大家一起用 gtk 编程 4(使用 GDB 调试程序)

作者：lvjinhua at gmail dot com
(本文档最后由杨小邪编辑整理)
2009.1.7

·6、使用 GDB 调试程序

上回话说使用 Makefile 来组织源代码，这回简单地介绍下如何使用 GDB 来调试我们的程序；关于 GDB 的其它应用将在后续的章节中逐渐深入。

首 先需要肯定一点，GDB 是个功能异常强大的调试工具，其本身只能运行于字符模式，但是当前众多基于 GUI 的调试器/IDE，无论是自由软件还是商业软件， 绝大多数都使用 GDB 作为其后端（但这些基于 GUI 的调试器都不太稳定），因此 GDB 是个不二的选择（笔者推荐的 GUI 调试器：insight 和 ddd）。

这里使用 hello_gdb.c 作为例子，如果你从前面一直看过来，对这个程序一定不会陌生，hello_gdb.c 主要在 hello_dubuntu2.c 的基础上，添加了几个整型和字符串型变量，来演示 gdb 的一些基本功能：
运行效果图：

hello_gdb.c

```
/* 本例的主要目的是在窗口中显示一个按钮，
 * 单击按钮退出程序，并构建几个变量来演示 GDB 功能。
 */
#include<gtk/gtk.h>

void
cb_button(GtkWidget *widget, gpointer data)
{// 按钮"button"的回调函数
        gint i=5;
        gint j=++i;
        gtk_main_quit();
}

//此函数用于演示 GDB 直接调用被调试程序的函数
```

```
gint
gdb_test(gint arg)
{
        g_print("arg=%d\n",arg);
        arg++;
        return arg;
}

int
main(int argc, char *argv[])
{
        GtkWidget *main_window; //主窗口对象
        GtkWidget *button;              //将要放置到主窗口中的按钮对
象

        //构造两个变量用于演示 GDB 功能
        gint a=5;
        gchar *name="Dubuntu-6.06";
        //
        gtk_init(&argc, &argv);

        main_window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_window_set_title(GTK_WINDOW(main_window),
"Hello,Dubuntu2!");
        //设置窗口的默认大小（宽 200，高度 50）
        gtk_window_set_default_size(GTK_WINDOW(main_window),
200,50);

        button = gtk_button_new_with_label("退出程序");
        gtk_container_add(GTK_CONTAINER(main_window), button);
        //为"button"连接“单击事件”要调用的回调函数
        g_signal_connect(G_OBJECT(button), "clicked",
G_CALLBACK(cb_button),NULL);

        gtk_widget_show(button);
        gtk_widget_show(main_window);
        //上边的两句可以合为 gtk_widget_show_all(window)

        g_signal_connect(G_OBJECT(main_window), "destroy",
G_CALLBACK(cb_button),NULL);

        gtk_main();
        return 0;
}
```

编译： gcc -g -Wall -o hello_gdb hello_gdb.c `pkg-config --cflags --libs gtk+-2.0`
注意：-g 参数用于为可执行文件生成调试信息；
-Wall 用于在编译程序时打印所有的警告信息

好了，程序就是上边这个(hello_gdb.c)，我们使用如下的命令对它进行编译：

```
gcc -g -Wall -o hello_gdb hello_gdb.c `pkg-config --cflags --libs gtk+-2.0`
```

编译完后，如果没有错误，将生成 hello_gdb 可执行文件，此可执行文件将携带 gdb 调试时所需要的调试信息，有了这些调试消息，我们就可以在调试程序的时候查看函数名，变量名，源代码。

好了，开始调试吧：
1）在命令行下，通过如下命令加载刚编译生成的 hello_gdb 程序：

```
gdb  ./hello_gdb
```

加载成功后，将得到如下提示信息，并进入 gdb 的命令行模式：

```
dubuntu@dubuntu:~/Desktop/gnome-gtk-prog/hello_gtk/src$ gdb ./hello_gdb
GNU gdb 6.5.50.20060605-cvs
Copyright (C) 2006 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 "i686-pc-linux-gnu"...Using host libthread_db
library "/lib/tls/i686/cmov/libthread_db.so.1".

(gdb)
```

2）好，现在已经成功启动 gdb 并且加载了可执行程序 hello_gdb，接下来的命令绝大部分都是争对 hello_gdb 程序，下面将列举几个最常用的命令：

小技巧：在 gdb 命令中，只需要输入命令或参数的前几个字符，再按键盘上的“TAB”键，那 gdb 将自动补齐命令或参数，如果有多个候选者，那么 gdb 将把它们都列举出来。

