# gdb & gdbserver

开发板：

gdbserver 192.168.183.127:2345 ./test\_debug

其中192.168.183.127：本开发板的ip

123：端口号，自己随便写的

./test\_debug：要调试的程序

PC机：

gdb ./test-debug target remote 192.168.183.127:2345

gdbserver 192.168.90.180:2345 ./onvifsvr &

gdb ./onvifsvr target remote 192.168.91.106:2345

# gdb sample

arm-linux-androideabi-gdb ./onvifsvr core

源程序：tst.c

1 #include

2

3 int func(int n)

4 {

5 int sum=0,i;

6 for(i=0; i<7; i++) {

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

2

3 int func(int n)

4 {

5 int sum=0,i;

6 for(i=0; 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。

(gdb) s <--------------------- 单条语句执行，step into。

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>

# Gdb cmd

===============================================================================

run（r） //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。

continue（c） //继续执行，到下一个断点处（或运行结束）

next（n） //单步跟踪程序，当遇到函数调用时，直接调用，不进入此函数体；

step（s） //单步调试如果有函数调用，则进入函数；与命令n不同，n是不进入调用的函数的

until //运行程序直到退出循环体; / until+行号： 运行至某行

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。

call 函数(参数) //调用“函数”，并传递“参数”，如：call gdb\_test(55)

quit( q ) //退出gdb

break n（b n） //在第n行处设置断点 ;可以带上代码路径和代码名称： b OAGUPDATE.cpp:578）

break func //在函数func()的入口处设置断点，如：break cb\_button

delete 断点号n //删除第n个断点

disable 断点号n //暂停第n个断点

enable 断点号n //开启第n个断点

clear 行号n //清除第n行的断点

info breakpoints（info b） //显示当前程序的断点设置情况

list（l） //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。

print 表达式 (p ) //表达式”可以是任何当前正在被测试程序的有效表达式，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

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

where/bt //当前运行的堆栈列表

set args 参数 //指定运行时的参数

show args //查看设置好的参数

info program //来查看程序的是否在运行，进程号，被暂停的原因

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

===============================================================================

break：断点

break 10 //设置本地行号，即list命令可见的行号

break test.c:10 //设置指定文件的行号

break main //函数，可以是自定义函数也可是库函数，如open

break test.c:main //指定文件中的函数

break system

break open

break \*0x40fe00 //设置地址为0x40fe00位置的断点：可通过info add function查看函数的地址

break 10 if index == 3 //当变量index为3时，设置断点为第10行

info break //查看断点信息

tbreak //跟break类似，只在断点停一次，过后会自动将断点删除

rbreak list\_\* //即在所有以 list\_ 为开头字符的函数地方都设置断点

delete 5 //删除断点5

===============================================================================

info

info add RTSPServer::RTSPClientSession::handleCmd\_SETUP //查看指定函数的地址： 0x40fe00

break \*0x40fe00 //设置地址为0x40fe00处为断点

list \*0x40fe00 //查看0x40fe00处的源码

directory 源代码路径 //加载源代码

===============================================================================

directory: 增加源文件路径

directory /home/yhong/src/live/live\_20161101/liveMedia

# Gdb cmd

|  |  |  |
| --- | --- | --- |
| **命令** | **解释** | **示例** |
| file <文件名> | 加载被调试的可执行程序文件。 因为一般都在被调试程序所在目录下执行GDB，因而文本名不需要带路径。 | (gdb) file gdb-sample |
| r | Run的简写，运行被调试的程序。 如果此前没有下过断点，则执行完整个程序；如果有断点，则程序暂停在第一个可用断点处。 | (gdb) r |
| c | Continue的简写，继续执行被调试程序，直至下一个断点或程序结束。 | (gdb) c |
| b <行号> b <函数名称> b \*<函数名称> b \*<代码地址>  d [编号] | b: Breakpoint的简写，设置断点。两可以使用“行号”“函数名称”“执行地址”等方式指定断点位置。 其中在函数名称前面加“\*”符号表示将断点设置在“由编译器生成的prolog代码处”。如果不了解汇编，可以不予理会此用法。  d: Delete breakpoint的简写，删除指定编号的某个断点，或删除所有断点。断点编号从1开始递增。 | (gdb) b 8 (gdb) b main (gdb) b \*main (gdb) b \*0x804835c  (gdb) d |
| s, n | s: 执行一行源程序代码，如果此行代码中有函数调用，则进入该函数； n: 执行一行源程序代码，此行代码中的函数调用也一并执行。  s 相当于其它调试器中的“Step Into (单步跟踪进入)”； n 相当于其它调试器中的“Step Over (单步跟踪)”。  这两个命令必须在有源代码调试信息的情况下才可以使用（GCC编译时使用“-g”参数）。 | (gdb) s (gdb) n |
| si, ni | si命令类似于s命令，ni命令类似于n命令。所不同的是，这两个命令（si/ni）所针对的是汇编指令，而s/n针对的是源代码。 | (gdb) si (gdb) ni |
| p <变量名称> | Print的简写，显示指定变量（临时变量或全局变量）的值。 | (gdb) p i (gdb) p nGlobalVar |
| display ...  undisplay <编号> | display，设置程序中断后欲显示的数据及其格式。 例如，如果希望每次程序中断后可以看到即将被执行的下一条汇编指令，可以使用命令 “display /i $pc” 其中 $pc 代表当前汇编指令，/i 表示以十六进行显示。当需要关心汇编代码时，此命令相当有用。  undispaly，取消先前的display设置，编号从1开始递增。 | (gdb) display /i $pc  (gdb) undisplay 1 |
| i | Info的简写，用于显示各类信息，详情请查阅“help i”。 | (gdb) i r |
| q | Quit的简写，退出GDB调试环境。 | (gdb) q |
| help [命令名称] | GDB帮助命令，提供对GDB名种命令的解释说明。 如果指定了“命令名称”参数，则显示该命令的详细说明；如果没有指定参数，则分类显示所有GDB命令，供用户进一步浏览和查询。 | (gdb) help display |