

gdb 操作说明

调试准备

- 安装 gdb
- 生成可调试的程序

```
gcc -g test.c -o test
```

- 启动 gdb

```
# gdb
```

进入后，用 **file** 命令加载执行程序

```
# gdb test
```

基本命令和功能

- **list** 列出多行源代码
- **运行程序** **start**、**run**、**step**、**next**、**continue**、**finish**
- **break** 设置断点
- **disassemble** 反汇编
- **Info** 查看信息
- **x** 查看内存
- **watch** 监视变量值的变化
- **display** 跟踪查看某个变量
- **file** 重新载入待调试程序
- **backtrace** 查看当前函数及其被调用的信息
- **help** 查看命令帮助
- **quit** 退出 gdb 环境

显示程序

list 列出多行源代码

命令的 简写形式: **l**

根据行号和函数名查看。除了可以查看本文件源码，还可以查看其他文件的源码。

```
(gdb) l          # 每执行1次显示10行, 再执行1次显示次10行
(gdb) l 15        # 显示第15行, 此时会将15行显示在屏幕窗口中央, 方便查看前后的代码
(gdb) l main      # 显示本文件的main函数
```

```
(gdb) l tools.cpp:15      # 在tools.cpp中, 显示第15行附近的代码
(gdb) l tools.cpp:sum     # 在tools.cpp中, 查看sum函数的代码
```

列出源代码的函数是可以设置的。

```
(gdb) show list          # 显示行数      show  listsize
(gdb) set list 20        # 设置行数      set    listsize  20
```

显示程序

list 列出多行源代码

- **help list** 或者 **help l**, 显示**list** 的各种操作

```
(gdb) help list
List specified function or line.
With no argument, lists ten more lines after or around previous listing.
"list -" lists the ten lines before a previous ten-line listing.
One argument specifies a line, and ten lines are listed around that line.
Two arguments with comma between specify starting and ending lines to list.
Lines can be specified in these ways:
  LINENUM, to list around that line in current file,
  FILE:LINENUM, to list around that line in that file,
  FUNCTION, to list around beginning of that function,
  FILE:FUNCTION, to distinguish among like-named static functions.
  *ADDRESS, to list around the line containing that address.
With two args, if one is empty, it stands for ten lines away from
the other arg.

By default, when a single location is given, display ten lines.
This can be changed using "set listsize", and the current value
can be shown using "show listsize".
```

显示程序

list 列出多行源代码

- **|** : lists ten more lines after or around previous listing
- **| -** : lists the ten lines before a previous ten-line listing
- **| 15** : list around that line **in current file**
- **| 15,30**
- **| main** : 显示 main 函数
- **| *address | *\$rip rip** 对应的源程序
- **| test.c:15** : 指明当前文件为 **test.c**
- **| test.c:main**

每次显示的行数可以修改，缺省情况下，每次列出 **10**行；

show listsize ; 显示当前显示行数 设置的值

set listsize 20 ; 将每次列出的代码行数设为 **20**

程序的运行

start、run、step、next、continue

命令的简写形式：**start、r、s、si、n、c**

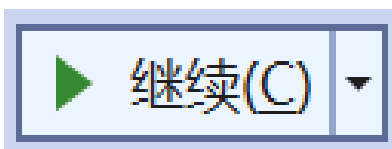
- **start**: 开始执行程序，在第一条可执行语句处停下来；
- **run** : 运行程序，直到第一个断点或程序结束；
- **step** : 执行下一条语句，如果该语句为函数调用，则执行所调用函数执行其第一行语句；
- **si [N]** : 执行一(N)条机器指令，s 是执行一条C语句
对于 **call** 指令，进入被调函数；
- **next**: 执行下一条语句，如果该下一条语句为函数调用，不会进入函数内部执行(即不会一步步调试函数内部语句)。
- **nexti [N]** : 类似 si，不进入 call 调用的函数
- **continue**: 继续程序的运行，直到下一个断点。如果没有下一个端点，则程序执行结束。

