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。

用GDB调试程序（三）

分类： 编程工具　　2003-07-21 18:36　　12401人阅读　　评论(8)　收藏　举报

四、维护停止点

上面说了如何设置程序的停止点，GDB中的停止点也就是上述的三类。在GDB中，如果你觉得已定义好的停止点没有用了，你可以使用delete、clear、disable、enable这几个命令来进行维护。

```
clear
    清除所有的已定义的停止点。

clear <function>
clear <filename:function>
    清除所有设置在函数上的停止点。

clear <linenum>
clear <filename:linenum>
    清除所有设置在指定行上的停止点。

delete [breakpoints] [range...]
    删除指定的断点，breakpoints为断点号。如果不指定断点号，则表示删除所有的断点。range 表示断点号的范围（如：3-7）。其简写命令为d。
```

比删除更好的一种方法是disable停止点，disable了的停止点，GDB不会删除，当你还需要时，enable即可，就好像回收站一样。

```
disable [breakpoints] [range...]
    disable所指定的停止点，breakpoints为停止点号。如果什么都不指定，表示disable所有的停止点。简写命令是dis.

enable [breakpoints] [range...]
    enable所指定的停止点，breakpoints为停止点号。

enable [breakpoints] once range...
    enable所指定的停止点一次，当程序停止后，该停止点马上被GDB自动disable。

enable [breakpoints] delete range...
    enable所指定的停止点一次，当程序停止后，该停止点马上被GDB自动删除。
```

五、停止条件维护

前面在说到设置断点时，我们提到过可以设置一个条件，当条件成立时，程序自动停止，这是一个非常强大的功能，这里，我想专门说说这个条件的相关维护命令。一般来说，为断点设置一个条件，我们使用if关键词，后面

(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。。。。

跟其断点条件。并且，条件设置好后，我们可以用condition命令来修改断点的条件。（只有break和watch命令支持if，catch目前暂不支持if）

```
condition <bnum> <expression>
    修改断点号为bnum的停止条件为expression。

condition <bnum>
    清除断点号为bnum的停止条件。
```

还有一个比较特殊的维护命令ignore，你可以指定程序运行时，忽略停止条件几次。

```
ignore <bnum> <count>
    表示忽略断点号为bnum的停止条件count次。
```

六、为停止点设定运行命令

我们可以使用GDB提供的command命令来设置停止点的运行命令。也就是说，当运行的程序在被停止住时，我们可以让其自动运行一些别的命令，这很有利行自动化调试。对基于GDB的自动化调试是一个强大的支持。

```
commands [bnum]
... command-list ...
end
```

为断点号bnum指写一个命令列表。当程序被该断点停住时，gdb会依次运行命令列表中的命令。

例如：

```
    break foo if x>0
    commands
    printf "x is %d/n",x
    continue
    end
```

断点设置在函数foo中，断点条件是x>0，如果程序被断住后，也就是，一旦x的值在foo函数中大于0，GDB会自动打印出x的值，并继续运行程序。

如果你要清除断点上的命令序列，那么只要简单的执行一下commands命令，并直接在打个end就行了。

七、断点菜单

在C++中，可能会重复出现同一个名字的函数若干次（函数重载），在这种情况下，break <function>不能告诉GDB要停在哪个函数的入口。当然，你可以使用break <function(type)>也就是把函数的参数类型告诉GDB，以指定一个函数。否则的话，GDB会给你列出一个断点菜单供你选择你所需要的断点。你只要输入你菜单列表中的编号就可以了。如：

```
(gdb) b String::after
[0] cancel
[1] all
[2] file:String.cc; line number:867
[3] file:String.cc; line number:860
[4] file:String.cc; line number:875
[5] file:String.cc; line number:853
[6] file:String.cc; line number:846
[7] file:String.cc; line number:735
> 2 4 6
Breakpoint 1 at 0xb26c: file String.cc, line 867.
```

```
Breakpoint 2 at 0xb344: file String.cc, line 875.
Breakpoint 3 at 0xafcc: file String.cc, line 846.
Multiple breakpoints were set.
Use the "delete" command to delete unwanted
 breakpoints.
(gdb)
```

可见，GDB列出了所有after的重载函数，你可以选一下列表编号就行了。0表示放弃设置断点，1表示所有函数都设置断点。

八、恢复程序运行和单步调试

当程序被停住了，你可以用continue命令恢复程序的运行直到程序结束，或下一个断点到来。也可以使用step或next命令单步跟踪程序。

```
continue [ignore-count]
c [ignore-count]
fg [ignore-count]
```

恢复程序运行，直到程序结束，或是下一个断点到来。ignore-count表示忽略其后的断点次数。continue，c，fg三个命令都是一样的意思。

```
step <count>
```

单步跟踪，如果有函数调用，他会进入该函数。进入函数的前提是，此函数被编译有debug信息。很像VC等工具中的step in。后面可以加count也可以不加，不加表示一条条地执行，加表示执行后面的count条指令，然后再停住。

```
next <count>
```

同样单步跟踪，如果有函数调用，他不会进入该函数。很像VC等工具中的step over。后面可以加count也可以不加，不加表示一条条地执行，加表示执行后面的count条指令，然后再停住。

```
set step-mode
set step-mode on
```

