Reaching puberty:
How Genode is becoming
a general-purpose OS

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

2. Noux runtime for Unix software

3. Challenges of dynamic system composition

4. Fundamental features

5. Current ventures
1. Background

2. Noux runtime for Unix software

3. Challenges of dynamic system composition

4. Fundamental features

5. Current ventures
Genode in a nut shell

→ Application-specific TCB
Combined with virtualization
Genode OS Framework

FIASCO.OC  OKL4  CODEZER0
FIASCO  NOVA  Microhypervisor
L4Ka  MicroBlaze

Reaching puberty: How Genode is becoming a general-purpose OS
Preservation of special kernel features
- OKLinux on OKL4,
- L4Linux on Fiasco.OC,
- Vancouver on NOVA,
- Real-time priorities on L4/Fiasco

Uniform API $\rightarrow$ kernel-independent components

Many ready-to-use device drivers, protocol stacks, and 3rd-party libraries
Eating our own dog food

Fundamentals

- VIM
- Tool chain
- Shell
- Fallback VM
- Web browser
- PDF viewer
- Tiled window manager
- Git client
- GNUPG
- SSH client, Rsync
- Persistent storage
- IM client
Noux runtime for Unix software

Idea: Provide Unix kernel interface as a service

fundamentals
- write, read
- stat, lstat, fstat, fcntl
- ioctl
- open, close, lseek
- dirent
- getcwd, fchdir
- select
- execve, fork, wait4
- getpid
- pipe
- dup2
- unlink, rename, mkdir

networking
- socket
- getsockopt, setsockopt
- accept
- bind
- listen
- send, sendto
- recv, recvfrom
- getpeername
- shutdown
- connect
- getaddrinfo

In contrast, Linux has more than 300 syscalls
Things we don’t need to consider

- Interaction with device drivers
- Unix initialization sequence
- Users, groups
  *Instance never shared by multiple users*
  *The opposite: One user may run many instances*
- Multi-threading
- Scalability of a single instance
  *Each instance serves one specific (limited) purpose*
  *Run many instances in order to scale!*
Noux runtime for Unix software (3)
noux config

<config>
  <fstab> <tar name="vim.tar" />
  <start name="/bin/vim">
    <env name="TERM" value="linux" />
    <arg value="--noplugi" />
    <arg value="-n" />
    <arg value="-N" />
  </start>
</config>
noux config

<config>
  <fstab>
    <tar name="coreutils.tar" />
    <tar name="vim.tar" />
    <tar name="bash.tar" />
    <dir name="home"> <fs label="home" /> </dir>
    <dir name="ram"> <fs label="root" /> </dir>
    <dir name="tmp"> <fs label="tmp" /> </dir>
  </fstab>
  <start name="/bin/bash">
    <env name="TERM" value="linux" />
  </start>
</config>
ram_fs config

<config>
  <content>
    <dir name="tmp">
      <rom name="init" as="something" />
    </dir>
    <dir name="home">
      <dir name="user">
        <rom name="timer" />
      </dir>
    </dir>
  </content>
  <policy label="noux -> root" root="/" />
  <policy label="noux -> home" root="/home/user" writeable="yes" />
  <policy label="noux -> tmp" root="/tmp" writeable="yes" />
</config>
Noux features

- Executes unmodified GNU software
  Bash, VIM, GCC, Coreutils, Lynx...

- Supports stacked file systems

- Instance starts in fraction of a second

- Uses original GNU build system → Porting software is easy

- Two versions
  - noux/minimal
  - noux/net (includes TCP/IP)

*less than 5,000 LOC*
1. Background

2. Noux runtime for Unix software

3. Challenges of dynamic system composition

4. Fundamental features

5. Current ventures
Unified configuration concept

```xml
<config>
  <parent-provides> ... </parent-provides>
  <default-route> ... </default-route>
  ...
  <start name="nitpicker">
    ...
  </start>
  <start name="launchpad">
    ...
    <config>
      <launcher>
        <filename>init</filename>
        <config>
          <parent-provides> ... </parent-provides>
          <default-route>
            <any-service> <any-child/> <parent/> </any-service>
          </default-route>
          <start name="nit_fb">
            <resource name="RAM" quantum="6M"/>
            <config xpos="400" ypos="270" width="300" height="200"/>
            <provides> <service name="Input"/>
              <service name="Framebuffer"/>
            </provides>
          </start>
          <start name="14linux">
            <resource name="RAM" quantum="1G"/>
            <config args="mem=52M 14_x_rd=initrd.gz"/>
          </start>
        </config>
      </launcher>
    </config>
  </start>
</config>
```

