Debugging Tools

CS 270: Systems Programming

Instructor:
Raphael Finkel

Gdb: The Gnu debugger

- It runs on most computers and operating systems.
- It allows you to examine a running executable program.
- It does not require the source code that created the executable.
- If you have the source code, compile the program to include debugging information (such as symbol tables and line numbers):
  - `gcc –g hello.c`
- To run gdb on an executable file:
  - `gdb a.out`
- Gdb then presents a command line interface.
- Using the command line interface, you can get help by typing
  - `help`
  - `help` *`subtopic`* (where *subtopic* is one of the main subtopic areas listed by typing help). Examples include:
    - `help data`
    - `help breakpoints`
  - `help` *`command`* (where *command* is a gdb command). Examples include:
    - `help print`
    - `help run`

Breakpoints

- **Breakpoints** are points in the program where you would like the program to stop so that you can examine what is going on (e.g., look at the contents of memory, registers, variables).
- In the absence of breakpoints, the program runs as normal until it completes or encounters an error.
- Typically you want to set a breakpoint near the point in the program where you think there is a problem before you run the program within gdb.
- You can enable and disable breakpoints.
- There are several ways to set a breakpoint. Examples include:
  - `break` *`function_name`*
  - `break` *`linenumber`*, or `break` *`filename:linenumber`*
  - `break` *`address`*

Running a program

- `run`
  - Run the program within gdb
- `continue`
  - Continue the program after encountering a breakpoint
- `step`
  - Run the program until it encounters the next line in the source file.
- `stepi`
  - Run the program until it encounters the next machine instruction
- `next`
  - Like `step`, but continue through procedure calls

Examining memory

- `x/nfu address`
  - Examine memory or register at location *address*, where
    - n is the number of items to examine
    - f is the format to use when displaying values
      - `s` for strings
      - `d` for integers
      - `u` for unsigned integers
      - `x` for hexadecimal
      - `a` for address/pointer
      - `f` for floating point
    - u is the unit
      - `b` for byte
      - `h` for halfword
      - `w` for word
      - `g` for giant word

Printing memory contents

`print exp`
Print the value of expression *exp*; let gdb determine the right format.

`print /f exp`
Print the value of expression *exp* using format *f* (see format specifications above).

`printf string, expressions`
print the list of *expressions* using the c-style format string *string*.

`display /f exp`
Print the value of expression *exp* using format *f* every time there is a break in execution.

Examining registers

- `info registers`
  - Print a list of the register contents
- Registers can be referenced in expressions by a predefined variable name that corresponds to the usual term used for the register. Variable names start with `$`. For example:
  - `x/d $eax`
    - Prints the contents of the eax register as a number
  - `x/s $rax`
    - Prints the string starting at the address represented by the address held in the rax register
  - `print $pc`
    - Prints the address in the pc register

Miscellaneous commands

- `bt`
  - Print the contents of the current stack (backtrace).
- `info frame`
  - Print information about the current stack frame.
- `info args`
  - Print information about the arguments passed to the current procedure.
- `info locals`
  - Print information about the current local variables.
- `list` *`linenumber`*
  - Show the source code lines around the given linenumber.

Help/tutorial pages

- Documentation
  - http://web.mit.edu/gnu/doc/html/gdb_1.html
  - ftp://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_toc.html
  - http://www.cs.hmc.edu/~geoff/classes/hmc.cs105.200901/labs/gdbinfo.html
- Tutorials
  - Google for “gdb tutorial”
  - You can find lots of good tutorials