



帐号☐ 自动登录 找回密码

密码 骑士注

技术 ♦ 学习 新闻 ♦ 快讯 观点 ♦ 热议 软件 ♦ 分享 **论坛** **投稿**

Locez 新手指南： [下载 Linux »](#) [安装 Linux »](#) [安装软件 »](#) [基础命令 »](#)

请注册后再搜索 搜索



技术 ♦ 学习 查看内容

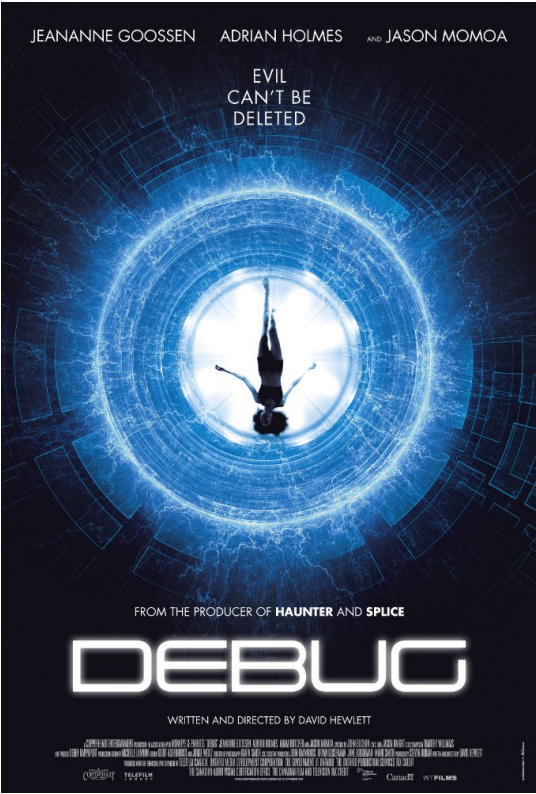
五个 Linux 下用户空间的调试工具

编译自：<http://linoxide.com/linux-how-to/user-space-debugging-tools-linux/>
原创：LCTT <https://linux.cn/article-5047-1.html>
本文地址：<https://linux.cn/article-5047-1.html>

作者：B N Poornima
译者：mtunique

2015-3-13 10:37 评论: 12 收藏: 11 分享: 14

根据定义，调试工具是那些那些使我们能够监测、控制和纠正其他程序的程序。我们为什么应该用调试工具呢？在有些情况下，运行一些程序的时候我们会被卡住，我们需要明白究竟发生了什么。例如，我们正在运行应用程序，它产生了一些错误消息。要修复这些错误，我们应该先找出为什么产生这些错误的消息和这些错误消息从哪里产生的。一个应用程序可能突然挂起，我们必须了解其他什么进程同时在运行。我们可能还必须弄清楚某个进程挂起的时候在做什么。为了剖析这些细节， 我们需要调试工具的帮助。



（题图来自：axxomovies.org）

有几个Linux下的用户空间调试工具和技术，它们用来分析用户空间的问题相当有用。它们是：

- 'print' 语句
- 查询 (/proc, /sys 等)
- 跟踪 (strace/ltrace)



热点评论

山鹰Sniper03 2015-3-13 13:55

gdb。。。这东西是因为实在没得用了才用的。。。确实是调试起来很麻烦，我还试过KDBG，界面难看我就不说了，断点的位置很难设置。最后我用上了codeblocks。。。。。

赞 8

Linux中国 2015-3-13 10:22

回复@纤夫张:哦，对，这个是错了。。除非删除这条微博了。。

赞 1



本文导航

- 1. 'print' 语句
- 2. 查询
- 3. 跟踪
 - strace:
 - ltrace:
- 4. Valgrind
- 5. GDB
 - 编译程序:

相关阅读

调试	Debug
Linux 内核的测试和调试（1）	2014-8-14
Linux 内核测试与调试（3）	2014-8-19
Linux 内核测试和调试（4）	2014-8-26
Linux 内核测试和调试（5）	2014-8-27
Linux 内核的测试和调试（6）	2014-8-28
使用GDB命令行调试器调试C/C++程序	2014-11-25

