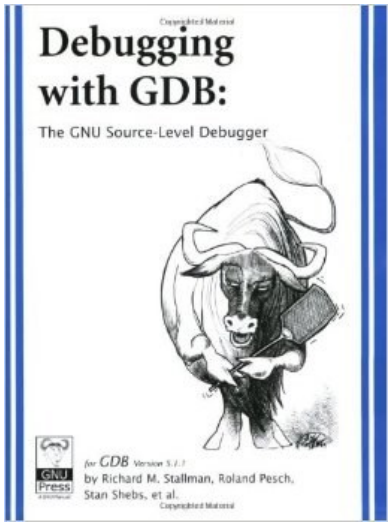
GDB and Assembly

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

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

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 = atol(argv[i]);
        f = fact(n);
        printf("fact(%2lu) is %lu\n", n, f);
    }
    return 0;
}
```

```asm
; 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:

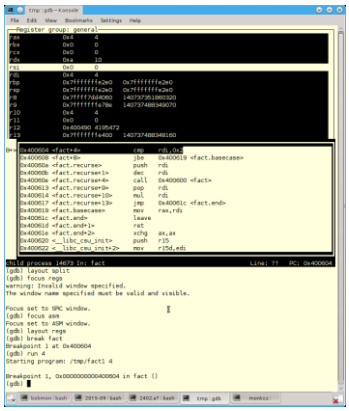
- `nasm -f elf64 -l fact1.lst -g fact.asm`
  - `-f elf64` - produce 64-bit Linux code
  - `-l fact1.lst` - produce a listing file as well
  - `-g` - include debugging symbols
- `gcc -Wall -o fact -g factmain.c fact.o`
  - `-Wall` - turn on all warnings and errors
  - `-o fact` - create an executable 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

```
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                      .recurse:
16 0000000A 57                      push rdi        ; save n for later
17 0000000B 48FFCF                  dec rdi         ; n - 1 ...
18 0000000E E8EDFFFFFF              call fact
19 00000013 5F                      pop rdi
20 00000014 48F7E7                  mul rdi         ; rax <-- rax * rdi
21 00000017 EB03                    jmp .end
22
23                      .basecase:
24 00000019 4889F8                  mov rax, rdi
25                      .end:
26 0000001C C9                      leave
27 0000001D C3                      ret
28                      ;-----------------
```

- open a tall terminal shell next to the editor



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

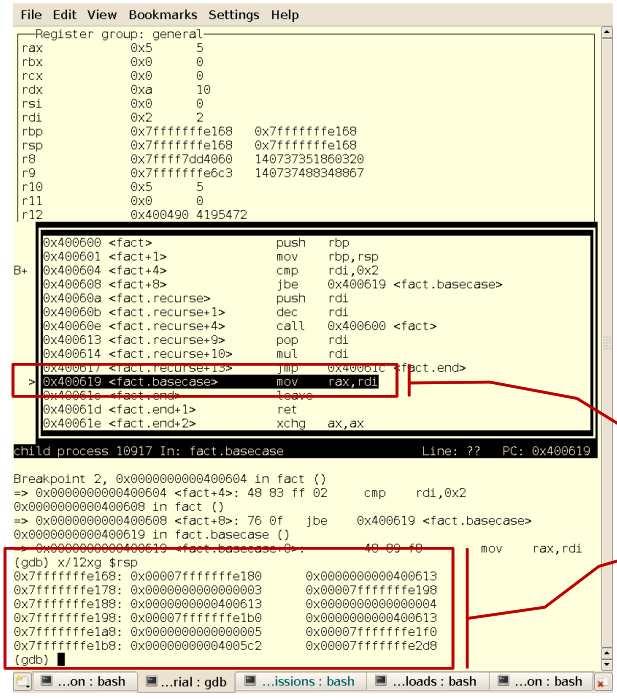
- The "layout" and "focus" commands set up the window.
  - layout split
    - » show C source code and its disassembled version
  - focus asm
    - » move the focus to the disassembled code pane
  - layout regs
    - » add a pane showing the processor registers
  - these can also go into ~/.gdbinit

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



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