Introduction to GDB for System Administrators and Programmers

Nikolai Fetissov
New York City BSD User Group. June 2, 2010
About GDB

* Source-level (symbolic) and machine-level debugger

* Originally written by R. Stallman => GPL (boo)

* Currently supported by FSF (oh, NO ...)

* Latest version as of June 2010 is 7.1 (which doesn’t support BSDs yet)

* De-facto debugger on open-source Unixen
What is a Debugger?

Wikipedia says:

A **debugger** or **debugging tool** is a [computer program](#) that is used to test and [debug](#) other programs (the "target" program).

Don’t you love recursive definitions?
What is a Bug?

1947: First actual case of bug being found

Clearly then, GDB is a tool for removing insects from your code
GDB allows you to

* Debug programs in asm, C, C++, Objective-C, Fortran, Pascal, Modula-2, and Ada
* Debug under Unix, Windows*, RT and embedded
  * Debug on many platforms
* Debug remotely over serial or TCP/IP
  * Debug live and dead kernels
    * Script it with Python
      * Bake muffins*
    * and many other things

BUT ...

* extra pain
This is an Introduction

So we will cover the basics:

* Running live under debugger
* Looking at dead process (core)
* Dealing with forks and threads
  * Dealing with C and C++

* And what can sysadmin do ...
Prerequisites

* Compile with -g or -ggdb for debug info
  * GCC deals with -g and -O together
  * NO RUNTIME COST, so don’t strip(1)

~$ cc -Wall -pedantic -o fib fib.c
~$ gdb --silent fib
(no debugging symbols found)...
(gdb)

~$ cc -Wall -pedantic -ggdb -o fib fib.c
~$ gdb fib
(gdb)
Basics: starting

* Start gdb, load the binary
* Then run, Forest, run

```
~$ gdb fib
(gdb) tb main
(gdb) run 10
```

OR

```
~$ gdb fib
(gdb) start 10
```

- Sets temporary breakpoint at program entry
- Starts program with given arguments
- Later versions provide single ‘start’ command
Basic basics

* Looking at the source
  * Line editing
  * Getting help
  * Getting annoyed

(gdb) list [location]
(gdb) ^w^a^e ...

(gdb) apropos <word>
(gdb) help <word>
(gdb) info <data>
(gdb) show <setting>

(gdb) l
...
(gdb) ↓
...

View program source
Emacs line editing
Empty line repeats last command (but not all of them)
Live Process: attaching

* Find the process
* Attach
* GDB takes over - process is STOPPED

```
~$ ps ax | grep fib
  1292  0 R  0:03.83 ./fib 100

~$ gdb fib 1292

~$ gdb -p 1292

~$ gdb fib
(gdb) attach 1292
```
(gdb) break fibonacci-main.c:28
Breakpoint 2 at 0x80484f0: file fibonacci-main.c, line 28.
(gdb) continue
Continuing.
Breakpoint 2, main(argc=2, argv=0xbfbfbebfbf4) at fibonacci-main.c:28
28   printf( "Fibonacci[%d] = %zu\n", ...
From Here

* Continue/break, or
* Step by step, or
* Detach or kill

(gdb) **next**  
Run next source line, step over function calls

(gdb) **step**  
Run next source line, step into function calls

(gdb) **nexti**  
Run next source line, step into function calls

(gdb) **stepi**  
Run next instruction, step over function calls

(gdb) **finish**  
Run next instruction, step into function calls

(gdb) **until**
From Here

* Continue/break, or
* Step by step, or
* Detach or kill

(gdb) detach

Release the debugged process

(gdb) kill

Kill process under debugger
Autopsy: core file

* Find what program produced the core
* Load program and core into GDB
* Examine the corpse

```bash
~$ file core.1290
core.1290: ELF 32-bit LSB core file Intel 80386, version 1 (FreeBSD), FreeBSD-style, from ‘fib’

~$ gdb fib core.1290

OR

~$ gdb -c core.1290
...
(gdb) symbol-file fib
```

Make sure GDB reads right symbols
Examination

* Find where we are
* Find values of variables
* Registers, signals, etc.

(gdb) p "We can just continue listing GDB commands one by one, but is it really fun?"