程序的运行

start 、 run、 step、 next、 continue
命令的 简写形式: **start、 r、 s、 n、 c**

▶ 开始调试(S)	F5
▶ 开始执行(不调试)(H)	Ctrl+F5

start 、 run



continue

↓ 逐语句(S)	F11
↺ 逐过程(O)	F10
↑ 跳出(T)	Shift+F11

step

next

finish 跳出函数

until 跳出循环

程序的运行

start、run、step、next、continue

命令的简写形式: **start、r、s、n、c**

```
(gdb) list
1      #include <stdio.h>
2      int main()
3      {
4          int  x, y, z;
5          x=10;
6          y=20;
7          z=x*3+y*6+4*8;
8          printf("z=%d \n", z);
9          return 0;
10     }
(gdb) start
Temporary breakpoint 1 at 0x652: file test.c, line 5.
Starting program: /home/test

Temporary breakpoint 1, main () at test.c:5
5          x=10;
(gdb)
```

程序的运行

start、**run**、**step**、**next**、**continue**

命令的 简写形式: **start**、**r**、**s**、**n**、**c**

```
5          x=10;
6          y=20;
7          z=x*3+y*6+4*8;
8          printf("z=%d \n", z);
9          return 0;
10         }
(gdb) start
Temporary breakpoint 1 at 0x652: file test.c, line 5.
Starting program: /home/test

Temporary breakpoint 1, main () at test.c:5
5          x=10;
(gdb) s
6          y=20;
(gdb) n
7          z=x*3+y*6+4*8;
(gdb)
```

程序的运行

同样可以使用 **help start**、**help run**、**help step**、**help next**、**help continue** 查看命令的帮助信息。

```
(gdb) help run
Start debugged program.
You may specify arguments to give it.
Args may include "*", or "[...]"; they are expanded using the
shell that will start the program (specified by the "$SHELL" environment
variable). Input and output redirection with ">", "<", or ">>"
are also allowed.

With no arguments, uses arguments last specified (with "run" or
"set args"). To cancel previous arguments and run with no arguments,
use "set args" without arguments.

To start the inferior without using a shell, use "set startup-with-shell off".
(gdb)
```

设置断点

- **break (b)** : 命令之后可接源文件行、函数名称、地址等:
 - break [line-num]** : 按代码行来设置断点;
 - break [function-name]**: 按函数名来设置断点;
 - break *地址**: 按地址来设置断点;
 - break if <condition>**: 条件成立时程序停住;
 - info break**: 查看断点

```
(gdb) b 9
Breakpoint 2 at 0x8000693: file test.c, line 9.
```

```
(gdb) disass
Dump of assembler code for function main:
   0x000000000800064a <+0>:      push    %rbp
   0x000000000800064b <+1>:      mov     %rsp, %rbp
   0x000000000800064e <+4>:      sub     $0x10, %rsp
   0x0000000008000652 <+8>:      movl   $0xa, -0xc(%rbp)
```

```
(gdb) break *0x800068e
Breakpoint 4 at 0x800068e: file test.c, line 8.
```

取消断点

- **break (b)** : 命令之后可接源文件行、函数名称、地址等;
- **Help break** : 可以看到 **break** 的使用帮助
- **d 3** 删除第 3 个断点
 - **dis 4** 将第 4 个断点设为无效
 - **ena 4** 将第 4 个断点设为有效

```
(gdb) break main
Breakpoint 5 at 0x8000652: file test.c, line 5.
(gdb) info break
```

Num	Type	Disp	Enb	Address	What
2	breakpoint	keep	y	0x0000000008000693	in main at test.c:9
3	breakpoint	keep	y	0x00000000000c3544	
4	breakpoint	keep	y	0x000000000800068e	in main at test.c:8
5	breakpoint	keep	y	0x0000000008000652	in main at test.c:5

```
(gdb)
```

