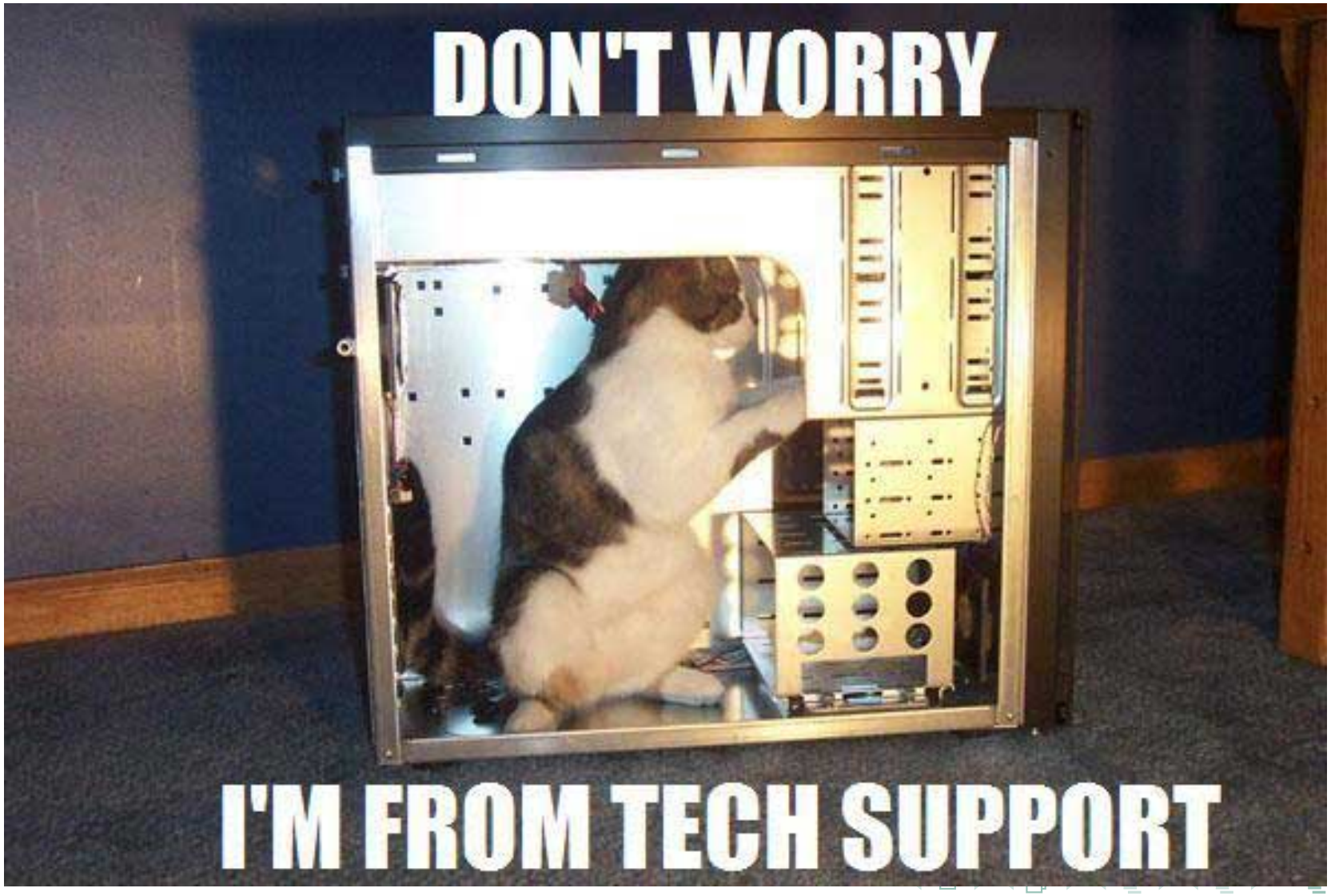
Unix Debugging at Ohio University

Dr. Shawn Ostermann



Talk Overview

- Helpful compiling tips for 'debuggability'
- gdb
- gdb interfaces
- examples
- Info on the web at:

  `http://ace/~osterman/talks/gdb`

Gcc
Object Files and Executables

To create an object from a C module **x1.c**, use the command

```
gcc -c x1.c
```

If this command is successful then **gcc** will create an object file **x1.o**.