- Valgrind (memwatch)
- GDB

让我们一个个地了解。

1. 'print' 语句

这是一个基本的原始的调试问题的方法。 我们可以在程序中插入print语句来了解控制流和变量值。 虽然这是一个简单的技术，但它有一些缺点。 程序需要进行编辑以添加'print'语句，然后必须重新编译，重新运行来获得输出。 如果要调试的程序相当大，这是一个耗时的方法。

2. 查询

在某些情况下，我们需要弄清楚在一个运行在内核中的进程的状态和内存映射。为了获得这些信息，我们不需要在内核中插入任何代码。 相反，可以用 /proc 文件系统。

/proc 是一个伪文件系统，系统一启动运行就收集着运行时系统的信息 (cpu信息, 内存容量等)。

```
mint@mint ~ $ ls /proc
1      2022 2789 3238 3540 5      dri      partitions
10     2028 28    325  3554 50     driver   sched_debug
11     2040 29    326  3588 51     execdomains schedstat
12     2079 3     327  359  52     fb        scsi
13     2099 30    328  3599 53     filesystems self
1366   21    3056 33   36   54     fs        slabinfo
1374   22    3057 3345 360  55     interrupts softirqs
1376   23    3058 3361 3609 558    iomem     stat
1387   24    3059 3367 3619 559    ioports   swaps
14     2457 3060 3386 3636 560    irq       sys
1413   25    3061 3388 37   561    kallsyms  sysrq-trigger
1419   2501 3076 3389 3730 573    kcore     sysvipc
1420   2581 3080 3391 3770 6     key-users timer_list
15     2588 3098 3393 3773 7     kmsg      timer_stats
1594   2595 31    34   3774 8     kpagecount tty
1656   2596 3158 3406 38   9     kpageflags uptime
1658   26    3173 343  3805 acpi      latency_stats version
17     2666 32    3460 3809 asound    loadavg   version_signature
18     2669 3203 35   3813 buddyinfo locks      vmallocinfo
1830   2687 3216 3500 3869 bus       mdstat    vmstat
1873   2688 3218 3501 3870 cgroups   meminfo   zoneinfo
1876   27    3221 3503 3886 cmdline  misc
19     2701 3224 3506 39   cpuinfo   modules
1980   2712 3225 3507 4     crypto    mounts
1999   2715 3230 3517 47    devices  mtrr
2      2721 3233 3519 48    diskstats net
20     2724 3235 352  49    dma       pagetypeinfo
```

'ls /proc'的输出

正如你看到的, 系统中运行的每一个进程在/proc文件系统中有一个以进程id命名的项。每个进程的细节信息可以在进程id对应的目录下的文件中获得。

```
mint@mint ~ $ ls /proc/10
attr      cpuset  latency mountstats sched  status
auxv      cwd     limits  net        schedstat syscall
cgroup    environ loginuid oom_adj   sessionid task
clear_refs exe     maps    oom_score smaps     wchan
cmdline   fd      mem     pagemap   stack
comm      fdinfo mountinfo personality stat
coredump_filter io       mounts  root      statm
```

'ls /proc/pid'的输出

解释/proc文件系统内的所有条目超出了本文的范围。一些有用的列举如下：

- /proc/cmdline -> 内核命令行
- /proc/cpuinfo -> 关于处理器的品牌，型号信息等
- /proc/filesystems -> 文件系统的内核支持的信息
- /proc/<pid>/cmdline -> 命令行参数传递到当前进程
- /proc/<pid>/mem -> 当前进程持有的内存
- /proc/<pid>/status -> 当前进程的状态

3. 跟踪

strace的和ltrace是两个在Linux中用来追踪程序的执行细节的跟踪工具。

strace:

strace拦截和记录系统调用及其接收的信号。对于用户，它显示了系统调用、传递给它们的参数和返回值。strace的可以附着到已在运行的进程或一个新的进程。它作为一个针对开发者和系统管理员的诊断、调试工具是很有用的。它也可以用来当做一个通过跟踪不同的程序调用来了解系统的工具。这个工具的好处是不需要源代码，程序也不需要重新编译。

