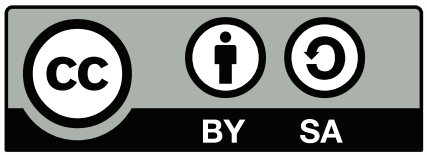
The VFS paradigm from the perspective of a component OS

Christian Helmuth

<christian.helmuth@genode-labs.com>

Copyright



Outline

1. Motivation
2. History of the VFS in Genode
3. Where are we now?
4. Main course finished, dessert anyone?

An OS without applications is of limited use

An OS without applications is of limited use

- Existing open-source applications can be ported
- Gradual decomposition (kernelization) of sensitive parts

An OS without applications is of limited use

- Existing open-source applications can be ported
- Gradual decomposition (kernelization) of sensitive parts
- Reconstruction of complex (single-purpose) software not appealing

An OS without applications is of limited use

- Existing open-source applications can be ported
- Gradual decomposition (kernelization) of sensitive parts
- Reconstruction of complex (single-purpose) software not appealing
- Consequently traditional applications will always be around

A square peg into a round hole?

Many attractive applications require a POSIX environment.

A square peg into a round hole?

Many attractive applications require a POSIX environment.

- Fairly global view on resources as a tree of directories and files
- File abstraction grants access to data on storage, configuration, hardware peripherals, (graphical) user interfaces, to some extent even networking

A square peg into a round hole?

Many attractive applications require a POSIX environment.

- Fairly global view on resources as a tree of directories and files
- File abstraction grants access to data on storage, configuration, hardware peripherals, (graphical) user interfaces, to some extent even networking
- Mostly traditional access control (user/group permission bits)
- C/C++ runtime
- Extensive system API (only a small share used)

A square peg into a round hole?

Our playground is Genode.

A square peg into a round hole?

Our playground is Genode.

- Microkernel-based
- Capability-based
  - No global namespace
  - Fine-grained access control to resources
  - Recursive system structure
- Component-based
  - Versatile combination of components according to use case
  - Spectrum from strong dependency to loose coupling
  - Only integrate what's needed to solve the task
  - Low software complexity as primary goal

A square peg into a round hole?

Application sandboxing seems a viable approach.

A square peg into a round hole?

Application sandboxing seems a viable approach.

- Customized C library
- Runtime environment including VFS

A square peg into a round hole?

Application sandboxing seems a viable approach.

- Customized C library
- Runtime environment including VFS
- How about the world beyond the frame of the sand pit?
- Connect to Genode services

Outline

1. Motivation
2. History of the VFS in Genode
3. Where are we now?
4. Main course finished, dessert anyone?

Customized C library

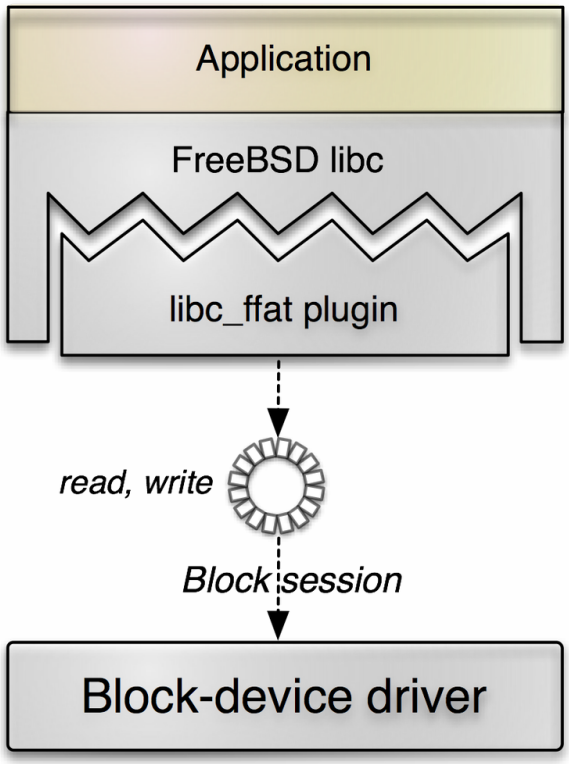
Customized C library

- Back end for used subset of C library functions
- Some emulated mechanisms to satisfy applications (e. g., UIDs, permissions, sysctl)

