Linux gdb调试

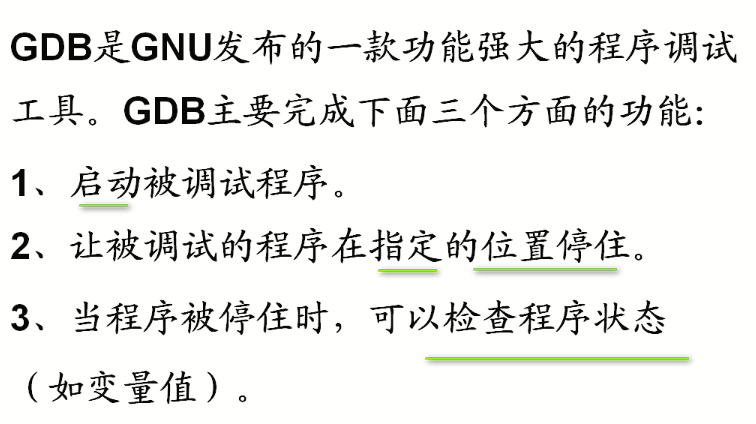


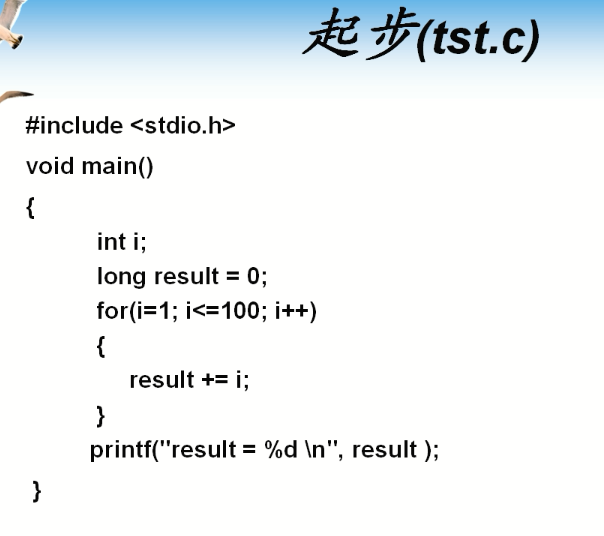
不一定每次运行 运行效果和期望的一模一样 编译错误可以编译解决 但是 逻辑错误没法避免 这个时候 要去调试问题所在

Linux中最常用的调试是GDB ---- 还有其他图形用户的调试器

实际上 所有的调试器 核心都是GDB 很多图形调试器 都是在GDB外面封装了图形的外壳

GDB有几个功能





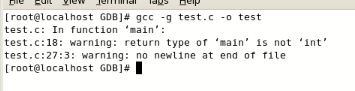


利用这个程序 来进行调试

首先编译这个程序

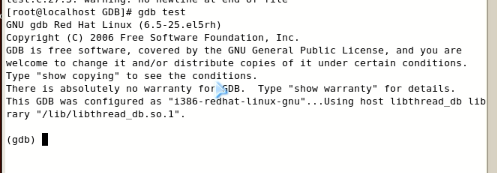
\*\*编译选项 ---- 提到了 如果想调试编译好的程序 必须添加-g这个选项 让编译的程序具有调试信息