使用strace的基本语法是：

strace 命令

strace有各种各样的参数。可以检查查看strace的手册页来获得更多的细节。

strace的输出非常长，我们通常不会对显示的每一行都感兴趣。我们可以用'-e expr'选项来过滤不想要的数

据。用 '-p pid' 选项来绑定到运行中的进程。

用'-o'选项，命令的输出可以被重定向到文件。

```
mint@mint ~ $ strace -e open ls
open("/etc/ld.so.cache", 0_RDONLY) = 3
open("/lib/libselinux.so.1", 0_RDONLY) = 3
open("/lib/librt.so.1", 0_RDONLY) = 3
open("/lib/libacl.so.1", 0_RDONLY) = 3
open("/lib/libc.so.6", 0_RDONLY) = 3
open("/lib/libdl.so.2", 0_RDONLY) = 3
open("/lib/libpthread.so.0", 0_RDONLY) = 3
open("/lib/libattr.so.1", 0_RDONLY) = 3
open("/proc/filesystems", 0_RDONLY|O_LARGEFILE) = 3
open("/usr/lib/locale/locale-archive", 0_RDONLY|O_LARGEFILE) = 3
open(".", 0_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_CLOEXEC) = 3
Desktop Documents Downloads Music Pictures Public Templates Videos
```

strace过滤成只有系统调用的输出

ltrace:

ltrace跟踪和记录一个进程的动态（运行时）库的调用及其收到的信号。它也可以跟踪一个进程所作的系统调用。它的用法是类似与strace。

ltrace command

'-i' 选项在调用库时打印指令指针。

'-S' 选项被用来现实系统调用和库调用

所有可用的选项请参阅ltrace手册。

```
mint@mint ~/Test $ ltrace -e strcmp find . -name test
strcmp("!", "(") = -1
strcmp("not", "(") = 1
strcmp("(", "(") = 0
strcmp("!", "(") = -1
strcmp("not", "(") = 1
strcmp("(", "(") = -1
strcmp(")", "(") = 0
strcmp("!", "print") = -1
strcmp("not", "print") = -1
strcmp("(", "print") = -1
strcmp(")", "print") = -1
strcmp(" ", "print") = -1
```

ltrace捕捉'STRCMP'库调用的输出

4. Valgrind

Valgrind是一套调试和分析工具。它的一个被广泛使用的默认工具——'Memcheck'——可以拦截malloc(), new(), free()和delete()调用。换句话说，它在检测下面这些问题非常有用：

- 内存泄露
- 重释放
- 访问越界
- 使用未初始化的内存
- 使用已经被释放的内存等。

它直接通过可执行文件运行。

Valgrind也有一些缺点，因为它增加了内存占用，会减慢你的程序。它有时会造成误报和漏报。它不能检测出静态分配的数组的访问越界问题。

为了使用它，首先请下载并安装在你的系统上。可以使用操作系统上的包管理器来安装。

使用命令行安装需要解压缩和解包下载的文件。

```
1. |tar -xjvf valgrind-x.y.z.tar.bz2 (where x.y.z is the version number you are trying to install)
```

进入新创建的目录（的valgrind-XYZ）内运行以下命令：

```
1. |./configure
2. |make
3. |make install
```

让我们通过一个小程序(test.c)来理解valgrind怎么工作的：

```
1. |#include <stdio.h>
2.
3. |void f(void)
4.
5. |{
6. |int x = malloc(10 * sizeof(int));
7.
8. |x[10] = 0;
9. |}
10.
11. |int main()
12. |{
13. |f();
14. |return 0;
15. |}
```

编译程序：

```
1. |gcc -o test -g test.c
```

现在我们有一个可执行文件叫做'test'。我们现在可以用valgrind来检测内存错误：

```
1. |valgrind -tool=memcheck -leak-check=yes test
```

