**用GDB调试程序**

**GDB概述  
————**

GDB是GNU开源组织发布的一个强大的UNIX下的程序调试工具。或许，各位比较喜欢那种图形界面方式的，像VC、BCB等IDE的调试，但如果你是在UNIX平台下做软件，你会发现GDB这个调试工具有比VC、BCB的图形化调试器更强大的功能。所谓“寸有所长，尺有所短”就是这个道理。

一般来说，GDB主要帮忙你完成下面四个方面的功能：

    1、启动你的程序，可以按照你的自定义的要求随心所欲的运行程序。  
    2、可让被调试的程序在你所指定的调置的断点处停住。（断点可以是条件表达式）  
    3、当程序被停住时，可以检查此时你的程序中所发生的事。  
    4、动态的改变你程序的执行环境。

从上面看来，GDB和一般的调试工具没有什么两样，基本上也是完成这些功能，不过在细节上，你会发现GDB这个调试工具的强大，大家可能比较习惯了图形化的调试工具，但有时候，命令行的调试工具却有着图形化工具所不能完成的功能。让我们一一看来。

**一个调试示例  
——————**

源程序：tst.c

     1 #include <stdio.h>  
     2  
     3 int func(int n)  
     4 {  
     5         int sum=0,i;  
     6         for(i=0; i<n; i++)  
     7         {  
     8                 sum+=i;  
     9         }  
    10         return sum;  
    11 }  
    12  
    13  
    14 main()  
    15 {  
    16         int i;  
    17         long result = 0;  
    18         for(i=1; i<=100; i++)  
    19         {  
    20                 result += i;  
    21         }  
    22  
    23        printf("result[1-100] = %d /n", result );  
    24        printf("result[1-250] = %d /n", func(250) );  
    25 }

编译生成执行文件：（[**Linux**](http://lib.csdn.net/base/linux)下）  
    hchen/test> cc -g tst.c -o tst

使用GDB调试：

hchen/test> gdb tst  <---------- 启动GDB  
GNU gdb 5.1.1  
Copyright 2002 Free Software Foundation, Inc.  
GDB is free software, covered by the GNU General Public License, and you are  
welcome to change it and/or distribute copies of it under certain conditions.  
Type "show copying" to see the conditions.  
There is absolutely no warranty for GDB.  Type "show warranty" for details.  
This GDB was configured as "i386-suse-linux"...  
(gdb) l     <-------------------- l命令相当于list，从第一行开始例出原码。  
1        #include <stdio.h>  
2  
3        int func(int n)  
4        {  
5                int sum=0,i;  
6                for(i=0; i<n; i++)  
7                {  
8                        sum+=i;  
9                }  
10               return sum;  
(gdb)       <-------------------- 直接回车表示，重复上一次命令  
11       }  
12  
13  
14       main()  
15       {  
16               int i;  
17               long result = 0;  
18               for(i=1; i<=100; i++)  
19               {  
20                       result += i;      
(gdb) break 16    <-------------------- 设置断点，在源程序第16行处。  
Breakpoint 1 at 0x8048496: file tst.c, line 16.  
(gdb) break func  <-------------------- 设置断点，在函数func()入口处。  
Breakpoint 2 at 0x8048456: file tst.c, line 5.  
(gdb) info break  <-------------------- 查看断点信息。  
Num Type           Disp Enb Address    What  
1   breakpoint     keep y   0x08048496 in main at tst.c:16  
2   breakpoint     keep y   0x08048456 in func at tst.c:5  
(gdb) r           <--------------------- 运行程序，run命令简写  
Starting program: /home/hchen/test/tst

Breakpoint 1, main () at tst.c:17    <---------- 在断点处停住。  
17               long result = 0;  
(gdb) n          <--------------------- 单条语句执行，next命令简写。  
18               for(i=1; i<=100; i++)  
(gdb) n  
20                       result += i;  
(gdb) n  
18               for(i=1; i<=100; i++)  
(gdb) n  
20                       result += i;  
(gdb) c          <--------------------- 继续运行程序，continue命令简写。  
Continuing.  
result[1-100] = 5050       <----------程序输出。

Breakpoint 2, func (n=250) at tst.c:5  
5                int sum=0,i;  
(gdb) n  
6                for(i=1; i<=n; i++)  
(gdb) p i        <--------------------- 打印变量i的值，print命令简写。  
$1 = 134513808  
(gdb) n  
8                        sum+=i;  
(gdb) n  
6                for(i=1; i<=n; i++)  
(gdb) p sum  
$2 = 1  
(gdb) n  
8                        sum+=i;  
(gdb) p i  
$3 = 2  
(gdb) n  
6                for(i=1; i<=n; i++)  
(gdb) p sum  
$4 = 3  
(gdb) bt        <--------------------- 查看函数堆栈。  
#0  func (n=250) at tst.c:5  
#1  0x080484e4 in main () at tst.c:24  
#2  0x400409ed in \_\_libc\_start\_main () from /lib/libc.so.6  
(gdb) finish    <--------------------- 退出函数。  
Run till exit from #0  func (n=250) at tst.c:5  
0x080484e4 in main () at tst.c:24  
24              printf("result[1-250] = %d /n", func(250) );  
Value returned is $6 = 31375  
(gdb) c     <--------------------- 继续运行。  
Continuing.  
result[1-250] = 31375    <----------程序输出。

