# what is Gdb

## 原理：

* 当你的程序被GDB停住时，所有的运行线程都会被停住 ，单运行时候全部激活。

In all-stop mode, whenever your program stops under gdb for any reason, all threads of

execution stop, not just the current thread.

Conversely, whenever you restart the program, all threads start executing

* 断点触发时候，会随机停留在其中的一个线程。

Whenever gdb stops your program, due to a breakpoint or a signal, it **automatically**

**selects the thread** where that breakpoint or signal happened.

## 命令



**Breakpoints, Watchpoints, and Catchpoints**

**在实际调试中，我们一般会用到三种断点：普通断点、条件断点和硬件断点**。

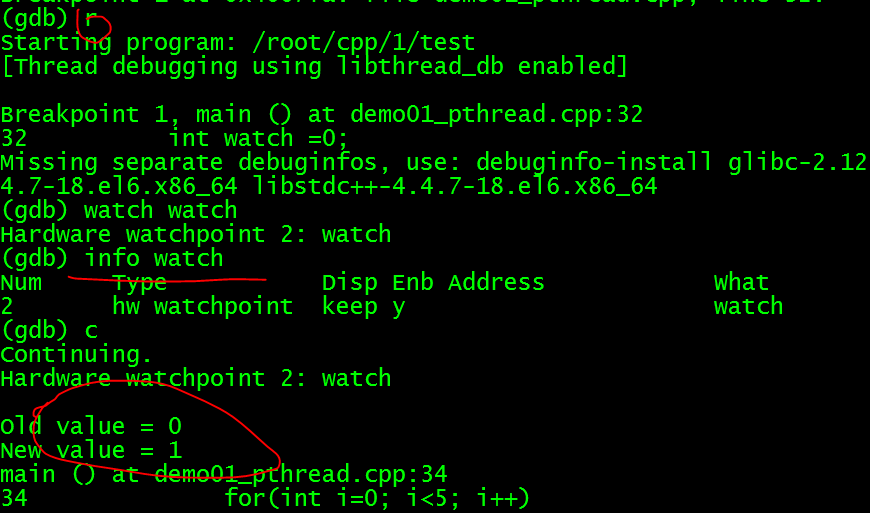
**更不没想到watch竟然也可以stop程序**

有一个变量其值被意外地改掉了，

通过单步调试或者挨个检查使用该变量的代码工作量会非常大，

如何快速地定位到该变量在哪里被修改了

**https://sourceware.org/gdb/onlinedocs/gdb/Set-Watchpoints.html**



**说明：如果你不指定thread <threadno>则表示你的断点设在所有线程上面**

Stopping and Starting Multi-thread Programs

catch syscall mmap

**处理信号**

handle SIGINT nostop print pass

signal SIGINT

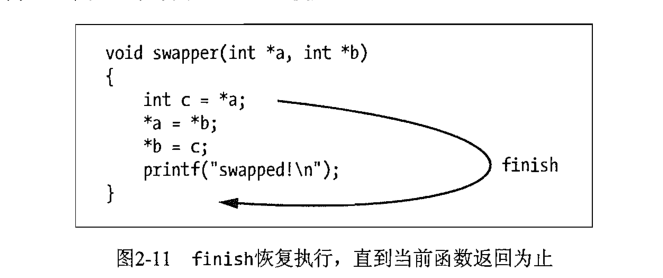
说明：

The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored

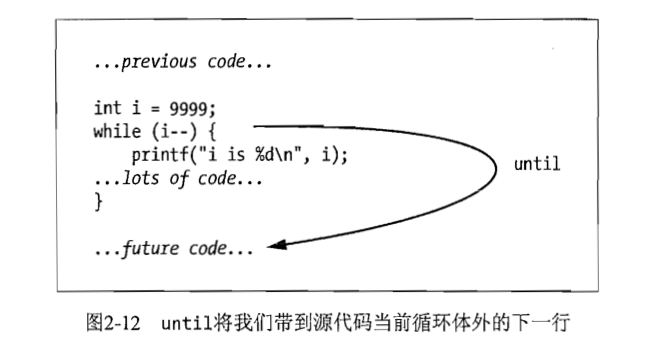
* continue    继续运行程序直到下一个断点（类似于VS里的F5）
* next        逐过程步进，不会进入子函数（类似VS里的F10）
* setp        逐语句步进，会进入子函数（类似VS里的F11）
* until        运行至当前语句块结束
* finish    运行至函数结束并跳出，并打印函数的返回值（类似VS的Shift+F11）