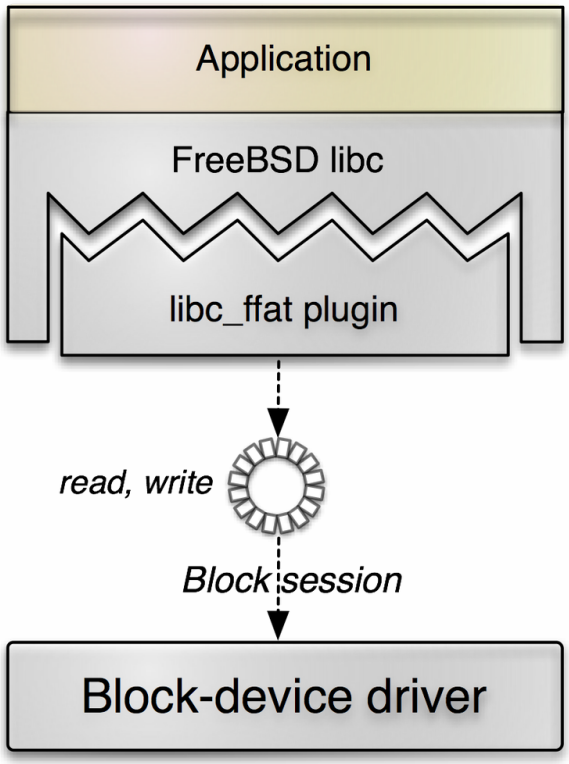
Customized C library

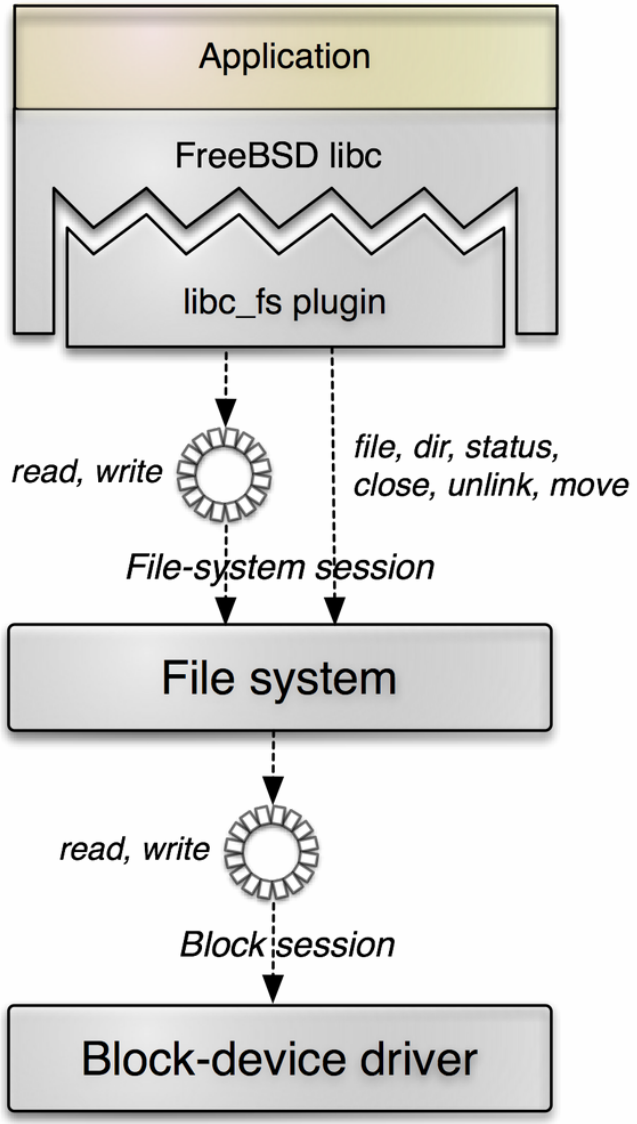
- Back end for used subset of C library functions
- Some emulated mechanisms to satisfy applications (e. g., UIDs, permissions, sysctl)
- Gradually develop C library plugins to access Genode services
  - File-system libraries (FFAT, FUSE) use Block service
  - Network-stack libraries (lwip, lxip) use Nic service
  - Terminal service
  - File-system service
  - Special-purpose (e. g., libdrm/gallium)

