# Homework #4

Advanced Programming in the UNIX Environment

Due: June 24, 2019

## Simple Instruction Level Debugger

In this homework, we are going 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. The commands you have to implement are summarized as follows:

```
- break {instruction-address}: add a break point
- cont: continue execution
- delete {break-point-id}: remove a break point
- disasm addr: disassemble instructions in a file or a memory region
- dump addr [length]: dump memory content
- exit: terminate the debugger
- get reg: get a single value from a register
- getregs: show registers
- help: show this message
- list: list break points
- load {path/to/a/program}: load a program
- run: run the program
- vmmap: show memory layout
- set reg val: get a single value to a register
- si: step into instruction
- start: start the program and stop at the first instruction
```

The details of each command are explained below. In a debugging process, you have to load a program first, configure the debugger, and start debugging by running the program. A debugger command may be only used in certain "states." The states include **any**, **loaded**, and **running**. **any** means that a command can be used at any time. **loaded** means that a command can be only used when a program is loaded. **running** means that a command can be only used when the program is running. We will use brackets right after a command to enclose the list of the state(s) that should be supported by the command.

- **break** or **b** [**loaded** and **running**]: Setup a break point. If a program is loaded but is not running, the address should be within the range specified by the text segment in the ELF file. When a break point is hit, you have to output a message and indicate the corresponding address and instruction.
- **cont** or **c** [**running**]: continue the execution when a running program is stopped (suspended).
- **delete** [**any**]: remove a break point.
- **disasm** or **d** [**loaded** and **running**]: Disassemble instructions in a file or a memory region. The address should be within the range specified by the text segment in the ELF file. You only have to dump 10 instructions for each command. If **disasm** command is executed without an address, it should disassemble the codes right after the previously disassembled codes. See the demonstration section for the sample output format.
- **dump** or **x** [**running**]: Dump memory content. You only have to dump 80 bytes from a given address. The output contains the addresses, the hex values, and printable ascii characters. If **dump** command is executed without an address, it should dump the region right after the previous dump.
- **exit** or **q** [**any**]: Quit from the debugger. The program being debugged should be killed as well.
- **get** or **g** [**running**]: Get the value of a register. Register names are all in lowercase.
- **getregs** [**running**]: Get the value of all registers.
- **help** or **h** [**any**]: Show the help message.
- **list** or **l** [**any**]: List break points, which contains index numbers (for deletion) and addresses.

- **load** [not **loaded**]: Load a program into the debugger. When a program is loaded, you have to print out the entry point, the address, the offset, and the size for the text segment.
- **run** or **r** [**loaded** and **running**]: Run the program. If the program is already running, show a warning message and continue the execution.
- **vmmap** or **m** [**loaded** and **running**]: Show memory layout for a running program. If a program is loaded but is not running, it should display the text segment address of the loaded program.
- **set** or **s** [**running**]: Set the value of a register
- **si** [**running**]: Run a single instruction, and step into function calls.
- **start** [**loaded**]: Start the program and stop at the first instruction.

For more details about the implementation, please check the demonstration section for the sample input and the corresponding output.

## Grading Policy

The grading policy for this homework is listed below:

1. Load and run the program.
2. Start the program, continue execution, and step into functions.
3. Dump memory content.
4. Set and get value from registers.
5. Disassemble assembly from the executable or from a memory region.
6. Handle break points.

## Homework Submission

Please pack your files into a single ZIP archive and submit your homework via the E3 system. Please also provide a Makefile (used for compiling and linking your codes) and a README file (indicating what features you have implemented).

## Demonstration

We use the hello world (hello64) and the guess (guess) program introduced in the class to demonstrate the usage of the simple debugger. User typed commands are marked in ***blue***.

### # Load a program, show maps, and run the program (hello64)

```
$ ./sdb
sdb> load sample/hello64
** program 'sample/hello64' loaded. entry point 0x4000b0, vaddr 0x4000b0, offset 0xb0, size 0x23
sdb> vmmap
00000000004000b0-00000000004000d3 r-x b0       sample/hello64
sdb> start
** pid 16328
sdb> vmmap
0000000000400000-0000000000401000 r-x 0        /home/chuang/unix_prog/hw4_sdb/sample/hello64
0000000000600000-0000000000601000 rwx 0        /home/chuang/unix_prog/hw4_sdb/sample/hello64
00007ffe29604000-00007ffe29625000 rwx 0        [stack]
00007ffe29784000-00007ffe29787000 r-- 0        [vvar]
00007ffe29787000-00007ffe29789000 r-x 0        [vdso]
7fffffffffffffff-7fffffffffffffff r-x 0        [vsyscall]
sdb> get rip
rip = 4194480 (0x4000b0)
sdb> run
** program sample/hello64 is already running.
hello, world!
** child process 16328 terminiated normally (code 0)
sdb>
```

### # Start a progrm, and show registers

```
./sdb sample/hello64
** program 'sample/hello64' loaded. entry point 0x4000b0, vaddr 0x4000b0, offset 0xb0, size 0x23
sdb> start
** pid 30433
sdb> getregs
RAX 0                 RBX 0                 RCX 0                 RDX 0
R8  0                 R9  0                 R10 0                 R11 0
R12 0                 R13 0                 R14 0                 R15 0
RDI 0                 RSI 0                 RBP 0                 RSP 7ffc51e88280
RIP 4000b0            FLAGS 0000000000000200
sdb>
```