反汇编

- **disassemble** 命令简写 **disass**
 - **Help disass** : 可以看到 **disass** 的使用帮助
 - **disass**
 - **disass /s** 列出源程序及反汇编程序 或 **/m**
 - **disass /r** 列出指令的机器码
 - **disass /rs** 列出指令的机器码、源程序
 - **disass main** 列出 **main** 函数的反汇编代码
 - **disass** 函数名, +长度; 函数前 **length** 字节反汇编
 - **disass** 起始地址, 终止地址 ; 对该区间的代码反汇编
 - **disass** 起始地址, +长度; 对该区间的代码反汇编
- 起始的表达式: 数字, 函数名, 变量名, 寄存器名

反汇编

```
(gdb) disass /rs phase_2, +20
Dump of assembler code from 0x8000f76 to 0x8000f8a:
phases.c:
51      {
      0x0000000008000f76 <phase_2+0>:      55      push    %rbp
      0x0000000008000f77 <phase_2+1>:      48 89 e5      mov     %rsp, %rbp
      0x0000000008000f7a <phase_2+4>:      48 83 ec 40    sub     $0x40, %rsp
      0x0000000008000f7e <phase_2+8>:      48 89 7d c8    mov     %rdi, -0x38(%rbp)
      0x0000000008000f82 <phase_2+12>:     64 48 8b 04 25 28 00 00 00      mov
```

```
(gdb) disass /s 0x8000fe2, 0x8000ff0
Dump of assembler code from 0x8000fe2 to 0x8000ff0:
phases.c:
67      for(i = 2; i < 6; i++) {
=> 0x0000000008000fe2 <phase_2+108>:      movl     $0x2, -0x24(%rbp)
      0x0000000008000fe9 <phase_2+115>:      jmp      0x8001012 <phase_2+122>

68      if (numbers[i] != numbers[i - 1] + i)
      0x0000000008000feb <phase_2+117>:      mov     -0x24(%rbp), %eax
      0x0000000008000fee <phase_2+120>:      cltq

End of assembler dump.
```

获得当前指令的地址，用 i reg rip

反汇编

disass \$rip, +20

disass \$rsp, +40 // 将堆栈中的数据当指令解析

Disass buf, +50 // 将变量buf中的数据当指令解析
buf 是一个字符数组类型的变量

disass &x, +4

例 : int x=10; p &x, x &x 等都可看 x的地址

disass /r &x, +4

0x00..... : 0a 00 or (%rax), %al

0x00..... : 00 00 add %al, (%rax)

disass &x, &y // 两个地址之间的内容的反汇编

反汇编

- 可以用不同格式显示反汇编代码
- **set disassembly-flavor intel or att**

```
(gdb) set disassembly-flavor att
(gdb) disass /r
Dump of assembler code for function main:
   0x000000000800064a <+0>:      55          push    %rbp
   0x000000000800064b <+1>:      48 89 e5     mov     %rsp, %rbp
   0x000000000800064e <+4>:      48 83 ec 10   sub     $0x10, %rsp
=>  0x0000000008000652 <+8>:      c7 45 f4 0a 00 00 00   movl    $0xa, -0xc(%rbp)
   0x0000000008000659 <+15>:     c7 45 f8 14 00 00 00   movl    $0x14, -0x8(%rbp)
```

```
(gdb) set disassembly-flavor intel
(gdb) disass /r
Dump of assembler code for function main:
   0x000000000800064a <+0>:      55          push    rbp
   0x000000000800064b <+1>:      48 89 e5     mov     rbp, rsp
   0x000000000800064e <+4>:      48 83 ec 10   sub     rsp, 0x10
=>  0x0000000008000652 <+8>:      c7 45 f4 0a 00 00 00   mov     DWORD PTR [rbp-0xc], 0xa
   0x0000000008000659 <+15>:     c7 45 f8 14 00 00 00   mov     DWORD PTR [rbp-0x8], 0x14
   0x0000000008000660 <+22>:     8b 55 f4     mov     edx, DWORD PTR [rbp-0xc]
```

```
(gdb) help set disassembly-flavor
Set the disassembly flavor.
The valid values are "att" and "intel", and the default value is "att".
(gdb) _
```

