Princeton University
COS 217: Introduction to Programming Systems
GDB Tutorial and Reference
for x86-64 Assembly Language

Part 1: Tutorial

Motivation

Suppose you're developing the **power.s** program. Further suppose that the program assembles and links cleanly, but is producing incorrect results at runtime. What can you do to debug the program?

One approach is temporarily to insert calls of **printf(...)** throughout the code to get a sense of the flow of control and the values of variables at critical points. That's fine, but often is inconvenient. It is especially inconvenient in assembly language: the calls of **printf()** will change the values of registers, and thus may corrupt the very data that you wish to view.

An alternative is to use **gdb**. **gdb** allows you to set breakpoints in your code, step through your executing program one line at a time, examine the contents of registers and memory at breakpoints, etc.

Building for gdb

To prepare to use **gdb**, build the program with **gcc217** using the **-g** option:

```
$ gcc217 -g power.s -o power
```

Running GDB

The next step is to run **gdb**. You can run **gdb** directly from the shell. But it's much handier to run it from within Emacs. So launch Emacs, with no command-line arguments:

```
$ emacs
```

Now call the **emacs gdb** function via these keystrokes:

```
<Esc key> x gdb <Enter Key> power <Enter key>
```

At this point you are executing **gdb** from within Emacs. **gdb** is displaying its **(gdb)** prompt.

Running Your Program

Issue the **run** command to run the program:

```
(gdb) run
```

gdb runs the program to completion, indicating that the "Program exited normally." (**gdb** also displays the cryptic message "Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.166.el6_7.1.x86_64". That message is innocuous; ignore it.)

Incidentally and importantly, command-line arguments and file redirection can be specified as part of the **run** command. For example the command **run 1 2 3** runs the program with command-line arguments 1, 2, and 3, and the command **run < myfile** runs the program with its stdin redirected to myfile.

Using Breakpoints

Set a breakpoint near the beginning of the **main()** function using the **break** command:

```
(gdb) break main
```

Run the program:

```
(gdb) run
```

gdb pauses execution at the beginning of the **main()** function. It opens a second window in which it displays your source code, with the about-to-be-executed line of code highlighted.

Issue the **continue** command to tell command **gdb** to continue execution past the breakpoint:

```
(gdb) continue
```

gdb continues past the breakpoint at the beginning of **main()**, and executes the program to completion.

Stepping Through the Program

Run the program again:

```
(gdb) run
```

Execution pauses at the beginning of the **main()** function. Issue the **next** command to execute the next instruction of your program:

```
        (gdb) next
```

Continue issuing the **`next`** command repeatedly until the next instruction to be executed is **`call printf`**.

The **`step`** command is the same as the **`next`** command, except that it commands **`gdb`** to step into a called function which you have defined. The **`step`** command will not cause **`gdb`** to step into a standard C function. Incidentally, the **`stepi`** (step instruction) command will cause **`gdb`** to step into any function, including a standard C function.

Examining Registers

Issue the **`info registers`** command to examine the contents of the registers:

```
        (gdb) info registers
```

Issue the **`print`** command to examine the contents of any given register. Some examples:

```
        (gdb) print/d $rsi    Print as a decimal integer the 8 bytes
                              which are the contents of register RSI
        (gdb) print/a $rdi    Print as a hexadecimal address the 8 bytes
                              which are the contents of register RDI
        (gdb) print/d $eax    Print as a decimal integer the 4 bytes
                              which are the contents of register EAX
```

Note that you must precede the name of the register with **`$`** rather than `%`.

Examining Memory

Issue the **`x`** command to examine the contents of memory at any given address. Some examples:

```
        (gdb) x/d &lBase       Examine as a decimal integer the 4 bytes
                               of memory at lBase (not really meaningful)
        (gdb) x/gd &lBase      Examine as a "giant" decimal integer the
                               8 bytes of memory at lBase
        (gdb) x/c &cResult     Examine as a char the 1 byte of memory
                               at cResult
        (gdb) x/s &cResult     Examine as a string the bytes in memory
                               at cResult
        (gdb) x/s $rdi         Examine as a string the bytes of memory
                               at the address contained in register RDI
```

Quitting GDB

Issue the **`quit`** command to quit **`gdb`**:

```
(gdb) quit
```

Then, as usual, type:

```
<Ctrl-x> <Ctrl-c>
```

to exit **emacs**.

