The Dynamic Debugger *gdb*

Introduction

This handout introduces the basics of using gdb, a very powerful dynamic debugging tool. No-one always writes programs that execute perfectly every time, and while reading the program source can help fiind bugs, some can only be discovered by running the program and seeing what happens. That's where a dynamic debugger comes in; it lets you stop execution during the run and look at variables.

Setting It Up

To use *gdb*, you must compile your program using *gcc* and give the –g flag:

```
gcc -ansi -g program.c -o program
```

The -g flag tells the compiler to add information for the debugger. To debug your program, simply say:

```
gdb program
```

If you omit *program*, gdb looks for an *a.out* file.

The rest of this handout contains several sample programs and how to use *gdb* to find the bugs. I recommend you use gdb to get used to it; in particular, make extensive use of its help command! Just type

```
help
```

to its prompt, and it will tell you what to do. One of the features that I did not show which you will find particularly useful is the backtrace facility, which shows you what routines have been called and the values of the parameters.

Example 1

Here's a program that is supposed to add 2 to a variable `j` every time through the for loop:

```
#include <stdio.h>

main()
{
      int i, j = 0;
      for(i = 0; i < 100; i++);
            j += 2;
      printf("The value of j is: %d\n", j);
}
```

I compile and run it, and it does not work:

```
% gcc -ansi -g -o sample1 sample1.c
% sample1
The value of j is: 2
```

Oops! Let's get out *gdb* and see what happens. In what follows, what I type is in **boldface**, what the computer prints is in plain face, and my comments are in *italics*.

```
% gdb sample1
GNU gdb 5.0rh-5 Red Hat Linux 7.1
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) l            list 10 lines
1     #include <stdio.h>
2
```

```
3       main()
4       {
5               int i, j = 0;
6               for(i = 0; i < 100; i++);
7                       j += 2;
8               printf("The value of j is: %d\n", j);
9       }
(gdb) b main            put in a breakpoint; the program will stop at main when it
                        is executed
Breakpoint 1 at 0x8048466: file sample1.c, line 5.
(gdb) r                 run the program
Starting program: /usr/home/bishop/sample1

Breakpoint 1, main () at sample1.c:5
5               int i, j = 0;
(gdb) n                 do the next statement
6               for(i = 0; i < 100; i++);
(gdb) n
7                       j += 2;
(gdb) p i               print i's value
$1 = 100
(gdb) p j               print j's value
$2 = 0
                        aha! why is it not 200?  It's not getting incremented right,
                        so let's check the for loop … and sure enough, that's where
                        the problem is!
(gdb) q
The program is running.  Exit anyway? (y or n) y
```

Example 2

Now for a more complex example. Here's a program that's supposed to multiply `s` by 2 until `s` is greater than 100:

```
#include <stdio.h>

main()
{
        int i = 1, s;

        s = 3;
        while(i = 1){
                s += s;
                if (s > 100)
                        i = 0;
        }
}
```

I compile and run it, and it hangs; I need to kill it with control-C:

```
% gcc -ansi -g -o sample2 sample2.c
% sample2
^C
```

