陈皓专栏 【空谷幽兰，心如皓月】

芝兰生于深谷，不以无人而不芳；君子修道立德，不为困穷而改节。

我的BLOG

陈皓专栏（技 术）(RSS)
酷壳（编程和技术）(RSS)

个人资料

haoel

访问： 2334645次
积分： 17381分
排名： 第91名

原创： 120篇 转载： 6篇
译文： 15篇 评论： 4710条

文章搜索

文章分类

技术趋势 (13)
抄袭事件 (7)
编程工具 (19)
编程语言 (58)
职业心情 (23)
软件开发 (28)
项目管理 (9)

文章存档

2011年04月 (1)
2011年02月 (3)
2010年09月 (1)
2010年08月 (2)
2010年07月 (5)

展开

阅读排行

用GDB调试程序（一） (88102)
跟我一起写 Makefile（一 (87915)
C++ 虚函数表解析 (75463)
其实Unix很简单

2013年1月当选微软MVP名单揭晓！ CSDN博客频道年终送好礼获奖名单公布！
2012CSDN博客之星评选正式上线 2000元大奖征异构开发博文 2013年全国百所高校巡讲讲师招募

用GDB调试程序（四）

分类： 编程工具 2003-07-09 08:30 12858人阅读 评论(2) 收藏 举报

查看栈信息

当程序被停住了，你需要做的第一件事就是查看程序是在哪里停住的。当你的程序调用了一个函数，函数的地址，函数参数，函数内的局部变量都会被压入“栈”（Stack）中。你可以用GDB命令来查看当前的栈中的信息。

下面是一些查看函数调用栈信息的GDB命令：

```
backtrace
bt
    打印当前的函数调用栈的所有信息。如：

    (gdb) bt
    #0  func (n=250) at tst.c:6
    #1  0x08048524 in main (argc=1, argv=0xbffff674) at tst.c:30
    #2  0x400409ed in __libc_start_main () from /lib/libc.so.6

    从上可以看出函数的调用栈信息：__libc_start_main --> main() --> func()

backtrace <n>
bt <n>
    n是一个正整数，表示只打印栈顶上n层的栈信息。

backtrace <-n>
bt <-n>
    -n表一个负整数，表示只打印栈底下n层的栈信息。
```

如果你要查看某一层的信息，你需要在切换当前的栈，一般来说，程序停止时，最顶层的栈就是当前栈，如果你要查看栈下面层的详细信息，首先要做的是切换当前栈。

```
frame <n>
f <n>
    n是一个从0开始的整数，是栈中的层编号。比如：frame 0，表示栈顶，frame 1，表示栈的第二层。

up <n>
    表示向栈的上面移动n层，可以不打n，表示向上移动一层。

down <n>
```

(62817)
再谈“我是怎么招聘程序员
(54256)
一些重要的算法
(51996)
C++ 对象的内存布局（上
(46037)
哥是玩程序的
(43844)
清华大学出版社“抄袭事件
(43597)
恐怖的C++语言
(39271)

评论排行

清华大学出版社“抄袭事件 (323)
再谈“我是怎么招聘程序员 (264)
“清华大学出版社抄袭事件 (233)
我是怎么招聘程序员的 (233)
C++ 虚函数表解析 (203)
6个变态的C语言写的Hel (201)
恐怖的C++语言 (198)
优秀程序员的十个习惯 (165)
Java构造时成员初始化的 (146)
惹恼程序员的十件事 (130)

推荐文章

最新评论

跟我一起写 Makefile（一）
wangjilebeishang: 我的神啊，可以，顶《《《《《学习

五个方法成为更好的程序员
雨中风铃_: 受教！

C++ 虚函数表解析
Walkerinwind: 第一个例子结束实际运行"经"果如下或许应该是结果?

“抄袭事件”判决书
fengyishang: 很遗憾，时隔七年，我才看到七年前这一起关于著作权的纠纷；但很幸运，因为最后的结果正是我们想要的，虽然...

C++ 虚函数表解析
zhiweiarm: 受益匪浅，学习了

22条经典的编程引言
雨中风铃_: 太经典了！

C++ 对象的内存布局（下）
Huang850615: cout << " "; pFun(); //D::f();这个地方出错了，把它改成这样：cout...

五个方法成为更好的程序员
冰冻牙膏: 寻找不同观点，取长补短

跟我一起写 Makefile（一）
luomuxiaoxiao98: Mark！

其实Unix很简单
lovelock: @yhmhappy2006:呵呵，我真不知道你干了什么。。。。

表示向栈的下面移动n层，可以不打n，表示向下移动一层。

上面的命令，都会打印出移动到的栈层的信息。如果你不想让其打出信息。你可以使用这三个命令：

```
select-frame <n> 对应于 frame 命令。
up-silently <n> 对应于 up 命令。
down-silently <n> 对应于 down 命令。
```

查看当前栈层的信息，你可以用以下GDB命令：

```
frame 或 f
```

会打印出这些信息：栈的层编号，当前的函数名，函数参数值，函数所在文件及行号，函数执行到的语句。

```
info frame
info f
```

这个命令会打印出更为详细的当前栈层的信息，只不过，大多数都是运行时的内内地址。比如：函数地址，调用函数的地址，被调用函数的地址，目前的函数是由什么样的程序语言写成的、函数参数地址及值、局部变量的地址等等。如：

```
(gdb) info f
Stack level 0, frame at 0xbffff5d4:
 eip = 0x804845d in func (tst.c:6); saved eip 0x8048524
 called by frame at 0xbffff60c
 source language c.
 Arglist at 0xbffff5d4, args: n=250
 Locals at 0xbffff5d4, Previous frame's sp is 0x0
 Saved registers:
  ebp at 0xbffff5d4, eip at 0xbffff5d8
```

```
info args
```

