Debuggers

Pat Pannuto / Marcus Darden

Administrivia

- Check the score! Turning in Advanced Homeworks…
- 398 is an experimental course.
- Course evaluations, PLEASE!

What Does gdb Do?

Yes

- Start your program (with options and arguments)
- Stop your program
- Allow you to see into registers and memory
- Allow you to change values manually during execution

What Does gdb Do?

Yes

- Start your program (with options and arguments)
- Stop your program
- Allow you to see into registers and memory
- Allow you to change values manually during execution

No

- **MAGIC**

How Do I gdb?

To debug a program with gdb, simply put `gdb` in front of the program, i.e.:

How Do I gdb?

To debug a program with gdb, simply put `gdb` in front of the program, i.e.:

```
> ./prime # running normally
> gdb ./prime # debugging the program with gdb
```

How Do I gdb?

To debug a program with gdb, simply put `gdb` in front of the program, i.e.:

```
> ./prime # running normally
> gdb ./prime # debugging the program with gdb
```

One annoying gotcha shows up if the program to debug takes any options. The simple prime program does not, but if it did:

```
> ./prime --imaginary-option # running normally
> gdb ./prime --imaginary-option # will not work
gdb: unrecognized option '--imaginary-option'
> gdb --args ./prime --imaginary-option # gdb will ignore everything after --a
rgs
```

gdb Commands

`run`

- Starting gdb will not run your program by default. You must use the `run` command to begin execution.
- Using `run` will start your program with the options originally specified, or you can pass new options with `run`.

```
(gdb) run --different-option
```

- If your project is recompiled, each `run` will automatically reload the new version. Debugging is easier if you don't quit gdb, but leave it running in a separate terminal.

gdb Commands

`backtrace`, `up`, `down`, `frame`, `print`

- While your program is running, it has a function call stack that is built up with frames that hold parameters, locals, and register information for each invocation. Consider math.c:

```c
#include <stdio.h>
int subtract (int a, int b) { return a - b; }
int divide (int a, int* b) { return a / *b; }
int do_math (int x, int y, int z) {
    int temp = subtract(x, y);
    temp = divide(z, &temp);
    return temp;
}
int main () {
    int temp;
    temp = do_math(10, 10, 20);
    printf("Result: %d\n", temp);
    return 0;
}
```

Function call stack (growing to the right)

main

main -> do_math

main -> do_math -> subtract

main -> do_math

main -> do_math -> divide

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `set`

- Look at your source with `list` or `list <function>`

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `set`

- Look at your source with `list` or `list <function>`
- Stop and start your program with `break` and `continue`

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `set`

- Look at your source with `list` or `list <function>`
- Stop and start your program with `break` and `continue`
- Take things at your own pace with `step` (into) and `next`

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `set`

- Look at your source with `list` or `list <function>`
- Stop and start your program with `break` and `continue`
- Take things at your own pace with `step` (into) and `next`
- Make a change to variables and registers with `set`

More on breakpoints

- Generally specified by filename:linenumber
- Will also work in context
- List all current breakpoints with `info breakpoints`
- Remove with `delete <number>` or `disable <number>` until later
- Skip over working code with breakpoints on either side and `continue`

Attendance:

http://tinyurl.com/c4cs-f16-dbug

Open Problems with Debugging

Look at `inf.c`