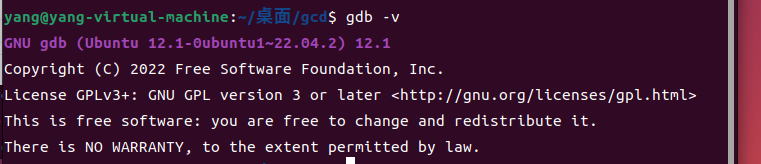
# Gdb试用

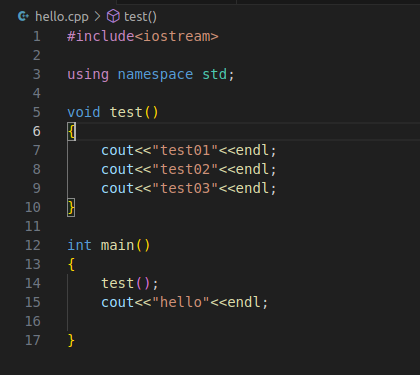
GDB 全称“GNU symbolic debugger”，是 Linux 下常用的程序调试器。发展至今，GDB 已经迭代了诸多个版本，当下的 GDB 支持调试多种编程语言编写的程序，包括 C、C++、Go、Objective-C、OpenCL、Ada 等。实际场景中，GDB 更常用来调试 C 和 C++ 程序。所以对于一名Linux下工作的c/c++程序员，对gdb的了解是必不可少的。