查看信息

➤ **info** 命令简写 **i**，可以查看的内容非常多

List of info subcommands:

```
info address -- Describe where symbol SYM is stored
info all-registers -- List of all registers and their contents
info args -- Argument variables of current stack frame
info auto-load -- Print current status of auto-loaded files
info auxv -- Display the inferior's auxiliary vector
info bookmarks -- Status of user-settable bookmarks
info breakpoints -- Status of specified breakpoints (all user-settable break
info checkpoints -- IDs of currently known checkpoints
info classes -- All Objective-C classes
info common -- Print out the values contained in a Fortran COMMON block
info copying -- Conditions for redistributing copies of GDB
info dcache -- Print information on the dcache performance
info display -- Expressions to display when program stops
info exceptions -- List all Ada exception names
info extensions -- All filename extensions associated with a source language
info files -- Names of targets and files being debugged
info float -- Print the status of the floating point unit
info frame -- All about selected stack frame
info frame-filter -- List all registered Python frame-filters
info functions -- All function names
```

查看信息

➤ 查看寄存器

➤ **info registers**

简化写法 **i registers** or **i reg** or **i r**

➤ **i register rax**

➤ **i all-registers** or **i all-register**

```
(gdb) i register eax
eax                0x800064a          134219338
(gdb)
```

查看信息

- **i b** 查看断点信息
- **i display** 查看跟踪查看哪变量
- **i source** 程序的信息
- **i stack**

查看内存

➤ **X /fmt address** **fmt** 是查看格式，**address** 为地址

➤ 地址的表达形式

简单类型的变量取地址，数组、指针、寄存器、函数名

int x ; → &x

int a[10] ; → a

对于寄存器，以寄存器中的内容为 地址

→ \$rip

显示栈顶的 **10**个字节单元（以**16**进制形式）

x /10xb \$rsp

显示 **main** 函数开头的 **20**个字节（以**16**进制形式）

x /20xb main

查看内存

➤ **x/fmt address** **fmt** 是查看格式，**address** 为地址

fmt 中包含的信息：**nfu**

显示的内存单元的个数(**n**)；缺省为 1；

显示的格式(**f**)：**x** 十六进制；**d** 十进制；**o** 八进制；**t** 二进制

u 十进制无符号数； **f** 浮点数 **c** 字符 **i** 指令地址

s 字符串

每个内存单元的大小(**u**)：**b**: 1字节，**h**: 2字节，**w**: 4字节，
g: 8字节 缺省为 4字节

```
(gdb) display x
4: x = 10
(gdb) display y
5: y = 20
(gdb) display z
6: z = 182
(gdb) x/12b &x
0x7fffffffef1b4: 0x0a    0x00    0x00    0x00    0x14    0x00    0x00    0x00
0x7fffffffef1bc: 0xb6    0x00    0x00    0x00
```

显示表达式的值

- **display /fmt exp** 以fmt 格式查看表达式exp 的值
- 每次运行暂停，显示**display**设置的要观察的表达式的值
- **i display** : 列出要显示的表达式
- **undisplay** 编号1 [,编号2] 取消表达式的显示

```
(gdb) display x
7: x = 10
(gdb) display x+15
8: x+15 = 25
(gdb) display /x x
9: /x x = 0xa
(gdb) display &x
10: &x = (int *) 0x7fffffffef1b4
(gdb)
```

监视表达式的值改变

➤ **watch exp**

当被设置的观察点变量发生变化时，打印显示！

与 **display** 命令一直显示某个变量的值的方式相比，用 **watch** 命令只显示变量值的变化情况，如果变量值不变则不显示，调试效率要高很多。

更多的用法

使用 help ， 获取帮助信息

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb)
```

更多的用法

➤ **help 类名**

命令被分成了多个类（如**data, running,files,...**），
得到该类下的命令；

➤ **help all**

获取所有的命令帮助信息

➤ **help 命令**

获取一个命令的帮助信息

更多的用法

➤ **help 类名** : 获取一个类的操作命令

```
(gdb) help running
Running the program.

