Test Suite: Minimal Debugger (Minidbg)

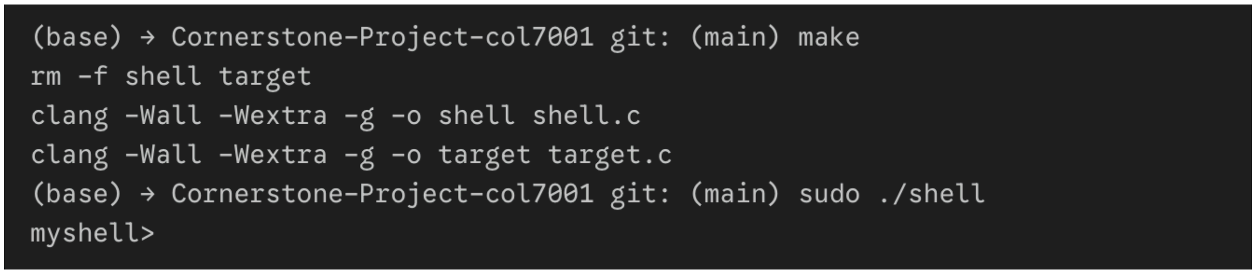
Student Name: Jaskaran Singh
Date: December 17, 2025
System Environment: macOS (Unix-based operating system, Intel)

Overview

This document demonstrates the functional verification of the minimal debugger (`minidbg`) integrated into the custom shell. Each test case targets a specific functional requirement from the Lab 2 specifications, including the optional memory inspection extension.

Test Environment Setup

- **Compilation command used:** `make clean && make`
- **Execution command used:** `sudo ./shell` (Root privileges required for Mach API access)

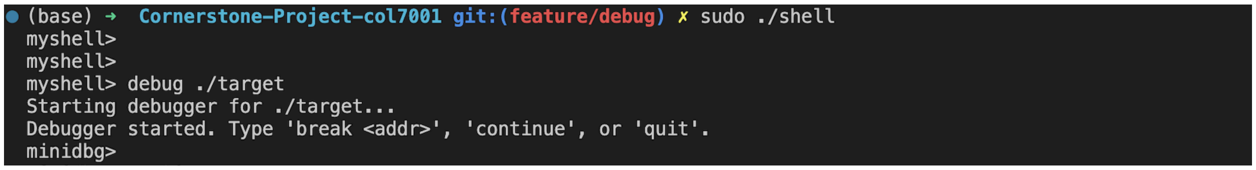


```
(base) → Cornerstone-Project-col7001 git: (main) make
rm -f shell target
clang -Wall -Wextra -g -o shell shell.c
clang -Wall -Wextra -g -o target target.c
(base) → Cornerstone-Project-col7001 git: (main) sudo ./shell
myshell>
```

1. Target Loading & Execution

Objective: To verify the debugger can correctly fork, trace, and execute a child process.

- **Command:** `debug ./target`
- **Expected Result:** The shell launches the target, which prints its start message. The debugger then pauses execution immediately (due to `ptrace` attach) and presents the `minidbg>` prompt.



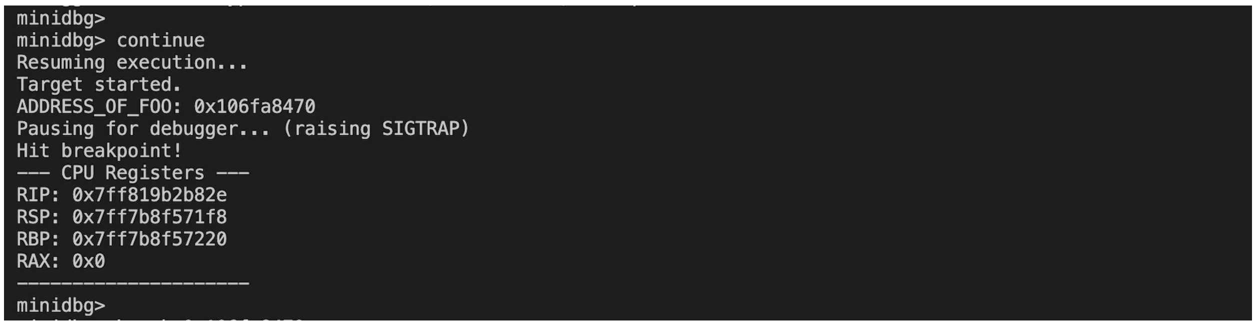
```
● (base) →  Cornerstone-Project-col7001 git:(feature/debug) ✗ sudo ./shell
  myshell>
  myshell>
  myshell> debug ./target
  Starting debugger for ./target...
  Debugger started. Type 'break <addr>', 'continue', or 'quit'.
  minidbg>
```

2. Address Space Layout Randomization (ASLR) Check

Objective: To confirm the user can determine the dynamic load address of functions at runtime.