Program exited with code 027. <--------程序退出，调试结束。  
(gdb) q     <--------------------- 退出gdb。  
hchen/test>

好了，有了以上的感性认识，还是让我们来系统地认识一下gdb吧。

**使用GDB  
————**

一般来说GDB主要调试的是C/C++的程序。要调试C/C++的程序，首先在编译时，我们必须要把调试信息加到可执行文件中。使用编译器（cc/gcc/g++）的 -g 参数可以做到这一点。如：

    > cc -g hello.c -o hello  
    > g++ -g hello.cpp -o hello

如果没有-g，你将看不见程序的函数名、变量名，所代替的全是运行时的内存地址。当你用-g把调试信息加入之后，并成功编译目标代码以后，让我们来看看如何用gdb来调试他。

启动GDB的方法有以下几种：

    1、gdb <program>   
       program也就是你的执行文件，一般在当然目录下。

    2、gdb <program> core  
       用gdb同时调试一个运行程序和core文件，core是程序非法执行后core dump后产生的文件。

    3、gdb <program> <PID>  
       如果你的程序是一个服务程序，那么你可以指定这个服务程序运行时的进程ID。gdb会自动attach上去，并调试他。program应该在PATH环境变量中搜索得到。

GDB启动时，可以加上一些GDB的启动开关，详细的开关可以用gdb -help查看。我在下面只例举一些比较常用的参数：

    -symbols <file>   
    -s <file>   
    从指定文件中读取符号表。

    -se file   
    从指定文件中读取符号表信息，并把他用在可执行文件中。

    -core <file>  
    -c <file>   
    调试时core dump的core文件。

    -directory <directory>  
    -d <directory>  
    加入一个源文件的搜索路径。默认搜索路径是环境变量中PATH所定义的路径。

**GDB的命令概貌  
———————**

启动gdb后，就你被带入gdb的调试环境中，就可以使用gdb的命令开始调试程序了，gdb的命令可以使用help命令来查看，如下所示：

    /home/hchen> gdb  
    GNU gdb 5.1.1  
    Copyright 2002 Free Software Foundation, Inc.  
    GDB is free software, covered by the GNU General Public License, and you are  
    welcome to change it and/or distribute copies of it under certain conditions.  
    Type "show copying" to see the conditions.  
    There is absolutely no warranty for GDB.  Type "show warranty" for details.  
    This GDB was configured as "i386-suse-[**Linux**](http://lib.csdn.net/base/linux)".  
    (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" followed by command name for full documentation.  
    Command name abbreviations are allowed if unambiguous.  
    (gdb)

gdb的命令很多，gdb把之分成许多个种类。help命令只是例出gdb的命令种类，如果要看种类中的命令，可以使用help <class> 命令，如：help breakpoints，查看设置断点的所有命令。也可以直接help <command>来查看命令的帮助。

gdb中，输入命令时，可以不用打全命令，只用打命令的前几个字符就可以了，当然，命令的前几个字符应该要标志着一个唯一的命令，在Linux下，你可以敲击两次TAB键来补齐命令的全称，如果有重复的，那么gdb会把其例出来。  
      
    示例一：在进入函数func时，设置一个断点。可以敲入break func，或是直接就是b func  
    (gdb) b func  
    Breakpoint 1 at 0x8048458: file hello.c, line 10.  
   
    示例二：敲入b按两次TAB键，你会看到所有b打头的命令：  
    (gdb) b  
    backtrace  break      bt  
    (gdb)

    示例三：只记得函数的前缀，可以这样：  
    (gdb) b make\_ <按TAB键>  
    （再按下一次TAB键，你会看到:）  
    make\_a\_section\_from\_file     make\_environ  
    make\_abs\_section             make\_function\_type  
    make\_blockvector             make\_pointer\_type  
    make\_cleanup                 make\_reference\_type  
    make\_command                 make\_symbol\_completion\_list  
    (gdb) b make\_  
    GDB把所有make开头的函数全部例出来给你查看。

    示例四：调试C++的程序时，有可以函数名一样。如：  
    (gdb) b 'bubble( M-?   
    bubble(double,double)    bubble(int,int)  
    (gdb) b 'bubble(  
    你可以查看到C++中的所有的重载函数及参数。（注：M-?和“按两次TAB键”是一个意思）

要退出gdb时，只用发quit或命令简称q就行了。

**GDB中运行UNIX的shell程序  
————————————**

在gdb环境中，你可以执行UNIX的shell的命令，使用gdb的shell命令来完成：

    shell <command string>  
    调用UNIX的shell来执行<command string>，环境变量SHELL中定义的UNIX的shell将会被用来执行<command string>，如果SHELL没有定义，那就使用UNIX的标准shell：/bin/sh。（在Windows中使用Command.com或cmd.exe）

