Debugging the `slen` Function with GDB

May 22, 2025

1 Introduction

This document describes the `slen` function, a C function that calculates the length of a null-terminated string, and details the debugging process using GDB on an embedded target. The debugging session history is included to illustrate the steps taken to verify the function's correctness.

2 The `slen` Function

The `slen` function takes a string pointer and returns the number of characters until the null terminator (`\0`). Below is the source code:

```
#include <stdint.h>

const char * ptr = "informatyk mpabi 0\n";

int slen ( char *s ) {
    int n;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}

int main () {
    int x = slen (ptr);
    return 0;
}
```

The string `ptr` points to `"informatyk mpabi 0\n"`, and `slen` is called to compute its length, stored in `x`.

3 Debugging Strategy

The debugging process uses GDB with a remote target (port 3333) to verify the `slen` function. The strategy includes:

- **Setup**: Connect to the target, load the program, and set breakpoints.

- **Execution Control**: Step through instructions and continue to breakpoints to inspect `slen`'s execution.
- **Memory Inspection**: Verify the string pointer and memory contents.
- **State Monitoring**: Use the GDB dashboard to track registers, stack, and memory.
- **Iterative Testing**: Reset and reload the program to ensure consistent behavior.

4 GDB Command History

The following GDB commands were executed to debug `slen`:

```
target extended-remote :3333
file prog
load
si
b main
dashboard -layout registers stack memory
c
si
c
si
c
laod  % typo, likely meant load
load
c
c
c
radix 16
set radix 16
p/x ptr
p/x *ptr
p/c *ptr
p/s *ptr
print &ptr
p ptr
dashboard memory set 0x800000c0 32
dashboard memory watch 0x800000c0 32
dash
p/x &ptr
p/x ptr
p/x *ptr
dash
dash
clear
dash
si
monitor reset halt
si
si
exit
```

```
target extended-remote :3333
dash
dashboard -layout registers stack memory breakpoints
file prog
load
si
exit
dashboard -layout registers stack memory breakpoints
exit
dashboard -layout registers memory breakpoints stack
file prog
load
si
exit
```

4.1 Explanation of Commands

- `target extended-remote :3333`: Connects to the embedded target on port 3333.
- `file prog`, `load`: Loads the executable `prog` onto the target.
- `b main`, `c`: Sets a breakpoint at `main` and continues execution to it.
- `si`: Steps through instructions, allowing inspection of `slen`'s loop.
- `p/x ptr`, `p/x *ptr`, `p/c *ptr`, `p/s *ptr`: Inspects the `ptr` address, its first character (hex and char), and the full string.
- `print &ptr`, `p/x &ptr`: Checks the address of `ptr` itself.
- `dashboard memory set 0x800000c0 32`, `dashboard memory watch 0x800000c0 32`: Monitors 32 bytes at address `0x800000c0`, likely where the string resides.
- `dashboard -layout ...`: Configures the GDB dashboard to show registers, stack, memory, and breakpoints.
- `monitor reset halt`: Resets the target and halts execution.
- `clear`, `exit`: Clears breakpoints and exits GDB sessions.
- `radix 16`, `set radix 16`: Sets hexadecimal output for memory and values.
- `dash`: Likely a custom alias or typo, possibly for dashboard commands.

5 Key Checks for `slen`

The debugging process verifies:

- **Pointer Validity**: `ptr` points to the correct string (checked via `p/s *ptr`).
- **Loop Correctness**: The loop in `slen` increments `n` and stops at `\0` (inspected via `si`).

- **Memory Access**: Memory at `0x800000c0` is valid and uncorrupted (via `dashboard memory watch`).

6 Conclusion

The GDB command history provides a detailed view of the debugging process, ensuring `slen` correctly computes the string length. The iterative use of stepping, memory inspection, and state monitoring confirms the function's behavior on the embedded target.