- list ：简记为 l ，其作用就是列出程序的源代码，默认每次显示 10 行。
  - list 行号：将显示当前文件以“行号”为中心的前后 10 行代码，如：list 12
  - list 函数名：将显示“函数名”所在函数的源代码，如：list main

- list ：不带参数，将接着上一次 list 命令的，输出下边的内容。

> 注意:如果运行 list 命令得到类似如下的打印,那是因为在编译程序时没有加入 -g 选项:
>
> ```
> (gdb) list
> 1       ../sysdeps/i386/elf/start.S: No such file or directory.
>         in ../sysdeps/i386/elf/start.S
> ```

- run：简记为 r ，其作用是运行程序，当遇到断点后，程序会在断点处停止运行，等待用户输入下一步的命令。
- set args：设置运行程序时的命令行参数，如：set args 33 55
- show args：显示命令行参数
- continue：简讯为 c ，其作用是继续运行被断点中断的程序。
- break：为程序设置断点。
    - break 行号：在当前文件的“行号”处设置断点，如：break 33
    - break 函数名：在用户定义的函数“函数名”处设置断点，如：break cb_button
    - info breakpoints：显示当前程序的断点设置情况
    - disable breakpoints Num：关闭断点“Num”，使其无效，其中“Num”为 info breakpoints 中显示的对应值
    - enable breakpoints Num：打开断点“Num”，使其重新生效
- step：简记为 s ，单步跟踪程序，当遇到函数调用时，则进入此函数体（一般只进入用户自定义函数）。
- next：简记为 n，单步跟踪程序，当遇到函数调用时，也不进入此函数体；此命令同 step 的主要区别是，step 遇到用户自定义的函数，将步进到函数中去运行，而 next 则直接调用函数，不会进入到函数体内。
- until：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。
- 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。
- stepi 或 nexti：单步跟踪一些机器指令。
- print 表达式：简记为 p ，其中“表达式”可以是任何当前正在被测试程序的有效表达式，比如当前正在调试 C 语言的程序，那么“表达式”可以是任何 C 语言的有效表达式，包括数字，变量甚至是函数调用。
    - print a：将显示整数 a 的值
    - print ++a：将把 a 中的值加 1,并显示出来
    - print name：将显示字符串 name 的值
    - print gdb_test(22)：将以整数 22 作为参数调用 gdb_test() 函数
    - print gdb_test(a)：将以变量 a 作为参数调用 gdb_test() 函

数

- bt：显示当前程序的函数调用堆栈。
- display 表达式：在单步运行时将非常有用，使用 display 命令设置一个表达式后，它将在每次单步进行指令后，紧接着输出被设置的表达式及值。如： display a
- watch 表达式：设置一个监视点，一旦被监视的“表达式”的值改变，gdb 将强行终止正在被调试的程序。如： watch a
- kill：将强行终止当前正在调试的程序
- help 命令：help 命令将显示“命令”的常用帮助信息
- call 函数(参数)：调用“函数”，并传递“参数”，如：
  call gdb_test(55)
- layout：用于分割窗口，可以一边查看代码，一边测试：

    - layout src：显示源代码窗口
    - layout asm：显示反汇编窗口
    - layout regs：显示源代码/反汇编和 CPU 寄存器窗口
    - layout split：显示源代码和反汇编窗口
    - Ctrl + L：刷新窗口

- quit：简记为 q ，退出 gdb

当然，gdb的功能远不止这些，包括多进程/多线程/信号/远程调试等功能在这里均没有提及，有需要的读者可以参考其它信息，本文的后续章节也将涉及一些内容。

推荐阅读：用gdb调试程序

3）调试实例：

```
gdb ./hello_gdb             #启动 gdb 并载入被调试程序
list gdb_test                  #显示函数 gdb_test 的代码
break cb_button           #在函数 cb_button 处设置断点
info breakpoints          #显示所有断点信息
enable breakpoints      #启动所有断点
run                                  # 开始运行程序
#注：现在程序正常运行，当单击按钮“退出程序”后，将在函数“cb_button”处停止，因为这里被设置了一个断点
bt                                     #显示当前的函数调用堆栈
display j                          #对变量 j 进行监视
next                                #运行下一条指令
print j                             #显示 j 的值
print gdb_test(j)           #调用函数 gdb_test 并使用 j 作为参数
call  gdb_test(++j)      #同上
next
```

```
finish                          #退出函数 cb_button
continue                        #继续运行完程序
quit                            # 退出 gdb
```

下集预告： 下节将简单地介绍如何使用 vbox/hbox 进行窗口布局，这样就能解决当前一个按钮占满整个窗体的问题了。