GDB

This work is a derivative of Using the GNU Debugger
We have a grutor!
- Mars Park
- Wednesdays: 7-9 in B105

For your lab submittal, if your github name isn’t clearly identifiable, add an AUTHOR.txt to your JOS repository:

```markdown
My name is John Smith (jsmith@hmc.edu)
```

Lecture recordings linked from schedule

Gradescope: when submitting, associate pages with homework question
• **From bootasm.S:**

```assembly
# Set up the stack pointer and call into C.
movl    $start, %esp
call    bootmain
```

• **Later, in `bootmain()`:**

```c
// Call the entry point from the ELF header.  
// Does not return!
entry = (void(*)(void))(elf->entry);
entry();
```
HW 1: What’s on the stack?

- call bootmain pushes a return address
- The prologue in bootmain makes a stack frame:

```assembly
push   %ebp
mov    %esp,%ebp
push   %edi
push   %esi
push   %ebx
sub    $0x10,%esp
```

- The call to entry pushes a return address
HW 1: The stack when we get to 0x1000c

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7bfc</td>
<td>0x00007da4</td>
<td>bootmain() return</td>
</tr>
<tr>
<td>0x7bf8</td>
<td>0x00000000</td>
<td>saved ebp</td>
</tr>
<tr>
<td>0x7bf4</td>
<td>0x00000000</td>
<td>saved edi</td>
</tr>
<tr>
<td>0x7bf0</td>
<td>0x00000000</td>
<td>saved esi</td>
</tr>
<tr>
<td>0x7bec</td>
<td>0x00000000</td>
<td>saved ebx</td>
</tr>
<tr>
<td>0x7be8</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x7be4</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x7be0</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x7bd8</td>
<td>0x00010054</td>
<td>local var: eph</td>
</tr>
<tr>
<td>0x7bd4</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x7bd0</td>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x7bcc</td>
<td>0x00007da4</td>
<td>entry() return</td>
</tr>
</tbody>
</table>
**HW 1: How we know which local variable**

```assembly
eph = ph + elf->phnum;
7d5a:  0f b7 05 2c 00 01 00    movzwl 0x1002c,%eax
7d61:  c1 e0 05                shl    $0x5,%eax
7d64:  01 f0                   add    %esi,%eax
7d66:  89 45 e4                mov    %eax,-0x1c(%ebp)
```
Must use `i386-elf-gdb`

When you run `make qemu-gdb` or `qemu-nox-gdb`:
- `make` creates a `.gdbinit` file
- You need `i386-elf-gdb` to read this file to know which process to communicate with:
  - Run `i386-elf-gdb` from the same directory (in a separate window)
  - For jos, you can make `gdb`

```bash
...  
echo + target remote localhost:28178\n
target remote localhost:28178

  
echo + symbol-file kernel\n
  symbol-file kernel

  .gdbinit
```
• **Run:**
  • `help`, or
  • `help command`

• **All commands may be abbreviated (if unambiguous)**
  • For example, `continue` or `cont` or `co` or `c`

• **There are some special abbreviations**
  • `stepi == si`
  • `nexti == ni`
GDB

- **Stepping**
  - `step`: steps a single source line
  - `stepi`: steps a single x86 instruction
  - `next`: steps a single source line (but skips over subroutine calls)
  - `nexti`: steps a single x86 instruction (but skips over CALL instructions)
GDB

- **continue**
  - Runs code until a breakpoint or Ctrl-C

- **finish**
  - Runs code until current function returns

- **advance location**
  - Runs code until instruction pointer reaches *location*
  - Shortcut for:
    - break *location*
    - cont
    - delete breakpoint
GDB: Breakpoints

• **break location**
  • Sets a breakpoint at *location*
  • *location* can be:
    - memory address: *0x7c00*
    - name: mon_backtrace, “monitor.c:71”

• **Modify breakpoints with:**
  • delete
  • disable
  • enable
GDB: Conditional breakpoints/watchpoints

