之前想验证一些关于堆栈的问题，但是没什么好方法，printf实在局限，流于表面，只间表象（值、范围、规律）不见真身（地址、寄存器、过程），所以想到了gdb——一个强大的调试工具，还能看汇编代码，现在先把这两天学的常用的命令做一个小结，以后有用到的可能再来更新一下：

括号内为全称补全，缩写全称均可用。

例：(e)x(amine)表示既可以用x又可以用examine

(gdb)代表gdb环境命令行提示符。

**关于缩写**，非常类似Linux的shell中的tab功能，但是与shell不同的是有默认选择：

你不一定要写全，也不一定只写首字母，比如(gdb) layout 命令，如果写个l，那么缺省的是list，抢不过，写layout——又太麻烦，你只要写上la、lay、layo都行，抢不上槽没关系，只要有一点不同，就默认是你了。

**1.进入gdb：**

#gdb test -q(uiet)

其中test为目标可执行文件，-q代表不打印那一大串版本版权信息之类的刷屏字幕。

这里有个小常识就是用gcc编译目标文件test时，记得-g，表示可调试。

另外，直接进入gdb而未加载可执行文件，或者加载了目标文件，想换一个其他的——可以使用

(gdb)file test2

或者

(gdb)exec(-file) test2

**2.断点的设立：**

(gdb)b(reakpoints) <rowNums...>

<rowNums...>代表想要设立断点的行数

忘了哪行是什么？没关系，你可以用list

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) list
2. 1   #include<stdio.h>
3. 2   **int** main()
4. 3   {
5. 4       **int** i = 10;
6. 5       i = 11;
7. 6       printf("the address of i is %p and the value of i is %d\n",&i,i);
8. 7   }
9. 8

(gdb) list

1 #include<stdio.h>

2 int main()

3 {

4 int i = 10;

5 i = 11;

6 printf("the address of i is %p and the value of i is %d\n",&i,i);

7 }

8

另外，list也可以设置显示行数和指定位置的

(gdb)list

(gdb)list 10

(gdb)list 5,10

(gdb)func

比如默认显示10行，可以指定第5到第10行，指定显示某函数代码，等等。

不过最好的建议还是开俩终端，一边看代码，一边调试，看着舒服。

另外

(gdb) layout

也可以显示程序代码，还是用框子圈起来的，高大上。

(gdb)b func

(gdb)b \*func

在函数func()设立断点，星号代表进入前，插结果——一目了然~！

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) b \*main
2. Breakpoint 1 at 0x80483e4: file testPC.c, line 3.
3. (gdb) b main
4. Breakpoint 2 at 0x80483ed: file testPC.c, line 4.
5. (gdb) info b
6. Num     Type           Disp Enb Address    What
7. 1       breakpoint     keep y   0x080483e4 in main at testPC.c:3
8. 2       breakpoint     keep y   0x080483ed in main at testPC.c:4

(gdb) b \*main

Breakpoint 1 at 0x80483e4: file testPC.c, line 3.

(gdb) b main

Breakpoint 2 at 0x80483ed: file testPC.c, line 4.

(gdb) info b

Num Type Disp Enb Address What

1 breakpoint keep y 0x080483e4 in main at testPC.c:3

2 breakpoint keep y 0x080483ed in main at testPC.c:4

其中

(gdb)info b(reakpoints)

相当于列表打印所有已设立的断点。有了断点，当然也可以删掉断点，看到列表中左边的”Num“了么，用得上：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) d 1
2. (gdb) info b
3. Num     Type           Disp Enb Address    What
4. 2       breakpoint     keep y   0x080483ed in main at testPC.c:4
5. (gdb)

(gdb) d 1

(gdb) info b

Num Type Disp Enb Address What

2 breakpoint keep y 0x080483ed in main at testPC.c:4

(gdb)

(gdb)d(elete) Num

代表删除第Num个断点。可以看到第一个断点被删了。

**3.基本调试流程：**

有了断点，就该用上了。

第一步，开始运行程序：

(gdb)r(un)

(gdb)n(ext)

(gdb)s(tep)

和其他调试相仿，这两条分别代表step over和step in，

(gdb)c(ontinue)

run和continue功能其实差不多，都是继续往下运行，直到下一个断点停下来，不过场合不一样罢了：run是开始运行前的启动命令，continue是运行中的命令。