这是valgrind呈现错误的输出：

```
==19182== Invalid write of size 4
==19182==    at 0x804838F: f (test.c:6)
==19182==    by 0x80483AB: main (test.c:11)
==19182== Address 0x1BA45050 is 0 bytes after a block of size 40 alloc'd
==19182==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182==    by 0x8048385: f (test.c:5)
==19182==    by 0x80483AB: main (test.c:11)
==19182== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==19182==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182==    by 0x8048385: f (test.c:5)
==19182==    by 0x80483AB: main (test.c:11)
```

valgrind显示堆溢出和内存泄漏的输出

正如我们在上面看到的消息，我们正在试图访问函数f未分配的内存以及分配尚未释放的内存。

5. GDB

GDB是来自自由软件基金会的调试器。它对定位和修复代码中的问题很有帮助。当被调试的程序运行时，它给用户控制权去执行各种动作，比如：

- 启动程序
- 停在指定位置
- 停在指定的条件

- 检查所需信息
- 改变程序中的数据 等。

你也可以将一个崩溃的程序coredump附着到GDB并分析故障的原因。

GDB提供很多选项来调试程序。 然而，我们将介绍一些重要的选择，来感受如何开始使用GDB。

如果你还没有安装GDB，可以在这里下载：[GDB官方网站](#)。

编译程序：

为了用GDB调试程序，必须使用gcc的'-g'选项进行编译。这将以操作系统的本地格式产生调试信息，GDB利用这些信息来工作。

下面是一个简单的程序（example1.c）执行被零除用来显示GDB的用法：

```
1. #include
2. int divide()
3. {
4.     int x=5, y=0;
5.     return x / y;
6. }
7.
8. int main()
9. {
10.    divide();
11. }
```

```
mint@mint ~/Test $ gdb example1
GNU gdb (GDB) 7.2-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/mint/Test/example1...done.
(gdb) run
Starting program: /home/mint/Test/example1

Program received signal SIGFPE, Arithmetic exception.
0x080483b0 in divide () at example1.c:7
7       return x / y;
(gdb) bt
#0  0x080483b0 in divide () at example1.c:7
#1  0x080483bd in main () at example1.c:12
(gdb) list
2
3
4     int divide()
```

展示GDB用法的例子

调用 GDB：

通过在命令行中执行'gdb'来启动gdb:

```
mint@mint ~ $ gdb
GNU gdb (GDB) 7.2-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb)
```

调用 gdb

调用后, 它将等待终端命令并执行，直到退出。

如果一个进程已经在运行，你需要将GDB连接到它上面，可以通过指定进程ID来实现。假设程序已经崩溃，要分析问题原因，则用GDB分析core文件。

启动程序：

一旦你在GDB里面，使用'run'命令来启动程序进行调试。

给程序传参数：

使用'set args'给你的程序传参数，当程序下次运行时将获得该参数。'show args'将显示传递给程序的参数。

检查堆栈：

每当程序停止，任何人想明白的第一件事就是它为什么停止，以及怎么停在那里的。该信息被称为反向跟踪。由程序产生每个函数调用和局部变量，传递的参数，调用位置等信息一起存储在堆栈内的数据块种，被称为一帧。我们可以使用GDB来检查所有这些数据。GDB从最底层的帧开始给这些帧编号。

- **bt**: 打印整个堆栈的回溯
- **bt** 打印n个帧的回溯
- **frame** : 切换到指定的帧，并打印该帧
- **up** : 上移'n'个帧
- **down** : 下移'n'个帧 (n默认是1)

检查数据：

程序的数据可以在里面GDB使用'print'命令进行检查。例如，如果'x'是调试程序内的变量，'print x'会打印x的值。

检查源码：

源码可以在GDB中打印。默认情况下，'list'命令会打印10行代码。

- **list** : 列出'linenum'行周围的源码
- **list** : 从'function'开始列出源码
- **disas** : 显示该函数机器代码

停止和恢复程序：

使用GDB，我们可以在必要的地方设置断点，观察点等来停止程序。

- **break** : 在'location'设置一个断点。当在程序执行到这里时断点将被击中，控制权被交给用户。
- **watch** : 当'expr'被程序写入而且它的值发生变化时GDB将停止
- **catch** : 当'event'发生时GDB停止
- **disable** : 禁用指定断点
- **enable** : 启用指定断点
- **delete** : 删除 断点/观察点/捕获点。 如果没有传递参数默认操作是在所有的断点
- **step**: 一步一步执行程序
- **continue**: 继续执行程序，直到执行完毕

