UMBC

gdb

CMSC 313

Raphael Elspas

Tools in this class

- Our development environment will be the **GL server** - make sure that you check that your codes runs on this server.
- Assembler: “**nasm**”. This is an assembler for x86-64 architecture
- We’ll be using **gcc** for linking. gcc is a full compiler like g++ but for C.
- There is a linker inside of gcc called “ld”
- We will use **gdb** for debugging

Steps for assembling

- Once you have written your assembly, **assemble** your code:

```
nasm –f elf64 assembly_file.asm
```

- **nasm** is our assembler.
- **-f elf64** chooses the output format. We will assemble to an elf executable object file with 64-bit values and operations.
- **assembly_file.asm** is your assembly code.

Steps for linking

- Once you have assembled your code, use the following to link your code

```
gcc –m64 –o executable_name object_file.o
```

- **gcc** includes our linker “ld”
- **-m64** forces 64 bit values and operations.
- **-o executable_name** specifies that you want to call your executable “executable_name”
- **object_file.o** is the name(s) of the file(s) you’d like to link.

Running program

- Use a “./” before the name of the executable to run your program

```
./executable_name
```

Gdb

- Gdb is a tool for inspecting the memory, registers and flags during the runtime of a program with the purpose of debugging

Running gdb

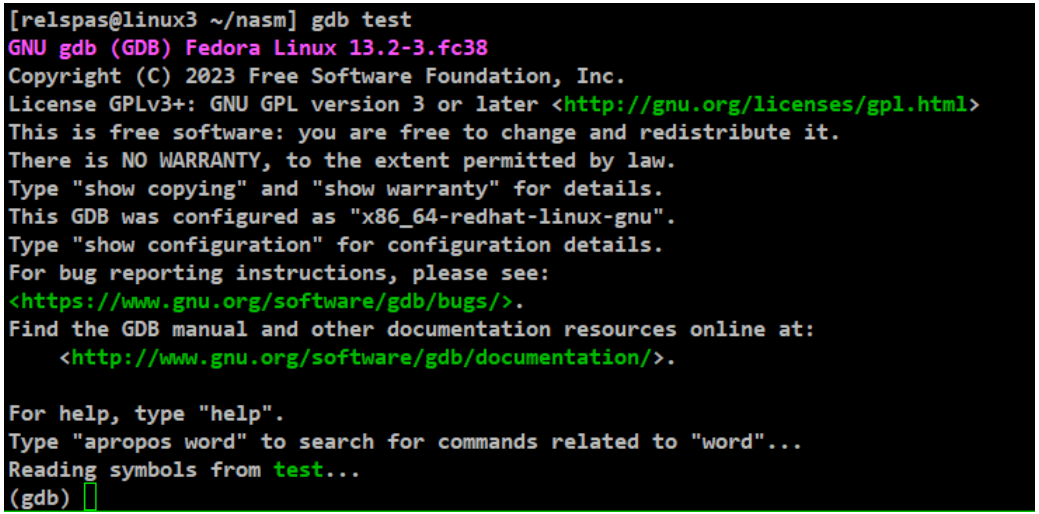
- To run gdb, use the command:

```
gdb executable_name
```

- **gdb** is our debugging tool
- **executable_name** is our program we assembled and linked

gdb

- Gdb looks like this:



- Prompt at the bottom allows us to run commands

Simple gdb commands

- The commands below are listed as <shorthand>(longhand). Either can be used as commands in gdb

- h (help) – starting point to find new commands
- disassemble (disas) – see assembly instructions at debug point
- b <label/line number> (break <label/line number>) – set a breakpoint in your code so that you can inspect execution at that point during runtime while debugging.
- r (run) – runs your program and pauses at breakpoints
- exit – exists gdb

Break points

- break main – will break at label main
- Break *0x401115 – will break at instruction address 0x401115. Use an asterisk to identify you are specifying an address.

Registers, disassembly

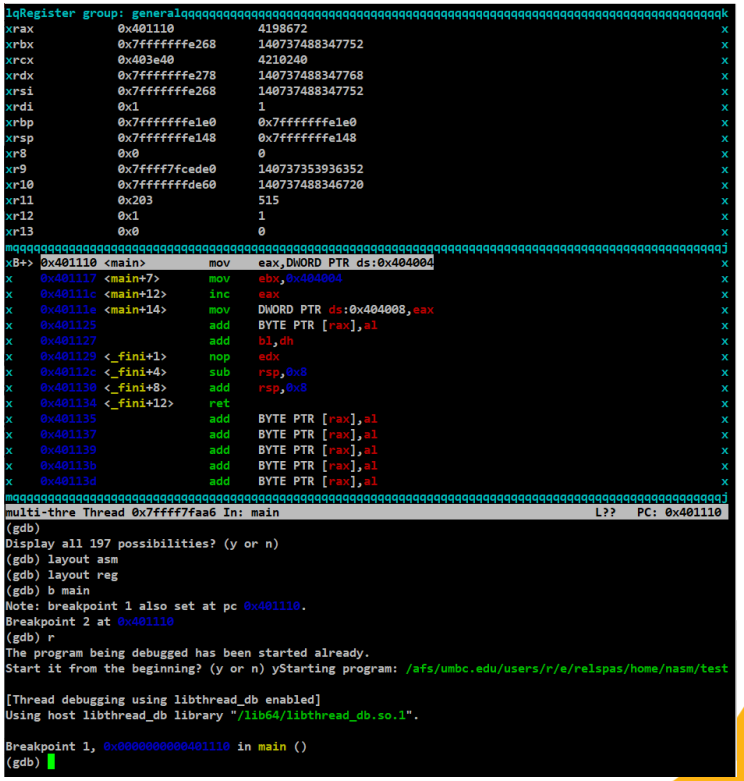
- To see registers use “info registers” or “i r”
- To see disassembly use “disassemble” or “disas” with an optional address to inspect