打开step-mode模式，于是，在进行单步跟踪时，程序不会因为没有debug信息而不停住。这个参数有很利于查看机器码。

```
set step-mod off
```

关闭step-mode模式。

```
finish
```

运行程序，直到当前函数完成返回。并打印函数返回时的堆栈地址和返回值及参数值等信息。

```
until 或 u
```

当你厌倦了在一个循环体内单步跟踪时，这个命令可以运行程序直到退出循环体。

```
stepi 或 si
nexti 或 ni
```

单步跟踪一条机器指令！一条程序代码有可能由数条机器指令完成，stepi和nexti可以单步执行机器指令。与之一样有相同功能的命令是“display/i $pc”，当运行完这个命令后，单步跟踪会在打出程序代码的同时打出机器指令（也就是汇编代码）

九、信号（Signals）

信号是一种软中断，是一种处理异步事件的方法。一般来说，操作系统都支持许多信号。尤其是UNIX，比较重要应用程序一般都会处理信号。UNIX定义了许多信号，比如SIGINT表示中断字符信号，也就是Ctrl+C的信号，SIGBUS表示硬件故障的信号；SIGCHLD表示子进程状态改变信号；SIGKILL表示终止程序运行的信号，等等。信号量编程是UNIX下非常重要的一种技术。

GDB有能力在你调试程序的时候处理任何一种信号，你可以告诉GDB需要处理哪一种信号。你可以要求GDB收到你所指定的信号时，马上停住正在运行的程序，以供你进行调试。你可以用GDB的handle命令来完成这一功能。

handle <signal> <keywords...>

在GDB中定义一个信号处理。信号<signal>可以以SIG开头或不以SIG开头，可以用定义一个要处理信号的范围（如：SIGIO-SIGKILL，表示处理从SIGIO信号到SIGKILL的信号，其中包括SIGIO，SIGIOT，SIGKILL三个信号），也可以使用关键字all来标明要处理所有的信号。一旦被调试的程序接收到信号，运行程序马上会被GDB停住，以供调试。其<keywords>可以是以下几种关键字的一个或多个。

nostop

当被调试的程序收到信号时，GDB不会停住程序的运行，但会打出消息告诉你收到这种信号。

stop

当被调试的程序收到信号时，GDB会停住你的程序。

print

当被调试的程序收到信号时，GDB会显示出一条信息。

noprint

当被调试的程序收到信号时，GDB不会告诉你收到信号的信息。

pass

noignore

当被调试的程序收到信号时，GDB不处理信号。这表示，GDB会把这个信号交给被调试程序会处理。

nopass

ignore

当被调试的程序收到信号时，GDB不会让被调试程序来处理这个信号。

info signals

info handle

查看有哪些信号在被GDB检测中。

十、线程（Thread Stops）

如果你程序是多线程的话，你可以定义你的断点是否在所有的线程上，或是在某个特定的线程。GDB很容易帮你完成这一工作。

break <linespec> thread <threadno>

break <linespec> thread <threadno> if ...

linespec指定了断点设置在的源程序的行号。threadno指定了线程的ID，注意，这个ID是GDB分配的，你可以通过“info threads”命令来查看正在运行程序中的线程信息。如果你不指定thread <threadno>则表示你的断点设在所有线程上面。你还可以为某线程指定断点条件。如：

```
(gdb) break frik.c:13 thread 28 if bartab > lim
```

当你的程序被GDB停住时，所有的运行线程都会被停住。这方便你你查看运行程序的总体情况。而在你恢复程序运行时，所有的线程也会被恢复运行。那怕是主进程在被单步调试时。

<-上一页 下一页->

分享到：

上一篇：用GDB调试程序（六）

下一篇：C/C++内存问题检查利器—Purify（一）

查看评论

7楼 syzcch 2012-11-21 09:57发表

感谢 总结的很到位

6楼 syzcch 2012-11-20 10:42发表

经典好文 收藏了

5楼 yuyantai1234 2012-06-29 16:20发表

厉害，学习了！

4楼 nianhuaxpj 2008-10-24 20:06发表

谢谢百忙之中的回复，谢谢。

3楼 nianhuaxpj 2008-10-23 22:42发表

您好，看了您写的用gdb调试程序后学到了很多。学习的过程中发现，如果源代码错误很多，就要不厌其烦的编译调试，在编译在调试，所以要在gdb和gcc中不停的跳越，请问：能不能在gdb中调用gcc？或者有没有将两者结合起来的方法？谢谢。

Re: haoel 2008-10-24 09:10发表

在GDB中使用shell来运行外部命令，或者是使用Emacs来集成。

2楼 nianhuaxpj 2008-10-23 22:42发表

您好，看了您写的用gdb调试程序后学到了很多。学习的过程中发现，如果源代码错误很多，就要不厌其烦的编译调试，在编译在调试，所以要在gdb和gcc中不停的跳越，请问：能不能在gdb中调用gcc？或者有没有将两者结合起来的方法？谢谢。

1楼 manio 2006-10-18 14:47发表

感谢好文章