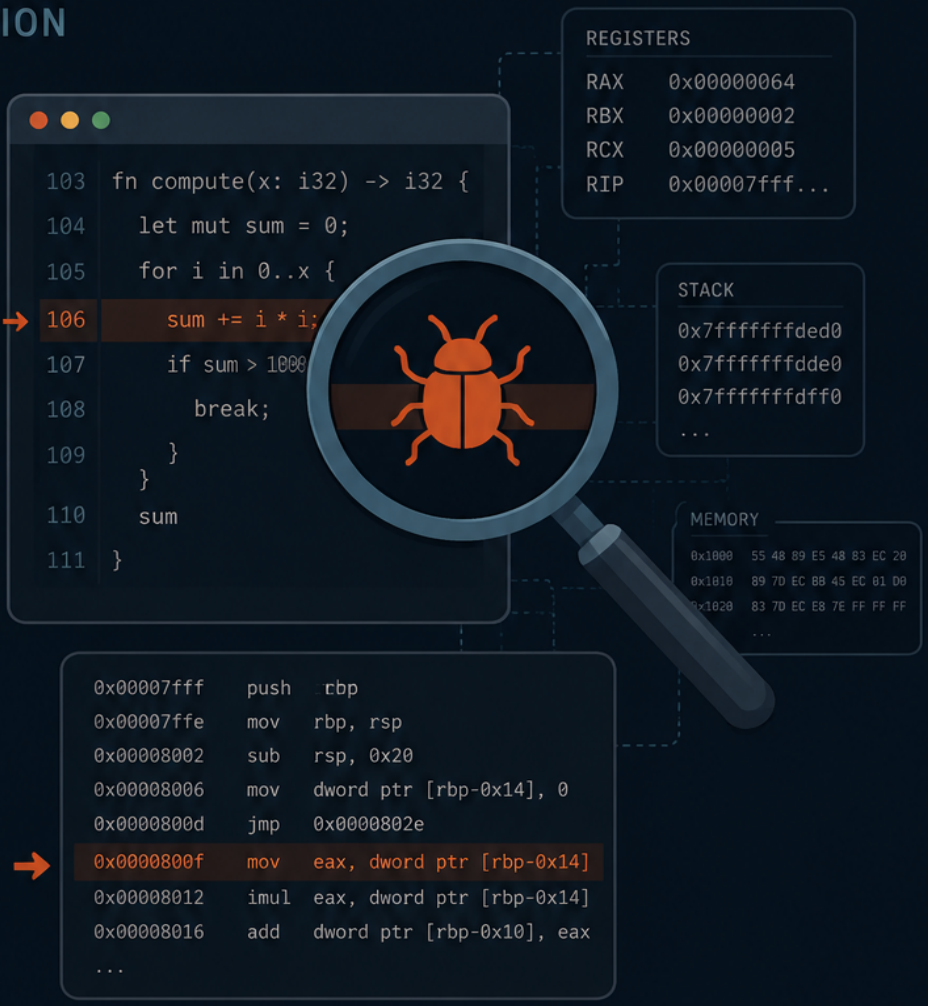
INSIDE THE DEBUGGER

BUILDING A SOURCE-LEVEL DEBUGGER IN RUST

A DEEP DIVE INTO PROCESSES, MEMORY, BREAKPOINTS, AND DWARF DEBUG INFORMATION

- PROCESSES
- MEMORY
- BREAKPOINTS
- SINGLE STEPPING
- REGISTERS
- STACK FRAMES
- DWARF DEBUG INFO
- SOURCE-LEVEL DEBUGGING
- WATCHPOINTS
- MULTITHREADED DEBUGGING
- REMOTE DEBUGGING



STEFAN REISIG

Inside the Debugger: Building a Source-Level Debugger in Rust

A Deep Dive into Processes, Memory, Breakpoints, and DWARF Debug Information

Steve T. Publications

This book is available at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust

