Matthew Dyer

Spring 2017

CMSC 461

GNU Debugger

Summary and Key Points

The GNU Debugger, otherwise known as GDB, is a powerful tool. It is one of the most popular and widely used debuggers in the UNIX family of systems. GDB is generally used for debugging code in C and C++. It also offers extensions for many other languages. GDB allows users extra tools that are useful in finding memory leaks, possible segmentation faults, checking variables while the program is running, and simply finding where errors may occur.

The following code snippet, written in C, is used as a basis for my examples:

```c
#include <stdio.h>

void printName(char* toPrint, int n){
    int i = 0;
    while(i < n){
        printf("%s\n", toPrint);
    }
}

int main(){

    char name[20] = "Example";
    int times = 5;

    printName(name,times);

    return 0;
}
```

In its current state, this code will continually print the word “Example” until an interrupt signal is sent to the program. What we really want it to do is print the word ”Example” only 5 times.

Seven Common Steps to Debugging in GDB

1) Compile your program to run in GDB.
    a. This is done by compiling with a “-g” flag in gcc or g++.
2) Open your executable in GDB.
    a. To run the GDB program you must type “gdb” on the command line.
    b. After opening the program, you must now load in your executable file by typing “file” followed by the name of your executable. (Example: file prog)
3) Learn the basic GDB Commands.
    a. break [function name]
        i. Set a point for your program to stop at while running.
    b. print [variable name]

        i. Print the value of a local variable.
    c. next
        i. Execute the next line/statement in the code.
    d. step
        i. This is the same as next except it stops if the next line is a function call.
    e. continue
        i. This will keep the program running until the end, a break, or a crash.
    f. backtrace
        i. Print a trace of the current stack, and is best used for recursion.
    g. up
        i. Move up a level in the stack.
    h. down
        i. Move down a level in the stack.
    i. run
        i. Run the program you loaded into gdb.
    j. quit
        i. Exit the gdb program.
4) Set a break at your functions.
    a. This is done using the “break” command followed by the name of a function.
    b. Using our above program as an example we would type, “break printName”.
5) Run and step through the program to locate the problem.
    a. To run the program simply type “run” and gdb will stop at any breakpoints you have set.
    b. After running you can use the navigation commands like, step, next, and continue to walk through your program line by line and diagnose the problem.
6) Print out variables to see how they are changing.
    a. You can print out the value of a variable by using the “print” command followed by the name of a variable. (Example: print i)
7) Fix any problems you encounter and compile again.
    a. In our example, we found that the loop in the “printName” function was not being incremented so we added the proper code and revised it to be the following:

```
3 void printName(char* toPrint, int n){
4     int i = 0;
5     while(i < n){
6         printf("%s\n", toPrint);
7         ++i;
8     }
9 }
```

A cheat-sheet style guide to using GDB can be found at:

http://www.yolinux.com/TUTORIALS/GDB-Commands.html

You can also check the man and info pages of a linux system for more information on using gdb.