Valgrind简单用法

Valgrind的主要作者Julian Seward刚获得了今年的Google-O'Reilly开源大奖之一——Best Tool Maker。让我们一起来看一下他的作品。Valgrind是运行在Linux上一套基于仿真技术的程序调试和分析工具，它包含一个内核——一个软件合成的CPU，和一系列的小工具，每个工具都可以完成一项任务——调试，分析，或测试等。Valgrind可以检测内存泄漏和内存违例，还可以分析cache的使用等，灵活轻巧而又强大，能直穿程序错误的心脏，真可谓是程序员的瑞士军刀。

一. Valgrind概观

Valgrind的最新版是3.2.0，它一般包含下列工具：

1.Memcheck

最常用的工具，用来检测程序中出现的内存问题，所有对内存的读写都会被检测到，一切对malloc()/free()/new/delete的调用都会被捕获。所以，它能检测以下问题：

1.对未初始化内存的使用；
2.读/写释放后的内存块；
3.读/写超出malloc分配的内存块；
4.读/写不适当的栈中内存块；
5.内存泄漏，指向一块内存的指针永远丢失；
6.不正确的malloc/free或new/delete匹配；
7,memcpy()相关函数中的dst和src指针重叠。

这些问题往往是C/C++程序员最头疼的问题，Memcheck在这里帮上了大忙。

2.Callgrind

和gprof类似的分析工具，但它对程序的运行观察更是入微，能给我们提供更多的信息。和gprof不同，它不需要在编译源代码时附加特殊选项，但加上调试选项是推荐的。Callgrind收集程序运行时的一些数据，建立函数调用关系图，还可以有选择地进行cache模拟。在运行结束时，它会把分析数据写入一个文件。callgrind_annotate可以把这个文件的内容转化成可读的形式。

3.Cachegrind

Cache分析器，它模拟CPU中的一级缓存I1，Dl和二级缓存，能够精确地指出程序中cache的丢失和命中。如果需要，它还能够为我们提供cache丢失次数，内存引用次数，以及每行代码，每个函数，每个模块，整个程序产生的指令数。这对优化程序有很大的帮助。

4.Helgrind

它主要用来检查多线程程序中出现的竞争问题。Helgrind寻找内存中被多个线程访问，而又没有一贯加锁的区域，这些区域往往是线程之间失去同步的地方，而且会导致难以发掘的错误。Helgrind实现了名为“Eraser”的竞争检测算法，并做了进一步改进，减少了报告错误的次数。不过，Helgrind仍然处于实验阶段。

5. Massif

堆栈分析器，它能测量程序在堆栈中使用了多少内存，告诉我们堆块，堆管理块和栈的大小。Massif能帮助我们减少内存的使用，在带有虚

随笔档案

拟内存的现代系统中，它还能够加速我们程序的运行，减少程序停留在交换区中的几率。
此外，lackey和nulgrind也会提供。Lackey是小型工具，很少用到；Nulgrind只是为开发者展示如何创建一个工具。我们就不做介绍了。
二. 使用Valgrind
Valgrind的使用非常简单，valgrind命令的格式如下：
valgrind [valgrind-options] your-prog [your-prog options]
一些常用的选项如下：
选项
作用
-h --help
显示帮助信息。
--version
显示valgrind内核的版本，每个工具都有各自的版本。
-q --quiet
安静地运行，只打印错误信息。
-v --verbose
打印更详细的信息。
--tool= [default: memcheck]
最常用的选项。运行valgrind中名为toolname的工具。如果省略工具名，默认运行memcheck。
--db-attach= [default: no]
绑定到调试器上，便于调试错误。
我们通过例子看一下它的具体使用。我们构造一个存在内存泄漏的C程序，如下：

```
#include
#include
void f(void)
{
int* x = malloc(10 * sizeof(int));
x[10] = 0; // problem 1: heap block overrun
} // problem 2: memory leak -- x not freed
int main(void)
{
int i;
f();
printf("i=%d/n",i); //problem 3: use uninitialised value.
return 0;
}
```

