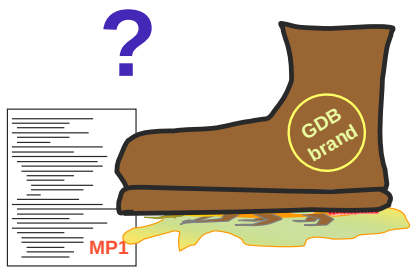
Today's Topic: Bugs...



Rationale and Pointers

- purpose of the lecture
  - illustrate the use of a symbolic debugger
  - make your life easier (really!)
- sources of information on gdb
  - info gdb (from command line on EWS)
  - man gdb
  - help (within gdb)
- user interface extensions
  - within Emacs
  - several graphical front ends available

Organization

- illustration of use through running example
  - binary search procedure
  - main procedure designed for testing
- interspersed with summaries
  - preparations and startup
  - basic commands
  - models of use
  - showing program state
  - controlling the program
  - breakpoints

- a guide to the color scheme
  - prompts appear in dark blue
  - your commands appear in green
  - responses appear in black
  - editorial comments appear in [purple]
  - some highlights boxed in yellow
- but before we begin, let's look at the code...
  - one source file: my_search.c
  - compiled to executable "my_search"

Getting the Code

- Copy tutorial code from class drive to current directory
  - `cp /ece391/gdb-tutorial-code.tar.gz .`
- Gunzip and untar the package
  - `tar zxvf gdb-tutorial-code.tar.gz`
- Change directory into the **`gdb-tutorial-code`** directory
  - `cd gdb-tutorial-code`
- Use your favorite editor (vim, emacs) to open **`my_search.c`**


```
int main (int argc, char* argv[])
{
   const char* const my_list[] =
      {"a","fine","test,","yes %%%","PASS"};
   int idx, item;

   for (idx = 0; idx < 4; idx++) {
      item = binary_search
               (my_list[idx], my_list, 4);
      printf (my_list[item]);
      puts ("");
   }
   item = binary_search ("miss", my_list, 4);
   printf (my_list[item]);
   puts ("");
   return 0;
}
```

```
static int binary_search (const char* find,
   const char* const* elts, int n_elts)
{
   int left = 0, right = n_elts - 1, middle;
   int comparison;

   while (left != right) {
      middle = (left + right) / 2;
      comparison = strcmp (find,elts[middle]);
      if (comparison == 0)
         return middle;
      if (comparison < 0)
         right = middle;
      left = middle;
   }
   return -1;
}
```


Preparing to Use GDB

- compile **and link** with -g option
  - compiler includes symbolic information with object and executable files
  - examples: type information, function and variable names
- debugging optimized code (-g and -O)
  - will work with gcc (not with most others)
  - but gdb cannot undo optimizations
  - if you can't either, safer to turn off optimizer


```
[user gdb-tutorial-code]$ ulimit –c unlimited
[user gdb-tutorial-code]$ ./my_search
Segmentation fault (core dumped)
[user gdb-tutorial-code]$ gdb my_search
   core.10676 [substitute your core file]
GNU gdb Red Hat Linux (5.1.90CVS-5)
              [some text elided]
Core was generated by `./my_search'.
Program terminated with signal 11,
   Segmentation Fault.
              [some text elided]
(gdb) backtrace
                    […]
```



```
(gdb) backtrace
#0 0x42050396 in vfprintf () from
   /lib/i686/libc.so.6
#1 0x4205a1dc in printf () from
   /lib/i686/libc.so.6
#2 0x08048553 in main (argc=1,
   argv=0xbffffb74) at my_search.c:65
#3 0x42017499 in __libc_start_main () from
   /lib/i686/libc.so.6
```

our test function!

