

Debugging Tools and Techniques

Thus Spake The Master Programmer: “When you have learned to snatch the error code from the trap frame, it will be time for you to Leave.”

- The Tao Of Programming

Ananth Shrinivas

Solaris Engineering

Sun Microsystems

Approx & Non-Linear Agenda

- **Common Problems and Common Tools**
- **The Problems in Theory**
 - Memory Management
 - Profiling and Execution Tracing
 - Core Dumps, Network Monitoring
- **The Solutions in Practice**
 - GDB, Netcat, Wireshark (*The Swiss Army Knives*)
 - strace and ltrace (*Dynamic Execution Tracers*)
 - Valgrind and Friends (*Emulators and Interposing Libraries*)
 - gprof and oprofile (*Instrumentation and Sampling Profilers*)
- **The One True Tool: State of Art in Debugging**

Common Problems

- **Memory Management:** Invalid Pointers, Buffer Overflows, Double Frees, Memory Leaks
- **Tracing Execution and Flow Control:** Profiling and Performance analysis, Code path verification, Code coverage, Debugging Logical errors
- **Multi-threaded Programming:** Race Conditions, Deadlocks, Lock Contention
- **Advanced Problems:** Core Dumps, Disassembly, Debugging your operating system, Compiler Bugs, Hardware Bugs, Debugger Bug, Understanding foreign code !

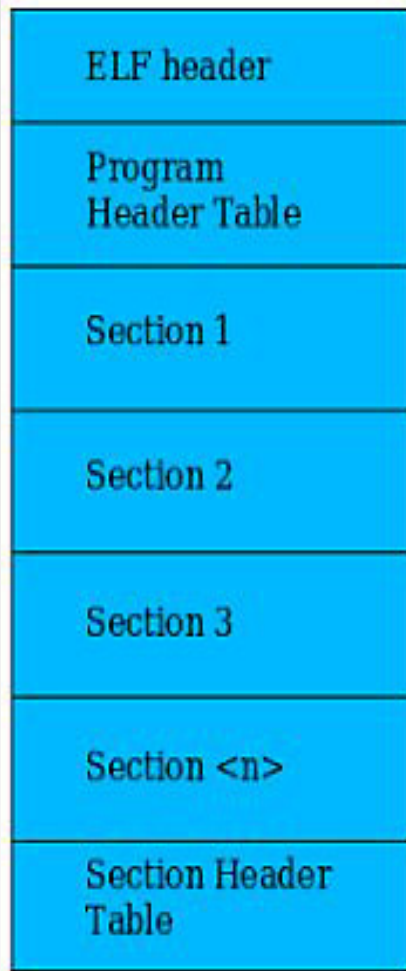
Common Linux Debugging Tools

- **Memory:** valgrind, Insure++, Purify, memwatch
- **Execution Tracing:** strace, ltrace, gdb
- **Process Monitoring:** pmap, lsof, top, /proc
- **Profiling:** gprof, oprofile, CodeAnalyst, vTune
- **Code Coverage:** gcov
- **Multithreaded Programming:** helgrind, \$BRAIN
- **General Purpose Debuggers:** gdb, dbx, DDD
- **Static Code Analyzers:** gcc, lint, splint, Purify
- **Grokkers:** cflow, cscope, ctags, lxr, opengrok

