GDB - DEBUGGING PROGRAMS

Getting Started: Starting and Stopping

- gcc -g myprogram.c
    - Compiles myprogram.c with the debugging option $-g$. You still get an a.out, but it contains debugging information that lets you use variables and function names inside GDB, rather than raw memory locations $notfun$.
- gdb a.out
    - Opens GDB with file a.out, but does not run the program. You'll see a prompt gdb - all examples are from this prompt.
- r
- r arg1 arg2
- r < file1
    - Three ways to run "a.out", loaded previously. You can run it directly r, pass arguments $rarg1arg2$, or feed in a file. You will usually set breakpoints before running.
- help
- h breakpoints
    - Lists help topics $help$ or gets help on a specific topic $hbreakpoints$. GDB is well-documented.
- q - Quit GDB

Stepping through Code

Stepping lets you trace the path of your program, and zero in on the code that is crashing or returning invalid input.

- l
- l 50
- l myfunction
    - Lists 10 lines of source code for current line l, a specific line $l50$, or for a function $lmyfunction$.
- next
    - Runs the program until next line, then pauses. If the current line is a function, it executes the entire function, then pauses. **next** is good for walking through your code quickly.
- step
    - Runs the next instruction, not line. If the current instruction is setting a variable, it is the same as **next**. If it's a function, it will jump into the function, execute the first statement, then pause. **step** is good for diving into the details of your code.
- finish
    - Finishes executing the current function, then pause $alsocalledstepout$. Useful if you accidentally stepped into a function.

Breakpoints or Watchpoints

Breakpoints play an important role in debugging. They pause $break$ a program when it reaches a certain point. You can examine and change variables and resume execution. This is helpful when some input failure occurs, or inputs are to be tested.

- break 45
- break myfunction
    - Sets a breakpoint at line 45, or at myfunction. The program will pause when it reaches the breakpoint.
- watch x == 3
    - Sets a watchpoint, which pauses the program when a condition changes $whenx == 3changes$. Watchpoints are great for certain inputs $myPtr ! = NULL$ without having to break on *every* function call.
- continue
    - Resumes execution after being paused by a breakpoint/watchpoint. The program will continue until it hits the next breakpoint/watchpoint.
- delete N
    - Deletes breakpoint N $breakpointsarenumberedwhencreated$.

Setting Variables

Viewing and changing variables at runtime is a critical part of debugging. Try providing invalid inputs to functions or running other test cases to find the root cause of problems. Typically, you will view/set variables when the program is paused.

- print x
    - Prints current value of variable x. Being able to use the original variable names is why the $-g$ flag is needed; programs compiled regularly have this information removed.
- set x = 3
- set x = y
    - Sets x to a set value 3 or to another variable y
- call myfunction
- call myotherfunction$x$
- call strlen$mystring$
    - Calls user-defined or system functions. This is extremely useful, but beware of calling buggy functions.
- display x
    - Constantly displays the value of variable x, which is shown after every step or pause. Useful if you are constantly checking for a certain value.
- undisplay x
    - Removes the constant display of a variable displayed by display command.

Backtrace and Changing Frames

A stack is a list of the current function calls - it shows you where you are in the program. A *frame* stores the details of a single function call, such as the arguments.

- bt
    - **Backtraces** or prints the current function stack to show where you are in the current program. If main calls function a, which calls b, which calls c, the backtrace is

```
c <= current location
b
a
main
```

- up
- down
    - Move to the next frame up or down in the function stack. If you are in **c,** you can move to **b** or **a** to examine local variables.
- return
    - Returns from current function.

Handling Signals

Signals are messages thrown after certain events, such as a timer or error. GDB may pause when it encounters a signal; you may wish to ignore them instead.

- handle [signalname] [action]
- handle SIGUSR1 nostop
- handle SIGUSR1 noprint
- handle SIGUSR1 ignore
    - Instruct GDB to ignore a certain signal $SIGUSR1$ when it occurs. There are varying levels of ignoring