(gdb) p "What is that fibonacci thing?"
Leonardo of Pisa

* Leonardo Fibonacci (1170-1250)
* Fibonacci Numbers: \( F_n = F_{n-1} + F_{n-2} \)
* What is wrong with software development?
Recursive - Stack

* Don’t worry - I don’t have this in production
  * But I have seen this many times

/* first attempt - too much Lisp? */

size_t fibonacci( size_t n )
{
    switch ( n )
    {
    case 0:  return 0;
    case 1:  /* FALLTHRU */
    case 2:  return 1;
    default: return fibonacci( n - 1 ) + fibonacci( n - 2 );
    }
}
Where: backtrace

~$ gdb --silent fib-rec
(gdb) tb main
Breakpoint 1 at 0x8048490: file fibonacci-main.c, line 17.
(gdb) run 100
... main (argc=2, argv=0xbfbfedb3) at fibonacci-main.c:17
17    {
(gdb) b fibonacci
Breakpoint 2 at 0x8048527: file fibonacci-recursive.c, line 13.
(gdb) ignore 2 5
Will ignore next 5 crossings of breakpoint 2.
(gdb) c
Continuing.
Breakpoint 2, fibonacci (n=95) at fibonacci-recursive.c:13
13        switch ( n )
(gdb) bt
#0  fibonacci (n=95) at fibonacci-recursive.c:13
#1  0x0804855b in fibonacci (n=96) at fibonacci-recursive.c:18
#2  0x0804855b in fibonacci (n=97) at fibonacci-recursive.c:18
#3  0x0804855b in fibonacci (n=98) at fibonacci-recursive.c:18
#4  0x0804855b in fibonacci (n=99) at fibonacci-recursive.c:18
#5  0x0804855b in fibonacci (n=100) at fibonacci-recursive.c:18
#6  0x080484fb in main (argc=2, argv=0xbfbfbeb4) at fibonacci-main.c:28
(gdb) bt full
#0  fibonacci (n=95) at fibonacci-recursive.c:13
No locals.
#1  0x0804855b in fibonacci (n=96) at fibonacci-recursive.c:18
No locals.
#2  0x0804855b in fibonacci (n=97) at fibonacci-recursive.c:18
No locals.
#3  0x0804855b in fibonacci (n=98) at fibonacci-recursive.c:18
No locals.
#4  0x0804855b in fibonacci (n=99) at fibonacci-recursive.c:18
No locals.
#5  0x0804855b in fibonacci (n=100) at fibonacci-recursive.c:18
No locals.
#6  0x080484fb in main (argc=2, argv=0xbfbfebfbf4) at fibonacci-main.c:28

    num = 100
(gdb) up
(gdb) down
(gdb) frame 6
#6 0x080484fb in main (argc=2, argv=0xbfbfebfbf4) at fibonacci-main.c:28
28 printf( "Fibonacci[%d] = %zu\n", num, fibonacci( num ));
(gdb) print argc
$1 = 2
(gdb) p argv
$2 = (char **) 0xbfbfebfbf4
(gdb) p *argv@argc
$3 = {0xbfbfed84 "fib-rec", 0xbfbfedb3 "100"}
(gdb) set print array on
(gdb) set print pretty on
Processes - Forks

* Don’t worry - I don’t have this in production
  * But I have seen something very close

/* second attempt - over-engineering or stupidity? */
size_t fibonacci( size_t n )
{
    pid_t pid1, pid2; size_t fib; int status;
    switch ( n )
    {
    case 0:  return 0; case 1:  /* FALLTHRU */
    case 2:  return 1; default: break;
    }
    if (( pid1 = fork() ) == 0 ) exit( fibonacci( n - 1 ) );
    if (( pid2 = fork() ) == 0 ) exit( fibonacci( n - 2 ) );
    waitpid( pid1, &status, 0 ); fib = WEXITSTATUS( status );
    waitpid( pid2, &status, 0 ); return fib + WEXITSTATUS( status );
}
Parent-Child Relationship

(gdb) set follow-fork-mode child | parent
(gdb) set follow-exec-mode new | same
(gdb) set detach-on-fork on | off
(gdb) catch fork
(gdb) info inferiors
(gdb) inferior N