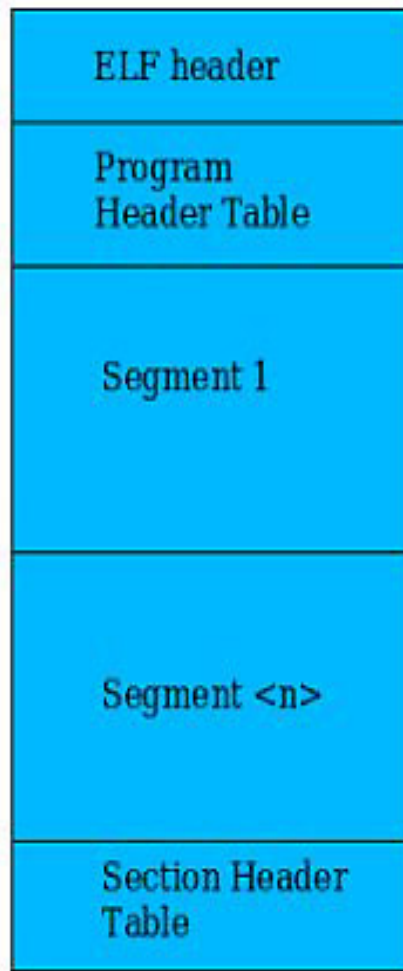
Underlined, Italicized = Proprietary Tools

A Program on Disk - Executable

Linking View



Execution View



Executable and Linkable Format

Relocatable files (gcc -c)
Shared objs (gcc -shared)
Executable files (ld)

readelf – Read elf headers, sections and symbol tables

Objdump – Disassemble elf objects and hack around

Neat Tools: **nm**, **strings**, **od**

Courtesy:

http://www.linuxforums.org/misc/understanding_elf_using_readelf_and_objdump.htm

A Program in Memory - Process

High Address

Args and env vars

Stack



Unused memory



Heap

Uninitialized Data Segment (bss)

Initialized Data Segment

Text Segment

Low Address

Text Segment – Machine Code. Shared, Read-Only.

Data Segment – Initialized global variables from executable

BSS – Has uninitialized global variables set to zero (Block started by symbol)

Stack – Collection of stack frames. Grows Downward

Heap – Dynamic memory for programs and libraries. Grows Upward

The GNU Debugger (GDB)

- Source/Instruction => Process/Core/Kernel(kgdb)
- Frequently used commands (TAB Autocompletes)
 - **file/attach** – load a binary file for execution
 - **kill/run** – the loaded file or process
 - **list** – list a file, funcs, lines, addr.
 - **break/clear** – breakpoint at func, lines, addr.
 - **step/stepi** – single step one source line/mi
 - **next/nexti** – step over subroutines
 - **cont** – continue until next breakpoint or end
 - **disable/enable** – breakpoint manipulation

The GNU Debugger (GDB)

- Frequently used commands (continued)
 - **bt** – Print backtrace of all stack frames
 - **frame N** – switch stack frame context
 - **display/print** – variables, expressions
 - **ptype** – print type of variable (stabs/ctf)
 - **info** – shows a huge number of useful things
- Tools useful in conjunction with GDB
 - **pmap** – Display process address layout
 - **elfsh** – interactive shell for elfdump !
 - **biew** – ncurses gui to explore elf objects

Preparing for a GDB session

- **Don't strip** – Cost of Disk \ll Cost of Engineering
- Never ever omit the frame pointer (**-fomit-frame-pointer** is *evil*)
- Add the enhanced symbol table (**gcc -g**)
- Disable optimizations when creating a debug executable (**-O0**)
- Add GNU specific extensions for a lot of extra debugging power (**-ggdb3**)
- **.gdbinit** – put redundant commands into this file

DDD - GDB for X

The screenshot displays the DDD interface for debugging a C program. The main window shows the source code of `prog.c` with the following content:

```
20     printf("Usage: %s [strings]\n", argv[0]);
21     exit(1);
22 }
23
24 for ( i = 0 ; i < argc ; i ++ ) {
25     do_print(i, argv[i]);
26 }
27
28 printf("Total: %d\n", i);
29
30 return 0;
31 }
32
```

The registers window shows the following values:

Register	Value
eax	0x42
ecx	0x0
edx	0xb7f680b
ebx	0xb7f66ff
esp	0xbfdb84a
ebp	0xbfdb84e
esi	0xb7f97ce
edi	0x0
eip	0x8048430
eFlags	0x200282
cs	0x73
ss	0x7b

The execution window shows the current state of the program:

```
Usage: /home/ananth/Desktop/debugging/examples/gdb/prog [strings]
```