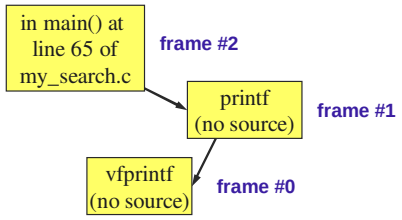
Starting GDB

- with a core file (*i.e.*, after a program crashes)
  `[user]$ gdb <executable> core`
- before starting a program
  `[user]$ gdb <executable>`
  `(gdb) run <arguments>`
- after starting a program
  `[user]$ gdb <exec.> <proc. id>` or
  `(gdb) attach <pid>`
- use "file" command or restart gdb after recompilation

Some Basic Commands

- backtrace or where
  - show the stack trace (sequence of procedure calls)
- moving from one stack frame to another
  - up: move one frame higher
  - down: move one frame lower
  - frame <n>: move to frame **n**
- list: show the source code corresponding to the current stack frame (by default)

```
#0 0x42050396 in vfprintf () from
   /lib/i686/libc.so.6
#1 0x4205a1dc in printf () from
   /lib/i686/libc.so.6
#2 0x08048553 in main (argc=1,
   argv=0xbffffb74) at my_search.c:65
```




```
(gdb) frame 2
#2 0x08048553 in main (argc=1,
   argv=0xbffffb74) at my_search.c:65
65              printf (my_list[item]);
(gdb) list
    [some text elided--starts on line 60]
63          for (idx = 0; idx < 4; idx++) {
64              item = binary_search
   (my_list[idx], my_list, 4);
65              printf (my_list[item]);
66              puts ("");
67          }
     [some text elided--ends on line 69]
```

We've located the problem, but how can we tell what is going wrong?

Basics of Showing Program State

- print
  - evaluate an expression (usually in C/C++)
  - show the result
- display (like print)
  - evaluate and show result of expression
  - each time the program stops
  - returns an integer identifier
- undisplay <#>: stop displaying something
- info locals: show the values of all variables defined within the current frame


```
(gdb) print item
$1 = -1
(gdb) print my_list[-1]
$2 = 0x20001 <Address 0x20001 out of bounds>
(gdb) l binary_search [abbreviated as "l"]
          [some text elided]
37      static int
39      binary_search (const char* find,
   const char* const* elts, int n_elts)
     [some text elided--ends on line 45]
(gdb)
44              middle = (left + right) / 2;
          [some text elided]
51          }
52                  n_elts
53          return -1;
```

pressing return repeats the last command

the culprit!

- We fix the bug,
- recompile,
- and try again...


```
[user gdb-tutorial-code]$ ./my_search
PASS
fine
test,
^C
[user gdb-tutorial-code]$ gdb my_search
          [gdb intro text elided]
(gdb) run [this program needs no arguments]
Starting program:
   /homedir/gdb-tutorial-code/my_search
PASS
fine
test,
^C
Program received signal SIGINT, Interrupt.
        [frame #0 information elided]
```

Three Models of Use

- program is running
  - use CTRL-C to stop it
  - otherwise, keyboard I/O goes to program
- program is stopped
  - gdb commands can be entered
  - show/change state, set breakpoints, perform function calls, *etc.*
- post-mortem analysis (*e.g.*, on a core file); no changes to program allowed


```
(gdb) bt [an abbrevation for backtrace]
#0  0x4207fa94 in strcmp () from
   /lib/i686/libc.so.6
#1  0x080484bf in binary_search
   (find=0x8048651 "yes %%%", elts=0x8048638,
   n_elts=4) at my_search.c:45
#2  0x0804852f in main (argc=1,
   argv=0xbffffb64) at my_search.c:64
#3  0x42017499 in __libc_start_main() from
   /lib/i686/libc.so.6
```

We're stuck in the search function.


```
(gdb) up
#1  0x080484bf in binary_search
   (find=0x8048651 "yes %%%", elts=0x8048638,
   n_elts=4) at my_search.c:45
45              comparison = strcmp (find,
   elts[middle]);
(gdb) info locals
left = 2
right = 3
middle = 2
comparison = 1
```

Seems ok...let's let it run a little longer.

