# UP24 HW3

Due Date: 2024-06-03



## 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 should implement the debugger by using the `ptrace` interface in C/C++. The commands you have to implement are detailed in the Commands Requirements.

- To simplify your implementation, your debugger only has to handle **64-bit static-**

**nopie** programs on **x86-64**.

- We use the sample program (https://up.zoolab.org/unixprog/hw03/hw3_testing_program.zip) to demonstrate how to use the debugger.

### Usage

- You can load a program after/when the debugger starts. See the load program section for the details.
- You should print " `(sdb)` " as the prompt in every line, no matter whether you have loaded the program.

```
# Launch the debugger directly
./sdb
# Launch the debugger with a program
./sdb [program]
```

## Commands Requirements

> We will not test any error handling not mentioned in this spec. You can determine how to handle the other errors by yourself.

### Load Program

- Command: `load [path to a program]`
- Load a program after the debugger starts.
  - You should output `** please load a program first.` if you input any other commands before loading a program.
- When the program is loaded:
  - The debugger should print the **name of the executable and the entry point address**.
  - Before waiting for the user's input, the debugger should **stop at the entry point** and **disassemble 5 instructions** starting from the current program counter (rip).
- Sample output of `./sdb`

```
(sdb) info reg
** please load a program first.
(sdb) load ./hello
** 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)
```

- Sample output of `./sdb ./hello`

```
** 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)
```

## Disassemble

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

1. address, e.g. `401005`
2. raw instructions in a grouping of 1 byte, e.g., `48 89 e5`
3. mnemonic, e.g., `mov`
4. operands of the instruction, e.g., `edx, 0xe`

And make sure that

- The output is aligned with the columns.
- If the disassembled instructions are less than 5, output `** the address is out of the range of the text section.`

Sample output:

```
(sdb) si
hello world!
      401026: e8 10 00 00 00                    call      0x40103b
      40102b: b8 01 00 00 00                    mov       eax, 1
      401030: 0f 05                             syscall
      401032: c3                                ret
      401033: b8 00 00 00 00                    mov       eax, 0
(sdb) si
      40103b: b8 3c 00 00 00                    mov       eax, 0x3c
      401040: 0f 05                             syscall
** the address is out of the range of the text section.
```

Note:

- You should only disassemble the program when the program is loaded or when using `si`, `cont` and `syscall` commands.
- If the `break` command **sets a breakpoint** using patched instructions like `0xcc` (int3), it should not appear in the output.
- If the `patch` command is used in the text section, the disassembled code should be the patched value, see the patch section for examples.

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

## Step Instruction

- Command: `si`
- Execute a single instruction.
  - If the program hits a breakpoint, output `** hit a breakpoint at [addr].`
  - If the program terminates, output `** the target program terminated.`
- Sample output:

```
(sdb) break 40103b
** set a breakpoint at 0x40103b.
(sdb) si
** 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 section.
(sdb) si
      401040: 0f 05                             syscall
** the address is out of the range of the text section.
(sdb) si
** the target program terminated.
```

## Continue

- Command: `cont`
- Continue the execution of the target program. The program should keep running until it terminates or hits a breakpoint.
  - If the program hits a breakpoint, output `** hit a breakpoint at [addr].`
  - If the program terminates, output `** the target program terminated.`
- Sample output:

```
(sdb) break 0x40100d
** set a breakpoint at 0x40100d.
(sdb) cont
** hit a breakpoint at 0x40100d.
      40100d: 48 8d 05 ec 0f 00 00              lea       rax, [rip + 0xfec]
      401014: 48 89 c6                          mov       rsi, rax
      401017: bf 01 00 00 00                    mov       edi, 1
      40101c: e8 0a 00 00 00                    call      0x40102b
      401021: bf 00 00 00 00                    mov       edi, 0
(sdb) cont
hello world!
** the target program terminated.
```

Note: You can only use **two ptrace (PTRACE_SINGLE_STEP) and two int3 at most** in the implementation of cont, or you will get 0 points.