退出 GDB：

用'quit'命令还从GDB中退出。

GDB还有更多的可用选项。里面GDB使用help选项了解更多详情。


```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb) |
```

在GDB中获得帮助

总结

在这篇文章中，我们已经看到不同类型的Linux用户空间的调试工具。总结以上内容，如下是什么时候使用该什么的快速指南：

- 基本调试，获得关键变量 - `print` 语句
- 获取有关文件系统支持，可用内存，CPU，运行程序的内核状态等信息 - 查询 `/proc` 文件系统
- 最初的问题诊断，系统调用或库调用的相关问题，了解程序流程 – `strace` / `ltrace`
- 应用程序内存空间的问题 – `valgrind`
- 检查应用程序运行时的行为，分析应用程序崩溃 – `gdb`

via: <http://linoxide.com/linux-how-to/user-space-debugging-tools-linux/>

作者： **B N Poornima** 译者： **mtunique** 校对： **wxy**

本文由 **LCTT** 原创翻译， **Linux中国** 荣誉推出

编译自：<http://linoxide.com/linux-how-to/user-space-debugging-tools-linux/>
原创：LCTT <https://linux.cn/article-5047-1.html>

作者： **B N Poornima**
译者： **mtunique**

本文由 LCTT 原创翻译，Linux中国首发。也想加入译者行列，为开源做一些自己的贡献么？欢迎加入 LCTT！

翻译工作和译文发表仅用于学习和交流目的，翻译工作遵照 CC-BY-NC-SA 协议规定，如果我们的工作有侵犯到您的权益，请及时联系我们。

欢迎遵照CC-BY-NC-SA 协议规定转载，敬请在正文中标注并保留原文/译文链接和作者/译者等信息。

文章仅代表作者的知识和看法，如有不同观点，请楼下排队吐槽 :D

上一篇：理解和解决 MySQL 乱码问题

下一篇：原生体验挡不住！JavaScript开源跨平台框架NativeScript


发表评论

验证码

换一个

评论

最新评论

 **lestat_henry** 2015-3-14 18:33 新浪微博网友评论

@我的印象笔记

赞 回复

 **shenwei5566** 2015-3-13 14:03 新浪微博网友评论

@mark

赞 回复

 **Sniper03** 2015-3-13 13:55

gdb。。。这东西是因为实在没得用了才用的。。。确实是调试起来很麻烦，我还试过KDBG，界面难看我就不说了，断点的位置很难设置。最后我用上了codeblocks。。。。。

赞 8 回复

lyhabc 2015-3-13 13:12

先收藏了

赞 回复

 **Linux中国** 2015-3-13 10:22 新浪微博网友评论

回复@纤夫张:哦，对，这个是错了。。除非删除这条微博了。。

赞 1 回复

 **纤夫张** 2015-3-13 10:22 新浪微博网友评论

回复@Linux中国:微博里标题后面摘要要是错的，老是提google code关闭的事情，但到你们网站的链接是对的。

赞 回复

 **Linux中国** 2015-3-13 10:03 新浪微博网友评论

回复@纤夫张:这条微博附带的链接，是这个 DEBUG 的啊，之前是出了一些事情，现在调整好了，也许是你那边的浏览器的缓冲？——手机浏览器好像不能刷新？

赞 回复

 **纤夫张** 2015-3-13 10:03 新浪微博网友评论

回复@Linux中国:文不对题。新发的好了，老的没改？

赞 回复

 **Linux中国** 2015-3-13 10:03 新浪微博网友评论

回复@纤夫张:具体情况？您说说？

赞 回复

 **纤夫张** 2015-3-13 10:03 新浪微博网友评论

依旧，或许微博有缓存。

赞 回复

 **Linux中国** 2015-3-13 10:03 新浪微博网友评论

回复@纤夫张:临时问题，现在好了吧？

赞 回复

 **纤夫张** 2015-3-13 09:33 新浪微博网友评论

你们的转帖机器人挂了。

赞 回复