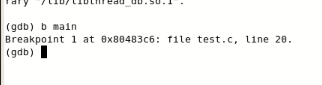
编译程序



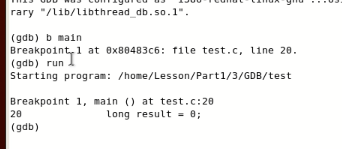
编译完成了

启动GDB GDB跟上你要调试的程序名



然后在主函数这打一个断点 

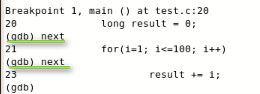
然后 运行这个程序 run



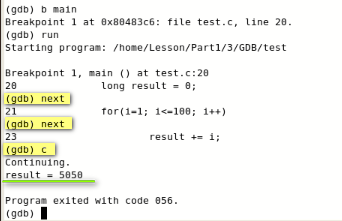
这样 运行起来之后 停在我们源代码的第20行

程序确实停在了这一行



 连续敲入两个next指令 就会单步执行两次

如果我想让这个程序连续执行 敲入c 就是continue的意思



直到下一个断点 或者 程序结束

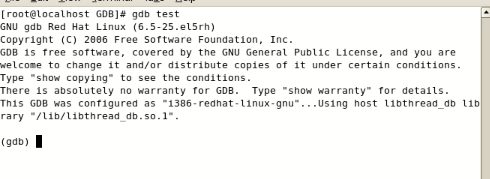
退出 输入quit指令

【所以没有输入quit之前 还在gdb这个shell里面 所以 可以再次输入你要添加断点的位置 然后 执行run命令】

--------------------------- 介绍GDB的命令

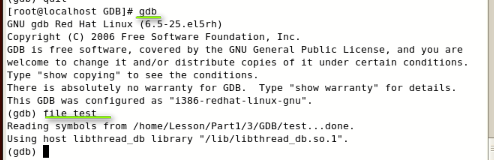
Gdb的命令非常多

 两个方法 启动gdb



这个是第一种方法

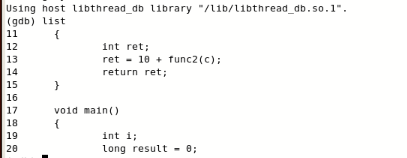
或者直接敲入gdb 然后 使用file命令指明哪一个程序需要被调试



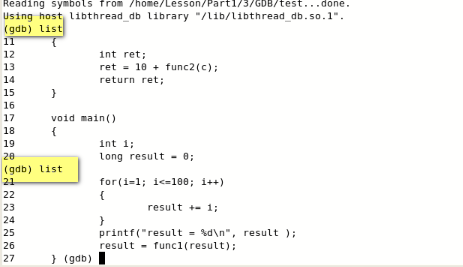
打断点



---使用l list命令 查看程序



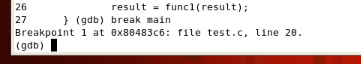
再次输入list 又会显示一段程序



这个时候 可以选择在哪一行打断点

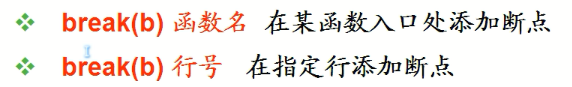
打断点使用b 或者break

如果要在函数的入口处打断点 break(b) 函数名



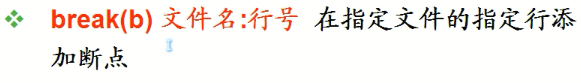
这样断点打在第20行

另一种是

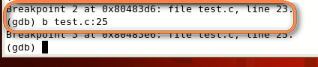


断点打在第23行



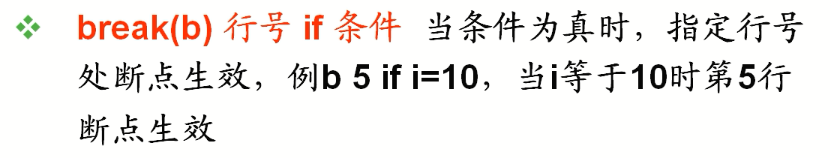


如果我的项目多个文件构成 就用文件名+行号

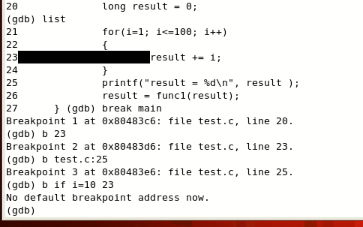


【注意 这个时候 我没有运行我的程序 所以 这个过程就相当于我打开了eclipse 然后双击某一行 添加断点一样 把想要添加的断点都加上去之后 然后点击debug按钮的过程 点击debug按钮 就相当于在gdb中输入了run命令】

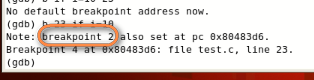
下面是条件断点



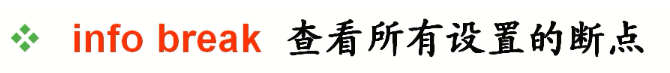
现在在23行打一个条件断点



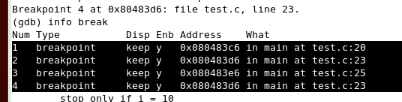
这个地方打了一个条件断点 报错了 提示 是没有默认的断点地址 所以 就是没有给出这个断点需要打的位置