• break location if condition
  • Breaks at location if condition is true

  (gdb) b test_backtrace if x == 3

• watch expression
  • Breaks if value of expression changes

  (gdb) watch myvar

• watch -l address
  • Breaks if contents of memory at address changes

rwatch will stop when expression is read
GDB: Examining Memory

- **x** prints the raw contents of memory in whatever format you want:
  - **x/x**: hex
  - **x/d**: decimal
  - **x/i**: instructions

- **print expression**
  - Evaluates the C expression and prints it

```
(gdb) p *((struct elfhdr *) 0x10000)
$2 = {magic = 1179403647,
    elf = "\001\001\001\000\000\000\000\000\000\000\000", type = 2, machine = 3,
    version = 1, entry = 1048588, phoff = 52, shoff = 148104, flags = 0,
    ehsiz = 52, phentsize = 32, phnum = 1, shentsize = 40, shnum = 18,
    shstrndx = 15}
```

```
(gdb) x/10x 0x10000
0x10000: 0x464c457f 0x00010101 0x00000000 0x00000000
0x10010: 0x00030002 0x00000001 0x0010000c 0x00000034
0x10020: 0x00024288 0x00000000
```
GDB: Examining

(gdb) info registers
eax            0x40     64
ecx            0x0      0
edx            0x1f0    496
ebx            0x0      0
esp            0x7bc4   0x7bc4
ebp            0x7bf8   0x7bf8
esi            0x0      0
edi            0x0      0
eip            0x7d3f   0x7d3f
eflags         0x46     [ PF ZF ]
cs             0x8      8
ss             0x10     16
ds             0x10     16
es             0x10     16
fs             0x0      0
gs             0x0      0

(gdb) info frame
Stack level 0, frame at 0xf0117f80:
eip = 0xf0100040 in test_backtrace (kern/init.c:13); saved eip 0xf0100069
called by frame at 0xf0117fa0
source language c.
Arglist at 0xf0117f78, args: x=2
Locals at 0xf0117f78, Previous frame's sp is 0xf0117f80
Saved registers:
eip at 0xf0117f7c
GDB: Examining

(gdb) backtrace
#0  test_backtrace (x=2) at kern/init.c:13
#1  0xf0100069 in test_backtrace (x=3) at kern/init.c:16
#2  0xf0100069 in test_backtrace (x=4) at kern/init.c:16
#3  0xf0100069 in test_backtrace (x=5) at kern/init.c:16
#4  0xf010010e in i386_init () at kern/init.c:44
#5  0xf010003e in ?? () at kern/entry.S:80
Backtrace stopped: Not enough registers or memory available to unwind further
Layouts

layout asm

remote Thread 1 In: test_backtrace  Line: 13  PC: 0xf0100040
(gdb)

layout reg

Register group: general

eax    0x2    2
ecx    0x3d4  980
edx    0x3d5  981
ebx    0x3     3
esp    0xf0117f7c 0xf0117f7c
ebp    0xf0117f98 0xf0117f98
Other tricks

- **set var** will change the value of a variable
  - `set var foo=3`

- GDB reads symbol file to determine variable names, source location, etc.
  - xv6 (homeworks) start up with:
    - `-symbol-file kernel`
  - JOS (labs) start up with:
    - `-symbol-file obj/kern/kernel`
  - That’s why in lab 1: `break bootmain` doesn’t work

- If you want to debug a JOS user program, use:
  - `-symbol-file obj/user/progname`
Summary

• Learn gdb well

• A good cheatsheet:
  • https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

Also, learn bash and vim/emacs well!
Running

# gdb <program> [core dump]
Start GDB (with optional core dump).

# gdb --args <program> <args...>
Start GDB and pass arguments

# gdb --pid <pid>
Start GDB and attach to process.

set args <args...>
Set arguments to pass to program to be debugged.

run
Run the program to be debugged.

kill
Kill the running program.

Breakpoints

break <where>
Set a new breakpoint.

delete <breakpoint#>
Remove a breakpoint.

clear
Delete all breakpoints.

enable <breakpoint#>
Enable a disabled breakpoint.

disable <breakpoint#>
Disable a breakpoint.