Customized C library



Customized C library





Noux runtime for Unix software

Noux runtime for Unix software

- UNIX command-line utilities running in shell

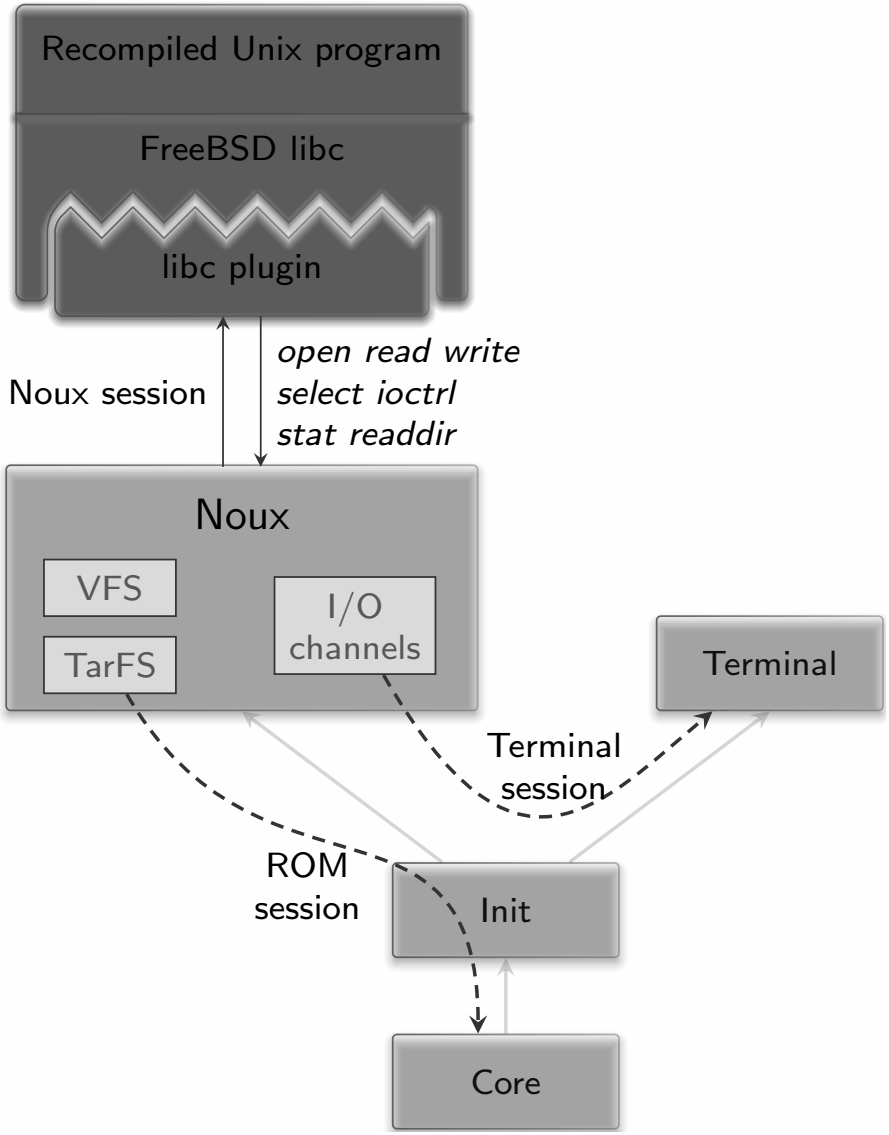
Noux runtime for Unix software

- UNIX command-line utilities running in shell
- Fork/exec child processes
- Pipes, process management

Noux runtime for Unix software

- UNIX command-line utilities running in shell
- Fork/exec child processes
- Pipes, process management
- Configure and build with original GNU build system
- From process-local to Noux-wide resource representation

Noux runtime for Unix software



Plug VFS into C library

Plug VFS into C library

- VFS and plugins in Noux and stand-alone C applications alike
- Gradually replace C library plugins by VFS plugins

Plug VFS into C library

- VFS and plugins in Noux and stand-alone C applications alike
- Gradually replace C library plugins by VFS plugins
- Process-local VFS instance
- Individual configuration