打了一个条件断点



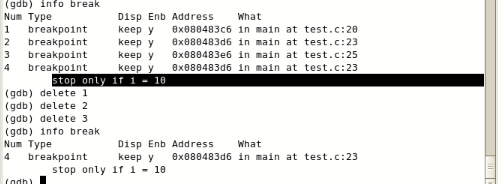
查看程序中所有的断点



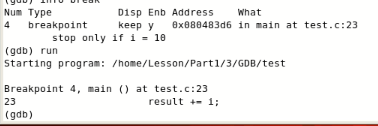
可以看到第四个断点下面写的是 stop only if i=10



删除断点



打好之后 可以运行这个程序



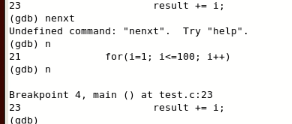
停在这个断点处 查看i的值





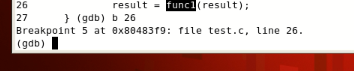
使用print查看一个变量值

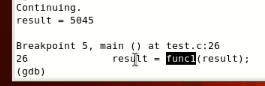
还可以使用next执行单步运行 简写成n



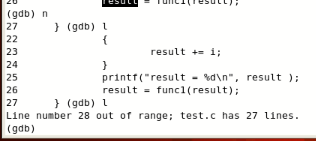
还有一个step 和next的区别就是 --- 遇到函数调用 step会进入这个子函数 next则不会

现在在26行敲入一个断点





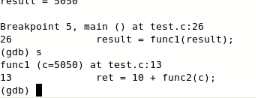
执行一个next



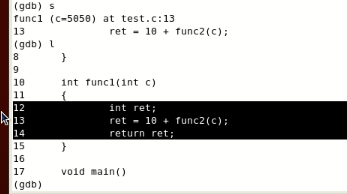
这时候 没有进入到func1中运行

换成step命令 s命令

在26行

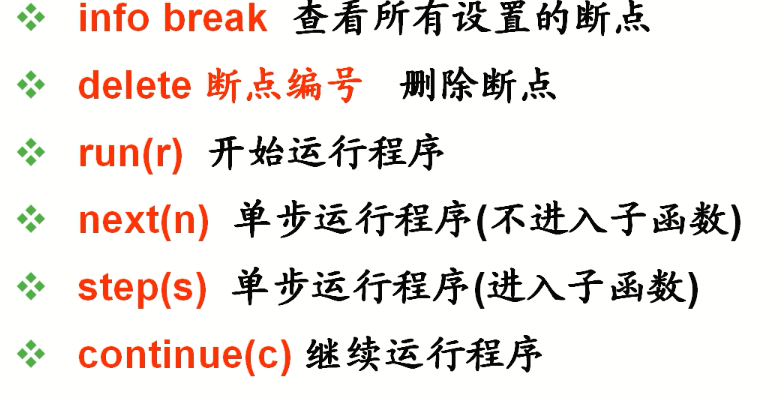


使用l

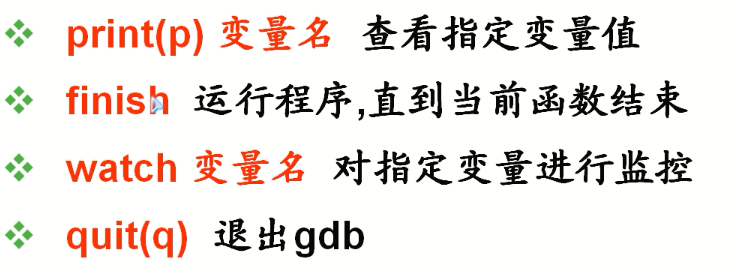


看到 进入了子函数里面

这就是s和n的区别



Continue是继续运行 连续运行 continue后面没有遇到断点是不会停下来的



finish 到当前函数结束

watch对某一个变量进行监控

quit是退出gdb