Reaching puberty: How Genode is becoming a general-purpose OS
→ Uniform syntax

→ Extensible through custom tags at each level

→ XML parser adds less than 300 LOC to TCB
Dynamic system configuration

Problems

- Change screen resolution at runtime
- Audio-mixing parameters
- Touchscreen calibration
- Resizing terminal windows
- Policy for hot-plugged device resources
Dynamic system configuration (2)

**Straight-forward approach**
Introduce problem-specific RPC interfaces

**Disadvantages**
- New RPC interfaces $\rightarrow$ added complexity
- Specific to the server *implementation*
- Redundancy to existing (static) configuration concept
Generalized solution

- Turn static config mechanism into dynamic mechanism

How?

- Add single RPC function to ROM session interface:
  ```c
  void sigh(Signal_context_capability sigh)
  ```
- Client responds to signal by re-acquiring session resources
Dynamic system configuration (4)

- Codec (avplay)
  - SDL audio
  - SDL video

- ROM "mediafile"

- ROM "config"
  
  <config>
  <sdl_audio_volume value="85"/>
  </config>

- GUI
  - virtual ROM
  - virtual frame buffer

- ROM service

- Nitpicker GUI server

- Frame buffer

- Nitpicker

- Nitpicker

Reaching puberty: How Genode is becoming a general-purpose OS
Loader service

Challenges

- Start and stop subsystems at runtime
- Controlled by software
- Decouple started subsystem from controlling software

Solution

- Trusted *loader* service
- Client pays
- Client configures subsystem
- Client cannot interfere during runtime
Loader service
Outline

1. Background

2. Noux runtime for Unix software

3. Challenges of dynamic system composition

4. Fundamental features

5. Current ventures
File-system infrastructure

**FreeBSD libc turned into modular C runtime**

libports/lib/mk/libc.mk

libports/lib/mk/libc_log.mk

libports/lib/mk/libc_fs.mk

libports/lib/mk/libc_rom.mk

libports/lib/mk/libc_lwip.mk

libports/lib/mk/libc_ffat.mk

libports/lib/mk/libc_lock_pipe.mk

→ *application-specific plugins*
File-system infrastructure (2)

Application

FreeBSD libc

libc_ffat plugin

read, write

Block session

Block-device driver
Media playback

Codec (avplay)
- SDL audio
- SDL video

ROM "mediafile"

ROM "config"

<config>
  <sdl_audio_volume value="85"/>
</config>

GUI
- virtual ROM
- virtual frame buffer

ROM service

Frame buffer

Nitpicker GUI server

Nitpicker

Reaching puberty: How Genode is becoming a general-purpose OS
User-level debugging

Genode Process

Service

RAM RM CPU Core

session
User-level debugging (2)
User-level debugging (3)

Genode Process as Debugging Target

RAM | RM | CPU

GDB Monitor

Service

Noux

GNU Debugger

RAM | RM | CPU

Core

session
Construction sites

- Kernels
- Base system
- C runtime, 3rd-party libraries
- Noux
- Porting the tool-chain components
  \emph{GCC, binutils, GNU make, findutils}

→ Insightful lessons about application performance
Outline

1. Background

2. Noux runtime for Unix software

3. Challenges of dynamic system composition

4. Fundamental features

5. Current ventures
User interface concept

Genode’s architecture calls for tailored UI concept

**Ingredients**  Nitpicker, framebuffer drivers, input drivers

**Desired**
- Convenient command-line interface
- Scripting
- Flexibility
  - multi-head, virtual desktops, different window layouts
- Resource management
Performance and scalability

- Multi-processor support
  - NUMA
  - Challenge: Platform-independent API
  - Facilitating Genode’s recursive structure

- Storage
  \textit{I/O scheduling, caching}

- Networking (i.e., TCP/IP performance)

- Tools
  \textit{Profiling, debugging, tracing}
Networking and security

- IOMMU support on NOVA
- Trusted computing
  → Network of Genode systems
- Capability-based security on Linux
Noux: Unix networking tools

**Needed command-line tools**

- netcat, wget, ...
- Lynx + SSL
- SSH
- Git

**Approach**

Integrate lwIP into Noux runtime

→ One TCP/IP stack per Noux instances
More light-weight device-driver environments (e.g., OSS)

ARM TrustZone

Hardware support (e.g., ARM SoCs)

HelenOS Spartan kernel

“Real” file system

Virtual NAT

Genode on FPGA softcores
Thank you

Genode OS Framework
http://genode.org

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

Source code at GitHub
http://github.com/genodelabs/genode