{ Only in Version 7 }
(gdb) set detach-on-fork off
(gdb) catch fork
(gdb) continue
Continuing.
Catchpoint 2 (forked process 1477), 0x00007fff78f4876 in fork () from /lib64/libc.so.6
(gdb) info inferiors
    Num  Description       Executable
         * 1    process 1474   /home/nickf/.home/code/cssource/gdb/fib-fork
(gdb) next
Single stepping until exit from function fork,
which has no line number information.
[New process 1477]
[Thread debugging using libthread_db enabled]
fibonacci (n=15) at fibonacci-fork.c:31
31        if ( pid1 == 0 ) exit( fibonacci( n - 1 ));
(gdb) info inferiors
    Num  Description       Executable
         2    process 1477   /home/nickf/.home/code/cssource/gdb/fib-fork
         * 1    process 1474   /home/nickf/.home/code/cssource/gdb/fib-fork
(gdb) inferior 2
[Switching to inferior 2 [process 1477] (/home/nickf/.home/code/cssource/gdb/fib-fork)]
[Switching to thread 2 (Thread 0x7ffffff7f3700 (LWP 1477))]
#0 0x00007fff78f4876 in fork () from /lib64/libc.so.6
Threads

* Don’t worry - I don’t have this in production
* But I see something like this every day

/* third attempt - look, ma, cores! */
static void* fibonacci_thread( void* arg )
{
    size_t n = ( size_t )arg, fib; int ern;
    pthread_t th1, th2; void* pvalue;
    switch ( n )
    { case 0: return (void*)0; case 1: /* FALLTHRU */
      case 2: return (void*)1; default: break; }

    pthread_create( &th1, 0, fibonacci_thread, (void*)(n - 1));
    pthread_create( &th2, 0, fibonacci_thread, (void*)(n - 2));

    pthread_join( th1, &pvalue ); fib = ( size_t )pvalue;
    pthread_join( th2, &pvalue ); fib += ( size_t )pvalue;

    return ( void* )fib;
}
Threading for dummies

(gdb) hbreak fibonacci_thread
Hardware assisted breakpoint 2 at 0x4007c4: file fibonacci-threads.c, line 17.

(gdb) ignore 2 3
Will ignore next 3 crossings of breakpoint 2.

(gdb) c
Continuing.

[New Thread 0x7ffffff84a710 (LWP 2066)]
[New Thread 0x7ffffff7049710 (LWP 2067)]
[New Thread 0x7ffffff848710 (LWP 2069)]
[Switching to Thread 0x7ffffff848710 (LWP 2069)]
Breakpoint 2, fibonacci_thread (arg=0x8) at fibonacci-threads.c:17
17 size_t n = ( size_t )arg, fib;

(gdb) info threads
[New Thread 0x7ffffff5846710 (LWP 2070)]
  5 Thread 0x7ffffff846710 (LWP 2070) 0x00007ffffff92f541 in clone () from /lib64/libc.so.6
* 4 Thread 0x7ffffff6848710 (LWP 2069) fibonacci_thread (arg=0x8) at fibonacci-threads.c:17
  3 Thread 0x7ffffff7049710 (LWP 2067) 0x00007ffffff92f541 in clone () from /lib64/libc.so.6
  2 Thread 0x7ffffff84a710 (LWP 2066) 0x00007ffffff92f541 in clone () from /lib64/libc.so.6
  1 Thread 0x7ffffff7ff3700 (LWP 2063) 0x00007ffffff7bc7f7d in pthread_join () from /lib64/libpthread.so.0

(gdb) thread 1
[Switching to thread 1 (Thread 0x7ffffff7ff3700 (LWP 2063))]#0 0x00007ffffffbc7f7d in pthread_join () from /lib64/libpthread.so.0

(gdb) thread apply all bt full
One Ping to Rule Them All

(gdb) thread apply all bt full

Thread 5 (Thread 0x7ffffff5846710 (LWP 2070)):
#0 0x00007ffff792f541 in clone () from /lib64/libc.so.6
No symbol table info available.
#1 0x00007ffff7bc68c0 in ?? () from /lib64/libpthread.so.0
No symbol table info available.
#2 0x00007ffff5846710 in ?? ()
No symbol table info available.
#3 0x0000000000000000 in ?? ()
No symbol table info available.

