

UP23 HW2

Due Date: 2023-06-05

- UP23 HW2
 - Simple Instruction Level Debugger
 - Launch the program
 - Disassemble
 - Step Instruction
 - Continue
 - Breakpoint
 - Time Travel
 - Examples
 - Example 1 (10pt)
 - Example 2 (10pt)
 - Example 3 (10pt)
 - Grading

Simple Instruction Level Debugger

In this homework, you have to implement a simple instruction-level debugger that allows a user to debug a program interactively at the assembly instruction level. You can implement the debugger by using the `ptrace` interface.

To simplify your program, your debugger only has to handle static-nopie programs.

We use `hello64` (<https://up23.zoolab.org/up23/hw2/hello64>), `hello` (<https://up23.zoolab.org/up23/hw2/hello>), `guess` (<https://up23.zoolab.org/up23/hw2/guess>) to demonstrate the usage of the debugger.

Launch the program

Unlike `gdb` and `lldb`, your debugger launches the target program when the debugger starts. The program should stop at the entry point, waiting for the user's `cont` or `si` commands.

```
# usage: ./sdb [program]
./sdb ./hello64
```

When the program is launched, the debugger should print the name of the executable and the entry point address. Before waiting for the user's input, the debugger should disassemble 5 instructions starting from the current program counter (rip). The detail requirement is described in the following paragraph.

```
** program './hello64' loaded. entry point 0x4000b0
    4000b0: b8 04 00 00 00      mov     eax, 4
    4000b5: bb 01 00 00 00      mov     ebx, 1
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80              int     0x80
(sdb)
```

Disassemble

When returning from execution, the debugger should disassemble 5 instruction starting from the current program counter. The address of the 5 instructions should be within the range of the text segment specified in the ELF file. We do not care about the format, but in each line, there should be

1. address, eg. 40000b0
2. raw instructions in grouping of 1 byte, eg. b8 04 00 00 00
3. mnemonic, eg. mov
4. operands of the instruction, eg. eax, 4

And make sure the output is aligned with the columns.

Hint: You can link against the capstone library for disassembling.

1. After typing an invalid command or using a command which is not `si`, `cont`, `timetravel`, the debugger should not disassemble the program.
2. Patched instructions like `0xcc` (`int3`) should not appear in the output.

```

(sdb) si
4000c4: cd 80          int      0x80
4000c6: b8 01 00 00 00 mov     eax, 1
4000cb: bb 00 00 00 00 mov     ebx, 0
4000d0: cd 80          int      0x80
4000d2: c3            ret

(sdb) si
hello, world!
4000c6: b8 01 00 00 00 mov     eax, 1
4000cb: bb 00 00 00 00 mov     ebx, 0
4000d0: cd 80          int      0x80
4000d2: c3            ret
** the address is out of the range of the text segment.
(sdb)
(sdb) si
4000cb: bb 00 00 00 00 mov     ebx, 0
4000d0: cd 80          int      0x80
4000d2: c3            ret
** the address is out of the range of the text segment.

```

Step Instruction

When the user use `si` command, the target program should execute a single instruction.

```

(sdb) si
4000c4: cd 80          int      0x80
4000c6: b8 01 00 00 00 mov     eax, 1
4000cb: bb 00 00 00 00 mov     ebx, 0
4000d0: cd 80          int      0x80
4000d2: c3            ret

(sdb) si
hello, world!
4000c6: b8 01 00 00 00 mov     eax, 1
4000cb: bb 00 00 00 00 mov     ebx, 0
4000d0: cd 80          int      0x80
4000d2: c3            ret
** the address is out of the range of the text segment.
(sdb)

```

Continue

The `cont` command continues the execution of the target program. The program should keep running until it terminates or hits a breakpoint.

You should not use `ptrace(PTRACE_SINGLE_STEP)` or patching interrupt instructions to implement this part, or you will get 0 points.

```

** program './hello64' loaded. entry point 0x4000b0
    4000b0: b8 04 00 00 00      mov     eax, 4
    4000b5: bb 01 00 00 00      mov     ebx, 1
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80               int     0x80
(sdb) break 0x4000ba
** set a breakpoint at 0x4000ba.
(sdb) cont
** hit a breakpoint at 0x4000ba.
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80               int     0x80
    4000c6: b8 01 00 00 00      mov     eax, 1
    4000cb: bb 00 00 00 00      mov     ebx, 0
(sdb) cont
hello, world!
** the target program terminated.

```

Breakpoint

A user can use `break <address in hexadecimal>` to set a breakpoint. The target program should stop before the instruction at the specified address is executed. Then it should print a message about the program. If the user resumes the program with `si` instead of `cont`, the program should not stop at the breakpoint twice. The debugger still needs to print the message.

```

** program './hello64' loaded. entry point 0x4000b0
    4000b0: b8 04 00 00 00      mov     eax, 4
    4000b5: bb 01 00 00 00      mov     ebx, 1
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80               int     0x80
(sdb) break 0x4000ba
** set a breakpoint at 0x4000ba.
(sdb) si
    4000b5: bb 01 00 00 00      mov     ebx, 1
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80               int     0x80
    4000c6: b8 01 00 00 00      mov     eax, 1
(sdb) si
** hit a breakpoint 0x4000ba.
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80               int     0x80
    4000c6: b8 01 00 00 00      mov     eax, 1
    4000cb: bb 00 00 00 00      mov     ebx, 0

```

Time Travel

Sometimes you might see some bugs that are hard to replicate. Use the `anchor` command set a checkpoint and use the `timetravel` command to restore the process status.

Hint:

There are two ways to implement this feature.

1. Snapshot the process memory and general purpose registers.
2. Patch `fork` into the target process and stop the parent or child as the checkpoint.