This version was published on 2026-07-13

Contents

A Deep Dive into Processes, Memory, Breakpoints, and DWARF Debug Information

A debugger is not magic. It is a careful orchestration of operating system primitives, CPU features, and compiler-generated metadata that together let you pause a running program, inspect its state, and control its execution one instruction at a time. This book teaches you how to build a fully functional source-level debugger from scratch in Rust. Starting from the fundamentals of processes, virtual memory, and executable formats, we progressively implement every feature a real debugger needs: process attachment, software and hardware breakpoints, single stepping, register inspection, memory read and write, symbol resolution, stack unwinding, call frame inspection, DWARF debug information parsing, source-level debugging, expression evaluation, watchpoints, multithreaded debugging, shared library support, signal handling, and remote debugging concepts. Every concept is accompanied by complete, production-quality Rust code that builds upon previous chapters into a cohesive debugger. By the end of this book, you will have built a capable debugger entirely in Rust and gained deep insight into how programs execute, how operating systems manage processes, and how compilers encode the information needed to map machine code back to source code.

Introduction

Type `gdb ./my_program`, set a breakpoint on line 42, type `run`, and watch the program halt exactly where you told it to. Inspect variables, step through code one line at a time, watch the call stack unfold, modify memory in real time. It feels almost like magic. But underneath that polished interface lies a machinery of operating system calls, CPU features, and compiler-generated metadata that is both elegant and deeply instructive.

This book is about peeling back that interface and building the machinery yourself.

Why Build a Debugger?

You might never need to write your own debugger in production. GDB, LLDB, and Visual Studio are mature, well-tested tools with decades of engineering behind them. But building a debugger teaches you things no other project can: how processes actually execute, how the operating system mediates between programs and hardware, how compilers encode debugging information, and how the CPU itself supports introspection. These are foundational systems concepts that inform everything from security research to compiler design to performance profiling.

A debugger sits at the intersection of every layer of a computing system. It talks to the kernel through tracing APIs. It reads executable formats and debug information embedded by compilers. It manipulates CPU registers and memory pages. It interprets the semantics of programming languages through type information. Building one forces you to understand all of these layers and how they interact.

What This Book Covers

We build a complete source-level debugger in Rust, starting from nothing. The debugger we create can:

- Launch and attach to processes
- Set software and hardware breakpoints
- Step through code, one instruction or one source line at a time
- Read and write CPU registers
- Read and write process memory
- Resolve symbols and demangle names
- Unwind the call stack and display backtraces with source locations
- Inspect local variables by evaluating DWARF location expressions
- Handle signals, crashes, and multithreaded programs
- Debug programs with dynamically loaded shared libraries

The implementation targets Linux using the `ptrace` system call, with discussion of portability to macOS and Windows throughout. The code is written for x86_64, with notes about ARM64 where the concepts differ.

Prerequisites

You should be comfortable writing Rust. You do not need prior experience with systems programming, operating systems internals, or debugger implementation. Every low-level concept is explained from first principles: what a process is, how virtual memory works, what CPU registers do, how executable files are structured, and how debug information encodes the mapping between source code and machine instructions.

How to Read This Book

The book is organized to build understanding progressively. The first few chapters establish the systems-level foundation: processes, memory, CPU architecture, and executable formats. Then we dive into the debugging API (`ptrace`) and begin implementing our debugger. Each subsequent chapter adds a new feature, building on the code from previous chapters. By the final chapter, you will have a complete, working debugger.

Every chapter contains substantial Rust code. You can follow along by typing the code yourself, or you can study it as documentation of how each feature works. The code is designed to be readable and well-structured, not minimal or clever. We prioritize clarity over brevity throughout.

What About the Code?