## Info Registers

- Command: `info reg`

- Show all the registers and their corresponding values in hex.
  - You should output 3 registers in each line.
  - Values should be printed in 64-bit hex format.
- Sample output:

```
(sdb) info reg
$rax 0x0000000000000001    $rbx 0x0000000000000000    $rcx 0x0000000000000000
$rdx 0x000000000000000e    $rsi 0x0000000000402000    $rdi 0x0000000000000001
$rbp 0x00007ffdc479ab68    $rsp 0x00007ffdc479ab60    $r8  0x0000000000000000
$r9  0x0000000000000000    $r10 0x0000000000000000    $r11 0x0000000000000000
$r12 0x0000000000000000    $r13 0x0000000000000000    $r14 0x0000000000000000
$r15 0x0000000000000000    $rip 0x0000000000401030    $eflags 0x0000000000000000
```

## Breakpoint

- Command: `break [hex address]`
- Set up a break point at the specified address. The target program should stop before the instruction at the specified address is executed. If the user resumes the program with `si` , `cont` or `syscall` , the program should not stop at the same breakpoint twice.
  - On success, output `** set a breakpoint at [hex address].`
- Sample output:

```
(sdb) break 0x401005
** set a breakpoint at 0x401005.
(sdb) break 40100d
** set a breakpoint at 0x40100d.
(sdb) si
** hit a breakpoint at 0x401005.
      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]
      401014: 48 89 c6                        mov       rsi, rax
      401017: bf 01 00 00 00                  mov       edi, 1
(sdb) si
      401008: ba 0e 00 00 00                  mov       edx, 0xe
      40100d: 48 8d 05 ec 0f 00 00            lea       rax, [rip + 0xfec]
      401014: 48 89 c6                        mov       rsi, rax
      401017: bf 01 00 00 00                  mov       edi, 1
      40101c: e8 0a 00 00 00                  call      0x40102b
```

## Info Breakpoints

- Command: `info break`
- List breakpoints with index numbers (for deletion) and addresses.
  - The index of the breakpoints starts from `0`.
  - If no breakpoints, output `** no breakpoints.`
  - If a breakpoint is deleted, the index of the other breakpoints should remain the same.
- Sample output:

```
(sdb) info break
** no breakpoints.
(sdb) break 0x4000ba
** set a breakpoint at 0x4000ba.
(sdb) break 0x4000bf
** set a breakpoint at 0x4000bf.
(sdb) info break
Num     Address
0       0x4000ba
1       0x4000bf
(sdb) delete 0
** delete breakpoint 0.
(sdb) info break
Num     Address
1       0x4000bf
```

## Delete Breakpoints

- Command: `delete [id]`
- Remove a break point with the specified id. The id is corresponding to the index number in Info Breakpoints.
  - On success, output `** delete breakpoint [id].`
  - If the breakpoint id does not exist, output `** breakpoint [id] does not exist.`
- Sample output:

```
(sdb) break 0x4000ba
** set a breakpoint at 0x4000ba.
(sdb) info break
Num     Address
0       0x4000ba
(sdb) delete 0
** delete breakpoint 0.
(sdb) delete 0
** breakpoint 0 does not exist.
```

## Patch Memory

- Command: `patch [hex address] [hex value] [len]`
- Patch memory starts at the `address` with the `value` of `len` bytes. The `value` will be integer value represented in hex and its length (in byte) is determined by the `len`, which can be either 1, 2, 4, or 8.
    - On success, output `** patch memory at address [hex address].`

> Note:
> - If you patch on an instruction that has been set as a breakpoint, the breakpoint should still exist, but the original instruction should be patched.
> - You don't have to handle cases where the patch causes the program to crash. We just want to make sure that this function works. 🫶