Again, let's use *gdb* to figure out what happened:

```
% gdb sample2
GNU gdb 5.0rh-5 Red Hat Linux 7.1
Copyright 2001 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) l
1       #include <stdio.h>
2
3       main()
4       {
5               int i = 1, s;
6
7               s = 3;
8               while(i = 1){
9                       s += s;
10                      if (s > 100)
(gdb) b 9
Breakpoint 1 at 0x804844b: file sample2.c, line 9.
(gdb) r
Starting program: /usr/export/home/bishop/ECS-40/gdb/sample2

Breakpoint 1, main () at sample2.c:9
9                       s += s;
(gdb) p s
$1 = 3
(gdb) n
10                      if (s > 100)
(gdb) c                 continue the run
Continuing.

Breakpoint 1, main () at sample2.c:9
9                       s += s;
(gdb) p s
$2 = 6
                        seems to be doubling s okay
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9                       s += s;
(gdb) p s
$3 = 12
                        I'm tired of always typing "p s" when the program stops, so
                        I associate commands with that breakpoint
(gdb) commands 1
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
p s
end
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9                       s += s;
$4 = 24
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9                 s += s;
$5 = 48
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9                 s += s;
$6 = 96
(gdb) p i
$7 = 1
(gdb) c
Continuing.

Breakpoint 1, main () at sample2.c:9
9                 s += s;
$8 = 192
                  wait … s is high here.  Let's watch the value of i
(gdb) watch i     break whenever i changes
Hardware watchpoint 2: i
(gdb) c
Continuing.
Hardware watchpoint 2, i      i just changed value

Old value = 1
New value = 0
main () at sample2.c:12
12          }
(gdb) c
Hardware watchpoint 2: i

Old value = 0
New value = 1
main () at sample2.c:9
9                       s += s;
(gdb) c
Continuing.

Hardware watchpoint 2, i      why does i change -- the "if" is not executed!
                              Look in the while expression; we used =, not ==
Old value = 1
New value = 0
main () at sample2.c:12
12          }
(gdb) q
The program is running.  Exit anyway? (y or n) y
```

Example 3

This one is a character counter (with apologies to Kernighan and Ritchie, from whose book it was mangled):

```
#include <stdio.h>

main()            /* counts digits, white space, others */
{
      int c, i, num_digits[10], num_white, num_other;
      num_white = num_other = 0;
      for(i = 0; i < 10; i++)       /* initialize */
            num_digits[i] = 0;
      while(c = getchar() != EOF){
            switch(c){
            case '0': case '1': case '2': case '3':
            case '4': case '5': case '6': case '7':
            case '8': case '9':
                  num_digits[c - '0']++;
                  break;
            case ' ': case '\t': case '\n':
                  num_white++;
                  break;
            default:
                  num_other++;
                  break;
            }
      }
      printf("digits =");
      for(i = 0; i < 10; i++)
            printf(" %d", num_digits[i]);
      printf("white space = %d, other = %d\n",
            num_white, num_other);
}
```

Here's a data file for our testing and debugging pleasure:

```
7 4 88 6 6  3 4 7 87 8 2 34
  5 7  3 21 123q52
```

I compile and run it on a Sun workstation (*not* on the Linux workstations in the CSIF), and it hangs:

```
% gcc -ansi -g -o sample3 sample3.c
% sample3 < sample3.data
^C %
```

So I run *gdb*:

```
% gdb sample3
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.10.pl1 (sparc-sun-sunos4.1),
Copyright 1993 Free Software Foundation, Inc...
(gdb) l
1     #include <stdio.h>
2
3     main()      /* counts digits, white space, others */
4     {
5           int c, i, num_digits[10], num_white, num_other;
6           num_white = num_other = 0;
7           for(i = 0; i <= 10; i++)      /* initialize */
8                 num_digits[i] = 0;
9           while(c = getchar() != EOF){
```

```
10                  switch(c){
(gdb) b 8
Breakpoint 1 at 0x22f0: file sample3.c, line 8.
(gdb) comm 1
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
p i
end
(gdb) r
Starting program: /usr/export/home/bishop/ECS-40/gdb/sample3

Breakpoint 1, main () at sample3.c:8
8                   num_digits[i] = 0;
$1 = 0
(gdb) s             s means to skip to the next statement; if it calls a
                    function, you'll stop in that function.  n means to skip to
                    the next statement in this function; you skip over any
                    functions that are called.
7            for(i = 0; i <= 10; i++)      /* initialize */
(gdb) n

Breakpoint 1, main () at sample3.c:8
8                   num_digits[i] = 0;
$2 = 1
(gdb) cond 1 i>=9 this means to skip breakpoint 1 until i >= 9 is true.
(gdb) c
Continuing.

Breakpoint 1, main () at sample3.c:8
8                   num_digits[i] = 0;
$3 = 9
(gdb) c
Continuing.

Breakpoint 1, main () at sample3.c:8
8                   num_digits[i] = 0;
$4 = 10             now we should leave the loop
(gdb) c
Continuing.

Breakpoint 1, main () at sample3.c:8
8                   num_digits[i] = 0;
$5 = 9              huh?  This probably means we're overwriting i -- or our
                    array has one less element than we think.
(gdb) q
The program is running.  Quit anyway (and kill it)? (y or n) y
```

