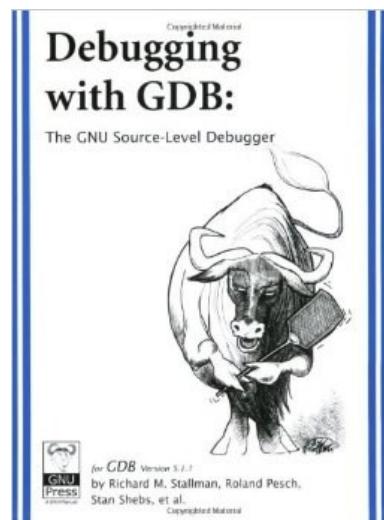


GDB and Assembly

A sample run of debugging
assembly code

What is GDB?

- Gnu DeBugger - gdb
- Originally created for C and C++
- Works well with other languages
- We'll use it for the assembly code version of the factorial program



Factorial function in C and Assembly

```

/* factorial driver
 * 2015-11-18
 */
#include<stdio.h>
#include<stdlib.h> // atoi()

long unsigned fact(long unsigned n);

int main(int argc, char **argv)
{
    unsigned i;
    long unsigned n, f;
    for (i = 1; i < argc; i++) {
        n = atoi(argv[i]);
        f = fact(n);
        printf("fact(%2lu) is %lu\n", n, f);
    }
    return 0;
}

```

```

; asm implementation of recursive factorial
; 2015-11-18
    global fact
    section .text

; expects:
;   RDI - n
; returns:
;   RAX - factorial(n)
fact:
    push rbp
    mov rbp, rsp

    cmp rdi, 2
    jbe .basecase

.recurse:
    push rdi        ; save n for later
    dec rdi         ; n - 1 ...
    call fact
    pop rdi
    mul rdi          ; rax <-- rax * rdi
    jmp .end

.basecase:
    mov rax, rdi

.end:
    leave
    ret

```

Assemble/compile the source for debugging:

- `nasm -f elf64 -l fact1.lst -g fact.asm`
 - `-f elf64` - produce 64-bit Linux code
 - `-l fact1.lst` - produce a listing file as well
 - `-g` - include debugging symbols
- `gcc -Wall -o fact -g factmain.c fact.o`
 - `-Wall` - turn on all warnings and errors
 - `-o fact` - create an executable named "fact"
 - `-g` - include debugging symbols
- The resulting executable includes information that gdb can use for displays

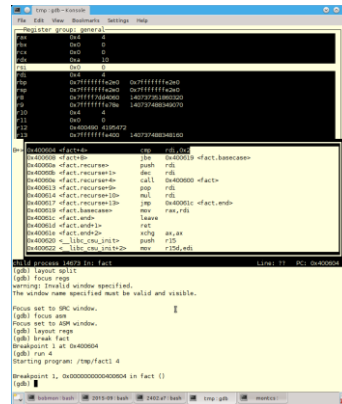
Getting ready to debug

- open up the listing file in an editor
- open a tall terminal shell next to the editor

```

1      ; asm implementation of recursive factorial
2      ; 2015-11-18
3      global fact
4      section .text
5      ; expects:
6      ;   RDI = n
7      ; returns:
8      ;   RAX = factorial(n)
9      fact:
10     00000000 55      push rbp
11     00000001 4889E5      mov rbp, rsp
12
13     00000004 4883FF02     cmp rdi, 2
14     00000008 760F      jbe .basecase
15
16     0000000A 57      .recurse: push rdi      ; save n for later
17     0000000B 48FFCF      dec rdi      ; n = 1 ...
18     0000000E E8EDFFFF      call fact
19     00000013 5F      pop rdi
20     00000014 48F7E7      mul rdi      ; rax <- rax * rdi
21     00000017 EB03      jmp .end
22
23     00000019 4889F8      .basecase: mov rax, rdi
24
25     0000001C C9      .end: leave
26     0000001D C3      ret
27
28

```



A few basic gdb commands

- **gdb -tui fact**
 - start debugging fact
 - use the "terminal user interface"
- **then...**
 - **set disassembly-flavor intel**
 - » use the Intel/nasm assembly syntax
 - **set disassemble-next-line on**
 - » after each step, show the next instruction to be fetched and executed
 - Automate these by putting them in "~/.gdbinit"

Basic gdb commands - 2

- The "layout" and "focus" commands set up the window.
 - **layout split**
 - » show C source code and its disassembled version
 - **focus asm**
 - » move the focus to the disassembled code pane
 - **layout regs**
 - » add a pane showing the processor registers
 - these can also go into ~/.gdbinit

Running the program

- **break fact**
 - set a breakpoint where the "fact" label occurs
- **run 5**
 - start running the program, with a command-line argument of "5"
 - execution proceeds until the breakpoint is reached
- **stepi**
 - execute one assembly instruction at a time
 - abbreviated as "si"
- **C**
 - continue - run until the next breakpoint (if any)

