C Bootcamp

Day 4

CS3600, Northeastern University

Alan Mislove

Slides adapted from Anandha Gopalan's CS132 course at Univ. of Pittsburgh
and the CS240 course at Purdue

C Debugging

Debugging with gdb

GDB is a debugger that helps you debug your program

Time you spend learning gdb will save you days of debugging time

You need to compile with the `-g` option to use gdb

The `-g` option adds debugging information to your program

```
gcc -g -o hello hello.c
```

Should be done automatically in all `Makefile`s we give you

Running gdb

To run a program with gdb type

```
gdb exename
....
(gdb)
```

Then set a breakpoint in the main function

- Marker in your program that will make the program stop
- Return control back to gdb

```
(gdb) break main
```

Now run your program

- If your program has arguments, you can pass them after run

```
(gdb) run arg1 arg2 ... argN
```

Stepping through

Your program will start and will stop at `main`

```
gdb>
```

You have the following commands to run your program step by step

```
(gdb) step
```

It will run the next line of code and stop

If it is a function call, it will enter into it

```
(gdb) next
```

It will run the next line of code and stop

If it is a function call, it will go through it

If the program is running without stopping, regain control `CTRL-C`

Setting breakpoints

You can set breakpoints in a program in multiple ways:

```
(gdb) break function
```

Set a breakpoint in a function

```
(gdb) break line
```

Set a break point at a line number in the current file

```
(gdb) break file:line
```

Set a break point at a line number in a specific file

Inspecting the stack

The command

```
(gdb) where
```

- Prints the current function being executed
- And the chain of functions that are calling that function
  - This is also called the backtrace

Example:

```
(gdb) where
#0  main () at test_mystring.c:22
#1  test () at test_mystring.c:38
(gdb)
```

Inspecting variables

To print the value of a variable

```
(gdb) print variable
```

Will automatically print char*s and arrays

```
(gdb) print i
$1 = 5
(gdb) print s1
$1 = 0x10740 "Hello"
(gdb) print stack[2]
$1 = 56
(gdb) print stack
$2 = {0, 0, 56, 0, 0, 0, 0, 0, 0, 0}
(gdb)
```

Catching seg faults

If your program seg faults, gdb will catch it

```
(gdb) run
Starting program: /home/amislove/a.out
test string

Program received signal SIGSEGV, Segmentation fault.
0x4007fc13 in _IO_getline_info () from /lib/libc.so.6

(gdb) backtrace
#0  0x4007fc13 in _IO_getline_info () from /lib/libc.so.6
#1  0x4007fb6c in _IO_getline () from /lib/libc.so.6
#2  0x4007ef51 in fgets () from /lib/libc.so.6
#3  0x80484b2 in main (argc=1, argv=0xbffffaf4) at segfault.c:10
#4  0x40037f5c in __libc_start_main () from /lib/libc.so.6
```

Other C debugging tools

Purify

- Checks code at runtime
- Looks for errors like buffer overflows, accessing unallocated memory

Valgrind

- Tool to help find memory leaks
- Tracks allocation, tells you where memory allocated but never freed

Shark, Performance Tools

- OS X has many tools built into Developer Tools

Using Makefiles

Makefiles

`make` is an early precursor to `ant`

- Uses a `Makefile`, which holds the build instructions

In this class, I'll give you the `Makefile`

- But, you may want/need to extend it

Basic idea: Dependency graph

- `make` determines what requires what
  - Builds graph
- Also determines what needs to be updated
  - Based on file timestamps
- Executes commands, stops if error occurs

Makefile format

Unfortunately, `Makefile`s have a somewhat archaic format

```
target: [dependency1] [dependency2] ... [dependencyN]
        command1
        command2
        ...
        commandN
```

Basically, says `target` depends on targets `dependency[1–N]`

- And, if those exist, build `target` by executing `command[1–N]`

Note that `commands` *must* be indented with <tab> characters

- Otherwise, you'll be debugging your `Makefile`

Makefile variables

- All variables are accessed with `$(name)`
  - Defined with `=`
  - Built-in variables include `$(input)` [`$<`], `$(output)` [`$@`], `$(inputs)` [`$^`]

- A number of built-in functions
  - Use file wildcards with `$(wildcard` *pattern*`)`
  - Remove/add suffixes with `$(addsuffix` *suffix paths*`)`, `$(basename` *paths*`)`

- Can express patterns with the `%` character

```
CC = gcc

%.o: %.c
   $(CC) -c $< -o $@
```

Example Makefile

```
CFILES = $(wildcard *.c)

cp%: cp%.c
        gcc -std=c99 -O0 -g -lm -Wall -pedantic -Werror -o $@ $<

all: $(basename $(CFILES))

test: all
        ./test $(basename $(CFILES))

clean:
        rm $(basename $(CFILES))
```

Debugging Makefiles

Sometimes, `make` will use built-in rules

- E.g., compile C files with `gcc`
- Can disable these with `make -r`

Sometimes, `make` doesn't do what you want

- Executes different commands than you expect
- Can debug with `make -n`
  - Just prints commands to be executed

UNIX Shell

Shell environment

Shell environment

- Consists of a set of variables with values
- Important for the shell and the programs that run from the shell
- You can define new variables, change the values

Usually set up in `.bashrc`, `.tshrc` files

Examples

- `PATH` determines where to look for executables
- `SHELL` indicates the type of shell you are using

```
bash% echo $PATH
/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
```

Viewing/setting env variables

```
bash% export FOO=BAR
bash% echo $FOO
BAR
bash% unset FOO
bash% echo $FOO

bash% export
declare -x CLICOLOR="1"
declare -x COMMAND_MODE="unix2003"
declare -x HOSTNAME="joshua"
....
```

Configuration files

When bash is executed, it reads and runs certain configuration files:

- `.profile`, `.bash_profile`: runs when you log in
  - Contains one time initialization, like `TERM`, `HOME` etc

- `.bashrc`: run each time another bash process is invoked
  - Sets lots of variables, like `PATH`, `HISTORY` etc

Only modify the lines that you fully understand!

- Can cause very bad errors if not careful

E.g., Adding the line `logout` to the `.profile` file is bad

- Will cause you to be logged out every time you log in
- Probably not what you want