Thread 4 (Thread 0x7ffffff6848710 (LWP 2069)):
#0 fibonacci_thread (arg=0x8) at fibonacci-threads.c:17
  n = 0
  fib = 0
  ern = 0
  th1 = 0
  th2 = 0
  pvalue = 0x0
#1 0x00007ffff7bc6980 in start_thread () from /lib64/libpthread.so.0
No symbol table info available.
#2 0x00007ffff792f57d in clone () from /lib64/libc.so.6
No symbol table info available.

Thread 3 (Thread 0x7ffffff7049710 (LWP 2067)):
#0 0x00007ffff792f541 in clone () from /lib64/libc.so.6
No symbol table info available.
#1 0x00007ffffbc732b in pthread_create@GLIBC_2.2.5 () from /lib64/libpthread.so.0
No symbol table info available.
#2 0x00007ffff792f57d in clone () from /lib64/libc.so.6
No symbol table info available.

Thread 2 (Thread 0x7ffffff7049710 (LWP 2067)):
#0 0x00007ffff792f541 in clone () from /lib64/libc.so.6
No symbol table info available.
#1 0x00007ffffbc732b in pthread_create@GLIBC_2.2.5 () from /lib64/libpthread.so.0
No symbol table info available.
#2 0x00007ffff792f57d in clone () from /lib64/libc.so.6
No symbol table info available.

...
How to See a Deadlock

* Deadlock - reversed order of resource acquisition

(gdb) thread apply all bt

Thread 2 (Thread 0x7f9eca220710 (LWP 2527)):
  #0 0xca5a42e4 in __l1l_lock_wait () from /lib64/libpthread.so.0
  #1 0xca59f459 in _L_lock_953 () from /lib64/libpthread.so.0
  #2 0xca59f27b in pthread_mutex_lock () from /lib64/libpthread.so.0
  #3 0x0040091e in fibonacci_thread (arg=0x0) at fibonacci-deadlock.c:39
  #4 0xca59c980 in start_thread () from /lib64/libpthread.so.0
  #5 0xca30557d in clone () from /lib64/libc.so.6

Thread 1 (Thread 0x7f9eca9ca700 (LWP 2526)):
  #0 0xca5a42e4 in __l1l_lock_wait () from /lib64/libpthread.so.0
  #1 0xca59f459 in _L_lock_953 () from /lib64/libpthread.so.0
  #2 0xca59f27b in pthread_mutex_lock () from /lib64/libpthread.so.0
  #3 0x004009b1 in fibonacci (n=100) at fibonacci-deadlock.c:63
  #4 0x00400881 in main (argc=2, argv=0x7fffbb4bf678) at fibonacci-main.c:28
Threads: Off to the Races

* Race condition - not enough mutual exclusion
* The most painful kind of bug
* So watch it

(gdb) watch question
Hardware watchpoint 2: question

(gdb) b fibonacci_thread
Breakpoint 3 at 0x4007c0: file fibonacci-race.c, line 23.

(gdb) c
Continuing.
[New Thread 0x7ffff784a710 (LWP 2395)]
[Switching to Thread 0x7ffff784a710 (LWP 2395)]
Breakpoint 3, fibonacci_thread (arg=0x0) at fibonacci-race.c:23
23     size_t prev = 0, next = 1, i;

(gdb) c
Continuing.
[Switching to Thread 0x7ffff7ff3700 (LWP 2392)]
Hardware watchpoint 2: question
Old value = 100
New value = 18
fibonacci (n=100) at fibonacci-race.c:50
50     if (( ern = pthread_join( th, 0 ))) err( ern, "pthread_join" );
Wonderland: C++

