Assignment 2

Minimal Debugger, mdb

Overview

In this assignment you must build a minimal debugger (mdb), which is able to run and debug ELF programs in a Linux-based operating system. mdb must be able to load an ELF executable and run it on demand. In addition, mdb must be able to set multiple software breakpoints on code locations of the debugged program and inspect the running code.

More precisely, here are the commands that mdb should support.

(1) Load and prepare an ELF binary for execution

mdb should take one parameter with the full path of the binary that is debugged. The format that is supported is only ELF for Linux-based systems. You are free to use either libelf or libbfd for loading binaries.

Upon loading the binary, mdb should output a command-line prompt for receiving commands. The commands that are supported are discussed below.

(2) Add software breakpoints (command 'b')

mdb must support an arbitrary list of software breakpoints. A breakpoint should be added in the list using the command 'b' using a symbol or a hexadecimal address of the program. If an address is given then the address should start with the character '*'. Otherwise, the parameter should be treated as a symbol. If the symbol is not found, mdb should report that the given software breakpoint cannot be established. Examples:

```
> b foo
> b *00000000aabbccdd
```

(3) List currently enabled software breakpoints (command 'l')

With 'l', mdb should output a list with the currently enabled breakpoints. Each breakpoint should be given a number.

(4) Delete software breakpoints (command ‘d’)

With ‘d’ and a number as a parameter, mdb must delete the specified breakpoint. For example, the command ‘d 2’ deletes the 2nd breakpoint if it does exist.

(5) Run the program (command ‘r’)

With ‘r’ mdb should run the program until a breakpoint is reached or the program exits. If the program reaches a breakpoint, then mdb should print the disassembly of the current instruction (to be executed) and 10 more instructions or until the end of a function is reached. The output of the disassembly should be in the form of:

```
> address:  instruction
  address:  instruction
  ...
```

Address should be in hex and each printed instruction should be the symbolic representation of each opcode.

(6) Continue the program (command ‘c’)

With ‘c’ the program should continue until a breakpoint is reached or the program exits.

Bonus (additional 10%)

Implement the ‘si’ command where the program, if stopped, progresses by executing a single command and the ‘disas’ command where mdb should output the disassembly of the current instruction (to be executed) and 10 more instructions or until the end of a function is reached.

Submission

You should submit your source code and a Makefile for building it as a compressed tarball (.tgz). Your program should be able to build and run on the provided VM or on the Unix lab.

Deadline

Submit your work using Blackboard by the 27th of March (end of day).

Good luck!