An *object file* is a file that contains linkable code plus symbol table information.)

If one of the n modules x1.c, ..., xn.c contains the "main" function, then we can create an executable from the object files as follows. (Here I assume that $n = 4$.)

```
gcc x1.o x2.o x3.o x4.o
```

This will create the executable **a.out**. To create an executable named **x**, use the command

```
gcc -o x x1.o x2.o x3.o x4.o
```

More useful command-line options

- Useful gcc options

  `-g` Include debugging information into the object/executable. (Note that this must be done at both the object creation and linking stage.)
  `-Wall` Generate all warning messages.
  `-Werror` Make all warning messages fatal.
  `-O` Optimize (same as -O1, the default).
  `-O0` Do **NOT** optimize.
  `-O2` Perform more optimizations.
  `-O3` Perform even more optimizations.

- I usually use

  ```
  -g -Wall -Werror -O2
  ```

Starting GDB

- Normally, to debug a program `myprog`, you would type

```
gdb myprog
```

- If the program has “dumped core” and you want to see what happened, you would type

```
gdb myprog core
```

  - Mostly useful if the problem can’t be easily reproduced
  - Often less reliable than the first version when the problem **can** be reproduced

Running GDB

- To run the program:

```
run
```

- To run the program with args

```
run -f thisfile thatarg
```

- You don't need to re-specify the arguments the next time you type `run`, it will remember them
- If the program always "dumps core", then just type `run` and wait for it to die, then see what the state of the program is
  - Much more efficient, often, than using print statements!

Running GDB (continued)

- You can also redirect input/output

```
run -f thisfile thatarg > output
run -f thisfile thatarg 2> errfile
run < infile
```

- You can even pipe the output

```
run -this -that | more
```

Show Examples

- coredump
  - where
  - list
  - print
- memory
  - where
  - list
  - print
- optim3
- optim0

Optimizing and Debugging

- Optimizing your code has several effects
  - Re-arranges lines to make things run faster
  - “inlines” function calls
  - “unrolls” loops
  - Removed unnecessary code
  - Removed unused code
- Obviously this will be confusing when debugging your program
- Suggestions
  - Don’t try to debug a `-O3` program!
  - When all else fails, recompile with `-O0` (“Oh zero”)

Typing to GDB

- Uses emacs command line editing

control-p	previous command
control-n	next command
control-f	forward a character
control-b	back a character
control-k	kill the rest of the line
control-a	beginning of line
control-e	end of line
control-d	delete a character
control-s	forward search for command
control-r	backward search for command

  - Other characters “auto insert”

Printing

- You can use most C/C++ formats for printing
  - `print x`
  - `print x+2`
  - `print ptr`
  - `print *ptr`
  - `print ptr->next`
  - `print/x ptr->next`
  - `print func(2,3,4)`
  - `print func(2,3,4)+5`
  - `print (void) printf("testing 1 2 3")`

Listing the program

- List “next screen”

```
list
```

- List starting at a particular line

```
list 1
```

- List starting at a particular line by file

```
list main.c:1
```

Stack Frames

- Useful commands:

```
(gdb) help
List of classes of commands:
(gdb) where
#0 0x10dd0 in meaning_of_life (param=0) at debug.c:55
#1 0x10e10 in meaning_of_life (param=-2147483648) at debug.c:58
#2 0x10e10 in meaning_of_life (param=536870912) at debug.c:58
...
#16 0x10e10 in meaning_of_life (param=2) at debug.c:58
#17 0x10d6c in main (argc=1, argv=0xffbee97c) at debug.c:30
```