Basic configuration

Basic configuration

VFS instance as XML node in component configuration

```
<config> <vfs>...</vfs> </config>
```

Basic configuration

VFS instance as XML node in component configuration

```
<config> <vfs>...</vfs> </config>
```

Directories

```
<vfs> <dir name="dev">...</dir> </vfs>
```

Basic configuration

VFS instance as XML node in component configuration

```
<config> <vfs>...</vfs> </config>
```

Directories

```
<vfs> <dir name="dev">...</dir> </vfs>
```

Plugins as nodes in the tree

```
<vfs> <dir name="dev"> <log/> </dir> </vfs>
```

Basic configuration

VFS instance as XML node in component configuration

```
<config> <vfs>...</vfs> </config>
```

Directories

```
<vfs> <dir name="dev">...</dir> </vfs>
```

Plugins as nodes in the tree

```
<vfs> <dir name="dev"> <log/> </dir> </vfs>
```

Configure C library mechanisms to use VFS nodes

```
<libc stdout="/dev/log" stderr="/dev/log"/>
```

Basic configuration

```
<config>
  <vfs>
    <dir name="dev"> <log/> </dir>
  </vfs>
  <libc stdout="/dev/log" stderr="/dev/log"/>
</config>
```

- Precise *declaration* of resource representation in VFS
- Plugins can be single files or whole directory sub-trees
- Nodes in a directory organized as stack (or union mount)
- Tweaking of C library behavior

VFS plugins at a glance

Files backed by ROM service

```
<rom name=".vimrc" label="vimrc.txt"/>
<rom name="avatar.png"/>
```

VFS plugins at a glance

Files backed by ROM service

```
<rom name=".vimrc" label="vimrc.txt"/>
<rom name="avatar.png"/>
```

Inline-defined file contents

```
<inline name="app.config">avatar = /avatar.png</inline>
```

VFS plugins at a glance

Files backed by ROM service

```
<rom name=".vimrc" label="vimrc.txt"/>
<rom name="avatar.png"/>
```

Inline-defined file contents

```
<inline name="app.config">avatar = /avatar.png</inline>
```

Handy tools

```
<null/> <zero/> <symlink name="editor" target="/bin/vim"/>
```

VFS plugins at a glance

RAM-backed storage (like tmpfs)

```
<ram/>
```

VFS plugins at a glance

RAM-backed storage (like tmpfs)

```
<ram/>
```

Integration of package-archive trees

```
<tar name="vim.tar"/>
<tar name="vim-syntax.tar"/>
```

VFS plugins at a glance

RAM-backed storage (like tmpfs)

```
<ram/>
```

Integration of package-archive trees

```
<tar name="vim.tar"/>
<tar name="vim-syntax.tar"/>
```

File-to-service wrappers

```
<log/> <rtc/> <terminal/> <block/>
```

VFS plugins at a glance

Expandable by custom shared objects

```
<jitterentropy name="random"/>
<gtotp name="gtop.service.net" secret="IMGLPG6VANGX3UCP"/>
```

VFS plugins at a glance

Expandable by custom shared objects

```
<jitterentropy name="random"/>
<gtotp name="gtop.service.net" secret="IMGLPG6VANGX3UCP"/>
```

File-system session to use persistent storage

```
<fs name="home"/>
<fs name="cfg" label="config" writeable="yes"/>
<fs name="bin" label="bin" root="/usr"/>
```

VFS plugins at a glance

Expandable by custom shared objects

```
<jitterentropy name="random"/>
<gtotp name="gtop.service.net" secret="IMGLPG6VANGX3UCP"/>
```

File-system session to use persistent storage

```
<fs name="home"/>
<fs name="cfg" label="config" writeable="yes"/>
<fs name="bin" label="bin" root="/usr"/>
```

VFS configuration is component-local → access control by policies in parent and service components (e. g., file-system service)

```
<policy label="app -> "       root="/home" writeable="yes"/>
<policy label="app -> config" root="/app/config" writeable="no"/>
<policy label="app -> bin"    root="/" writeable="no"/>
```

Outline

1. Motivation
2. History of the VFS in Genode
3. Where are we now?
4. Main course finished, dessert anyone?

Huh, a VFS server?