List of commands:

advance -- Continue the program up to the given location (same form as
attach -- Attach to a process or file outside of GDB
continue -- Continue program being debugged
detach -- Detach a process or file previously attached
detach checkpoint -- Detach from a checkpoint (experimental)
detach inferiors -- Detach from inferior ID (or list of IDs)
disconnect -- Disconnect from a target
finish -- Execute until selected stack frame returns
handle -- Specify how to handle signals
inferior -- Use this command to switch between inferiors
interrupt -- Interrupt the execution of the debugged program
jump -- Continue program being debugged at specified line or address
kill -- Kill execution of program being debugged
kill inferiors -- Kill inferior ID (or list of IDs)
next -- Step program
nexti -- Step one instruction
queue-signal -- Queue a signal to be delivered to the current thread wh
reverse-continue -- Continue program being debugged but run it in rever
```

更多的用法

➤ **help 命令** : 获取一个操作命令的帮助信息

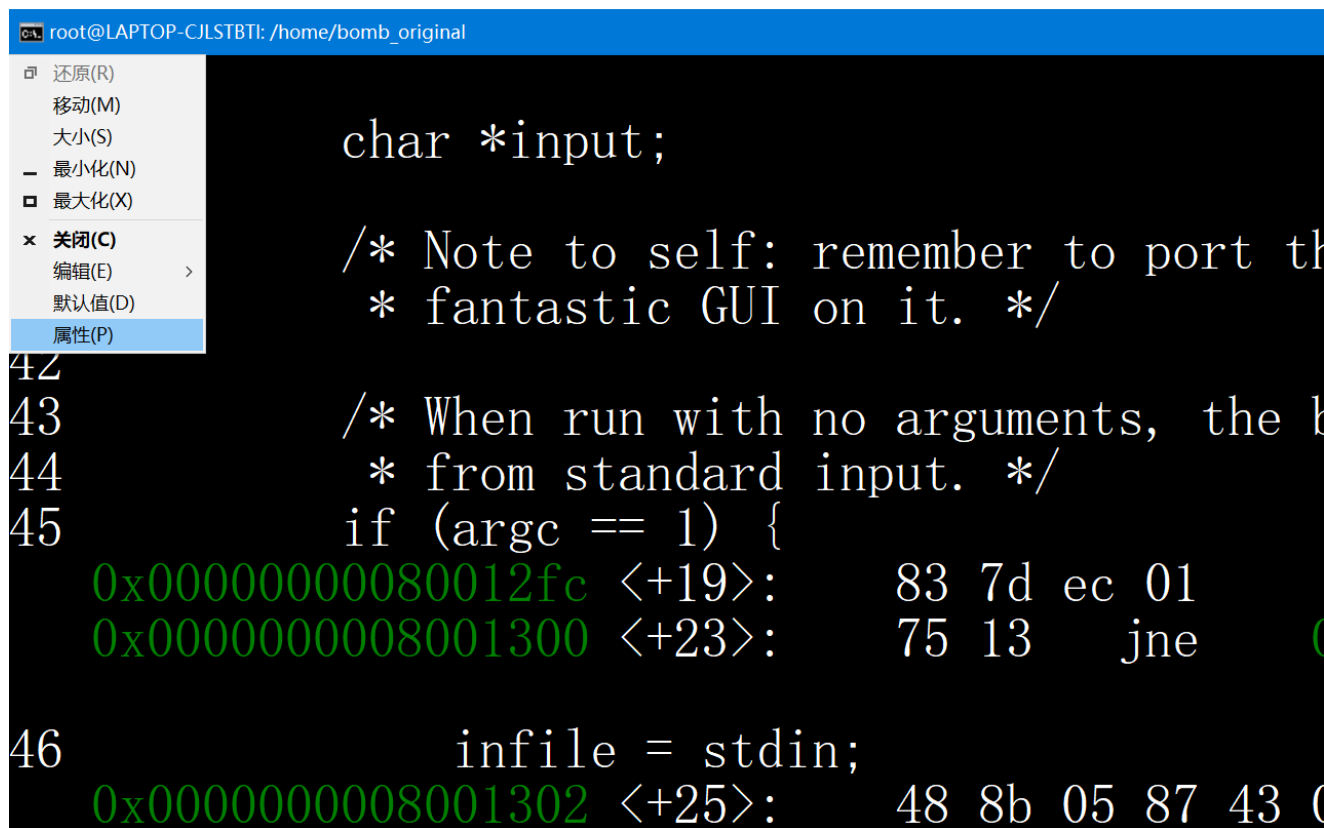
```
(gdb) help continue
Continue program being debugged, after signal or breakpoint.
Usage: continue [N]
If proceeding from breakpoint, a number N may be used as an argument,
which means to set the ignore count of that breakpoint to N - 1 (so that
the breakpoint won't break until the Nth time it is reached).