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000401110 <+0>:     mov    eax,DWORD PTR ds:0x404004
   0x0000000000401117 <+7>:     mov    ebx,0x404004
   0x000000000040111c <+12>:    inc    eax
   0x000000000040111e <+14>:    mov    DWORD PTR ds:0x404008,eax
End of assembler dump.
```

```
0x000000000040111c in main ()
(gdb) i r
rax            0xd                 13
rbx            0x404004            4210692
rcx            0x403e40            4210240
rdx            0x7fffffffe278      140737488347768
rsi            0x7fffffffe268      140737488347752
rdi            0x1                 1
rbp            0x7fffffffe1e0      0x7fffffffe1e0
rsp            0x7fffffffe148      0x7fffffffe148
r8             0x0                 0
r9             0x7ffff7fcede0      140737353936352
r10            0x7fffffffde60      140737488346720
r11            0x203               515
r12            0x1                 1
r13            0x0                 0
r14            0x7ffff7ffd000      140737354125312
r15            0x403e40            4210240
rip            0x40111c            0x40111c <main+12>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

Permanent registers and disassembly

- “layout reg” for disassembly and registers
- “layout asm” for disassembly
- “focus reg” to switch to register pane
- “focus asm” to switch to disassembly pane
- “focus cmd” to switch to command line pane
- “Ctrl+x”, then “a” to exit graphical interface



Syntax flavor

- By default AT&T syntax is displayed

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000401110 <+0>:     mov    0x404004,%r8d
   0x0000000000401118 <+8>:     xor    %r8b,%r8b
End of assembler dump.
```

- To switch to Intel syntax use the gdb command:

```
set disassembly-flavor intel
```

- To make it persistent for every run, write that line to the file: ~/.gdbinit

Step through instructions

- The gdb command **stepi** is short for step instruction
- stepi (or **si** for short) executes only one instruction at a time
- This allows you to closely examine the effects of each individual instruction on the program's state, including changes to memory and registers.
- Equivalent to **“step into”**

- **nexti** (or **ni** for short) will also execute the next instruction, but if a subroutine is reached, the subroutine will be executed in one step.
- nexti is equivalent to **“step over”**

Viewing memory

- Use the command x (stands for examine)

```
x/[count][format][size] address
```

- count: Specifies the number of units to display.
- format: Specifies the format of the data to be displayed.
- size: Specifies the size of each unit.

Viewing memory

```
x/[count][format][size] address
```

Count
Number of elements of size given by “size specifier”

Format specifier	
specifiers	**displayed**
x	hexadecimal
d	Signed decimal
u	Unsigned decimal
f	Floating point
s	String
i	Machine instruction

Size specifier	
Size specifiers	**Size of segment**
b	1 byte
h	2 bytes
w	4 bytes
g	8 bytes

Viewing memory examples

- Examine 10 4-byte segments of memory starting at address 0x1000 in hexadecimal format:

```
(gdb) x/10wx 0x1000
```

- Examine 5 floating-point numbers starting at address 0x2000:

```
(gdb) x/5f 0x2000
```

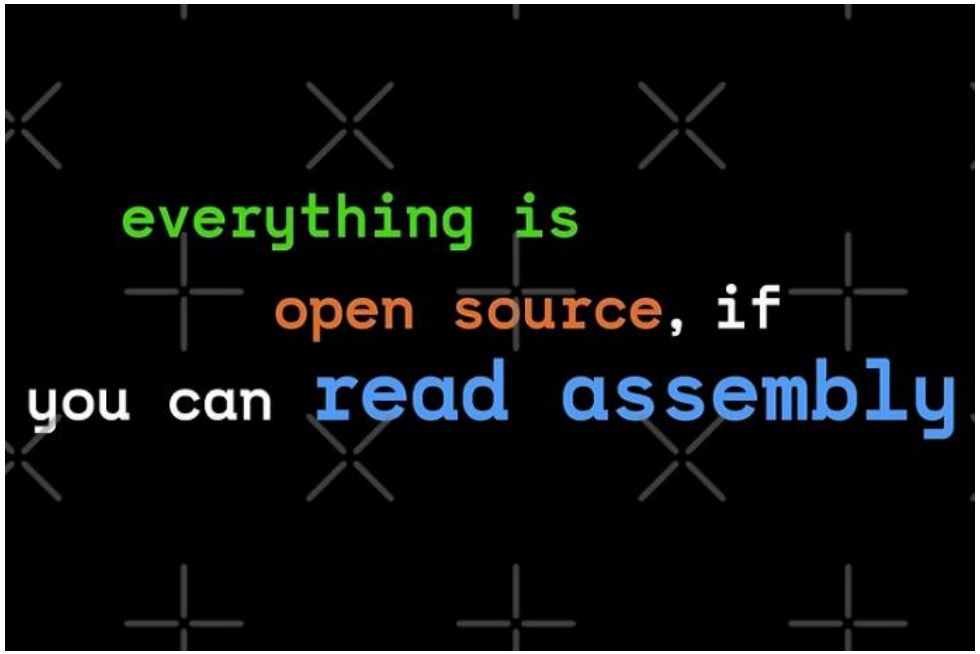
- Examine a null-terminated string starting at address 0x3000:

```
(gdb) x/s 0x3000
```

- Examine 12 1-byte segments as decimal from a label:

```
(gdb) x/12bd &data_label
```

Some wisdom



References

- Ivan Sekyonda’s slides