**4.汇编style：**

基本的流程走完了，该引入汇编了。

i代表指令（instruction）

不很确定，至少你不能用instruction代替i，至少，先理解为汇编的意思。

前边的指令加上i就显示了汇编代码，例如：

n(ext)i

s(tep)i

要想一步一步看汇编代码和执行过程，

(gdb)ni

(gdb)si

是必不可少的，不过你可以用回车表示继续使用上一次的命令。

前边提到list和layout显示源代码，其实layout还可以扩展一下用途

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb)layout asm

(gdb)layout asm

以窗口形式显示汇编代码

**5.print：**

gdb提供了打印功能：

示例：

(gdb)p(rint) i

打印i变量当前的值。

不仅程序中的变量，寄存器的值也能打印

(gdb) p $pc

两个小疑问：

**5.1.$pc代表什么，除了它还能打印什么？**

**这句话其实就是打印程序计数器的值。**

先说寄存器，除了$pc，还有%esp，%edp等等等等，

具体到底能打印那些，又要牵扯到另一条命令了，下面看一例：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb)i(nfo) r(eg)
2. (gdb)
3. eax            0x80484f0    134513904
4. ecx            0xbffff304    -1073745148
5. edx            0xb    11
6. ebx            0xb7fc2ff4    -1208209420
7. esp            0xbffff240    0xbffff240
8. ebp            0xbffff268    0xbffff268
9. esi            0x0    0
10. edi            0x0    0
11. eip            0x8048406    0x8048406 <main+34>
12. eflags         0x200282    [ SF IF ID ]
13. cs             0x73    115
14. ss             0x7b    123
15. ds             0x7b    123
16. es             0x7b    123
17. fs             0x0    0
18. gs             0x33    51

(gdb)i(nfo) r(eg)

(gdb)

eax 0x80484f0 134513904

ecx 0xbffff304 -1073745148

edx 0xb 11

ebx 0xb7fc2ff4 -1208209420

esp 0xbffff240 0xbffff240

ebp 0xbffff268 0xbffff268

esi 0x0 0

edi 0x0 0

eip 0x8048406 0x8048406 <main+34>

eflags 0x200282 [ SF IF ID ]

cs 0x73 115

ss 0x7b 123

ds 0x7b 123

es 0x7b 123

fs 0x0 0

gs 0x33 51

上边看到的都可以print，而且能发现个小规律，这个info reg打印的，除了最左边是寄存器名称外，中间是寄存器存的值（也就是一个内存地址），右边是这个值对应的内存地址中的值。打印一下$eax可验证：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) p $eax
2. $3 = 134513904

(gdb) p $eax

$3 = 134513904

其实用法远不止于此，比如p $打印上一次打印的值，$$打印上上次打印过的值。print其实是有计数器的，每次print打印，其实都有一个类似count++在内部发生，使用print $num 能显示第num个打印结果（如上，**p $3**就等价于**p $eax**），其他还有blabla~~~

至于为什么是$，贪心的外国人把各种变量都弄成美元$了，所以这个也是gdb下设置的环境变量~~开个玩笑。

其实，我猜，$也是为了区分变量和表达式吧~print可以打印表达式的。

与其说print不只打印变量（左值），还打印表达式（右值），不如说，按描述，print本来就是打印表达式的，只不过表达式包括变量。

欲知详情，可以使用help查看使用说明

**5.2.C语言中printf有打印格式控制，那么gdb的print呢？**

也有~

(gdb)p i  
(gdb)p/a i  
(gdb)p/c i  
(gdb)p/f i  
(gdb)p/x i  
(gdb)p/o i  
(gdb)p/d i  
(gdb)p/t i  
......

反斜杠后边这几个参数分别控制打印的进制与格式：  
f浮点，c字符。。。  
t为二进制，o八，x十六，d十  
另外：a和x同样是打印十六进制，区别呢？可能就是不同名但同功能

**理念有点像C语言编程时候加printf打印变量来监视程序。在gdb中你也可以随时打印各变量的值，而且更为强大（不用像C到处插打印命令，还能逐条执行，打印变量加地址加寄存器你说强大不）。**

**6.display：**

这是一种设置，设置好了调试过程中每一步都回显一次，有点像echo吧~~

示例：

(gdb) display /3i $pc  
中，3指的是一次显示几行，不输入，缺省为1  
但是~~~怎么修改，而且有一种错觉，通常都是一次定义以后，再怎么定义都不会变（有时候确实会变~！！！）~~~~~~~~~~~~  
找到了