If non-stop mode is enabled, continue only the current thread,
otherwise all the threads in the program are continued. To
continue all stopped threads in non-stop mode, use the -a option.
Specifying -a and an ignore count simultaneously is an error.
(gdb)
```

GDB 显示界面的调整

通过 **CMD** 窗口属性的设置来调整

单击窗口左上角的图标，在弹出菜单上单击“属性”



The screenshot shows a GDB terminal window with a blue title bar. A context menu is open on the left side of the terminal, listing standard window management options. The '属性(P)' (Properties) option is highlighted in blue. The terminal background is black with white text for C code and green text for assembly code. The code includes variable declarations, comments, and assembly instructions with memory addresses.

```
root@LAPTOP-CJLSTBTI: /home/bomb_original
还原(R)
移动(M)
大小(S)
最小化(N)
最大化(X)
关闭(C)
编辑(E)
默认值(D)
属性(P)

char *input;

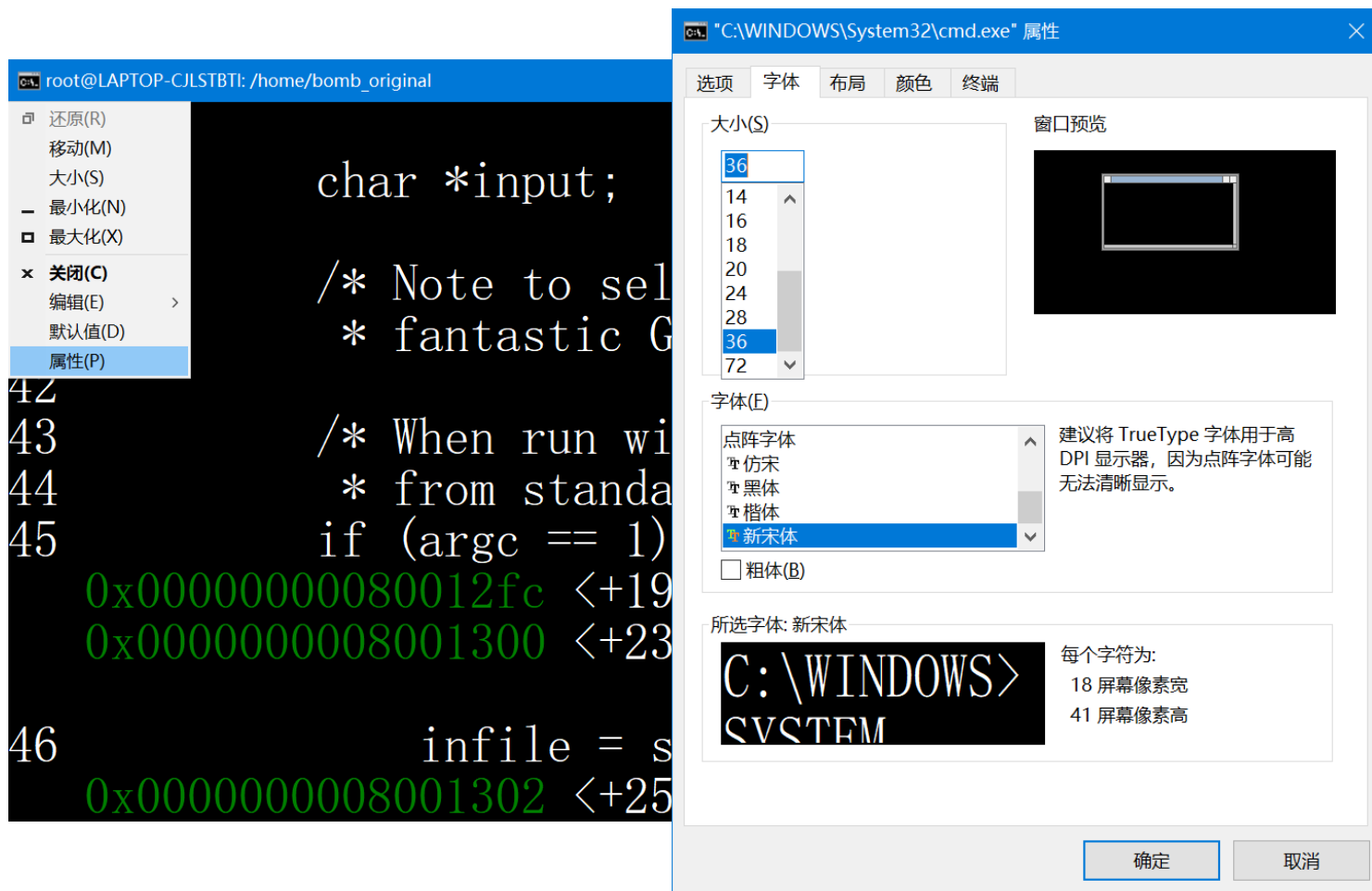
/* Note to self: remember to port th
 * fantastic GUI on it. */

