AstraKernel Documentation

A Minimal Kernel for QEMU's VersatilePB (ARM926EJ-S) Platform

Written in Modern C and ARM Assembly

Version 0.1

By Chris Dedman

This page is intentionally left blank.

Contents

Preface

This documentation serves as a comprehensive guide to the AstraKernel project, a minimal operating system kernel written in modern C and ARM assembly. Designed to run on QEMU's VersatilePB (ARM926EJ-S) emulated platform, AstraKernel is intended to provide a clear and approachable introduction to the fundamental concepts of operating system design and development. This project also reflects my personal journey in learning about kernel development and systems programming.

This project was developed with a focus on clarity, simplicity, and educational value. Rather than attempting to recreate the complexity of established operating systems, AstraKernel's goal is to strip away unnecessary abstractions and present a clean, understandable codebase for anyone interested in the "bare metal" foundations of computing.

Through hands-on implementation of kernel bootstrapping, direct hardware communication, and basic user interaction, AstraKernel demonstrates how fundamental OS components come together. The project showcases how modern C best practices can be utilized in a systems programming context to create code that is maintainable, portable, and robust, while still being accessible to those new to kernel development. The design of this kernel emphasizes modularity and extensibility, allowing developers to easily add new features or modify existing ones. This makes it ideal for educational purposes, as it provides a clear structure that can be followed and built upon.

It is my hope that AstraKernel will not only serve as a foundation for those wishing to understand kernel development, but also inspire curiosity and confidence in exploring lower-level aspects of computer systems.

INFO:

> This documentation is a work in progress and may be updated as the project evolves. I welcome contributions, feedback, and suggestions for improvement. You can find the source code on GitHub: https://github.com/sandbox-science/AstraKernel

Resources

To guide my learning and support development of AstraKernel, I am using the following resources, which are particularly valuable for foundational and practical understanding of OS design:

- **Operating Systems: Three Easy Pieces** by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
- **The Little Book About OS Development** by Erik Helin and Adam Renberg

Disclaimer

AstraKernel is currently in its early stages of development and is not intended for production use. It is primarily an educational and experimental project. The code is provided as-is, without any warranties or guarantees of any kind. Use it at your own risk.

Chapter 1

Introduction

I. Getting Started

AstraKernel begins its life in a small bootstrap routine, written in ARM assembly, that prepares the processor's state before passing control to the main C kernel. This bootstrap code is responsible for setting up the stack pointer, clearing the uninitialized data section (`.bss`), and ensuring a clean environment for the kernel's entry point.

Below is the initial assembly code that executes at startup:

```
    .section .text
    .global _start

_start:
    // Set up the stack pointer
    LDR sp, =_estack
    BIC sp, sp, #7
    // Zero the .bss section
    LDR R0, =__bss_start   // Start address (symbol from linker script)
    LDR R1, =__bss_end     // End address (symbol from linker script)
    MOV R2, #0             // init zero-value for BSS clearing

zero_bss:
    // Check if we are done zeroing the BSS
    CMP R0, R1             // Compare current address to end
    BGE bss_done           // If done, skip zeroing
    STR R2, [R0], #4       // Store zero at [r0], increment r0 by 4
    B zero_bss

bss_done:
    // Call kernel_main function
    BL kernel_main

hang:
    // Halt if kernel_main returns (should not happen)
    B hang    // Infinite loop
```

Listing 1.1: Initial bootstrap code for AstraKernel.

This startup sequence is the essential first step for any kernel, ensuring the CPU is properly initialized and memory is in a known state before higher-level code takes over. Once these preparations are complete, the `kernel_main` function from `kernel/kernel.c` is called, marking the transition from low-level assembly to the C code that forms the core of AstraKernel.

A. Prerequisites

Before you can build and run AstraKernel, please ensure you have the following tools installed on your system:

- **ARM Cross-Compiler:** A cross-compiler targeting ARM is required to build the kernel. It is recommended to use `arm-none-eabi-gcc`, `arm-none-eabi-ld`, and `arm-none-eabi-objcopy` for ARM926EJ-S, which is the target architecture for AstraKernel.
  - Example installation: `arm-none-eabi-xxx` (available via package managers such as `brew`, `apt`, or direct download from ARM's website).
- **QEMU Emulator:** QEMU is used to emulate the ARM VersatilePB (ARM926EJ-S) platform for kernel development and testing.
  - Ensure your QEMU installation supports the `versatilepb` machine.
  - Example installation: `qemu-system-arm` via qemu https://www.qemu.org/download/.
- **Build Tools:** Standard build tools such as `make` are required to compile the kernel.
  - Example installation: `make` (available via package managers such as `brew`, `apt`, or direct download https://www.gnu.org/software/make/#download).

For best results, ensure all tools are up-to-date. Consult the official documentation of each tool for installation instructions on your operating system.

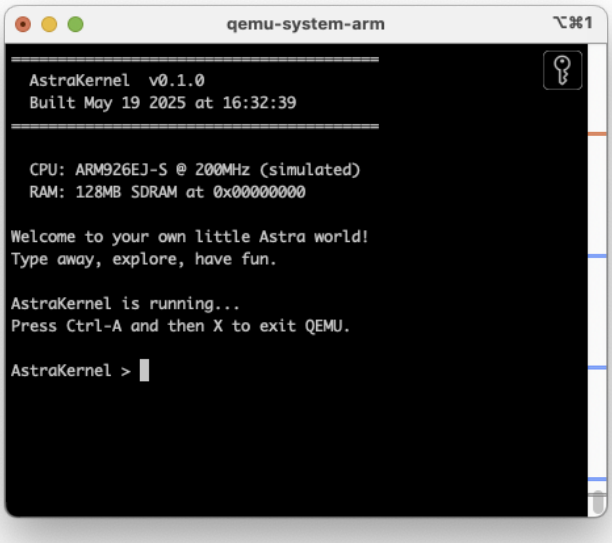

Figure 1.1: AstraKernel booted in QEMU.