Command Abbreviations

The most commonly used **gdb** commands have one-letter abbreviations (**r**, **b**, **c**, **n**, **s**, **p**). Also, pressing the Enter key without typing a command tells **gdb** to reissue the previous command.

Part 2: Reference

gcc217 ... -o *program*	Assemble and link with debugging information
gdb [-d *sourcefiledir*] [-d *sourcefiledir*] ... *program* [*corefile*]	Run `gdb` from a shell
ESC x gdb [-d *sourcefiledir*] [-d *sourcefiledir*] ... *program* [*corefile*]	Run `gdb` from Emacs

Miscellaneous	
quit	Exit `gdb`.
directory [*dir1*] [*dir2*] ...	Add directories *dir1*, *dir2*, ... to the list of directories searched for source files, or clear the directory list.
help [*cmd*]	Print a description command *cmd*

Running the Program	
run [*arg1*],[*arg2*] …	Run the program with command-line arguments *arg1*, *arg2*, ...
set args *arg1 arg2* ...	Set program's the command-line arguments to *arg1*, *arg2*, ...
show args	Print the program's command-line arguments.

Using Breakpoints	
info breakpoints	Print a list of all breakpoints.
break *label*	Set a breakpoint at the memory address denoted by *label*.
break *fn*	Set a breakpoint at the third instruction of function *fn*.
condition *bpnum expr*	Break at breakpoint *bpnum* only if expression *expr* is non-zero (TRUE).
commands [*bpnum*] *cmd1 cmd2* ...	Execute commands *cmd1*, *cmd2*, ... whenever breakpoint *bpnum* (or the current breakpoint) is hit.
continue	Continue executing the program.
kill	Stop executing the program.
delete [*bpnum1*][,*bpnum2*]...	Delete breakpoints *bpnum1*, *bpnum2*, ..., or all breakpoints.
clear [**addr*]	Clear the breakpoint at memory address *addr*, or the current breakpoint.
clear [*fn*]	Clear the breakpoint at function *fn*, or the current breakpoint.
disable [*bpnum1*][,*bpnum2*]...	Disable breakpoints *bpnum1*, *bpnum2*, ..., or all breakpoints.
enable [*bpnum1*][,*bpnum2*]...	Enable breakpoints *bpnum1*, *bpnum2*, ..., or all breakpoints.

Stepping through the Program	
next	"Step over" the next instruction.
step	"Step into" the next instruction.
finish	"Step out" of the current function.

Examining Registers and Memory	
info registers	Print the contents of all registers.
print/*f* $*reg*	Print the contents of register *reg* using format *f*. The format can be x (hexadecimal), d (decimal), u (unsigned decimal), o (octal), a (address), c (character), or f (floating point).
x/*rsf addr*	Examine the contents of memory at address *addr* using repeat count *r*, size *s,* and format *f*. The repeat count is optional; it defaults to 1. The size is optional; it can be b (1 byte), h (2 bytes), w (4 bytes), or g (8 bytes). The format can be x (hexadecimal), d (decimal), u (unsigned decimal), o (octal), a (address), c (character), f (floating point), s (string), or i (instruction).
x/*rsf* $*reg*	Examine the contents of memory at the address contained in register *reg*.
info display	Print the display list.
display/*f* $*reg*	At each break, print the contents of register *reg* using format *f* (as with a print command).
display/*s*i *addr*	At each break, print the contents of memory at address *addr* using size *s* (as with an x command).
display/*s*s *addr*	At each break, print the string of size *s* that begins in memory at address *addr* (as with an x command).
undisplay *displaynum*	Remove *displaynum* from the display list

Examining the Call Stack	
where	Print the call stack.
backtrace	Print the call stack.
frame	Print the top of the call stack.
up	Move the context toward the bottom of the call stack.
down	Move the context toward the top of the call stack