This functionality is inspired by the Checkpoint/Restore In Userspace(CRIU)

(https://criu.org/Main_Page). `gdb` also has a similar feature `checkpoint` which is implemented in a different way.

```

** program './hello64' loaded. entry point 0x4000b0
    4000b0: b8 04 00 00 00      mov     eax, 4
    4000b5: bb 01 00 00 00      mov     ebx, 1
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80               int     0x80
(sdb) anchor
** dropped an anchor
(sdb) break 0x4000cb
** set a breakpoint at 0x4000cb
(sdb) cont
hello, world!
** hit a breakpoint at 0x4000cb
    4000cb: bb 00 00 00 00      mov     ebx, 0
    4000d0: cd 80               int     0x80
    4000d2: c3                 ret
** the address is out of the range of the text segment.
(sdb) timetravel
** go back to the anchor point
    4000b0: b8 04 00 00 00      mov     eax, 4
    4000b5: bb 01 00 00 00      mov     ebx, 1
    4000ba: b9 d4 00 60 00      mov     ecx, 0x6000d4
    4000bf: ba 0e 00 00 00      mov     edx, 0xe
    4000c4: cd 80               int     0x80
(sdb) cont
hello, world!
** hit a breakpoint at 0x4000cb
    4000cb: bb 00 00 00 00      mov     ebx, 0
    4000d0: cd 80               int     0x80
    4000d2: c3                 ret
** the address is out of the range of the text segment.

```

Examples

Example 1 (10pt)

- Command: `./sdb ./hello`
- Inputs: `cont`

```
** program './hello' loaded. entry point 0x401000
    401000: f3 0f 1e fa                endbr64
    401004: 55                          push    rbp
    401005: 48 89 e5                    mov     rbp, rsp
    401008: ba 0e 00 00 00             mov     edx, 0xe
    40100d: 48 8d 05 ec 0f 00 00       lea     rax, [rip + 0xfec]
(sdb) cont
hello world!
** the target program terminated.
```

Example 2 (10pt)

- Command: ./sdb ./hello
- Inputs:

```
break 0x401030
break 0x40103b
cont
cont
si
si
```

```

** program './hello' loaded. entry point 0x401000
    401000: f3 0f 1e fa                endbr64
    401004: 55                          push     rbp
    401005: 48 89 e5                    mov     rbp, rsp
    401008: ba 0e 00 00 00             mov     edx, 0xe
    40100d: 48 8d 05 ec 0f 00 00       lea     rax, [rip + 0xfec]
(sdb) break 0x401030
** set a breakpoint at 0x401030
(sdb) break 0x40103b
** set a breakpoint at 0x40103b
(sdb) cont
** hit a breakpoint at 0x401030
    401030: 0f 05                      syscall
    401032: c3                          ret
    401033: b8 00 00 00 00             mov     eax, 0
    401038: 0f 05                      syscall
    40103a: c3                          ret
(sdb) cont
hello world!
** hit a breakpoint at 0x40103b
    40103b: b8 3c 00 00 00             mov     eax, 0x3c
    401040: 0f 05                      syscall
** the address is out of the range of the text segment.
(sdb) si
    401040: 0f 05                      syscall
** the address is out of the range of the text segment.
(sdb) si
** the target program terminated.

```

Example 3 (10pt)

- Command: `./sdb ./guess`
- Inputs:

```

break 0x4010bf
break 0x40111e
cont
anchor
cont
haha
timetravel
cont
42
cont

```



```

** program './guess' loaded. entry point 0x40108b
    40108b: f3 0f 1e fa                endbr64
    40108f: 55                          push    rbp
    401090: 48 89 e5                    mov     rbp, rsp
    401093: 48 83 ec 10                  sub     rsp, 0x10
    401097: ba 12 00 00 00              mov     edx, 0x12
(sdb) break 0x4010bf
** set a breakpoint at 0x4010bf
(sdb) break 0x40111e
** set a breakpoint at 0x40111e
(sdb) cont
guess a number > ** hit a breakpoint at 0x4010bf
    4010bf: bf 00 00 00 00              mov     edi, 0
    4010c4: e8 67 00 00 00              call    0x401130
    4010c9: 48 89 45 f8                  mov     qword ptr [rbp - 8], rax
    4010cd: 48 8d 05 3e 0f 00 00        lea     rax, [rip + 0xf3e]
    4010d4: 48 89 c6                     mov     rsi, rax
(sdb) anchor
** dropped an anchor
(sdb) cont
haha

no no no
** hit a breakpoint at 0x40111e
    40111e: bf 00 00 00 00              mov     edi, 0
    401123: e8 10 00 00 00              call    0x401138
    401128: b8 01 00 00 00              mov     eax, 1
    40112d: 0f 05                       syscall
    40112f: c3                           ret
(sdb) timetravel
** go back to the anchor point
    4010bf: bf 00 00 00 00              mov     edi, 0
    4010c4: e8 67 00 00 00              call    0x401130
    4010c9: 48 89 45 f8                  mov     qword ptr [rbp - 8], rax
    4010cd: 48 8d 05 3e 0f 00 00        lea     rax, [rip + 0xf3e]
    4010d4: 48 89 c6                     mov     rsi, rax
(sdb) cont
42

yes
** hit a breakpoint at 0x40111e
    40111e: bf 00 00 00 00              mov     edi, 0
    401123: e8 10 00 00 00              call    0x401138
    401128: b8 01 00 00 00              mov     eax, 1
    40112d: 0f 05                       syscall
    40112f: c3                           ret
(sdb) cont
** the target program terminated.

```

Grading

1. [30%] Your program has the correct output for all test cases listed in the examples section.
2. [70%] We use N additional test cases to evaluate your implementation. You get $\frac{60}{N}$ points for each correct test case.