Huh, a VFS server?

- Robust implementation of file-system service
- File-system itself implemented as VFS plugin

Huh, a VFS server?

- Robust implementation of file-system service
- File-system itself implemented as VFS plugin
- One VFS configuration for multiple components
- Multiplex access to all aggregated resources
- Differentiate client permissions by policies

Huh, a VFS server?

- Dynamic reconfiguration of server applies to all clients

Huh, a VFS server?

- Dynamic reconfiguration of server applies to all clients
- All VFS plugins can be used in the server

Huh, a VFS server?

- Dynamic reconfiguration of server applies to all clients
- All VFS plugins can be used in the server
- Now, at the latest, HURD translators come into mind...

Is there more about that server?

Shared resources could be provided by large plugins.

Is there more about that server?

Shared resources could be provided by large plugins.

- Dedicated rump (ext2) server superseded by plugin

Is there more about that server?

Shared resources could be provided by large plugins.

- Dedicated rump (ext2) server superseded by plugin
- Network stack based on lxip/lwip

```
<!-- server -->
<vfs> <lxip/> </vfs>

<!-- client -->
<vfs> <dir name="/socket"> <fs/> </dir> </vfs>
<libc socket=/socket>
```

Outline

1. Motivation
2. History of the VFS in Genode
3. Where are we now?
4. Main course finished, dessert anyone?

Review our goals

- Audited system resource discovery and access
- C library integration enables existing POSIX applications
- Versatile combination inside component and via VFS server
- Abstraction from Genode services
- Future extension is easy (e. g., USB service adapter for libusb)

Outlook

- Extend VFS interface to support plugins using other plugins

Outlook

- Extend VFS interface to support plugins using other plugins
- Plugin-based filter chains become possible
  - Mangling/routing of mouse/keyboard input events
  - File systems using block-device nodes

Outlook

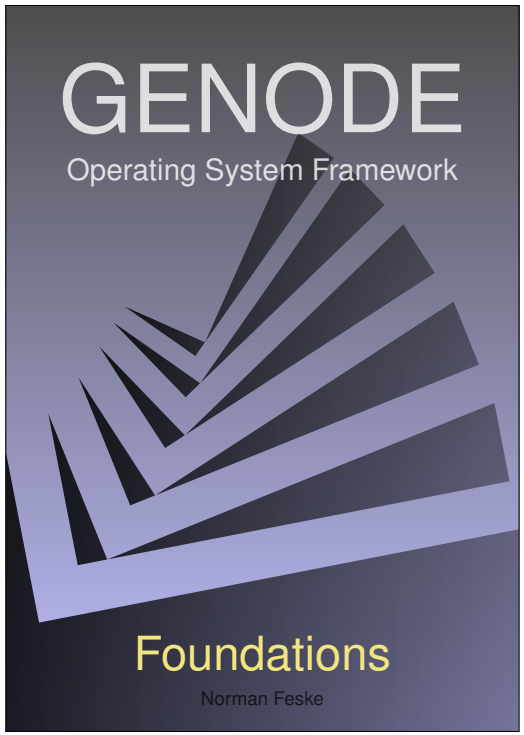
- Extend VFS interface to support plugins using other plugins
- Plugin-based filter chains become possible
  - Mangling/routing of mouse/keyboard input events
  - File systems using block-device nodes
- Split applications scenarios driven by security considerations
  - PDF reader with file-to-HTTP plugin in separate component
  - Separate domains for edit-compile-test-push development workflow

Outlook

- Extend VFS interface to support plugins using other plugins
- Plugin-based filter chains become possible
  - Mangling/routing of mouse/keyboard input events
  - File systems using block-device nodes
- Split applications scenarios driven by security considerations
  - PDF reader with file-to-HTTP plugin in separate component
  - Separate domains for edit-compile-test-push development workflow

Application-stack architectures range from multiple components connected by file-system sessions to unikernel-like monoliths.

Thank you



Genode OS Framework

`https://genode.org/`

Genode Labs GmbH

`https://www.genode-labs.com/`

Source code on GitHub

`https://github.com/genodelabs/genode`

Genode Foundations book

`https://genode.org/documentation/genode-foundations-16-05.pdf`