved robustness and security
  - Stepping-stone to s/w designs for heterogeneous compute

Integrating GStreamer with Your App

- As part of your app process itself
  - **appsrc** and **appsink** are your friends ☺
  - Prefer **native callbacks** to signals
  - Ensure you properly support **GST threading** model
- As external process communicating with the app
  - Data exchange over **sockets** (good for distributed designs)
  - Data exchange over **shared memory** (better – *if available*)

Integrating GStreamer with Your App

- As part of your app process itself
  - **appsrc** and **appsink** are your friends ☺
  - Prefer **native callbacks** to signals
  - Ensure you properly support **GST threading** model
- As external process communicating with the app
  - Data exchange over **sockets** (good for distributed designs)
  - Data exchange over **shared memory** (better – *if available*)

Our focus today

GST elements **shmsrc** and **shmsink**

- Available as part of “GStreamer Bad Plug-ins”
  - Implemented and contributed by **Collabora** engineers in 2009
  - **Local sockets** for control and sharing memory between processes
  - **Shared memory** for data exchange (**shm_open** / **shm_unlink**)
- Limitations, concerns and practical problems
  - Available as GST elements only (**no ready external client library**)
  - *Security:* **shm_open** exposes shared mem. objects to all processes
  - **shm_open** is removed on Android (*due to security risks*)

My Project: ***in*GST** (*pron. "ingest"*)

- Standalone command line tool + client library for apps
  - Taking the same pipeline description syntax as GStreamer tools
  - *Custom elements "**in**" and "**out**" for sources and sinks used by app*
- Basic design approach
  - Built on top of core mechanisms and code from **shmsrc** / **shmsink**
  - Replacing **shm_open** with **memfd_create** on Linux
  - Passing shared memory file descriptors over local sockets

*in*GST: Current Status and Plans

- Happily using it in recent developments ☺
- Caveats
  - Immediate focus is on modern Linux systems (w/ **memfd_create**)
  - Not intended to be backward compatible with **shmsrc** / **shmsink**
- Near-Term Plans and Research Topics
  - Passing meta info together with input / output media frames
  - More GST event triggers for apps (*forcing key frames is available*)
  - *Open-sourcing on GitHub during Oct 2023*

*in*GST: Longer-Term Plans and Ideas

- Releasing my ***in*GST**-based video coder for WebRTC
- Rust library for clients
- Adding cross-platform support (beyond Linux)
- Supporting heterogeneous systems (CXL, *etc.*)
- *Looking forward to others' feedback too!*

Thank you for your time and interest!

Vladimir Beloborodov

linkedin.com/in/vladimirtechman

@VladimirTechMan

github.com/VladimirTechMan