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) undisplay <dnums...>

(gdb) undisplay <dnums...>

<dnums...>为编号，但是直观感觉上像是覆盖的，至少不知道怎么调回原来的设置  
另外还有

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. delete display <dnums...>
2. disable display <dnums...>
3. enable display <dnums...>

delete display <dnums...>

disable display <dnums...>

enable display <dnums...>

至于怎么灵活用？是先info一下，然后再enable一下，就代表当前使用这种显示？  
还是他们同时显示？所以造成了我“有些设置不起作用，有些能起作用”的错觉。  
因为（行数）比原来多能“立刻见效”，比原来少则不能？

清空了重新si一遍，真正的原因是同时显示好几份。终端刷屏相似度太高眼花缭乱啊有木有~又没有clear功能~  
一行的也有，二行的也有，三行的也有。所有设置的顺次显示一遍

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) si
2. 0xb7fec1ec in ?? () from /lib/ld-linux.so.2
3. 7: x/i $pc
4. => 0xb7fec1ec:    mov    %eax,%edi
5. 6: x/2i $pc
6. => 0xb7fec1ec:    mov    %eax,%edi
7. 0xb7fec1ee:    shr    $0x8,%edi
8. 5: x/3i $pc
9. => 0xb7fec1ec:    mov    %eax,%edi
10. 0xb7fec1ee:    shr    $0x8,%edi
11. 0xb7fec1f1:    mov    %edi,%ecx
12. 4: x/i $pc
13. => 0xb7fec1ec:    mov    %eax,%edi

(gdb) si

0xb7fec1ec in ?? () from /lib/ld-linux.so.2

7: x/i $pc

=> 0xb7fec1ec: mov %eax,%edi

6: x/2i $pc

=> 0xb7fec1ec: mov %eax,%edi

0xb7fec1ee: shr $0x8,%edi

5: x/3i $pc

=> 0xb7fec1ec: mov %eax,%edi

0xb7fec1ee: shr $0x8,%edi

0xb7fec1f1: mov %edi,%ecx

4: x/i $pc

=> 0xb7fec1ec: mov %eax,%edi

通过info display打印显示表，可以查到自己的设置。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) info display
2. Auto-display expressions now in effect:
3. Num Enb Expression
4. 7:   y  /1bi $pc
5. 6:   y  /2bi $pc
6. 5:   y  /3bi $pc
7. 4:   y  /1bi $pc

(gdb) info display

Auto-display expressions now in effect:

Num Enb Expression

7: y /1bi $pc

6: y /2bi $pc

5: y /3bi $pc

4: y /1bi $pc

最后，想看一下全部汇编代码，直接

(gdb)disassemble

或者去用objdump（题外）

**7.bt**

最近调服务器发现普通打印法已经很难跟住bug了，另一个原因是段错误就系统重启（其实是工程设置的信号处理，SIGSEGV段错误信号重启系统。），而gdb能在重启之前截断程序运行，从而卡在出错点，防止重启系统。

不过光卡在那也不行啊，一层套一层，那么多的函数调用，你不进去看不到东西啊，而且再输入n(next)往下走是看不到已经运行完的错误的，所以就谈到bt（backtrace）命令——回溯。

**8.etc.**

其他的乱七八糟，例如examine。

(e)xamine：功能和display差不太多，区别就是display是一种设置，每次跳命令显示一次，x是主动显示。

x/3i $pc显示3条指令（3为示范，数字可选）  
(gdb)(e)x(amine)  
语法：  
x/<n/f/u> <addr>  
n选择从当前地址向后显示几个  
f是显示格式，还有s字符串和i整型  
u表示从当前地址往后请求的字节数，如果不指定的话，GDB默认是4个bytes。u参数可以用下面的字符来代替，b表示单字节，h表示双字节，w表示四字节，g表示八字节。当我们指定了字节长度后，GDB会从指内存定的内存地址开始，读写指定字节，并把其当作一个值取出来。  
  
例子：  
x/3uh 0x54320表示，从地址0x54320读取，h表示以双字节为单位，3表示三个单位，u表示十六进制

**0.HELP：**

前边print提到功能太多，方法太多，想知道最详细的，请

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/huqinwei987/article/details/23548239)[在CODE上查看代码片](https://code.csdn.net/snippets/288704)

1. (gdb) help print

(gdb) help print

关于help的强大和使用方法，不赘述了，使用

(gdb) help

就什么都知道了。

----------------------------------------------------------------------------------------------------------------

**ADDITIONAL：**

GDB7.0以上（7.4）

可用如下套路：

(gdb)set disassemble-next-on

(gdb)b main  
(gdb)r  
(gdb)ni  
(gdb)ni  
.....  
这个也是比较不错比较直观的方式  
  
disas /m main  
让C和汇编同时显示