42
43 /* When run with no arguments, the k
44 * from standard input. */
45 if (argc == 1) {
0x00000000080012fc <+19>: 83 7d ec 01
0x0000000008001300 <+23>: 75 13 jne

46 infile = stdin;
0x0000000008001302 <+25>: 48 8b 05 87 43 0
```

GDB 显示界面的调整

在“属性”窗口中，可设置“字体”，“颜色”等



GDB 显示界面的调整

<https://sourceware.org/gdb/>

在gdb 中使用命令: `set style textType attr value`

`textType`: 源码 (`source`) 、 地址 (`address`)

`attr` : 背景颜色 (`background`) 、 字体颜色 (`foreground`)
粗细 (`intensity`)

`value` : 颜色或者粗细 (`normal`, `bold`, `dim`)

`set style address foreground green`

; 设置地址显示的前景色为 绿色

`set style address intensity bold`

; 设置地址显示粗体

GDB 显示界面的调整

layout 设置布局

(gdb) help layout // 显示 layout 的用法

layout src // 显示源程序窗、gdb命令窗

layout asm //源程序窗、反汇编窗、命令窗

layout regs // 增加 寄存器窗

layout prev // 前一个窗

layout next // 下一个窗

info wins // 显示窗口信息 行数,

winheight nme [+ /-] line

update 更新源代码窗和当前执行点

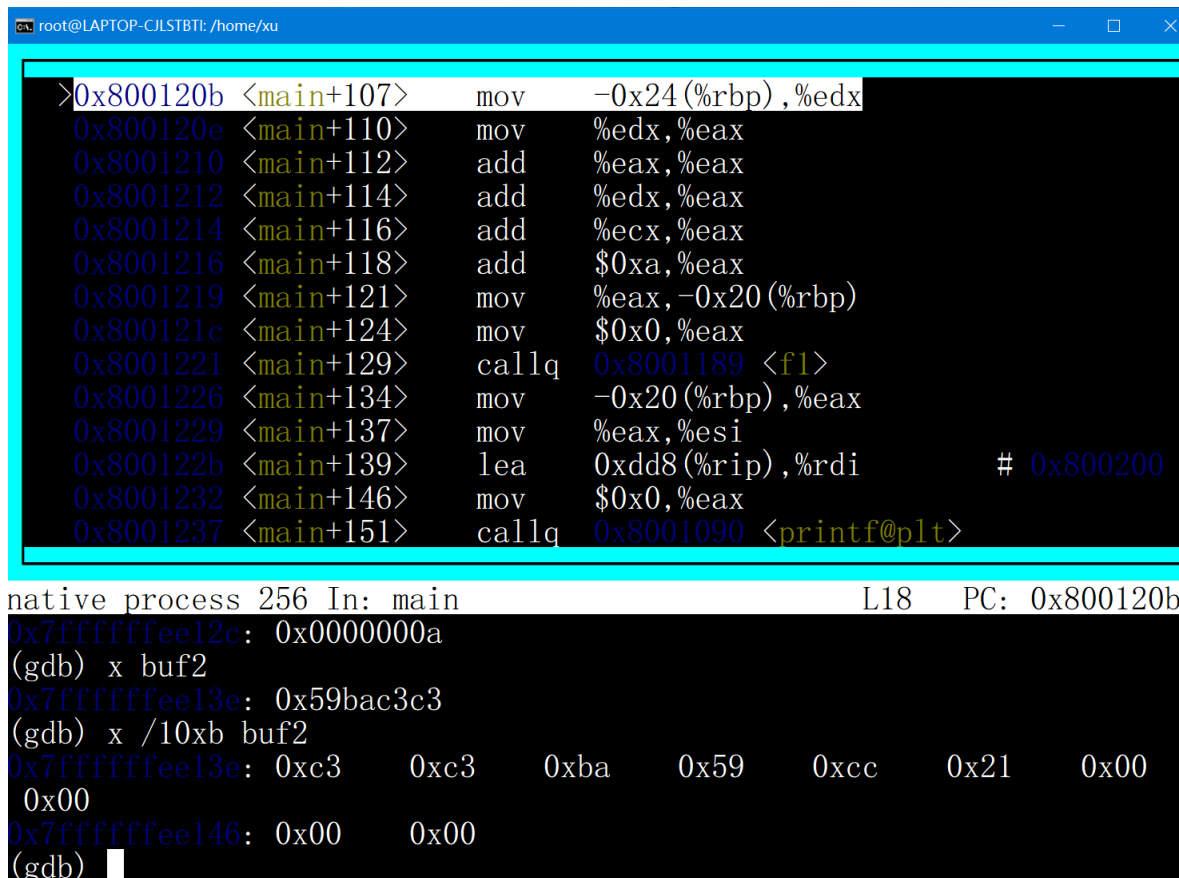
refresh 刷新所有窗口

focus cmd/src/asm/regs/next/prev

GDB 显示界面的调整