Controlling the Program

- continue: let the program run at full speed
- one step (**source line**!) at a time
  - next
    - step over any function calls
    - run until the function returns and any remaining code on the line finishes
  - step: step into function calls, to the first line
- finish: run until the current frame returns
- return [<value>]: cut the current frame short and return immediately


```
(gdb) display left
1: left = 2
(gdb) display right
2: right = 3
(gdb) continue
Continuing.
^C
Program received signal SIGINT, Interrupt.
binary_search (find=0x8048651 "yes %%%",
   elts=0x8048638, n_elts=4) at my_search.c:43
43          while (left != right) {
2: right = 3
1: left = 2
```

Nothing has changed!


```
(gdb) l
              [some text elided]
41          int comparison;
42
43          while (left != right) {
44              middle = (left + right) / 2;
              [some text elided]
(gdb) break 44
Breakpoint 1 at 0x8048490: file my_search.c,
   line 44.
(gdb) continue
Continuing.
Breakpoint 1, binary_search (find=0x8048651
   "yes %%%", elts=0x8048638, n_elts=4) at
   my_search.c:44
 [source line and displayed variables elided]
```

```
(gdb) next
45              comparison = strcmp (find,
  elts[middle]);
2: right = 3
1: left = 2 [displayed values elided below]
(gdb) [next..again]
46              if (comparison == 0)
(gdb) print comparison
$1 = 1
(gdb) n [an abbreviation for next]
48              if (comparison < 0)
(gdb) [and again]
50              left = middle + 1;
(gdb) print middle
$2 = 2
```

it's the same!


```
static int binary_search (const char* find,
  const char* const* elts, int n_elts)
{
  int left = 0, right = n_elts - 1, middle;
  int comparison;

  while (left != right) {
    middle = (left + right) / 2;
    comparison = strcmp (find,elts[middle]);
    if (comparison == 0)
      return middle;
    if (comparison < 0)
      right = middle - 1;
    left = middle + 1;
  }
  return n_elts;
}
```

We've fixed both of these!

time to try again...

But what was that "break" command that we used?

Using Breakpoints

- break [<file>:]<line>
  - set a breakpoint
  - returns an integer identifier
- break <function/label name>
- cond <breakpoint #> <logical expression>
- disable [<breakpoint #>]: disable temporarily
- enable [<breakpoint #>]: re-enable
- delete [<breakpoint #>]: remove permanently
- info breakpoints: show all breakpoints, conditions, *etc.*


```
[user gdb-tutorial-code]$ gdb my_search
          [gdb intro text elided]
(gdb) run
Starting program:
  /homedir/gdb-tutorial-code/my_search
^C [darn!]
Program received signal SIGINT, Interrupt.
        [frame #0 information elided]
(gdb) l binary_search
          [first 10 lines elided]
(gdb)
44              middle = (left + right) / 2;
(gdb) break 44 [the start of the loop again]
Breakpoint 1 at 0x8048490: file my_search.c,
  line 44.
```



```
(gdb) run
The program being debugged has been started
  already.
Start it from the beginning? (y or n) y
Starting program:
  /homedir/gdb-tutorial-code/my_search

Breakpoint 1, binary_search (find=0x8048664
  "a", elts=0x8048638, n_elts=4) at
  my_search.c:44
44              middle = (left + right) / 2;
(gdb) disp left [an abbreviation for display]
1: left = 0
(gdb) disp right
2: right = 3
```

```
(gdb) c [an abbreviation for continue]
Continuing.
Breakpoint 1, binary_search (find=0x8048664
   "a", elts=0x8048638, n_elts=4) at
   my_search.c:44
44              middle = (left + right) / 2;
2: right = 0
1: left = 2
(gdb) [Repeat continue command]
Continuing.
Breakpoint 1, binary_search (find=0x8048664
   "a", elts=0x8048638, n_elts=4) at
   my_search.c:44
44              middle = (left + right) / 2;
2: right = 0
1: left = 2
```