The assembly view shows the following instructions:

```
0x080483f4 <main+62>: movl $0x6,0xffffffe8(%ebp)
0x080483fb <main+69>: movl $0x7,0xfffffec(%ebp)
0x08048402 <main+76>: movl $0x8,0xffffff0(%ebp)
0x08048409 <main+83>: movl $0x9,0xffffff4(%ebp)
0x08048410 <main+90>: movl 0xffffffc8(%ebp),%eax
0x08048413 <main+93>: cmpl $0x1,(%eax)
0x08048416 <main+96>: jg 0x804843c <main+134>
0x08048418 <main+98>: movl 0xffffffc8(%ebp),%edx
0x0804841b <main+101>: movl 0x4(%edx),%eax
0x0804841e <main+104>: movl (%eax),%eax
0x08048420 <main+106>: movl %eax,0x4(%esp)
0x08048424 <main+110>: movl $0x8048580,(%esp)
0x0804842b <main+117>: call 0x80482b4 <printf@plt>
0x08048430 <main+122>: movl $0x1,(%esp)
0x08048437 <main+129>: call 0x80482c4 <exit@plt>
0x0804843c <main+134>: movl $0x0,0xfffffff8(%ebp)
0x08048443 <main+141>: jmp 0x8048466 <main+176>
```

The status bar at the bottom indicates "Showing integer registers only." and the system tray shows the date and time: "Sun Jan 7, 9:04:13 PM".

strace

- Powerful runtime tool to trace syscalls and signals.
- Restrict to system calls or classes using **-e trace (!) = syscall|set|process|network|ipc|file|desc**
- Attach to existing process using **-p**
- Follow children of fork() **-f** and vfork() **-F**
- Coarse profiling using **-tt, -r , -c**
- **-s <n>** display upto only n characters
- Symbol name demangling using **-C**
- Instruction Pointer at the time of trace **-i**
- Log output to a file using **-o** and **-ff**

Itrace

- Runtime tool to trace library calls
(How are library calls and syscalls different)
- Aggregate syscalls by count **-c**
- Use ldd to find out static link-time library dependencies and **-l** , **-L** to filter library names.
- Indent call flow using **-n**
- Trace system calls to using **-S** !
- Most other options are strace syntax compatible
- For Itrace Internals see PTRACE(2)
- **BUGGY** ! ELF32 only ! dlopen() not traced !

Profiling Tools

- Time from Shell / `gettimeofday()` / `clock()`
- Instrumentation Profilers
 - GPROF – Collection/Analysis of execution profile
 - GCOV - Hotspot detection using code coverage
 - Quantum Limitations – Heisenberg Principle
- Sampling Profilers
 - `oprofile` – Kernel, CPU supported counters and event monitors – understand your CPU well.
 - AMD CodeAnalyst and Intel vTune

Nifty Tools for Unlucky Days

➤ Networking Tools

- Wireshark – Brilliant protocol analyzer
- Netstat – A lot of useful statistics and views
- Netcat – TCP/IP Swiss Army Knife
- Nmap – Network Exploration / Port Scanner

➤ Filesystem Tools

- fuser – Identify processes using a file/socket
- lsof – List of open files. Command line hell
- watch (-d) – Repeatedly executes a command.
Waits for output to change. Highlights the change.

MM :: Valgrind / Cachegrind

- The **most advanced** MM debugging tool
- Use of uninitialized memory
- Reading/writing memory after it has been freed
- Reading/writing off the end of malloc() areas
- Reading/writing to wrong addresses on the stack !
- Memory leaks - i.e. malloc() pointers lost forever
- Mismatched use of malloc/new[] vs. free/delete[]
- Overlapping pointers in memcpy() and friends
- Some misuses of the POSIX pthreads API
- Memory hog ! 25-75 times slower ! -O0 works best

The One True Tool

If you thought Valgrind was mind-boggling wait until you see what is the *state-of-art* in debugging.

DTrace (OpenSolaris, FreeBSD, MacOS)