The debugger we build uses several crates from the Rust ecosystem:

- **nix**: Safe bindings to Unix system calls including `ptrace`, `fork`, `waitpid`, and signal handling
- **libc**: Low-level C type definitions like `user_regs_struct` and `siginfo_t`
- **gimli** and **addr2line**: DWARF debug information parsing and address-to-line lookup
- **object**: ELF file format parsing
- **rustc-demangle**: Rust symbol name demangling
- **nom**: Parser combinators for command-line input parsing

All code targets stable Rust. We use `unsafe` only where absolutely necessary (primarily for FFI boundaries with the kernel) and document the safety invariants carefully. The goal is production-quality code that demonstrates how to write safe systems software in Rust.

Let us begin.

Chapter 1: What Is a Debugger, Really?

The Debugger as an Operating System Feature

A debugger is not a standalone application that happens to talk to other programs. It is, fundamentally, an operating system feature exposed through a user-space interface. When you set a breakpoint in GDB and the target program halts at the right place, it is the kernel itself that intercepts the execution, stops the process, and notifies your debugger. The debugger is merely the user-space half of a cooperation between two processes mediated by the kernel.

This cooperation has a name: on Linux, it is called `ptrace`, short for "process trace." The `ptrace` system call provides one process (the tracer, or debugger) with the ability to observe and control another process (the tracee, or debuggee). Through `ptrace`, the debugger can read and write the tracee's memory, inspect and modify its CPU registers, pause and resume its execution, and intercept every signal the kernel delivers to it.

But `ptrace` is not the whole story. A debugger also needs to understand executable file formats (to find symbol tables and debug information), CPU architecture (to interpret register values and instruction semantics), compiler output (to map machine addresses back to source lines), and programming language semantics (to evaluate expressions in the target's context). `ptrace` gives you the raw power; everything else is interpretation.

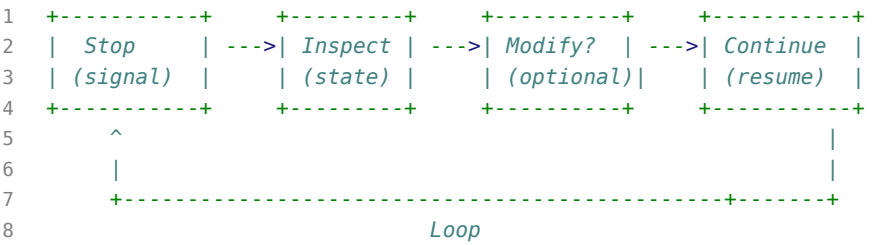
The Debugging Loop

Every debugger, from the simplest trace program to the most sophisticated IDE-integrated tool, implements the same fundamental loop:

1. **Stop**: The target process is paused at some point in its execution. This happens because of a breakpoint hit, a single-step completion, a signal delivery, or an explicit user request.

2. **Inspect**: The debugger reads the target's state. This includes CPU registers (especially the instruction pointer and stack pointer), memory contents, and derived information like the current function name and source line.
3. **Modify** (optional): The user may change register values, write to memory, or alter program state in some way.
4. **Continue**: The debugger resumes the target's execution, either by continuing normally, stepping one instruction, or stepping over a function call.

This loop repeats until the target process exits or the user detaches the debugger. The entire art of debugger implementation lies in making each step of this loop reliable, fast, and informative.



The "stop" event is the heartbeat of the debugger. On Linux, it arrives as a notification from `waitpid()`, telling the debugger that a traced child has entered a stopped state. The debugger must then determine *why* the process stopped: was it a breakpoint? A single-step completion? An unhandled signal like SIGSEGV? A fork or exec event? Each reason requires different handling, and getting this dispatch logic correct is one of the trickiest parts of debugger implementation.

What We Will Build

Our debugger, which we will call **rustdbg**, is designed to be a complete but focused implementation. It covers the essential features of a real debugger without attempting to match GDB or LLDB in every dimension. Here is what it will do:

Process management: Launch new processes under debug control, attach to running processes, and detach cleanly.

Breakpoints: Set software breakpoints by modifying code bytes (using the `int3` instruction on x86_64) and hardware breakpoints using CPU debug registers. Breakpoints can be set at specific addresses or resolved from source file and line number.