打印出当前函数的参数名及其值。

```
info locals
```

打印出当前函数中所有局部变量及其值。

```
info catch
```

打印出当前的函数中的异常处理信息。

查看源程序

一、显示源代码

GDB 可以打印出所调试程序的源代码，当然，在程序编译时一定要加上-g的参数，把源程序信息编译到执行文件中。不然就看不到源程序了。当程序停下来以后，GDB会报告程序停在了那个文件的第几行上。你可以用list命令来打印程序的源代码。还是来看一看查看源代码的GDB命令吧。

```
list <linenum>
```

显示程序第linenum行的周围的源程序。

```
list <function>
```

显示函数名为function的函数的源程序。

list
显示当前行后面的源程序。

list -
显示当前行前面的源程序。

一般是打印当前行的上5行和下5行，如果显示函数是是上2行下8行，默认是10行，当然，你也可以定制显示的范围，使用下面命令可以设置一次显示源程序的行数。

set listsize <count>
设置一次显示源代码的行数。

show listsize
查看当前listsize的设置。

list命令还有下面的用法：

list <first>, <last>
显示从first行到last行之间的源代码。

list , <last>
显示从当前行到last行之间的源代码。

list +
往后显示源代码。

一般来说在list后面可以跟以下这们的参数：

<linenum> 行号。
<+offset> 当前行号的正偏移量。
<-offset> 当前行号的负偏移量。
<filename:linenum> 哪个文件的哪一行。
<function> 函数名。
<filename:function> 哪个文件中的哪个函数。
<*address> 程序运行时的语句在内存中的地址。

二、搜索源代码

不仅如此，GDB还提供了源代码搜索的命令：

forward-search <regexp>
search <regexp>
向前面搜索。

reverse-search <regexp>
全部搜索。

其中，<regexp>就是正则表达式，也主一个字符串的匹配模式，关于正则表达式，我就不在这里讲了，还请各位查看相关资料。

三、指定源文件的路径

某些时候，用-g编译过后的执行程序中只是包括了源文件的名字，没有路径名。GDB提供了可以让你指定源文件

的路径的命令，以便GDB进行搜索。

directory <dirname ... >

dir <dirname ... >

加一个源文件路径到当前路径的前面。如果你要指定多个路径，UNIX下你可以使用“:”，Windows下你可以使用“;”。

directory

清除所有的自定义的源文件搜索路径信息。

show directories

显示定义了的源文件搜索路径。

四、源代码的内存

你可以使用info line命令来查看源代码在内存中的地址。info line后面可以跟“行号”，“函数名”，“文件名:行号”，“文件名:函数名”，这个命令会打印出所指定的源码在运行时的内存地址，如：

```
(gdb) info line tst.c:func
Line 5 of "tst.c" starts at address 0x8048456 <func+6> and ends at 0x804845d <func+13>.
```

还有一个命令（disassemble）你可以查看源程序的当前执行时的机器码，这个命令会把目前内存中的指令dump出来。如下面的示例表示查看函数func的汇编代码。

```
(gdb) disassemble func
Dump of assembler code for function func:
0x8048450 <func>:       push   %ebp
0x8048451 <func+1>:     mov    %esp,%ebp
0x8048453 <func+3>:     sub    $0x18,%esp
0x8048456 <func+6>:     movl   $0x0,0xfffffffc(%ebp)
0x804845d <func+13>:    movl   $0x1,0xfffffff8(%ebp)
0x8048464 <func+20>:    mov    0xfffffff8(%ebp),%eax
0x8048467 <func+23>:    cmp    0x8(%ebp),%eax
0x804846a <func+26>:    jle    0x8048470 <func+32>
0x804846c <func+28>:    jmp    0x8048480 <func+48>
0x804846e <func+30>:    mov    %esi,%esi
0x8048470 <func+32>:    mov    0xfffffff8(%ebp),%eax
0x8048473 <func+35>:    add    %eax,0xfffffffc(%ebp)
0x8048476 <func+38>:    incl   0xfffffff8(%ebp)
0x8048479 <func+41>:    jmp    0x8048464 <func+20>
0x804847b <func+43>:    nop
0x804847c <func+44>:    lea    0x0(%esi,1),%esi
0x8048480 <func+48>:    mov    0xfffffffc(%ebp),%edx
0x8048483 <func+51>:    mov    %edx,%eax
0x8048485 <func+53>:    jmp    0x8048487 <func+55>
0x8048487 <func+55>:    mov    %ebp,%esp
0x8048489 <func+57>:    pop    %ebp
0x804848a <func+58>:    ret
End of assembler dump.
```

<-上一页 下一页->

（版权所有，转载时请注明作者和出处）

上一篇：用GDB调试程序（五）

分享到：

下一篇：用GDB调试程序（七）

查看评论

2楼 zxj557 2012-01-31 17:17发表

写的挺全面的，正好小弟我想学习一下gdb。哈哈……

1楼 instructors 2011-05-05 15:32发表

[e01][e01]顶一下

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

联系邮箱：webmaster(at)csdn.net

Copyright © 1999-2012, CSDN.NET, All Rights Reserved