## 安装好gdb

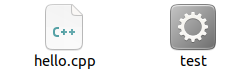


使用流程

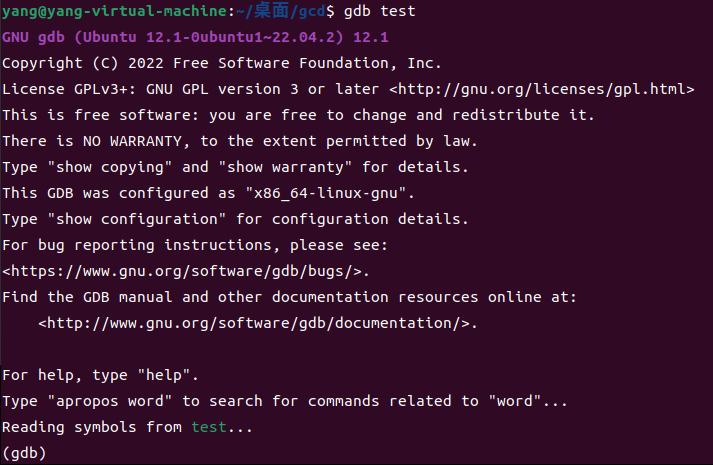
首先，写一个简单的c或c++程序



通过g++ hello.cpp -o test -g编译该文件（如果是c文件则用gcc开头）



## gdb的启动

输入gdb test 就可以直接对我们之前编译后的test文件进行调试，如果想退出调试，输入quit。

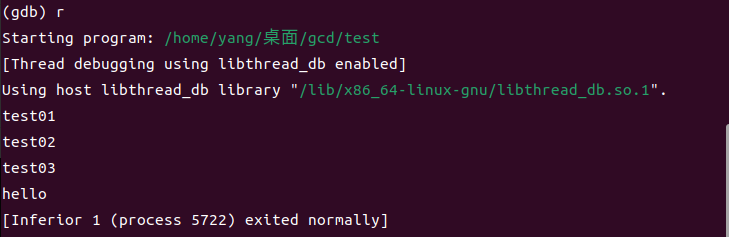
调试中查看源代码



用list或者l 0来显示（注意：源代码一次只显示十行，如果显示不全，按enter显示剩下的）

## 开始调试

使用指令r



如果想和平常的可视化调试器一样打断点用

b+行号      #对代码中的某行打断点

info + b    #查看我们所打的断点位置





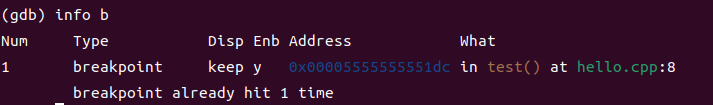
断点位置

带上断点后，再次运行



上面依次显示了调试到断点位置，断点运行前的结果，断点位置打印结果

查看断点

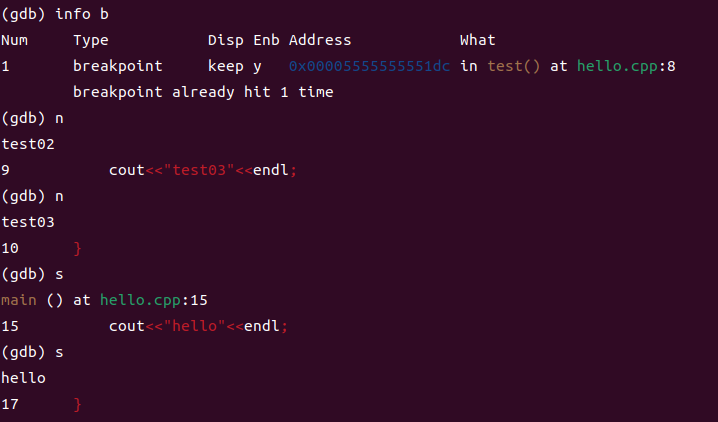


指令：

n           #逐过程调试，相当于F10

s           #逐语句调试，相当于F11

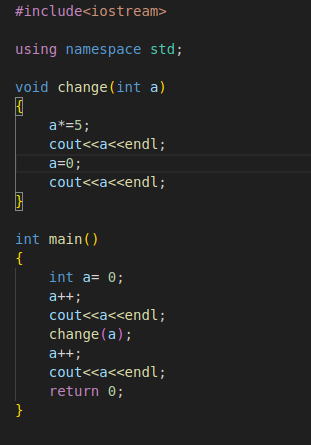
让我们继续上面的调试



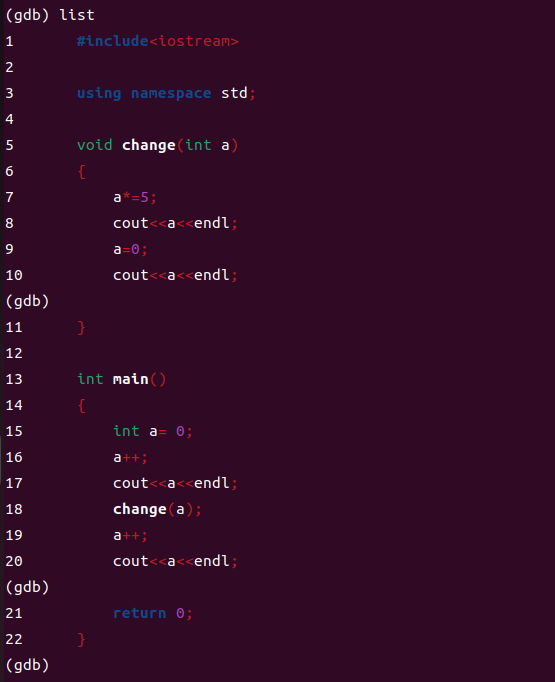
由于后面没有函数了，所以n和s都是一句句在调试，但实际他们是不同的

Gdb方便的点可以实时跟踪变量值的变化

新写一段代码



下面是进入gdb之后并打上断点





P+变量是显示变量当前的值，但每次打印都要输入指令



指令：

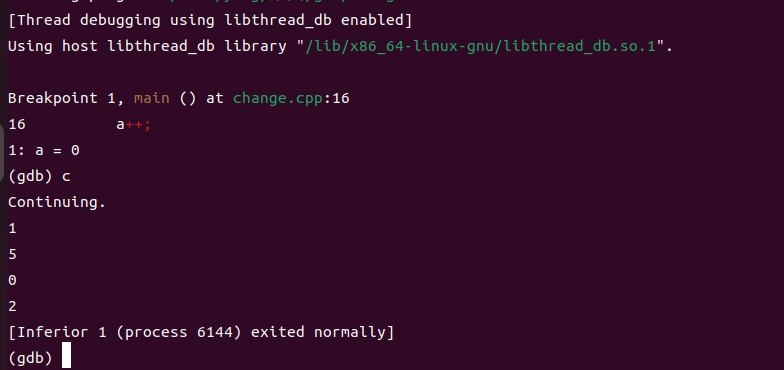
display + 变量名      #随着调试的进行常显示该变量。

undisplay + 编号      #取消该编号对应的变量常显示。