Stepping: Step one instruction at a time using the CPU's single-step mechanism. Step over function calls by temporarily setting a breakpoint at the return address. Step by source line using DWARF line table information.

Register operations: Read all general-purpose registers, the instruction pointer, stack pointer, and flags register. Write registers to modify program state. Display registers in hexadecimal with named bit fields for RFLAGS.

Memory operations: Read and write process memory in various formats (hex dump, bytes, words, formatted types). Handle page boundaries and invalid addresses gracefully.

Symbol resolution: Look up function names by address using the ELF symbol table. Demangle C++ (`__cxa_demangle`) and Rust (`rustc-demangle` crate) symbols for readable output.

Stack unwinding: Walk the call stack using both frame-pointer-based unwinding (following the RBP chain) and DWARF Call Frame Information (CFI) based unwinding. Display a backtrace with function names, source file locations, and argument values where available.

Source-level debugging: Map instruction addresses to source file paths and line numbers using the DWARF `.debug_line` section. Display the current source code context with the active line highlighted.

Variable inspection: Evaluate DWARF location expressions to find local variables within a stack frame. Read variable values from memory or registers, interpreting them according to their DWARF type information.

Watchpoints: Set hardware watchpoints using debug registers to break on memory reads, writes, or modifications at specific addresses.

Signal handling: Intercept and report signals delivered to the tracee. Distinguish between debugger-induced traps (breakpoints, single-step) and real signals (SIGSEGV for crashes, SIGABRT for aborts). Generate crash dumps with register state and stack traces.

Multithreaded debugging: Debug programs with multiple threads using `PTRACE_SEIZE` and group-stop notifications. Track all threads in a process.

Command interface: A line-based command interpreter supporting commands like `break`, `continue`, `step`, `next`, `backtrace`, `print`, `set`, `info registers`, `info breakpoints`, `finish`, `run`, `attach`, `detach`, and `quit`.

Why Rust for a Debugger?

Rust is an excellent choice for debugger implementation for several reasons:

Memory safety: A debugger manipulates raw memory addresses, reads binary data from executable files, and parses complex data structures like DWARF debug information. In C or C++, a single off-by-one error in a parser can lead to buffer overflows, use-after-free bugs, or arbitrary code execution. Rust's ownership system and borrow checker prevent entire classes of these bugs at compile time.

FFI support: Debuggers must call operating system APIs like `ptrace`, `fork`, and `waitpid`. Rust provides excellent foreign function interface support through the `libc` crate and the higher-level `nix` crate, giving us safe, idiomatic wrappers around Unix system calls.

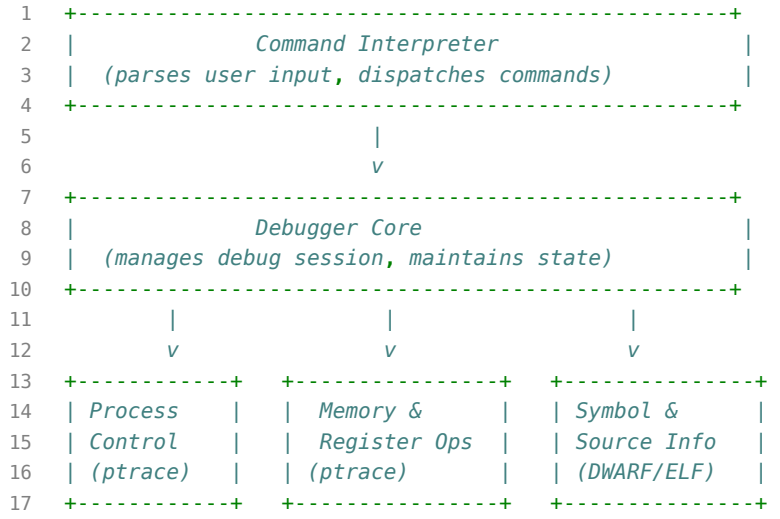
Ecosystem: The Rust ecosystem has mature libraries for every layer of our debugger. The `gimli` crate provides a fast, zero-copy DWARF parser. The `addr2line` crate wraps gimli with a convenient API for address-to-line lookups. The `object` crate parses ELF, Mach-O, and PE executable formats. The `nom` crate provides parser combinators for command-line input. All of these are production-quality libraries used in real tools.

