Homework #4

Advanced Programming in the UNIX Environment

Due: June 23, 2021

*** The difficulty of this homework has been simplified to handle only PIE-disabled program.

Simplified Scriptable 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** [**running**]: Setup a break point. If a program is loaded but is not running, you can simply display an error message. 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** [**running**]: remove a break point.
- **disasm** or **d** [**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 address of entry point.
- **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** [**running**]: Show memory layout for a running program. If a program is not running, you can simply display an error message.
- **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.

Your program may output some debug messages. In that case, please add "**" prefixes before your message. We will remove lines beginning with "**" when comparing outputs.

For more details about the implementation, please check the demonstration section for the sample input and the corresponding output. Your program should read user command from either user inputs (by default) or from a predefined script (if -s option is given). The usage of this homework is:

```
usage: ./hw4 [-s script] [program]
```

Homework Submission

We will compile your homework by simply typing 'make' in your homework directory. Please make sure your Makefile works and the output executable name is correct before submitting your homework.

Please pack your C/C++/Assembly code and Makefile into a **zip** archive. The directory structure should follow the below illustration. The ***id*** is your student id. Please note that you don't need to enclose your id with the braces.

```
{id}_hw4.zip
└── {id}_hw4/
        ├── Makefile
        └── (any other c/c++/assembly files if needed)
```

You have to submit your homework via the E3 system. Scores will be graded based on the completeness of your implementation.

Demonstration

We use the hello world and the guess.nopie 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
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 terminated normally (code 0)
sdb>
```

Start a progrm, and show registers

```
./sdb sample/hello64
** program 'sample/hello64' loaded. entry point 0x4000b0
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
sdb> start
** pid 28354
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
      4000d3: 00 68 65                      add    byte ptr [rax + 0x65], ch
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
sdb> start
** pid 16690
sdb> b 0x4000c6
sdb> i
  0:  4000c6
sdb> cont
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 terminated normally (code 0)
sdb>
```

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

```
$ ./sdb sample/guess.nopie
** program 'sample/guess' loaded. entry point 0x4006f0
sdb> start
** pid 17133
sdb> b 0x400879
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 5678
sdb> set rdx 5678
sdb> cont
Bingo!
** child process 17133 terminated normally (code 0)
sdb>
```

Sample scripts passed to your homework (with -s option) can be found here!

- hello1.txt
- hello2.txt
- hello3.txt
- hello4.txt
- guess.txt

Two examples of running scripts are given as follows

#1. Print 'hello, world!' for three times.

```
$ ./sdb -s scripts/hello3.txt 2>&1 | grep -v '^\*\*'
hello, world!
rip = 4194502 (0x4000c6)
hello, world!
rip = 4194502 (0x4000c6)
hello, world!
Bye.
```

#2. Auto debugger for guess

```
./sdb -s scripts/guess.txt sample/guess.nopie 2>&1 | grep -v '^\*\*'
1234
rax = 1234 (0x4d2)
rdx = 580655839 (0x229c1adf)
Show me the key: Bingo!
Bye.
```

Hints

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

- For disassembling, you have to link against the capstone library. You may refer to the official capstone C tutorial or the ptrace slide for the usage.