Operating Systems Lab (C+Unix)

Enrico Bini

University of Turin

Outline

Debugging

- Debugging is very helpful to find issues in programs
- `gdb` is the debugging engine
- as everything in Unix/Linux, it is very powerful and very cryptic

Launching gdb

1. As examples, we debug a code terminating with Segmentation Fault
   - *test-seg-fault.c*
2. To properly debug a program, it must be compiled with the flags:
   - -g, to add extra information to the object files
   - -O0, to turn all code optimization off -O0 is a valid alternative

```
gcc -g -O0 test-seg-fault.c
```

3. To run a program within the debugger

```
gdb <name-of-executable>
```

4. Even if the executable should have some command-line options, just ignore it

gdb commands 1/2

- It appears the prompt
  `(gdb)`
- here gdb commands may be entered
  - `help`, help on commands
  - `list`, list the source code
    - `list <line-number>`, list the source code starting from `<line-number>` of the current file being debugged
  - `break <line-number>`, insert a breakpoint at line `<line-number>` (always insert a breakpoint before running)
    - `break <filename>:<line-number>`, insert a breakpoint at line `<line-number>` in file `<filename>`
  - `info b`, show current breakpoints. Each breakpoint is identified by a numeric ID
  - `del <ID>`, delete the breakpoint number `<ID>`
  - `run`, run the executable (until the first breakpoint)
    - `run <command-line-args>`, run the executable with the specified command-line arguments

gdb commands 2/2

- `next`, execute a line of code: if a function call, invokes the call
- `step`, execute a line of code: if a function call, step in the function
- `cont`, continue the execution until the next breakpoint,
- `print <expression>`, evaluate and print `<expression>`
- `display <expression>`, evaluate and print `<expression>` **at every step**
- `bt`, "backtrace" shows all the called functions on the stack
- `quit`, to quit the debugger