Error handling: Debugging is inherently error-prone. Processes crash, memory reads fail, debug information is missing or malformed. Rust's `Result` type and the `?` operator make it natural to propagate errors through the call stack with context, producing clear diagnostic messages instead of cryptic segfaults.

Existing debugger ecosystem: Several projects have already demonstrated that Rust is suitable for debugger work. The `gdbstub` crate provides a GDB remote protocol server implementation. The `wasmtime-debug` project implements debugging support for the Wasmtime WebAssembly runtime. LLDB's expression evaluator is implemented in C++, but the Rust community has built several debugging front-ends and tools that demonstrate the language's fitness for this domain.

The Architecture of Our Debugger

Our debugger follows a modular architecture with clear separation of concerns:



The **command interpreter** reads user input from the terminal, parses it into commands, and dispatches them to the debugger core. It handles the interactive loop: prompt, read, parse, execute, repeat.

The **debugger core** maintains the state of the debug session: which process is being debugged, what breakpoints are set, the current stop reason, cached register values, loaded symbol tables, and parsed DWARF information. It implements the debugging loop described above.

The **process control** module wraps `ptrace` operations: launching processes, attaching to running ones, continuing execution, single-stepping, reading and writing registers and memory.

The **memory and register operations** module handles the low-level details of ptrace-based memory access (word-at-a-time reads via `PTRACE_PEEKDATA`, handling endianness) and register manipulation (`PTRACE_GETREGS/PTRACE_SETREGS`).

The **symbol and source information** module parses executable files, extracts symbol tables, loads DWARF debug information, maps addresses to source locations, and evaluates location expressions for variable inspection.

This modular design allows us to implement each component independently and test it in isolation. It also makes the code easier to understand: when you want to know how breakpoints work, you look at the process control module. When you want to understand source-level debugging, you look at the symbol and source information module.

A Brief History of Debuggers

The first software debugger was DDT (Digital Differential Analyzer) for the PDP-1 in 1959. It could display memory contents, set breakpoints (by replacing instructions with trap instructions, just as we will do), and single-step through code. The fundamental ideas have not changed in sixty-five years.

Unix introduced `ptrace` in Version 6 (1975), originally designed for the DBX debugger. System V and BSD extended it. Linux inherited the concept from SVr4 and has since added significant extensions like `PTRACE_SEIZE`, event notifications, and register set access.

GDB, released in 1986 by Richard Stallman, became the de facto standard Unix debugger. It introduced many features we will implement: command-line interface with history, source-level debugging using DWARF (originally STABS), multi-architecture support, and extensibility through extensions.

LLDB, developed as part of the LLVM project starting in 2009, modernized the approach with a C++ object-oriented architecture, native Python scripting, and tight integration with Clang's expression evaluator. It is now the default debugger on macOS and increasingly used on Linux.

Our debugger stands on these shoulders. We implement the same core ideas that DDT pioneered, using the same kernel interface that DBX started, following the same architectural patterns that GDB established. The only thing new is the language we write it in.

In the next chapter, we build the systems-level foundation: understanding processes, virtual memory, CPU registers, and how programs are loaded into memory. These concepts are essential for everything that follows.

