CS2720 Practical Software Development

Debug Tutorial Spring 2011

Instructor: Rex Forsyth
Office: C-558
E-mail: forsyth@cs.uleth.ca
Tel: 329-2496

Tutorial Web Page: http://www.cs.uleth.ca/~forsyth/cs2720/lab/lab.html

Debugging

1. Opinion
2. To use a debugger, you must compile your program with the **-g** option
3. To start the debugger
   (a) command line
   - `gdb progName`
   - `gdb progName core`
     to obtain a core file, use `ulimit -c 64`
   - `gdb pid`

   (b) inside emacs
   - select *debug* from the *tools* menu

   (c) **ddd**, GUI front end for gdb
4. To quit the debugger
   - `quit` or `q`

GDB commands

- may be abbreviated to just enough to be unique, often a single letter
- tab completion allowed for commands and arguments
- most may be repeated by pressing the *enter* key
- many accept a location as an argument. Locations may be any of :
  - line number
  - –offset
  - +offset
  - function name
  - *address
  - filename:lineNumber
  - filename:functionName

1. Checkpoints
   - allow you to specify a point that you may want to *rewind* to.
   - save a snapshot of the program's current state
   - commands
     - `checkpoint` – sets the checkpoint and assigns an id
     - `info checkpoints` – list the checkpoints that have been saved
     - `restart checkpoint-id` – restore program to its state at the checkpoint
     - `delete checkpoint checkpoint-id` – remove the checkpoint
2. Breakpoints
   - a *location* where you want the debugger to stop execution of your program.
   - to set a breakpoint, use the command `break location`
   - `break` without an argument sets a breakpoint at the next instruction

3. Watchpoints
   - an expression that you want the debugger to watch and stop execution of your program whenever the value of the expression changes.
   - the expression may be
     - a variable name
     - an expression eg. `a+b`
     - an address cast to an appropriate datatype eg `*(int*)0x34ab56f`
   - to set a watchpoint, use the command `watch expression`
4. Catchpoints
   - an event that you want the debugger to watch for and stop execution of your program whenever the event occurs.
   - the event may be *throw* or *catch*
   - WARNING
   - to set a catchpoint, use the command `catch event`

GDB assigns an ID to each break, watch or catch point. This ID may be used to modify the points. Break, watch and catch points are all handled identically.

- to get a list of IDs, use the command `info breakpoints` or `info break`
- you may *enable*, *disable* or *delete* any ID
- to remove ID(s), there are two commands
  1. clear – remove a breakpoint according to a location
     - `clear` – removes a breakpoint at the next instruction
     - `clear location` – removes a breakpoint at the specified location
  2. delete – remove a breakpoint by the ID number
     - `delete [ID] [range of IDs]` – remove the specified points. If no argument is given, it removes all points(with confirmation).
- to disable ID(s), use the command `disable [ID] [range]`
- to enable ID(s), use the command `enable [ID] [range]`

- to run your program, use the command `run`
  if no breakpoints have been set, the entire progam will run
- if the debugger halts execution at a breakpoint, you can start it running again using the `continue` or `c` command.

A typical debug session would be as follows :

- start the debugger
- set breakpoint(s)
- run the program

When a breakpoint is reached and execution stops, you may:

- `step` – executes the next instruction
- `step n` – execute the next **n** instructions
- `next` – like step, except that it treats function calls as a single instruction.
- `next n`
- `finish` – continue until the current function is complete
- `until` – continue until the next source line is reached
  used to avoid single stepping through loops
- `backtrace [n]` – examine the stack, how did I get here?
  `where` and `info s` are synonyms.
- change to a different frame
  - `frame n` – change to frame **n** where **n** is obtained from the `backtrace` command.
  - `up n` – move up **n** frames; defaults to 1

  - `down n` – move down **n** frames; defaults to 1
- `print var=value` – change the value of a variable
- `print expression` – print the value of the expression
- `print \f expression` – print value in format **f** where **f** may be :
  - x – as an integer in hexadecimal
  - d – as a signed integer in decimal
  - u – as an unsigned integer in decimal
  - o – as an integer in octal
  - t – as an integer in binary (two)
  - a – as an address
  - c – as a character
  - f – as a floating point number in decimal