We make the change, and run it:

```
% sample3a < sample3.data
digits = 0 0 0 0 0 0 0 0 0 0white space = 0, other = 48
```

We're making progress, but are not done yet. Back to *gdb*!

```
% gdb sample3
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
```

```
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.10.pl1 (sparc-sun-sunos4.1),
Copyright 1993 Free Software Foundation, Inc...
(gdb) l
1      #include <stdio.h>
2
3      main()      /* counts digits, white space, others */
4      {
5            int c, i, num_digits[10], num_white, num_other;
6            num_white = num_other = 0;
7            for(i = 0; i < 10; i++)       /* initialize */
8                  num_digits[i] = 0;
9            while(c = getchar() != EOF){
10                 switch(c){
(gdb) b 10         since the for loop seems to work, it's probably in the while
Breakpoint 1 at 0x23ac: file sample3.c, line 10.
(gdb) r < sample3.data  note I can redirect standard input, output, and error
                        as if I were in the shell
Starting program: /usr/export/home/bishop/ECS-40/gdb/sample3 < sample3.data

Breakpoint 1, main () at sample3.c:10
10                 switch(c){
(gdb) p c
$1 = 1             oops … there's no ^A (character 1) in the file!
(gdb) c
Continuing.

Breakpoint 1, main () at sample3.c:10
10                 switch(c){
(gdb) p c
$2 = 1             again!  looks like I assign when I should be comparing …
(gdb) quit
The program is running.  Quit anyway (and kill it)? (y or n) y
```

Example 4

Here's a program that I suspect everyone in this class can relate to. It's the word listing program of homework 3, done with a linked list. I compile it and run it, and get a core dump. This is on a FreeBSD system, so the core dump goes into a file called *a.out.core*. I now use *gdb* to find the problem.

```
> gdb a.out a.out.core
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-unknown-freebsd"...
Core was generated by `a.out'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /usr/lib/libc.so.4...done.
Reading symbols from /usr/libexec/ld-elf.so.1...done.
#0  0x8048910 in lprint (h=0x0) at ans3.c:182
182             printf("%s\n", h->data);
(gdb) where             This prints out the call stack, showing me what
```

```
                         functions were active when the core dump occurred
#0  0x8048910 in lprint (h=0x0) at ans3.c:182
#1  0x804892f in lprint (h=0x804f050) at ans3.c:187
#2  0x804892f in lprint (h=0x804f030) at ans3.c:187
#3  0x804879e in main () at ans3.c:95
#4  0x80485f5 in _start ()
                        main has called lprint, which called lprint, which
                        called lprint. The third call to lprint passed a NULL
                        pointer
(gdb) p h->data         try to print what would have been printed
Cannot access memory at address 0x0.
(gdb) p h               see what the pointer was
$1 = (struct word *) 0x0
(gdb) l
177     void lprint(struct word *h)
178     {
179             /*
180              * print current contents
181              */
182             printf("%s\n", h->data);
183
184             /*
185              * recurse
186              */
(gdb) up                go to the caller of the routine
#1  0x804892f in lprint (h=0x804f050) at ans3.c:187
187             lprint(h->next);
(gdb) p h               this prints the pointer value
$2 = (struct word *) 0x804f050
(gdb) p *h              this prints the structure pointed to and it’s passing
                        NULL. The invoked routine should have stopped and not
                        tried to print. So we’re missing a check.
$3 = {data = 0x804f060 "there", next = 0x0}
(gdb) quit
>
```