? No change again...


```
(gdb) undisplay
Delete all auto-display expressions?
  (y or n) y
(gdb) n
45              comparison = strcmp (find,
  elts[middle]);
(gdb)
46              if (comparison == 0)
(gdb)
48              if (comparison < 0)
(gdb)
49                  right = middle - 1;
(gdb)
50              left = middle + 1;
```

Oops! We shouldn't update both sides!


```
static int binary_search (const char* find,
   const char* const* elts, int n_elts)
{
   int left = 0, right = n_elts - 1, middle;
   int comparison;
   while (left != right) {
      middle = (left + right) / 2;
      comparison = strcmp (find,elts[middle]);
      if (comparison == 0)
         return middle;
      if (comparison < 0)
         right = middle - 1;
      else
         left = middle + 1;
   }
   return n_elts;
}
```

The new "else" here is critical.


```
(gdb) run
Starting program:
   /homesta/ece338/gdb/my_search
PASS
fine
test,
PASS
^C
Program received signal SIGINT, Interrupt.
0x080484cd in binary_search (find=0x8048667
   "miss", elts=0x8048638, n_elts=4) at
   my_search.c:46
46              if (comparison == 0)
```

A little farther, but still hanging...


```
(gdb) disp left
1: left = 2
(gdb) disp right
2: right = 1
(gdb) c
Continuing.
^C
Program received signal SIGINT, Interrupt.
0x080484cd in binary_search (find=0x8048667
   "miss", elts=0x8048638, n_elts=4) at
   my_search.c:46
46              if (comparison == 0)
2: right = 1
1: left = 2
```

? We shouldn't be in the loop when right is smaller than left!


```
(gdb) und [short for undisplay]
Delete all auto-display expressions?
  (y or n) y
(gdb) n
48              if (comparison < 0)
(gdb)
51                  left = middle + 1;
(gdb)
52          }
(gdb)
43          while (left != right) {
```

The loop condition is wrong!
(One more try...)

```
[user gdb-tutorial-code]$ ./my_search
a
fine
test,
yes %
PASS
```

The program appears to be working, for the most part.

But something is missing...

```
int main (int argc, char* argv[])
{
   const char* const my_list[] =
     {"a","fine","test,","yes %%%","PASS"};
```



```
[user gdb-tutorial-code]$ gdb my_search
          [gdb intro text elided]
(gdb) l main
           [listed lines elided]
(gdb)
65              item = binary_search
  (my_list[idx], my_list, 4);
66              printf (my_list[item]);
67              puts ("");
68          }
          [remaining lines elided]
(gdb) break 66
Breakpoint 1 at 0x804853f: file my_search.c,
  line 66.
(gdb) cond 1 item == 3
```



```
(gdb) info break
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x0804853f in
  main at my_search.c:66
        stop only if item == 3
(gdb) run
Starting program:
  /homesta/ece338/gdb/my_search
a
fine
test,

Breakpoint 1, main (argc=1, argv=0xbffffb64)
  at my_search.c:66
66              printf (my_list[item]);
```



```
Breakpoint 1, main (argc=1, argv=0xbffffb64)
  at my_search.c:66
66              printf (my_list[item]);
(gdb) print my_list[item]
$1 = 0x8048651 "yes %%%"
(gdb) n
67              puts ("");
(gdb)
yes %  [notice the output buffering effect]
64          for (idx = 0; idx < 4; idx++) {
```

Oops! Printf interprets its first argument as a formatting string!


```
int main (int argc, char* argv[])
{
   const char* const my_list[] =
      {"a","fine","test,","yes %?%","PASS"};
   int idx, item;

   for (idx = 0; idx < 4; idx++) {
      item = binary_search
               (my_list[idx], my_list, 4);
      printf ("%s\n", my_list[item]);
   }
   item = binary_search ("miss", my_list, 4);
   printf ("%s\n", my_list[item]);
   return 0;
}
```

new versions


```
[user gdb-tutorial-code]$ ./my_search
a
fine
test,
yes %%%
PASS
```

Hurrah!