- Other commands
  - `up`
  - `down`
  - `frame N`
  - `info locals`

Breakpoints

- Setting breakpoints by function:

```
break main
```

- Setting breakpoints by line:

```
break 10
```

- With multiple files:

```
break main.c:10
break util.c:subr
```

- Listing/deleting breakpoints

```
show breakpoints
delete 3
```

Conditional Breakpoints

- Sometimes stop at a particular

```
break 27 if (x == 10)
```

  - (beware the == !!!!)
- Keep in mind that when it stops (for ALL breakpoints), it has not yet run that line
- Conditional breakpoints can make the program VERY slow, so use them carefully!

Setting Variables

- You can also **change** variables

```
set x=5
print y=10
```

Useful Hints
attach

- For programs that “fork off into the background” or that you forgot to debug, you can use `attach`

```
KSH: ./myprog &
KSH:picard> /usr/bin/ps
PID TTY TIME CMD
29822 pts/5 0:00 ksh
6816 pts/5 0:00 pprint
KSH: gdb myprog
(gdb) attach 6816
(gdb)
```

Examples

- Work through example 2
  - Running
  - whatis
  - setting variables
  - list
  - continue
- Emacs interface
  - conditional breakpoints
  - calling routines

Information about Variables

- You can ask GDB what things are:

```
(gdb) whatis x
type = int
(gdb) whatis main
type = int (int, char **)
(gdb) whatis hidden
type = int ()
```

Useful Hints
Exit()

- How do you debug a program that exits, but doesn't die?

```
(gdb) break _exit()
(gdb) run
(gdb) where
```

Getting Help

- Type:

```
(gdb) help
List of classes of commands:

running - Running the program
stack - Examining the stack
data - Examining data
breakpoints - Making program stop at certain points
files - Specifying and examining files
status - Status inquiries
support - Support facilities
user-defined - User-defined commands
aliases - Aliases of other commands
obscure - Obscure features
internals - Maintenance commands
```

Getting Help

- You can then get help on specific stuff

```
(gdb) help running
List of commands:
attach - Attach to a process or file outside of GDB
continue - Continue program being debugged
detach - Detach a process or file previously attached
finish - Execute until selected stack frame returns
handle - Specify how to handle a signal
interrupt - Interrupt the execution of the debugged program
jump - Continue program being debugged at specified line or address
kill - Kill execution of program being debugged
next - Step program
[...]

(gdb) help attach
Attach to a process or file outside of GDB.
This command attaches to another target, of the same type as your las
[...]
```

Useful Hints
Startup File

- You can store useful startup commands in the file “.gdbinit” in your home directory
- For examply, mine contains “`set print pretty`”

Working at Breakpoints

- Execute the next line or instruction
  - step
  - stepi
  - step NUM
  - stepi NUM
- Same, but including functions
  - next
  - nexti
  - next NUM
  - nexti NUM

Misc

- Other useful commands:

control-c	stop the program
continue	start the program running from where you stopped it
finish	run until selected stack frame returns
return	make the current stack frame return (may be fatal)
step	execute until another line is reached
next	execute line, including function calls
disass	disassemble (look up args)
set print pretty	“prettier” printing of structures

- Unambiguous prefixes are good enough
  - “c” is continue
  - “n” is next

GDB Interfaces

- For quick debugging, I recommend just using the command line interface. It's extremely fast to start up
  - The correct place to start for finding where a 'coredump' is happening
- For more complicated debugging, expecially when you don't know the source code well, a 'bigger' interface is handy
  - Of course, my favorite interface is emacs
  - Another interface that some people like, but that I haven't used much, is 'ddd'

Advanced

- See the `attach` command for debugging a program that is already running
- Get the “pink sheet” in the lab (or from my website)
- Run gdb inside emacs!!!
- Find a good web site with a gdb tutorial and spend some time with it
- Grab one of the gdb books from an Ohio University computer, through `http://proquest.safaribooksonline.com/`