Chapter 2: The Building Blocks: Processes, Memory, and the CPU

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Processes and the Process Table

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Virtual Memory and Address Spaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The CPU Registers and Program Counter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

How Programs Start: Loading and Entry Points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

System Calls and the Kernel Boundary

Putting It All Together: A Process in Motion

Chapter 3: The ptrace System Call: Your Gateway to Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

ptrace Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

PTRACE_TRACEME: Voluntary Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

PTRACE_ATTACH and PTRACE_SEIZE: Attaching to Running Processes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Process Stop States and Wait

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Distinguishing Stop Reasons

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Reading and Writing Registers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Extended Registers with PTRACE_GETREGSET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Reading and Writing Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The /proc/[pid]/mem Alternative

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Single Stepping and Instruction Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

ptrace Security and Permissions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Complete ptrace Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 4: Executable Formats and DWARF Debug Information

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The ELF File Format

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Program Headers (Segments)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Section Headers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Symbol Tables and Relocations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Symbol Demangling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

DWARF Debug Information Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

DIEs: Debugging Information Entries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Abbreviations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Line Number Program

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Variable Locations and Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Reading DWARF in Rust with gimli

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

DWARF Version Differences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Putting It Together: From Address to Source Line

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 5: Project Setup and the Debugger Shell

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Project Structure and Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Error Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Debugger Core Trait and Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Command Parsing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Process Fork and Trace

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The First Debug Session

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The First Debug Session

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 6: Breakpoints: Stopping at Will

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Software Breakpoints and INT3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Breakpoint Lifecycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Breakpoint Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Handling Breakpoint Hits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Multiple Breakpoints and Conditional Execution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Testing Breakpoints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Common Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Troubleshooting Breakpoints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 7: Single Stepping and Register Inspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Single Stepping with PTRACE_SINGLESTEP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Single-Step Caveats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The x86_64 Register File

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Reading Registers After a Stop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Writing Registers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Step Over Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Instruction Length: Using iced-x86 for Reliable Decoding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Beyond Length: Full Disassembly Display

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Register Display Formatting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 8: Memory Inspection and Modification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Reading Process Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Memory Display Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Handling Page Faults and Invalid Addresses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The /proc/[pid]/mem Alternative

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Memory Region Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Writing Memory Safely

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 9: Symbol Resolution and Source-Level Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Symbol Lookup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Demangling C++ and Rust Symbols

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Source File Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Using addr2line::Loader for Clean Memory Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

When to Use Low-Level gimli Directly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Displaying Source Code with Current Line Highlighting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Step Over at the Source Level

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Frame Pointer Omission and Its Consequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 10: Stack Unwinding and Call Frames

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Call Stack Explained

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Frame Pointer-Based Unwinding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

DWARF Call Frame Information

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Full CFI Implementation with gimli

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Displaying the Backtrace

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Local Variable Inspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 11: Watchpoints, Hardware Breakpoints, and Advanced Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Hardware Debug Registers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Setting Hardware Breakpoints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The PTRACE_GETREGSET and NT_X86_DEBUGREGS Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Limitations and Fallback Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Watchpoint Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Tracepoints and Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 12: Multithreaded Debugging and Shared Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Thread Groups and ptrace

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

PTRACE_SEIZE and Group Stop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Debugging Existing Processes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Shared Library Loading Events

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Symbol Resolution Across Shared Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Thread-Specific Data and Stack Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 13: Expression Evaluation and Variable Modification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Expression Evaluation Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Building a Simple Expression Parser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Type Resolution from DWARF

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Set Command

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Limitations and Pragmatic Choices

Chapter 14: Signals, Exceptions, and Error Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Signal Delivery During Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Distinguishing Breakpoint Signals from Real SIGTRAPs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Signal Masking and Filtering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Handling Common Signals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Crash Dump Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Debugger Error Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Chapter 15: Architecture, Testing, and Going Further

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Complete Debugger Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Testing a Debugger

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Performance Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Portability: macOS ptrace and Windows Debug Active Process API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Remote Debugging Concepts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Extending Your Debugger

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Conclusion: What We Built and What It Teaches Us

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Layers of Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

What Makes a Good Debugger?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

The Bigger Picture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Final Thoughts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Appendix A: Complete Project Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Directory Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Module Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Shared Types and Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Error Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Memory Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

ptrace API Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Building and Running

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Testing Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

Companion Repository

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/insidethedebuggerbuildingasource-leveldebuggerinrust.