他追踪局部变量a的大小变化，虽然函数里有相同名称的变量，但是不被干扰

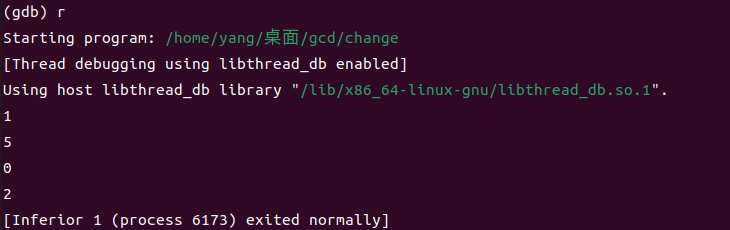
c       #运行到下一断点处



没有就全运行完了

d  + 编号      #编号是我们在info指令下查看的断点编号。

删除断点并再次运行



## 线程调试

$  info threads                 // 显示当前程序处于哪一个线程中

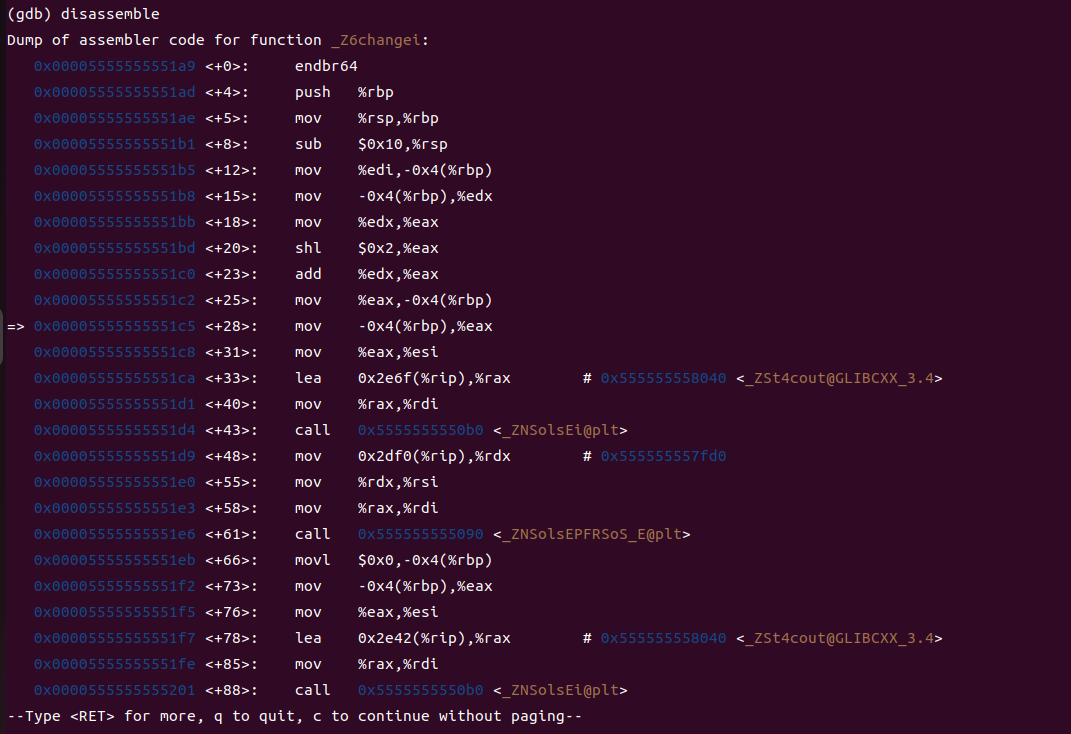
这里使用打上断点，让程序运行，查看断点

+

$ t 线程id   // 切换到指定线程执行

$ set scheduler-locking on   // 锁定， 在当前线程下执行，不会被其他线程抢占CPU，直到该线程运行完成。

$ disassemble          // 解析汇编指令， 显示当前接下来要执行的汇编指令



$ ni                // 执行汇编下一条汇编指令



main函数调用 call add函数时，将下一条指令的地址…5551e6压入栈中，然后跳转到函数add的地址…555090处去执行。



call 指令等价于：

push 

jmp 

ret 指令等效于：

pop

jmp

弹出并跳到，上面指令不太直观

## gdb汇编指令

下面是关于gdb汇编指令的参考资料

2.1 step/stepi

（1）step：用于单步执行程序并停在下一个源代码行，会进入函数。

使用 step 命令时，GDB 会执行当前行的代码，如果当前行包含函数调用，GDB 会跳进函数并停在函数内的第一行代码。如果当前行不包含函数调用，GDB 会停在下一行代码处。如果程序在当前行内包含多条语句，GDB 会执行完当前行内的所有语句并停在下一行代码处。

（2）stepi：以单步执行程序的汇编指令，汇编指令call 函数时，会进入函数。

用于单步执行一条汇编指令并停止。与 step 命令不同的是，step 命令会将当前源代码行（比如C源代码行）作为一个单独的步骤来执行，而 stepi 命令会将汇编指令作为执行的最小单位。