layout 设置布局

用箭头键，可以对焦点窗口上下、左右移动或翻页



```
root@LAPTOP-CJLSTBTI: /home/xu
>0x800120b <main+107>  mov    -0x24(%rbp),%edx
0x800120e <main+110>  mov    %edx,%eax
0x8001210 <main+112>  add    %eax,%eax
0x8001212 <main+114>  add    %edx,%eax
0x8001214 <main+116>  add    %ecx,%eax
0x8001216 <main+118>  add    $0xa,%eax
0x8001219 <main+121>  mov    %eax,-0x20(%rbp)
0x800121c <main+124>  mov    $0x0,%eax
0x8001221 <main+129>  callq  0x8001189 <f1>
0x8001226 <main+134>  mov    -0x20(%rbp),%eax
0x8001229 <main+137>  mov    %eax,%esi
0x800122b <main+139>  lea    0xdd8(%rip),%rdi    # 0x800200
0x8001232 <main+146>  mov    $0x0,%eax
0x8001237 <main+151>  callq  0x8001090 <printf@plt>

native process 256 In: main                               L18   PC: 0x800120b
0x7fffffffe12c: 0x0000000a
(gdb) x buf2
0x7fffffffe13e: 0x59bac3c3
(gdb) x /10xb buf2
0x7fffffffe13e: 0xc3    0xc3    0xba    0x59    0xcc    0x21    0x00
0x00
0x7fffffffe146: 0x00    0x00
(gdb)
```

Ctrl-x a : 切换 显示不同窗口个数的布局