- **Command:** `continue` (Run once to see the address)
- **Expected Result:** The `target` program runs, prints the specific memory address of function `foo` (e.g., `0x10...`), and then pauses itself by raising `SIGTRAP`.

Screenshot/Output:



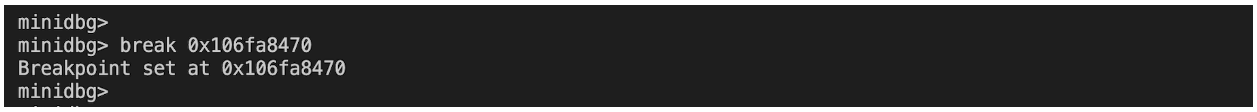
```
minidbg>
minidbg> continue
Resuming execution...
Target started.
ADDRESS_OF_FOO: 0x106fa8470
Pausing for debugger... (raising SIGTRAP)
Hit breakpoint!
--- CPU Registers ---
RIP: 0x7ff819b2b82e
RSP: 0x7ff7b8f571f8
RBP: 0x7ff7b8f57220
RAX: 0x0
---------------------
minidbg>
```

3. Setting Software Breakpoints

Objective: To verify the debugger can write trap instructions (`0xCC`) into the Read-Only text segment of the child process.

- **Command:** `break <ADDRESS_FROM_STEP_2>` (e.g., `break 0x10f28a470`)
- **Expected Result:** The debugger reports "Breakpoint set at..." without any "Invalid Argument" or permission errors, confirming successful use of `mach_vm_protect`.

Screenshot/Output:



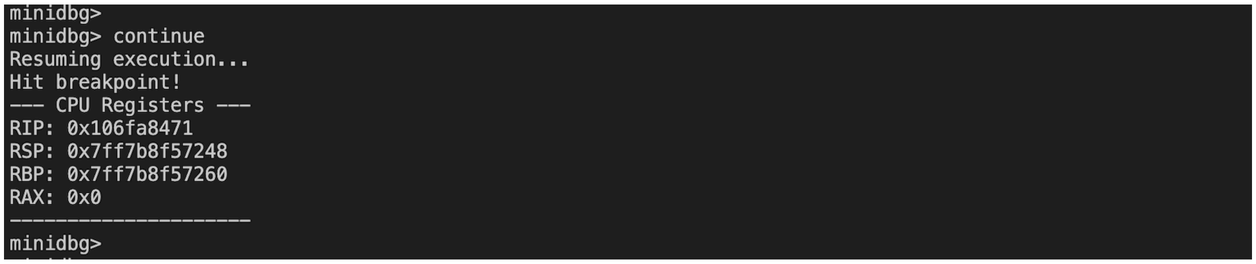
```
minidbg>
minidbg> break 0x106fa8470
Breakpoint set at 0x106fa8470
minidbg>
```

4. Triggering Breakpoints

Objective: To verify that execution stops at the injected breakpoint.

- **Command:** `continue`
- **Expected Result:** The target program resumes, attempts to call `foo`, and is immediately stopped by the OS. The debugger catches the signal and prints "Hit breakpoint!".

Screenshot/Output:



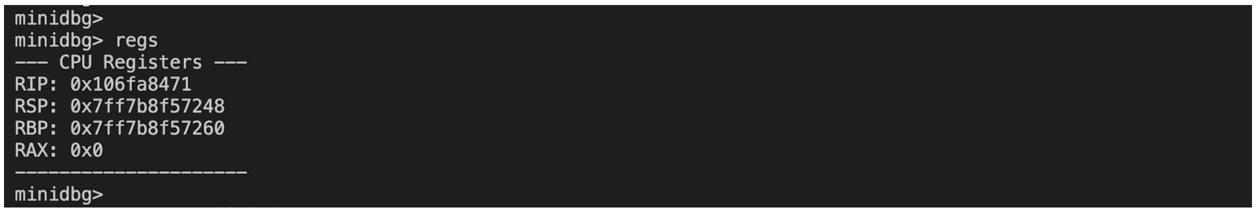
```
minidbg>
minidbg> continue
Resuming execution...
Hit breakpoint!
--- CPU Registers ---
RIP: 0x106fa8471
RSP: 0x7ff7b8f57248
RBP: 0x7ff7b8f57260
RAX: 0x0
---------------------
minidbg>
```

5. Register Inspection (`regs`)

Objective: To verify the debugger can read the CPU state of the suspended child process.