使用 stepi 命令时，GDB 会执行当前指令，并将 PC 寄存器指向下一条指令，然后停止程序的执行。如果当前指令是跳转指令，那么 stepi 命令会跳转到跳转指令的目标地址继续执行。如果当前指令是函数调用指令，那么 stepi 命令会跳转到被调用函数的入口地址，等待下一个命令。

stepi 命令是按照汇编指令执行的，而不是按照源代码行执行的。

2.2 next/nexti

next：用于单步执行程序并停在下一行源代码，不会进入函数。

使用 next 命令时，GDB 会执行当前行的代码，如果当前行包含函数调用，GDB 会跳过函数并停在函数调用的下一行代码。如果当前行不包含函数调用，GDB 会停在下一行代码处。如果程序在当前行内包含多条语句，GDB 只会执行当前行的第一条语句并停在下一行代码处。

nexti：用于执行下一条汇编指令并停止。

使用 nexti 命令时，GDB 会执行当前指令，并将 PC 寄存器指向下一条指令，然后停止程序的执行。如果当前指令是跳转指令，那么 nexti 命令会跳转到跳转指令的目标地址继续执行。如果当前指令是函数调用指令，那么 nexti 命令会将函数作为一个整体执行，并在函数返回时停止。

nexti 命令是按照汇编指令执行的，而不是按照源代码行执行的。

2.3 info registers

info registers 用于查看当前程序的寄存器状态，包括通用寄存器、浮点寄存器、协处理器寄存器等。

使用 info registers 命令时，GDB 会列出当前程序的寄存器状态，每个寄存器的名称、值和格式（十六进制或十进制）都会被显示出来。通常情况下，一般只需要查看一部分寄存器的值，可以使用 info registers 命令后面跟上寄存器名称的方式来查看指定寄存器的值。比如：

(gdb) info registers rsp

1

可以使用 set $register=value 命令来设置寄存器的值，以便进行调试和测试。

2.4 set

用于设置变量、内存地址、寄存器或者其他 GDB 配置选项的值。使用 set 命令可以在调试过程中修改变量的值或者更改 GDB 的配置选项，以便更好地进行调试和分析。

2.5 x

显示内存中的内容，格式如下：

x/[n][f][size] address：其中，n表示要显示的数据的数量，f表示要显示的数据的格式，size表示每个数据的大小，address表示要显示的内存地址。

f 选项可以用于指定要显示的数据的格式。下面是一些常见的数据格式：

x: 以十六进制格式显示数据。

d: 以十进制格式显示数据。

u: 以十进制无符号整数格式显示数据。

o: 以八进制格式显示数据。

t: 以二进制格式显示数据。

a: 以地址格式显示数据。

c: 以字符格式显示数据。

f: 以浮点数格式显示数据。

s: 以字符串格式显示数据。

i: 以指令格式显示数据。

size 选项可以用于指定要显示的每个数据的大小。下面是一些常见的数据大小选项：

b: 每个数据的大小为1字节（即8位）。

h: 每个数据的大小为2字节（即16位）。

w: 每个数据的大小为4字节（即32位）。

g: 每个数据的大小为8字节（即64位）。

2.6 rsp寄存器

rsp 寄存器是 x86\_64 架构中的一个 64 位寄存器，用于存储栈指针（Stack Pointer，SP）。栈指针是一个指向当前栈顶的内存地址的指针，用于指示下一个入栈或出栈操作的位置。

在函数调用时，rsp 寄存器会被用于存储当前函数的栈帧信息，包括函数的返回地址、参数、局部变量等。在函数返回时，rsp 寄存器会被重新设置为之前保存的值，以便恢复上一个函数的栈帧信息。

在程序执行过程中，rsp 寄存器的值会随着栈的变化而不断改变。当程序执行入栈操作时，rsp 寄存器的值会减小，指向新的栈顶位置。当程序执行出栈操作时，rsp 寄存器的值会增加，指向上一个栈顶位置。

在 GDB 中，可以使用 info registers 命令或 print $rsp 命令来查看 rsp 寄存器的值。

同时，可以使用 x 命令来查看栈内存的内容，例如：

(gdb) x/8xg $rsp

1

这样就可以以十六进制格式查看栈顶的 8 个 64 位数据。

2.7 rip 寄存器

rip 寄存器是 x86\_64 架构中的一个 64 位寄存器，用于存储指令指针（Instruction Pointer，IP）。指令指针是一个指向下一条将要执行的指令的地址的指针，用于指示程序的执行位置。

在程序执行过程中，rip 寄存器的值会随着指令的执行而不断改变。当程序执行一条指令时，rip 寄存器的值会自动增加，指向下一条指令的地址。如果程序遇到跳转指令，rip 寄存器的值会被修改为跳转目标的地址，以便程序跳转到指定位置继续执行。

在 GDB 中，可以使用 info registers 命令或 print $rip 命令来查看 rip 寄存器的值。

(gdb) print $rip

$1 = (void (\*)()) 0x400537 <add+10>

(gdb) info registers rip

rip 0x400537 0x400537 <add+10>

————————————————

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

原文链接：https://blog.csdn.net/weixin\_45030965/article/details/131795004