Documentation – `20f2946`

# blog_os

14.11.2025

ThePerkinrex

# Contents

**1. Introduction** **1**
1.1. Functionality . . . . . 1
1.2. Building . . . . . 1
1.3. Running . . . . . 2

**2. Packages and crates** **2**
2.1. `kernel`: The Kernel . . . . . 2
2.2. `kernel-libs`: Libraries used by the kernel . . . . . 2
2.3. `qemu-common`: Utilities for interfacing with QEMU and the runner . . . . . 3
2.4. `runner`: Cargo runner . . . . . 3
2.5. `userspace`: Anything userspace . . . . . 4

**3. Kernel startup** **4**

**4. Simple I/O** **4**

**5. Memory** **4**

**6. Multitasking** **4**

**7. Processes** **4**

**8. Interrupts & Syscalls** **4**

**9. Backtrace, unwinding, & DWARF** **4**

**10. The VFS & FS API** **4**

**11. Userspace API & programs** **4**

# 1. Introduction

This is an OS that initially was based on Philipp Oppermann's `blog_os`[1] and was later expanded through different sources, mainly from OSDev Wiki[2].

## 1.1. Functionality

The kernel currently has the following features:

- VGA framebuffer printing
- Serial printing
- Memory paging
- Cooperative kernel multitasking with stack switching
- Userspace process execution in ring3, with ELF loading
- Userspace calls into the kernel, with syscalls (`int 0x80`), and process switching on those syscalls.
- Process & task exiting.

The current WIP features are:

- VFS and user FS API
- Device buses & PCI
- Driver API

Future expected features are:

- StdIO for processes, that could be redirected to different outputs (serial terminals, files...)
- Preemptive task switching
- Advanced task scheduler
- RAM disk support
- Devices on FS tree
- Simple shell & utilities

## 1.2. Building

To build this OS, the `cargo-make` system is used, so to get a complete OS image, just one command is needed: `cargo make build` at the root of the project. The runner executable will print where the image is located, which will depend on the build profile.

- For debug builds, use `cargo make build`.
- For release builds, use `cargo make -p release build`

Other dev utilities are provided by the `cargo-make` system:

- `cargo make format`: Apply `cargo fmt` to the whole project

[1] https://os.phil-opp.com/
[2] https://wiki.osdev.org/Expanded_Main_Page

- `cargo make docs`: Apply `cargo doc` to the whole project. Each index for each crate/workspace is printed.
- `cargo make pdf`: (*`typst` executable is needed*) builds this pdf.

### 1.3. Running

To run this OS, `cargo-make` can also be used. QEMU for x86-64 needs to be installed.

- `cargo make run` will start the OS, with a VGA display, and serial output in the terminal.
- `cargo make -p no_display run` will start the OS, without a VGA display, and serial output in the terminal.
- `cargo make -p gdb run` will start the OS, without a VGA display, and serial output in the terminal, attaching the `gdb` debugger to it and stopping immediately, *before the bootloader runs and load the kernel in memory.*

## 2. Packages and crates

### 2.1. `kernel`: The Kernel

### 2.2. `kernel-libs`: Libraries used by the kernel

This workspace contains crates that will be used by the kernel, but also some that can also be used by external drivers. This split is useful for code that doesn't necessarily depend on other kernel code, allowing the use of the standard testing framework, and better compile times.

Here there are the following crates:

- VFS:
  - `blog_os_vfs`: Contains kernel-specific VFS code.
  - `blog_os_vfs_api`: VFS API, for FS drivers.
- Device:
- `blog_os-device`: Kernel specific device code, for providing support for the API.
- `blog_os-device-api`: Device API, for drivers (bus, bus device…)
- `blog_os-pci`: The PCI bus driver
- `kernel_utils`: Common utilities used by the kernel, and that can be reused by drivers and other code.
- `api-utils`: Common types used by APIs (common `cglue` code for FFI).

## 2.3. `qemu-common`: Utilities for interfacing with QEMU and the runner

This is a common crate shared by the kernel testing framework and the runner, allowing for some communication between them through QEMU-specific APIs.

## 2.4. `runner`: Cargo runner

This is a utility that is used as a cargo runner and a separate binary. It builds the OS image, bundling together the ELF and the bootloader (either for BIOS or UEFI boot). It also supports starting up `gdb`, with config setup for the kernel, and detecting when testing is going on, for better exit codes.

2.5. `userspace`: Anything userspace

3. Kernel startup

4. Simple I/O

5. Memory

6. Multitasking

7. Processes

8. Interrupts & Syscalls

9. Backtrace, unwinding, & DWARF

10. The VFS & FS API

11. Userspace API & programs