- **Command:** `regs`
- **Expected Result:** The output displays valid hexadecimal values for key registers. Crucially, `RIP` (Instruction Pointer) should match the breakpoint address set in Step 3.

Screenshot/Output:



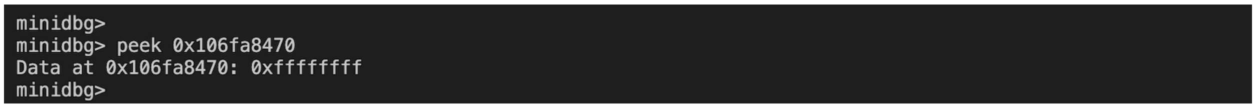
```
minidbg>
minidbg> regs
--- CPU Registers ---
RIP: 0x106fa8471
RSP: 0x7ff7b8f57248
RBP: 0x7ff7b8f57260
RAX: 0x0
---------------------
minidbg>
```

6. Memory Inspection (peek) - *Optional Extension*

Objective: To verify the debugger can read raw data from the child's memory address space.

- **Command:** `peek <ADDRESS_FROM_STEP_2>`
- **Expected Result:** The debugger prints the 4-byte hexadecimal word stored at that address.

Screenshot/Output:



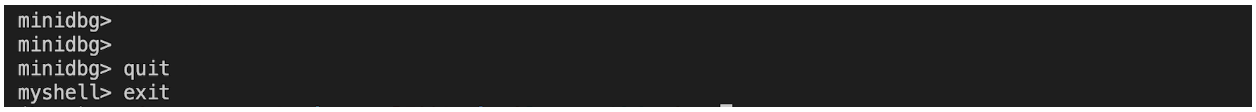
```
minidbg>
minidbg> peek 0x106fa8470
Data at 0x106fa8470: 0xffffffff
minidbg>
```

7. Graceful Termination

Objective: To verify the debugger can exit and kill the child process without leaving zombies or crashing the shell.

- **Command:** `quit`
- **Expected Result:** The debugger session ends, and control returns to the main `myshell>` prompt.

Screenshot/Output:



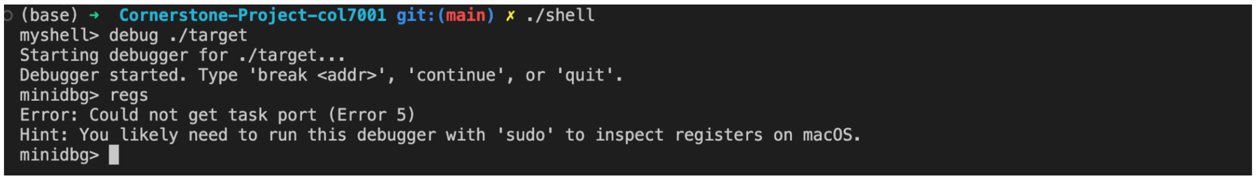
```
minidbg>
minidbg>
minidbg> quit
myshell> exit
```

8. Robustness & Error Handling

Objective: To verify that the debugger handles missing permissions (running without `sudo`).