Examining what's happened

- `x /16xg $rsp`
 - display 16 "giant words" (8 bytes) of memory, in hexadecimal, starting where the RSP (stack pointer) register points to
 - This lets you see the stack frame
- `info ...`
 - print information about variables, registers
 - "... " refers to various arguments, see "help info"
- `print ...`
 - print contents of a register; "... " is a register name
 - redundant with the "regs" pane in the layout

```

File Edit View Bookmarks Settings Help
Register group: general
rax    0x5      5
rbx    0x0      0
rcx    0x0      0
rdx    0xa     10
rsi    0x0      0
rdi    0x2      2
rbp    0x7fffffff168 0x7fffffff168
rsp    0x7fffffff168 0x7fffffff168
r8     0x7ffff7dd4060 140737351860320
r9     0x7fffffff6c3 140737489348867
r10    0x5      5
r11    0x0      0
r12    0x400490 4195472

0x400600 <fact>      push    rbp
0x400601 <fact+1>    mov     rbp, rsp
0x400604 <fact+4>    cmp     rdi, 0x2
0x400608 <fact+8>    jbe     0x400619 <fact.basecase>
0x40060a <fact.recurse> push    rdi
0x40060b <fact.recurse+1> dec     rdi
0x40060e <fact.recurse+4> call   0x400600 <fact>
0x400613 <fact.recurse+9> pop     rdi
0x400614 <fact.recurse+10> mul    rdi
0x400617 <fact.recurse+13> jmp     0x40061c <fact.end>
> 0x400619 <fact.basecase> mov     rax, rdi
0x40061c <fact>      ret
0x40061d <fact.end+1> ret
0x40061e <fact.end+2> xchg    ax, ax

Child process 10917 In: fact.basecase Line: ?? PC: 0x400619

Breakpoint 2, 0x0000000000400604 in fact ()
=> 0x0000000000400604 <fact+4>: 48 83 ff 02      cmp     rdi, 0x2
0x0000000000400608 in fact ()
=> 0x0000000000400608 <fact+8>: 76 0f jbe     0x400619 <fact.basecase>
0x0000000000400619 in fact.basecase ()
0x0000000000400619 <fact.basecase.0>: 48 03 40      mov     rax, rdi
(gdb) x/12xg $rsp
0x7fffffff168: 0x0000000000400613 0x0000000000400613
0x7fffffff178: 0x0000000000000003 0x0000000000000003
0x7fffffff188: 0x0000000000000013 0x0000000000000013
0x7fffffff198: 0x000000000000001b 0x000000000000001b
0x7fffffff1a8: 0x0000000000000005 0x0000000000000005
0x7fffffff1b8: 0x00000000004005c2 0x0000000000000000
(gdb)

```

a gdb session

- breakpoint at beginning of "fact" routine
- factorial program started with "run 5"
- "stepi" through the routine 3 times, arrived at "basecase"
- "`x/12xg $rsp`" shows top 12 entries into stack
 - each frame is three entries long

More information on commands

- **help <command>**
 - display help information about a command, within gdb
- http://www.csee.umbc.edu/~cpatel2/links/310/nasm/gdb_help.shtml
 - a nice writeup entitled "Using gdb for Assembly Language Debugging"

Command summary - 1

Command	short hand	Example	Description
run	r	run qwerty 3	start program, optional args
quit	q	q	quit out of gdb
cont	c	c	continue execution
break [addr]		break *main+5	set a breakpoint
delete n		delete 4	remove n th breakpoint
delete		delete	remove all breakpoints
info break		info break	list all breakpoints
stepi	si	si	execute 1 instruction
stepi [n]	si [n]	si 5	excute next n instructions
nexti	ni	ni	execute next instruction, stepping over function calls
nexti [n]	ni [n]	ni 5	execute next 5 instructions, stepping over function calls

Command summary - 2

Command	short hand	Example	Description
where		<code>where</code>	show where execution halted
disas [addr]		<code>disas fact</code>	disassemble at given address
info registers		<code>info regs</code>	show contents of all registers
print/d [expr]	p/d	<code>p/d \$ecx+4</code>	print expression, in decimal
print/x [expr]	p/x	<code>p/x \$rdi+4</code>	print expression, in hexadecimal
print/t [expr]	p/t	<code>p/t 55</code>	print expression, in binary
x /[FMT] [addr]		<code>x/16xg \$rsp</code>	examine memory contents in given format, at given address
display [expr]		<code>display \$rax</code>	automatically print expression each time execution stops
info display		<code>info display</code>	show list of automatic displays
undisplay [n]		<code>undisplay 3</code>	remove an automatic display
help [cmd]		<code>help x</code>	show help about a command