- Sample output:

```
(sdb) si
      401017: bf 01 00 00 00                  mov       edi, 1
      40101c: e8 0a 00 00 00                  call      0x40102b
      401021: bf 00 00 00 00                  mov       edi, 0
      401026: e8 10 00 00 00                  call      0x40103b
      40102b: b8 01 00 00 00                  mov       eax, 1
(sdb) patch 0x40101c 0x0090 2
** patch memory at 0x40101c.
(sdb) si
      40101c: 90                              nop
      40101d: 00 00                           add       byte ptr [rax], al
      40101f: 00 00                           add       byte ptr [rax], al
      401021: bf 00 00 00 00                  mov       edi, 0
      401026: e8 10 00 00 00                  call      0x40103b
(sdb)
```

### System Call

- Command: `syscall`
- The program execution should break at every system call instruction **unless it hits a breakpoint**.
  - If it hits a breakpoint, output `** hit a breakpoint at [hex address].`
  - If it enters a syscall, output `** enter a syscall([nr]) at [hex address].`
  - If it leaves a syscall, output `** leave a syscall([nr]) = [ret] at [hex address].`

> Note: You can ignore the cases where a breakpoint is set on a syscall instruction.

- Sample output:

```
(sdb) syscall
** hit a breakpoint at 0x401008
      401008: ba 0e 00 00 00                    mov       edx, 0xe
      40100d: 48 8d 05 ec 0f 00 00              lea       rax, [rip + 0xfec]
      401014: 48 89 c6                          mov       rsi, rax
      401017: bf 01 00 00 00                    mov       edi, 1
      40101c: e8 0a 00 00 00                    call      0x40102b
(sdb) syscall
** enter a syscall(1) 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) syscall
hello world!
** leave a syscall(1) = 14 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
```

# Examples

We use the sample program (https://up.zoolab.org/unixprog/hw03/hw3_testing_program.zip) to demonstrate the following examples.

## Example 1

- Requirements (basic): `load` `cont` `si` `disassemble`
- Launch debugger: `./sdb`
- Input:

```
si
load ./hello
si
si
cont
```

- Sample:

```
(sdb) si
** please load a program first.
(sdb) load ./hello
** 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) si
      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]
      401014: 48 89 c6                            mov       rsi, rax
(sdb) si
      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]
      401014: 48 89 c6                            mov       rsi, rax
      401017: bf 01 00 00 00                      mov       edi, 1
(sdb) cont
hello world!
** the target program terminated.
```

## Example 2

- Requirements (basic): `break` `info break` `info reg`
- Launch debugger: `./sdb ./hello`
- Input:

```
break 0x401005
break 40102b
info break
si
si
cont
info reg
cont
```

- Sample:

```
** 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 0x401005
** set a breakpoint at 0x401005
(sdb) break 40102b
** set a breakpoint at 0x40102b
(sdb) info break
Num     Address
0       0x401005
1       0x40102b
(sdb) si
      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]
      401014: 48 89 c6                            mov       rsi, rax
(sdb) si
** hit a breakpoint at 0x401005
      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]
      401014: 48 89 c6                            mov       rsi, rax
      401017: bf 01 00 00 00                      mov       edi, 1
(sdb) cont
** hit a breakpoint at 0x40102b
      40102b: b8 01 00 00 00                      mov       eax, 1
      401030: 0f 05                               syscall
      401032: c3                                  ret
      401033: b8 00 00 00 00                      mov       eax, 0
      401038: 0f 05                               syscall
(sdb) info reg
$rax 0x0000000000402000    $rbx 0x0000000000000000    $rcx 0x0000000000000000
$rdx 0x000000000000000e    $rsi 0x0000000000402000    $rdi 0x0000000000000001
$rbp 0x00007ffe0e5cd5b8    $rsp 0x00007ffe0e5cd5b0    $r8  0x0000000000000000
$r9  0x0000000000000000    $r10 0x0000000000000000    $r11 0x0000000000000000
$r12 0x0000000000000000    $r13 0x0000000000000000    $r14 0x0000000000000000
$r15 0x0000000000000000    $rip 0x000000000040102b    $eflags 0x0000000000000000
(sdb) cont
hello world!
** the target program terminated.
```