## # Start a program, set a break point, check assembly output, and dump memory (hello64)

```
$ ./sdb sample/hello64
** program 'sample/hello64' loaded. entry point 0x4000b0, vaddr 0x4000b0, offset 0xb0, size 0x23
sdb> disasm
** no addr is given.
sdb> disasm 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
      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> start
** pid 20354
sdb> b 0x4000c6
sdb> disasm 0x4000c6
      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
      4000d3: 00 68 65                       add    byte ptr [rax + 0x65], ch
      4000d6: 6c                             insb   byte ptr [rdi], dx
      4000d7: 6c                             insb   byte ptr [rdi], dx
      4000d8: 6f                             outsd  dx, dword ptr [rsi]
      4000d9: 2c 20                          sub    al, 0x20
      4000db: 77 6f                          ja     0x40014c
sdb> dump 0x4000c6
      4000c6: cc 01 00 00 00 bb 00 00 00 00 cd 80 c3 00 68 65  |..............he|
      4000d6: 6c 6c 6f 2c 20 77 6f 72 6c 64 21 0a 00 00 00 00  |llo, world!.....|
      4000e6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
      4000f6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 00  |................|
      400106: 01 00 b0 00 40 00 00 00 00 00 00 00 00 00 00 00  |....@...........|
sdb>
```

## # Load a program, disassemble, set break points, run the program, and change the control flow (hello64).

```
$ ./sdb sample/hello64
** program 'sample/hello64' loaded. entry point 0x4000b0, vaddr 0x4000b0, offset 0xb0, size 0x23
sdb> disasm 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
      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> b 0x4000c6
sdb> l
  0:  4000c6
sdb> run
** pid 16690
hello, world!
** breakpoint @       4000c6: b8 01 00 00 00                 mov    eax, 1
sdb> set rip 0x4000b0
sdb> cont
hello, world!
** breakpoint @       4000c6: b8 01 00 00 00                 mov    eax, 1
sdb> delete 0
** breakpoint 0 deleted.
sdb> set rip 0x4000b0
sdb> cont
hello, world!
** child process 16690 terminiated normally (code 0)
sdb>
```

# Load a program, disassemble, set break points, run the program, and change the control flow (guess).

```
$ ./sdb sample/guess
** program 'sample/guess' loaded. entry point 0x820, vaddr 0x820, offset 0x820, size 0x262
sdb> vmmap
0000000000000820-0000000000000a82 r-x 820      sample/guess
sdb> disasm 0x985
         985: 48 8d 3d 08 01 00 00           lea    rdi, qword ptr [rip + 0x108]
         98c: b8 00 00 00 00                 mov    eax, 0
         991: e8 0a fe ff ff                 call   0x7a0
         996: 48 8b 15 73 06 20 00           mov    rdx, qword ptr [rip + 0x200673]
         99d: 48 8d 45 d0                    lea    rax, qword ptr [rbp - 0x30]
         9a1: be 10 00 00 00                 mov    esi, 0x10
         9a6: 48 89 c7                       mov    rdi, rax
         9a9: e8 12 fe ff ff                 call   0x7c0
         9ae: 48 8d 45 d0                    lea    rax, qword ptr [rbp - 0x30]
         9b2: ba 00 00 00 00                 mov    edx, 0
sdb> disasm
         9b7: be 00 00 00 00                 mov    esi, 0
         9bc: 48 89 c7                       mov    rdi, rax
         9bf: e8 0c fe ff ff                 call   0x7d0
         9c4: 8b 15 52 06 20 00              mov    edx, dword ptr [rip + 0x200652]
         9ca: 89 d2                          mov    edx, edx
         9cc: 48 39 d0                       cmp    rax, rdx
         9cf: 75 0e                          jne    0x9df
         9d1: 48 8d 3d ce 00 00 00           lea    rdi, qword ptr [rip + 0xce]
         9d8: e8 93 fd ff ff                 call   0x770
         9dd: eb 0c                          jmp    0x9eb
sdb> b 0x9cc
sdb> start
** pid 17133
sdb> vmmap
00005559c2a73000-00005559c2a74000 r-x 0        /home/chuang/unix_prog/hw4_sdb/sample/guess
00005559c2c73000-00005559c2c75000 rw- 0        /home/chuang/unix_prog/hw4_sdb/sample/guess
00007f18475b4000-00007f18475db000 r-x 0        /lib/x86_64-linux-gnu/ld-2.27.so
00007f18477db000-00007f18477dd000 rw- 27000    /lib/x86_64-linux-gnu/ld-2.27.so
00007f18477dd000-00007f18477de000 rw- 0
```

```
00007ffd56d81000-00007ffd56da2000 rw- 0        [stack]
00007ffd56dd7000-00007ffd56dda000 r-- 0        [vvar]
00007ffd56dda000-00007ffd56ddc000 r-x 0        [vdso]
7fffffffffffffff-7fffffffffffffff r-x 0        [vsyscall]
sdb> cont
Show me the key: 1234
** breakpoint @ 5559c2a739cc: 48 39 d0                       cmp    rax, rdx
sdb> get rax
rax = 1234 (0x4d2)
sdb> get rdx
rdx = 17624781 (0x10ceecd)
sdb> set rax 17624781
sdb> cont
Bingo!
** child process 17133 terminiated normally (code 0)
sdb>
```

## Hints

Here we provide a number of hints for implementing this homework.

- For disassembling, you have to link against the capstone (http://www.capstone-engine.org/) library. You may refer to the official capstone C tutorial (https://www.capstone-engine.org/lang_c.html) or the ptrace slide for the usage.
- For handling ELF file, you have to link against the libelf library (with -lelf option). To reduce the workload for implemeting ELF file handling, we have a sample wrapper implementation for libelf so that you can read required information from ELF file. Please refer to the files elftool.h (elftool.h), elftool.c (elftool.c), and elfdemo.c (elfdemo.c).