What about Threads?

- two more commands...
  - info threads: list all threads and their frame #0s
  - thread <n>: switch to a different thread
- Solaris Lightweight Processes (LWPs)
  - kernel-level threads
  - used for scheduling
  - cause duplicate listings in gdb


```
eesn14> gdb relay
[gdb intro text elided]
(gdb) break main
Breakpoint 1 at 0x11234: file relay.c, line
  108.
(gdb) run eesn15 1234 target 4000
Starting program:
  /homesta/ece338/gdb/MP3/relay eesn15 1234
  target 4000
[New LWP    2        ]
[New LWP    3        ]
[New LWP    4        ]
Breakpoint 1, main (argc=5, argv=0xffbefac4)
  at relay.c:108
108         int fd = -1, cli_fd, addr_size,
  i;
```

Management threads are always present in threaded code.


```
(gdb) info threads
  7 Thread 3          0xff0cddbc in
  _reap_wait () from /usr/lib/libthread.so.1
  6 Thread 2 (LWP 2)  0xff29e99c in
  _signotifywait () from /usr/lib/libc.so.1
* 5 Thread 1 (LWP 1)  main (argc=5,
  argv=0xffbefacc) at relay.c:108
  4 LWP    4          0xff0ca728 in
  _lwp_start () from /usr/lib/libthread.so.1
  3 LWP    3          0xff29c540 in
  door_restart () from /usr/lib/libc.so.1
  2 LWP    2          0xff29e99c in
  _signotifywait () from /usr/lib/libc.so.1
  1 LWP    1          main (argc=5,
  argv=0xffbefacc) at relay.c:108
```



```
(gdb) c
Continuing.
chance to drop a UDP packet:    0%
chance to corrupt a UDP packet: 0%
maximum reordering delay:       0
  milliseconds
initial random seed:            327183211
logging calls:                  NO
[New LWP    5        ]
[New LWP    6        ]
  [keep going; also more threads than LWPs]
[New LWP    38       ]
^C
Program received signal SIGINT, Interrupt.
0xff29bc70 in _so_accept () from
  /usr/lib/libc.so.1
```



```
(gdb) info thread
  121 Thread 83 (LWP 35)  0xff29d1d8 in _poll
  () from /usr/lib/libc.so.1
  120 Thread 82 (LWP 32)  0xff29d1d8 in _poll
  () from /usr/lib/libc.so.1
  119 Thread 81           0xff0c826c in
  cond_wait () from /usr/lib/libthread.so.1
                  [etc.]
  3 LWP    3          0xff29d1d8 in _poll ()
  from /usr/lib/libc.so.1
  2 LWP    2          0xff29e99c in
  _signotifywait () from /usr/lib/libc.so.1
  1 LWP    1          0xff29bc70 in
  _so_accept () from /usr/lib/libc.so.1
```



```
(gdb) thread 121
[Switching to thread 121 (Thread 83 (LWP
  35))]
      [frame #0 information elided]
(gdb) where
      [frames #0-2 information elided]
#3  0x12a4c in udp_receiver (v_uct=0x26f20)
  at relay.c:755
(gdb) frame 3
#3  0x12a4c in udp_receiver (v_uct=0x26f20)
  at relay.c:755
755             if ((len = mp3_recvfrom (uct-
  >fd, packet, MAX_PKT_LEN, 0,
```

All operations now pertain to thread 121.

That code has
no bugs, of course.
As you all knew.

A Few More Comments

- gdb uses libreadline
  - standard GNU input control
  - also used in bash, for example
- when you print something
  - gdb assigns it a name ($1, for example)
  - you can use the name in later expressions
- if you want to change a variable...
  - set variable <name>=<value>
  - but not with a core file, of course
- do not use watch points...

One Last Command

- quit
  - Quit.
  - You may kill the program being debugged when you do so.