## Example 3

- Requirements (advanced): `delete  patch`
- Launch debugger: `./sdb ./guess`

- Input:

```
break 0x4010de
cont
1
patch 0x4010e8 0x9090 2
si
info break
delete 0
break 0x4010ea
delete 0
info break
cont
patch 0x402015 0x4e49570a 4
cont
```

- Sample:

```
** 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 0x4010de
** set a breakpoint at 0x4010de.
(sdb) cont
guess a number > 1
** hit a breakpoint at 0x4010de.
      4010de: 48 89 c7                            mov       rdi, rax
      4010e1: e8 1a ff ff ff                      call      0x401000
      4010e6: 85 c0                               test      eax, eax
      4010e8: 75 1b                               jne       0x401105
      4010ea: ba 06 00 00 00                      mov       edx, 6
(sdb) patch 0x4010e8 0x9090 2
** patch memory at address 0x4010e8.
(sdb) si
      4010e1: e8 1a ff ff ff                      call      0x401000
      4010e6: 85 c0                               test      eax, eax
      4010e8: 90                                  nop
      4010e9: 90                                  nop
      4010ea: ba 06 00 00 00                      mov       edx, 6
(sdb) info break
Num     Address
0       0x4010de
(sdb) delete 0
** delete breakpoint 0.
(sdb) break 0x4010ea
** set a breakpoint at 0x4010ea.
(sdb) delete 0
** breakpoint 0 does not exist.
(sdb) info break
Num     Address
1       0x4010ea
(sdb) cont
** hit a breakpoint at 0x4010ea.
      4010ea: ba 06 00 00 00                      mov       edx, 6
      4010ef: 48 8d 05 1f 0f 00 00                lea       rax, [rip + 0xf1f]
      4010f6: 48 89 c6                            mov       rsi, rax
      4010f9: bf 01 00 00 00                      mov       edi, 1
      4010fe: e8 25 00 00 00                      call      0x401128
(sdb) patch 0x402015 0x4e49570a 4
** patch memory at address 0x402015.
(sdb) cont

WIN
** the target program terminated.
```

## Example 4

- Requirements (advanced): `syscall`
- Launch debugger: `./sdb ./hello`
- Input:

```
break 0x401005
break 40102b
cont
syscall
syscall
syscall
syscall
syscall
```

- Sample:

```
** 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 0x401005
** set a breakpoint at 0x401005
(sdb) break 40102b
** set a breakpoint at 0x40102b
(sdb) cont
** hit a breakpoint at 0x401005
      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]
      401014: 48 89 c6                            mov       rsi, rax
      401017: bf 01 00 00 00                      mov       edi, 1
(sdb) syscall
** hit a breakpoint at 0x40102b
      40102b: b8 01 00 00 00                      mov       eax, 1
      401030: 0f 05                               syscall
      401032: c3                                  ret
      401033: b8 00 00 00 00                      mov       eax, 0
      401038: 0f 05                               syscall
(sdb) syscall
** enter a syscall(1) 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) syscall
hello world!
** leave a syscall(1) = 14 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) syscall
** enter a syscall(60) at 0x401040.
      401040: 0f 05                               syscall
** the address is out of the range of the text section.
(sdb) syscall
** the target program terminated.
```

# Homework Submission

- Due time: 2024-06-03 15:30

- Filename: `{studentID}_hw3.zip`
- Format:

```
+---{studentID}_hw3
|    Makefile
|    sdb.c/sdb.cpp
|    ...
```

## Grading

- [40%] Your program has the correct output for all example test cases.
- [60%] We use `N` hidden test cases to evaluate your implementation. You get `60/N` points for each correct test case.

> Plagiarism is not allowed. Any student who is caught plagiarizing will receive a zero.
> 👏
> We will find

## Demo

- Date: 2024-06-03

> TBA