Watchpoints

watch <where>
Set a new watchpoint.

delete(enable/disable) <watchpoint#>
Like breakpoints.

function_name
break/watch the named function.

line_number
Break/watch the line number in the current source file.

file:line_number
Break/watch the line number in the named source file.

Conditions

break/watch <where> if <condition>
Break/watch at the given location if the condition is met.
Conditions may be almost any C expression that evaluate to true or false.

condition <breakpoint#> <condition>
Set/change the condition of an existing break- or watchpoint.

Examining the stack

backtrace
where
Show call stack.

backtrace full
where full
Show call stack, also print the local variables in each frame.

frame <frame#>
Select the stack frame to operate on.

Stepping

step
Go to next instruction (source line), diving into function.
next
Go to next instruction (source line) but don't dive into functions.
finish
Continue until the current function returns.
continue
Continue normal execution.

Variables and memory

print/format <what>
Print content of variable/memory location/register.

display/format <what>
Like „print“, but print the information after each stepping instruction.

undisplay <display#>
Remove the „display“ with the given number.

enable display <display#>

disable display <display#>
En- or disable the „display“ with the given number.

x/nu <address>
Print memory.

n: How many units to print (default 1).
f: Format character (like „print“).
u: Unit.
Unit is one of:

b: Byte,
h: Half-word (two bytes)
w: Word (four bytes)
g: Giant word (eight bytes)).
GDB cheatsheet - page 2

Format

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Pointer.</td>
</tr>
<tr>
<td>c</td>
<td>Read as integer, print as character.</td>
</tr>
<tr>
<td>d</td>
<td>Integer, signed decimal.</td>
</tr>
<tr>
<td>f</td>
<td>Floating point number.</td>
</tr>
<tr>
<td>o</td>
<td>Integer, print as octal.</td>
</tr>
<tr>
<td>s</td>
<td>Try to treat as C string.</td>
</tr>
<tr>
<td>t</td>
<td>Integer, print as binary (t = &quot;two&quot;).</td>
</tr>
<tr>
<td>u</td>
<td>Integer, unsigned decimal.</td>
</tr>
<tr>
<td>x</td>
<td>Integer, print as hexadecimal.</td>
</tr>
</tbody>
</table>

<what>

expression
Almost any C expression, including function calls (must be prefixed with a cast to tell GDB the return value type).

file_name::variable_name
Content of the variable defined in the named file (static variables).

function::variable_name
Content of the variable defined in the named function (if on the stack).

{type}address
Content at address, interpreted as being of the C type type.

$register
Content of named register. Interesting registers are $esp (stack pointer), $ebp (frame pointer) and $eip (instruction pointer).

Manipulating the program

set var <variable_name> =<value>
Change the content of a variable to the given value.

return <expression>
Force the current function to return immediately, passing the given value.

Sources

directory <directory>
Add directory to the list of directories that is searched for sources.

list
list <filename>::<function>
list <filename>::<line_number>
list <first>,<last>
Shows the current or given source context. The filename may be omitted. If last is omitted the context starting at start is printed instead of centered around it.

set listsize <count>
Set how many lines to show in „list“.

Signals

handle <signal> <options>
Set how to handle siglines. Options are:
(no)print: (Don't) print a message when signals occurs.
(no)stop: (Don't) stop the program when signals occurs.
(no)pass: (Don't) pass the signal to the program.

Informations

disassemble
Disassemble the current function or given location.

info args
Print the arguments to the function of the current stack frame.

info breakpoints
Print informations about the break- and watchpoints.

info display
Print informations about the „displays“.

info locals
Print the local variables in the currently selected stack frame.

info sharedlibrary
List loaded shared libraries.

info signals
List all signals and how they are currently handled.

info threads
List all threads.

show directories
Print all directories in which GDB searches for source files.

show listsize
Print how many are shown in the „list“ command.

whatis <variable_name>
Print type of named variable.

Threads

thread <thread#>
Chose thread to operate on.