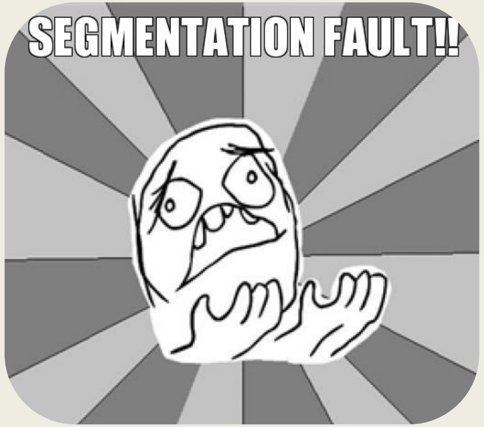
GDB

1. Что это ?
2. Зачем ?
3. Как работает ?
4. Как пользоваться ?

Что это?

A *debugger* can start some process and debug it, or attach itself to an existing process. It can single-step through the code, set breakpoints and run to them, examine variable values and stack traces.

Зачем?

1. нет нормального API для вывода на экран
2. работа с core dump
3. 

……..

Как работает?

> *man ptrace*

The *ptrace()* system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers.

Как работает?

Проблема: нужно узнать название функции, название переменных, их тип и т. д. Для этого в объектные файлы записывают дебажную информацию - структура, которая все это описывает.

Как работает?

gcc -g

gcc -g3

-g :

Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF 2). GDB can work with this debugging information.

-g3:

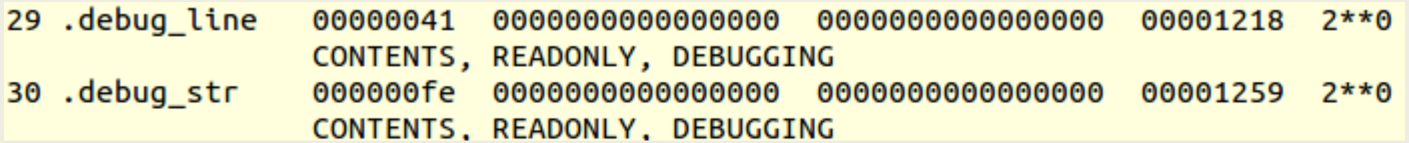
Level 3 includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use -g3.

Как работает?

DWARF is a complex format, building on many years of experience with previous formats for various architectures and operating systems. It has to be complex, since it solves a very tricky problem - presenting debugging information from any high-level language to debuggers, providing support for arbitrary platforms and ABIs.

Как работает?

ELF defines arbitrary sections that may exist in each object file. A *section header table* defines which sections exist and their names.
objdump -h - Display the contents of the section headers



```
29 .debug_line    00000041  0000000000000000  0000000000000000  00001218  2**0
                  CONTENTS, READONLY, DEBUGGING
30 .debug_str     000000fe  0000000000000000  0000000000000000  00001259  2**0
                  CONTENTS, READONLY, DEBUGGING
```

*.debug_* - DWARF debugging sections

Как работает?

```
objdump --dwarf=info
objdump -d
...
```

Starting

gdb -tui exe-file
  run
gdb -tui
  file exe-file
  break location ("b main", "b 5", "b hello.c:23")
  run [arg1] [arg2] [...] *(r)*
  help run
  quit (q)

Stepping and running

next(n)	Run to the next line of this function
step(s)	Step into the function on this line, if possible
stepi	Step a single assembly instruction
continue(c)	Keep running from here
CTRL-C	Stop running, wherever you are
finish(fin)	Run until the end of the current function
advance location	Advance to a location, line number, or file (e.g. "fun", "5", or "hello.c:23")
jump location (j)	Just like continue, except jump to a particular location first.

Examining and Modifying Variables

display expression (disp)	Display the value every step of the program
info display (i disp)	Show a list of expressions currently being displayed
undisplay num (undisp 2)	Stop showing an expr identified by its number (info display)
print expression (p var)	Print the value of a variable or expression
printf formatstr expressionlist	Do some formatted output with printf() e.g. printf "i = %d, p = %s\n", i, p
set variable expression	Set a variable to value, e.g. set variable x=20
set (expression)	Works like set variable

Examining and Modifying Variables

x/2wd $rsp - Examine two (4-byte) words starting at address in $rsp. Print in decimal

help x
x/[NUM][SIZE][FORMAT] where
    NUM = number of objects to display

    SIZE = size of each object
    (b=byte, h=half-word, w=word, g=giant (quad-word))

    FORMAT = how to display each object
    (d=decimal, x=hex, s=string, c=char, f=float etc.)

Examining and Modifying Variables

disp /16wd $rsp - Display examined two (4-byte) words starting at address in $rsp. Print in decimal

```
disp /16cb $rax

print /x ($rsp+8)
p *(int *) ($rsp+8)  = x /dw ($rsp + 8)
p (char *) 0xbfff890 = x /s 0xbfff890

p $xmm0
p $xmm0.v4_float; p $xmm0.v2_double; p $xmm0.uint128…

set $xmm0.uint128=0xFFFFFFFFFFFFFFFF
```

Window Commands

info win (i win)	Shows current window info
focus winname (fs)	Set focus to a particular window by name ("SRC", "CMD", "ASM", or "REG") or by position ("next" or "prev")
layout type	Set the window layout ("src", "asm", "split", or "reg")
tui reg type	Set the register window layout ("general", "float", "system", or "next")
winheight val (wh src + 20)	Set the window height (either an absolute value, or a relative value prefaced with "+" or "-")

Breakpoints and Watchpoints

watch expr (wat e)	Break when a variable is written to
rwatch expression	Break when a variable is read from
info break (i b)	Display breakpoint and watchpoint information and numbers
clear location	Clear a breakpoint from a location
delete num (d 1)	Delete a breakpoint or watchpoint by number
delete (d)	Delete all breakpoints
backtrace	Show the current stack

…

list fun (l fun)	list a few lines of the source code around fun

return (ret)
return expr
	cancel execution of a function call

backtrace (bt)	Show the current stack
info registers (i reg) i all-reg	Dump integer registers to screeni (reg esp ebx ecx)
TBD	

Core files

In computing, a core dump consists of the recorded state of the working memory of a program at a specific time, generally when the program crashed.

Including the processor registers, memory management information, and other processor and operating system flags and information.

Core files

```
ulimit -c unlimited
gdb -tui -c core exe-file
```

when that coredump was produced on another machine:

- on another computer:

  ```
  run
  i shared
  ```
  copy libraries to libs/
- on your computer:

```
gdb -tui exe-file
  set solib-absolute-prefix libs/
  set solib-search-path libs/
  core core
```