Pluggable Device Drivers for Genode

Norman Feske
<norman.feske@genode-labs.com>
1. Background

2. Re-stacking the GUI stack

3. Pluggable network drivers

4. Bottom line and Outlook
1. Background

2. Re-stacking the GUI stack

3. Pluggable network drivers

4. Bottom line and Outlook
Genode OS Framework
Fault encapsulation in Sculpt OS

≈60 sandboxes right after boot
Ambitions

**Long-running systems**
- Partial system updates / downgrades
- Self-healing as promoted by MINIX3

**Assurance and fail-safety**
- Low-complexity trusted computing base
- Drivers are known to be flaky

**Adaptiveness at runtime**
- Changing output / input devices on the fly
- Power-gating of individual devices  
  (*saving energy of mobile phones*)
Genode’s strict client-server model

- Clients and servers are sandboxed independently
- Clients lend resources to servers for their services
- Mutual distrust (confidentiality, integrity)
- Liveliness of a client depends on the used servers
- Servers do not depend on clients
Layered architectures

- Decorator
- Web Browser
- Terminal
- Text Editor
- Window Manager
- Pointer
- Backdrop
- Nitpicker GUI Server
- Input Filter
- USB HID Driver
- Intel FB Driver
- USB Host Driver
- PS/2 Driver
- Platform Driver
- Core / Init
- MMIO
- Kernel

Pluggable Device Drivers for Genode
Layered architectures (real world)

- Kernel
- Core / Init
- IRQ
- MMIO
- Platform Driver
- Platform
- Intel FB Driver
-Framebuffer
- USB Host Driver
- USB
- PS/2 Driver
- USB HID Driver
- Input Filter
- USB HID Driver
- Input
- Framed buffer
- Intel FB Driver
- USB
- Platform Driver
- Platform
- Core / Init
- Kernel

Exposed to flaky devices
Highly complex

Pluggable Device Drivers for Genode
1. Background

2. Re-stacking the GUI stack

3. Pluggable network drivers

4. Bottom line and Outlook
Re-stacking the GUI stack

Pluggable Device Drivers for Genode
Introducing the capture service interface

Framebuffer Server
Framebuffer
Framebuffer Client
refresh pixels
sync signal

Capture Server
Capture
Capture Client
request changes (on sync) pixels

Pluggable Device Drivers for Genode
Introducing the event service interface

Input Server
- Input
- Input Client
- request
- pending events
- event flow
- notify

Event Server
- Event
- Event Client
- push
- event batch
- event flow

Pluggable Device Drivers for Genode
Challenge

How to get from the old to the new architecture?

- Around 50 existing system scenarios, i.e., Sculpt OS

Arsenal of existing device drivers

- Framebuffer drivers
  - OMAP4, Exynos5, Intel, i.MX8, i.MX53, Boot, PL11x, Rpi, SDL, VESA
- Input drivers
  - PS/2, USB HID, touch, ACPI input

Nitpicker GUI server and companions

- Support new architecture
- Preserve nesting ability
- Reversal of the input filter
Starting point

Pluggable Device Drivers for Genode
Extending the Nitpicker GUI server

**Nitpicker GUI Server**

- GUI
- Input
- Framebuffer

**Event**

- Capture
- GUI

**Input** (optional)

**Framebuffer** (optional)

Pluggable Device Drivers for Genode
Cutting Nitpicker’s dependency to the Input service

- Input-Event Client
- Nitpicker GUI Server
- Input Filter
- USB HID Driver
- PS/2 Driver
Inverting the drivers piece by piece
All drivers reversed

Pluggable Device Drivers for Genode

20
Replacing the input filter by new event filter

![Diagram showing the replacement of an input filter by an event filter with PS/2 and USB HID connections.]
Complexity collapses

Pluggable Device Drivers for Genode
Puzzle pieces coming together

Platform driver
- Arbitration of bus access (access control)
- IOMMU address space per driver
- Power-gating PCI devices

Heartbeat monitoring
- Liveliness of components ↔ response to I/O

Fault injection mechanism
- Genode’s trace service
- Default trace points (RPC call, RPC request, ...)
- Policy code executed by traced subjects (thread-local)
- Division by zero as trace policy code
1. Background

2. Re-stacking the GUI stack

3. Pluggable network drivers

4. Bottom line and Outlook
Traditional architecture (single application)
Traditional architecture (practical scenarios)

Network Application
- Socket API
- TCP/IP Stack

Guest OS
- vNIC
- Virtual Machine

NIC
- Packet routing
- DHCP
- Port forwarding
- Link-state handling
- Runtime-adjustable policy

Platform Driver
- IRQ
- MMIO

Core / Init

Kernel

Pluggable Device Drivers for Genode
Traditional architecture (real world)

Network Application
  - Socket API
    - TCP/IP Stack

Network Application
  - Socket API
    - TCP/IP Stack

Guest OS
  - vNIC
    - Virtual Machine

NIC Router
  - Packet routing
  - DHCP
  - Port forwarding
  - Link-state handling
  - Runtime-adjustable policy

NIC Driver

Platform Driver

Core / Init

MMIO

IRQ

Kernel

Pluggable Device Drivers for Genode
Introducing the Uplink service interface

NIC Server

NIC

NIC Client

request

MAC and

link state

payload

link state

changed

NIC

NIC Server

session exists = link up

MAC as session argument

payload

Uplink Client

Uplink

(Uplink Server

(NIC router)
Disposable NIC drivers

- Network Application
  - Socket API
  - TCP/IP Stack
- NIC Driver
- Platform Driver
  - Platform
- Core / Init
- Kernel
- IRQ
- MMIO
- NIC Router
- Uplink
- NIC
- Guest OS
  - vNIC
  - Virtual Machine
1. Background

2. Re-stacking the GUI stack

3. Pluggable network drivers

4. Bottom line and Outlook
Limitations

Block-device drivers not covered
- Stateful, consistency, caching
- May be addressed separately...

Bus drivers e.g., USB host controller
- Need to be long-living
- May have far reach
Prospects

- Dynamic multi-head scenarios
- Screen capturing
- Virtual on-screen keyboard
- Swapping out drivers (fallbacks)
- Updating and downgrading of drivers during runtime
- Power saving (on-demand drivers, automated power gating)
- Simplify suspend/resume (driver life-cycle management)
Thank you

Genode OS Framework
https://genode.org

Sculpt OS download and manual
https://genode.org/download/sculpt

Genodians.org community blog
https://genodians.org

Genode Labs GmbH
https://www.genode-labs.com

Company newsletter
https://genode-labs.com/newsletter