还有一个gdb命令是make：  
    make <make-args>   
    可以在gdb中执行make命令来重新build自己的程序。这个命令等价于“shell make <make-args>”。

**在GDB中运行程序  
————————**

当以gdb <program>方式启动gdb后，gdb会在PATH路径和当前目录中搜索<program>的源文件。如要确认gdb是否读到源文件，可使用l或list命令，看看gdb是否能列出源代码。

在gdb中，运行程序使用r或是run命令。程序的运行，你有可能需要设置下面四方面的事。

1、程序运行参数。  
    set args 可指定运行时参数。（如：set args 10 20 30 40 50）  
    show args 命令可以查看设置好的运行参数。

2、运行环境。  
    path <dir> 可设定程序的运行路径。  
    show paths 查看程序的运行路径。  
    set environment varname [=value] 设置环境变量。如：set env USER=hchen  
    show environment [varname] 查看环境变量。

3、工作目录。  
    cd <dir> 相当于shell的cd命令。  
    pwd 显示当前的所在目录。

4、程序的输入输出。  
    info terminal 显示你程序用到的终端的模式。  
    使用重定向控制程序输出。如：run > outfile  
    tty命令可以指写输入输出的终端设备。如：tty /dev/ttyb

**调试已运行的程序  
————————**

两种方法：  
1、在UNIX下用ps查看正在运行的程序的PID（进程ID），然后用gdb <program> PID格式挂接正在运行的程序。  
2、先用gdb <program>关联上源代码，并进行gdb，在gdb中用attach命令来挂接进程的PID。并用detach来取消挂接的进程。

**暂停 / 恢复程序运行  
—————————**

调试程序中，暂停程序运行是必须的，GDB可以方便地暂停程序的运行。你可以设置程序的在哪行停住，在什么条件下停住，在收到什么信号时停往等等。以便于你查看运行时的变量，以及运行时的流程。

当进程被gdb停住时，你可以使用info program 来查看程序的是否在运行，进程号，被暂停的原因。

在gdb中，我们可以有以下几种暂停方式：断点（BreakPoint）、观察点（WatchPoint）、捕捉点（CatchPoint）、信号（Signals）、线程停止（Thread Stops）。如果要恢复程序运行，可以使用c或是continue命令。

**一、设置断点（BreakPoint）**  
      
    我们用break命令来设置断点。正面有几点设置断点的方法：  
      
    break <function>   
        在进入指定函数时停住。C++中可以使用class::function或function(type,type)格式来指定函数名。

    break <linenum>  
        在指定行号停住。

    break +offset   
    break -offset   
        在当前行号的前面或后面的offset行停住。offiset为自然数。

    break filename:linenum   
        在源文件filename的linenum行处停住。

    break filename:function   
        在源文件filename的function函数的入口处停住。

    break \*address  
        在程序运行的内存地址处停住。

    break   
        break命令没有参数时，表示在下一条指令处停住。

    break ... if <condition>  
        ...可以是上述的参数，condition表示条件，在条件成立时停住。比如在循环境体中，可以设置break if i=100，表示当i为100时停住程序。

    查看断点时，可使用info命令，如下所示：（注：n表示断点号）  
    info breakpoints [n]   
    info break [n] 

**二、设置观察点（WatchPoint）**  
      
    观察点一般来观察某个表达式（变量也是一种表达式）的值是否有变化了，如果有变化，马上停住程序。我们有下面的几种方法来设置观察点：  
      
    watch <expr>  
        为表达式（变量）expr设置一个观察点。一量表达式值有变化时，马上停住程序。  
          
    rwatch <expr>  
        当表达式（变量）expr被读时，停住程序。  
          
    awatch <expr>  
        当表达式（变量）的值被读或被写时，停住程序。  
      
    info watchpoints  
        列出当前所设置了的所有观察点。

**三、设置捕捉点（CatchPoint）**

    你可设置捕捉点来补捉程序运行时的一些事件。如：载入共享库（动态链接库）或是C++的异常。设置捕捉点的格式为：  
      
    catch <event>  
        当event发生时，停住程序。event可以是下面的内容：  
        1、throw 一个C++抛出的异常。（throw为关键字）  
        2、catch 一个C++捕捉到的异常。（catch为关键字）  
        3、exec 调用系统调用exec时。（exec为关键字，目前此功能只在HP-UX下有用）  
        4、fork 调用系统调用fork时。（fork为关键字，目前此功能只在HP-UX下有用）  
        5、vfork 调用系统调用vfork时。（vfork为关键字，目前此功能只在HP-UX下有用）  
        6、load 或 load <libname> 载入共享库（动态链接库）时。（load为关键字，目前此功能只在HP-UX下有用）  
        7、unload 或 unload <libname> 卸载共享库（动态链接库）时。（unload为关键字，目前此功能只在HP-UX下有用）

    tcatch <event>   
        只设置一次捕捉点，当程序停住以后，应点被自动删除。