* Prepare for ugly data printouts
* Name mangling
* Multiple symbols for overloading and virtuals
* Exceptions

```c
(gdb) ptype v
```
```c
type = class std::vector<unsigned long, std::allocator<unsigned long>> : protected
std::_Vector_base<unsigned long, std::allocator<unsigned long>> {
    public:
        void vector(void);
    ...
```
```c
(gdb) p v
```
```c
$3 = ((anonymous namespace)::fvec & ) @0x7fffffff82c0: {<std::_Vector_base<unsigned
long, std::allocator<unsigned long>> >> = {
    _M_impl = {<std::allocator<unsigned long>> >> = {
        __gnu_cxx::new_allocator<unsigned long>> = {
            No data fields}, <No data fields}, _M_start = 0x603010, _M_finish =
0x603060, _M_end_of_storage = 0x603108}}, <No data fields>
```
```c
(gdb) set print vtbl on
(gdb) set print object on
(gdb) set print pretty on
(gdb) catch throw
(gdb) catch catch
```
Back to the Basics

* Info
* Printing expressions
* Examining memory

(gdb) info registers|all-registers|float|vector|handle|scope
...
(gdb) p/x n
$4 = 0x9

(gdb) p/c n
$5 = 9 't'

(gdb) x/3i $pc
=> 0x400ad5 <next_fibonacci+13>: mov  -0x38(%rbp),%rax
   0x400ad9 <next_fibonacci+17>: mov %rax,%rdi
   0x400adc <next_fibonacci+20>: callq 0x400c3a
<_ZNKSt6vectorImSaImEE4sizeEv>
Back to the Basics

* Assigning variables
* Changing registers and memory
* Calling functions

(gdb) set variable n = 13
(gdb) set $rax = n
(gdb) set *( float* )&n = 3.14
(gdb) set *( unsigned char* )$pc = 0xCC
(gdb) call next_fibonacci( v )
* Automatic display at each break

(gdb) display/i $pc
(gdb) display/c n
(gdb) info display
Auto-display expressions now in effect:
Num  Enb  Expression
2:   y   /c n
1:   y   /lbi $pc

(gdb) c
Continuing.
Breakpoint 2, next_fibonacci (v=...) at fibonacci-cpp.cpp:18
18     size_t n( v.size());
2: /c n = 19 '\023'
1:  /i $pc
=> 0x400ad5 <next_fibonacci+13>: mov   -0x38(%rbp),%rax
What about the SysAdmin

* Very sharp tool
* Use other tools first :)

freebsd:~$ truss
freebsd:~$ dtrace
linux:~$ ltrace
linux:~$ strace
Linux:~$ pstack pmap pldd
both:~$ nm
both:~$ objdump
both:~# ldd
both:~$ gcore
define snap
  dont-repeat
  set pagination off
  set print vtbl on
  set print demangle on
  set print union on
  set print object on
  set print pretty on
  set print array on
  set print array-indexes on
  set print object on
  set print static-members on
  set logging file gdb.log
  set logging redirect on
  set logging on
  attach $arg0
  thread apply all bt full
  output $_siginfo
  detach
end
(gdb) snap PID
Advanced: i.e. not here

* Conditional break- and watchpoints
* Thread-specific breakpoints
* Breakpoint commands
* Non-stop async mode
* Stepping backwards
* Checkpoints and restart
* Remote debugging
* Kernel debugging
* User commands
* Python interface
* GUI interfaces
References and Links

* NYC*BUG - New York City BSD User Group: http://www.nycbug.org
* GDB home page: http://www.gnu.org/software/gdb/
* GDB user manual: http://sourceware.org/gdb/current/onlinedocs/gdb/
* GDB Wiki: http://sourceware.org/gdb/wiki/
* GDB STL support: http://sourceware.org/gdb/wiki/STLSupport
* x86 debug registers: http://en.wikipedia.org/wiki/X86_debug_register
* GDB Cheat Sheet: http://www.yolinux.com/TUTORIALS/GDB-Commands.html
* dotfiles .gdbinit: http://dotfiles.org/~mkfs/.gdbinit
* Data Display Debugger (DDD): http://www.gnu.org/software/ddd/
* KDE Debugger (KDbg): http://www.kdbg.org
* Valgrind Toolset: http://www.valgrind.org
* Sun DTrace: http://wikis.sun.com/display/DTrace/Documentation

* This presentation and all the silly source code: https://www.fetissov.org/wiki/GDB