Krishveer Grewal

GrooveOS Design Document

Introduction

GrooveOS is a microkernel-inspired, modular operating system designed for real-time embedded audio platforms, including grooveboxes, synthesizers, samplers, and other performance-critical devices. The architecture emphasizes strict determinism, component isolation, and message-driven communication between subsystems.

GrooveOS operates in two execution contexts:

Embedded Mode

Runs directly on embedded hardware (ARM SBCs, microcontrollers), providing:

- Tight CPU budget control
- Low-latency behavior
- Real-time scheduling precision
- Hardware abstraction via drivers

Simulation Mode

Runs as a standard application under Linux for:

- Debugging
- Development
- Instrumentation
- Performance profiling

Core Design Goals

1. Deterministic Task Scheduling
   Ensures predictable timing for audio and control events.
2. Strict Modularity
   Each subsystem (Kernel, Task Manager, MessageBus, SysLog, DriverManager, Shell) operates independently.
3. Message-Driven IPC
   All components publish or subscribe to messages rather than calling each other directly.

4. High Transparency
   The Shell exposes internal state to the developer at runtime.

2. High-Level Architecture Overview

GrooveOS consists of six principal subsystems:

- Kernel – Bootstraps the system and orchestrates lifecycle transitions
- Task Manager – Cooperative scheduler for all tasks
- MessageBus – Asynchronous publish/subscribe IPC system
- DriverManager – Manages all hardware and virtualized drivers
- SysLog – In-memory persistent logging system
- Shell – Interactive command-line interface

Kernel Architecture

The Kernel is the central coordinator of GrooveOS. It is intentionally minimal, handling only functionality that must be trusted and global.

Responsibilities

- Boot sequencing
- Subsystem initialization
- System-wide resource ownership
- Graceful shutdown coordination
- Main scheduling loop ownership

Boot Sequence

1. Allocate global resources
2. Initialize SysLog
3. Initialize MessageBus
4. Initialize DriverManager
5. Initialize TaskManager
6. Register and start system tasks (Shell, drivers, etc.)

Kernel APIs

- taskManager()
- driverManager()
- syslog()
- messageBus()
- shutdown()

Lifecycle States

- BOOTING
- RUNNING
- SHUTDOWN_REQUESTED
- STOPPED

Task Manager Architecture

The Task Manager implements a cooperative (non-preemptive) scheduler, chosen for determinism and low jitter required in real-time audio environments.

Task Model

A task is a struct or object with:

- id
- name
- update() (invoked each scheduler cycle)

Why Cooperative Scheduling?

Advantages for embedded audio:

- Zero context-switch overhead
- Deterministic execution order
- Simpler task model
- Tasks can be tuned for exact cycle times

Scheduling Cycle

1. Iterate tasks in strict registration order
2. Call task.update()
3. Task yields implicitly by returning
4. Move to next task

Task States

- READY – Eligible for execution
- RUNNING – Inside update()
- BLOCKED – Waiting for event/message

MessageBus IPC Architecture

The MessageBus is the central communications backbone of GrooveOS.

Message Structure

A message contains:

- topic (string)
- payload (string or structured object)

Publish/Subscribe Flow

1. A component publishes a message
2. MessageBus queues it
3. All subscribers to topic receive the message

Benefits of MessageBus

- Zero direct coupling between tasks
- Drivers can emit system events without explicit knowledge of listeners
- Shell can manually inject system messages
- SysLog can passively listen to all events

Typical Topics

- "log"
- "midi.in"

- "engine.event"
- "ui.button"

DriverManager Architecture

The DriverManager controls all hardware and virtualized drivers.

Responsibilities

- Driver registration
- Initialization
- Shutdown
- Metadata exposure
- Integration with MessageBus

Driver Interface

Every driver exposes:

- name()
- init()
- tick()
- shutdown()

Examples of Drivers

- ConsoleDriver
- MIDI driver
- Audio codec driver
- GPIO input driver
- Virtual simulation drivers

SysLog Architecture

SysLog provides persistent, in-memory logging for debugging and diagnostics.

Features

- Append-only event logging
- Timestamped entries
- Accessible via Shell
- Subscribers can listen to "log" topic
- Lightweight circular buffer structure

Shell (CLI) Architecture

The Shell is a REPL-style interface built as a task.

- Accept commands
- Tokenize input
- Dispatch commands
- Display system state
- Publish IPC messages
- Trigger shutdown

Built-In Commands

- help
- ps
- drivers
- send <topic> <msg>
- log
- shutdown

Command Processing Pipeline

1. Print prompt
2. Read input
3. Tokenize string
4. Match command
5. Execute appropriate code path

Future Extensions

Real-Time Audio Engine

- Sample-accurate event timing
- Polyphonic voice engine
- DSP integration with tasks

Filesystem Layer

- Read/write support
- Config file handling
- Patch storage

Network Support

- UDP control
- OSC messages
- Remote debugging shell