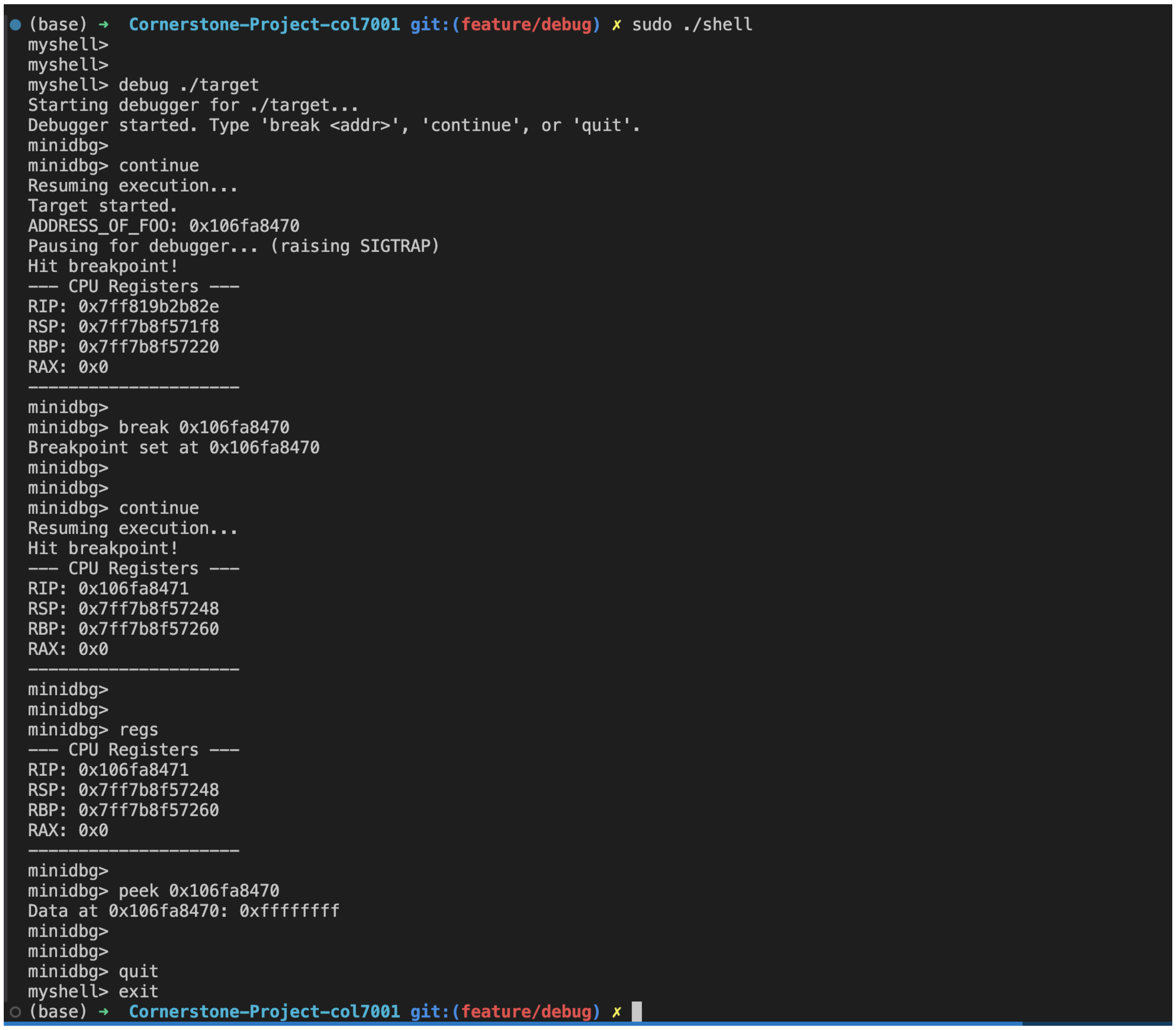
- **Command:** `regs` (Run in a session started *without* sudo)
- **Expected Result:** The debugger catches the kernel error and prints a descriptive message instead of crashing.

Screenshot/Output:



```
(base) ➜  Cornerstone-Project-col7001 git:(main) ✗ ./shell
myshell> debug ./target
Starting debugger for ./target...
Debugger started. Type 'break <addr>', 'continue', or 'quit'.
minidbg> regs
Error: Could not get task port (Error 5)
Hint: You likely need to run this debugger with 'sudo' to inspect registers on macOS.
minidbg> █
```

9. Complete Demo Session Output



```
(base) ➜  Cornerstone-Project-col7001 git:(feature/debug) ✗ sudo ./shell
myshell>
myshell>
myshell> debug ./target
Starting debugger for ./target...
Debugger started. Type 'break <addr>', 'continue', or 'quit'.
minidbg>
minidbg> continue
Resuming execution...
Target started.
ADDRESS_OF_FOO: 0x106fa8470
Pausing for debugger... (raising SIGTRAP)
Hit breakpoint!
--- CPU Registers ---
RIP: 0x7ff819b2b82e
RSP: 0x7ff7b8f571f8
RBP: 0x7ff7b8f57220
RAX: 0x0
---------------------
minidbg>
minidbg> break 0x106fa8470
Breakpoint set at 0x106fa8470
minidbg>
minidbg>
minidbg> continue
Resuming execution...
Hit breakpoint!
--- CPU Registers ---
RIP: 0x106fa8471
RSP: 0x7ff7b8f57248
RBP: 0x7ff7b8f57260
RAX: 0x0
---------------------
minidbg>
minidbg>
minidbg> regs
--- CPU Registers ---
RIP: 0x106fa8471
RSP: 0x7ff7b8f57248
RBP: 0x7ff7b8f57260
RAX: 0x0
---------------------
minidbg>
minidbg> peek 0x106fa8470
Data at 0x106fa8470: 0xffffffff
minidbg>
minidbg>
minidbg> quit
myshell> exit
(base) ➜  Cornerstone-Project-col7001 git:(feature/debug) ✗
```

<---------------------------------- END OF REPORT ---------------------------------->