使用finish恢复程序执行

* 想返回到单步进入被调用函数之前GDB所在的调用函数



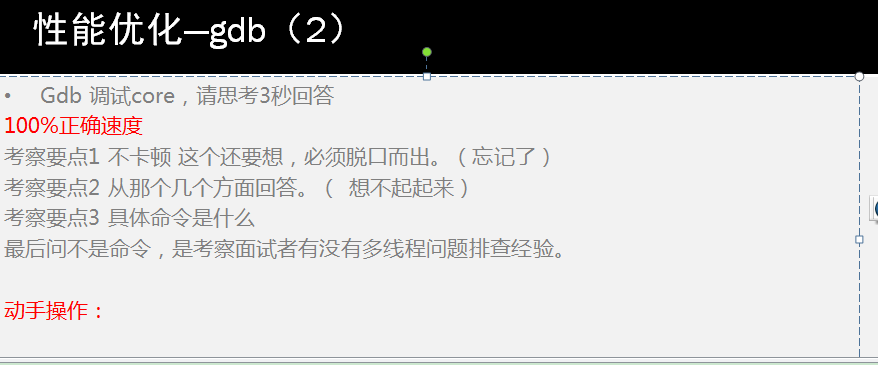
**使用until恢复程序执行**



* 当i很大是，使用next需要多次。而使用until会执行循环的其余部分

没有调试信息的发布版程序时，也只能通过反汇编代码去定位问题

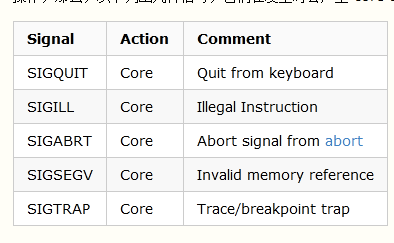
## **Disassemble**



Core文件产生流程

大概说一下从进程出现异常到生成core文件的流程，不会涉及太多Linux系统实现细节，具体细节可以参考相关文档和linux内核源码。

* 进程运行时发生一个异常，比如非法内存地址访问（即段错误），相应硬件会上报该异常，CPU检测到该异常时会进入异常处理流程,包括保存当前上下文，跳转到对应的中断向量执行入口等
* 在异常处理流程中如果判断该异常发生时是处于用户态，则该异常只会影响当前进程，此时向用户态进程发送相应的信号，如段错误就会发送SIGSEGV信号
* 当用户态进程被调度时会检查pending的信号，如果发现pending的信号是SIG\_KERNEL\_COREDUMP\_MASK中的一个，就会进入core文件内容收集存储流程，然后根据配置(core\_pattern等)生成core文件



dmesg addr2line

ulimit -c unlimited（单测）

info frame

 rip 指令寄存器，指向当前执行的代码位置

 $rsp 栈指针寄存器，指向当前栈顶

## 技巧：

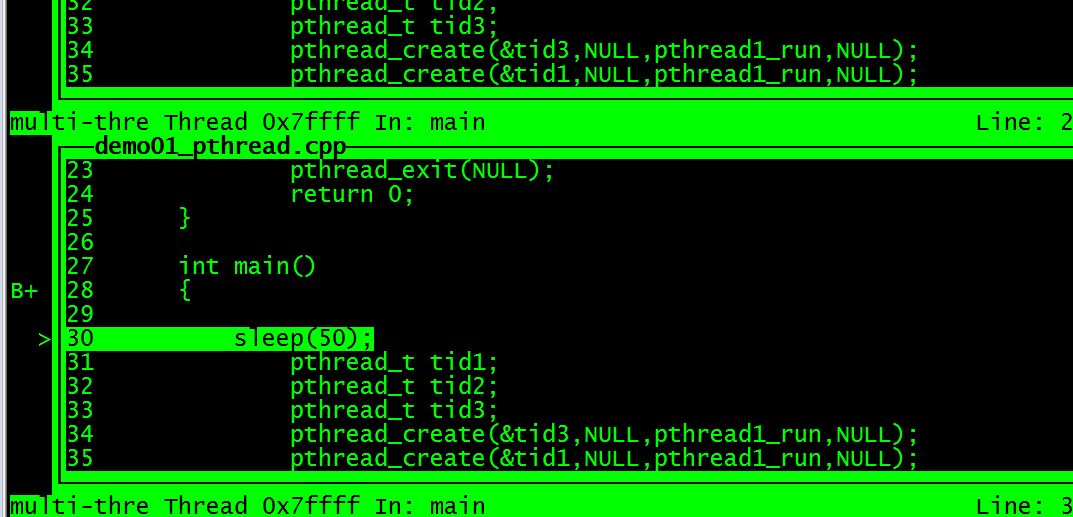
* **gdb默认不支持STL，gdbinit文件中，就可以使用pvector命令查看vector容器数据**
* GDB TUI——在 GDB 中显示程序源码

GDB TUI 提供了一个可视化的代码阅读功能

在需要的时候使用切换键 Ctrl + X + A 调出 GDB TUI 。

* GDB 高级扩展工具——CGDB

<https://github.com/leeyiw/cgdb-manual-in-chinese/blob/master/SUMMARY.md>



# 二．gdb调试中间件

## 2.1 Redis6.0

gcc version 7.5.0 (需要c++11 )

## 为了调试，在编译redis的源码的时候需要带上参数，默认情况下，Redis是使用 -O2选项编译

make CFLAGS="-g -O0" || make noopt

## 执行如下命令

Cd /root/local/redis-6.0.4/src

./redis-server ../redis.conf &

## 即可进入gdb的模式 如下

ps -ef |grep redis

gdb -p 2730

## 2.2.Nginx

## 2.3.Mysql

# 巨人的肩膀

<https://blog.csdn.net/jogger_ling/article/details/64443470>

https://blog.csdn.net/small\_prince\_/article/details/80682110

Demo <https://www.cnblogs.com/hazir/p/linxu_core_dump.html>

<https://wizardforcel.gitbooks.io/100-gdb-tips/catch-syscall.html>

<https://blog.csdn.net/Shinlyzsljay/article/details/105960824?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase>

<https://emacsist.github.io/2019/08/09/%E4%BD%BF%E7%94%A8gdb%E8%B0%83%E8%AF%95redis/#%E6%98%BE%E7%A4%BA%E5%8F%98%E9%87%8F-%E5%87%BD%E6%95%B0>

<https://emacsist.github.io/2019/08/09/%E4%BD%BF%E7%94%A8gdb%E8%B0%83%E8%AF%95redis/>

<https://blog.csdn.net/zhangjs0322/article/details/10152279>

https://blog.csdn.net/u010659887/article/details/89677351