保存为memleak.c并编译，然后用valgrind检测。

```
$ gcc -Wall -o memleak memleak.c
```

2011年6月 (1)
2011年5月 (3)
2011年3月 (4)
2011年2月 (3)
2011年1月 (4)
2010年12月 (13)
2010年11月 (11)
2010年10月 (12)
2010年9月 (13)
2010年8月 (23)
2010年7月 (15)
2010年6月 (5)
2010年5月 (11)
2010年4月 (9)
2010年3月 (5)
2010年2月 (2)
2010年1月 (13)
2009年12月 (16)
2009年11月 (20)
2009年10月 (2)
2009年9月 (8)
2009年8月 (8)
2009年7月 (38)

最新评论

1. Re:关于file_operations结构体
结构体file_operations根本不在 linux/fs.h文件里
--流年逝水sing
2. Re:RTS与CTS的含义
ttl串口大数据量通信时 这个是必须的吗
--Ardai
3. Re:getopt()简介
谢谢楼主分享
--青儿哥哥
4. Re:日语中di，ti，du，这些如何用片假名打出来
第3个片假名应该是「デュ」。
--cici920

```
$ valgrind --tool=memcheck ./memleak
```

我们得到如下错误信息：

```
==3649== Invalid write of size 4
==3649== at 0x80483CF: f (in /home/wangcong/memleak)
==3649== by 0x80483EC: main (in /home/wangcong/memleak)
==3649== Address 0x4024050 is 0 bytes after a block of size 40 alloc'd
==3649== at 0x40051F9: malloc (vg_replace_malloc.c:149)
==3649== by 0x80483C5: f (in /home/wangcong/memleak)
==3649== by 0x80483EC: main (in /home/wangcong/memleak)
```

前面的3649是程序运行时的进程号。第一行是告诉我们错误类型，这里是非法写入。下面的是告诉我们错误发生的位置，在main()调用的f()函数中。

```
==3649== Use of uninitialised value of size 4
==3649== at 0xC3A264: _itoa_word (in /lib/libc-2.4.so)
==3649== by 0xC3E25C: vfprintf (in /lib/libc-2.4.so)
==3649== by 0xC442B6: printf (in /lib/libc-2.4.so)
==3649== by 0x80483FF: main (in /home/wangcong/memleak)
```

这个错误是使用未初始化的值，在main()调用的printf()函数中。这里的函数调用关系是通过堆栈跟踪的，所以有时会非常多，尤其是当你使用C++的STL时。其它一些错误都是由于把未初始化的值传递给libc函数而被检测到。在程序运行结束后，valgrind还给出了一个小的总结：

```
==3649== ERROR SUMMARY: 20 errors from 6 contexts (suppressed: 12 from 1)
==3649== malloc/free: in use at exit: 40 bytes in 1 blocks.
==3649== malloc/free: 1 allocs, 0 frees, 40 bytes allocated.
==3649== For counts of detected errors, rerun with: -v
==3649== searching for pointers to 1 not-freed blocks.
==3649== checked 47,256 bytes.
==3649==
==3649== LEAK SUMMARY:
==3649== definitely lost: 40 bytes in 1 blocks.
==3649== possibly lost: 0 bytes in 0 blocks.
==3649== still reachable: 0 bytes in 0 blocks.
==3649== suppressed: 0 bytes in 0 blocks.
==3649== Use --leak-check=full to see details of leaked memory.
```

我们可以很清楚地看出，分配和释放了多少内存，有多少内存泄漏。这对我们查找内存泄漏十分方便。然后我们重新编译程序并绑定调试器：

```
$ gcc -Wall -ggdb -o memleak memleak.c
$ valgrind --db-attach=yes --tool=memcheck ./memleak
```

一出现错误，valgrind会自动启动调试器（一般是gdb）：

```
==3893== ---- Attach to debugger ? --- [Return/N/n/Y/y/C/c] ---- y
```

5. Re:vim常用命令
谢谢分享
--太行山

阅读排行榜
1. shell判断文件是否存在(153440)
2. Valgrind简单用法(50229)
3. gcc 编译器常用的命令行参数一览(43472)
4. linux下svn客户端安装及环境配置(39198)
5. vim常用命令(22767)

评论排行榜
1. shell判断文件是否存在(2)
2. 使用Linux命令来发送信息(1)
3. 关于file_operations结构体(1)
4. PCLint使用（一）(1)
5. Linux常用的二进制文件分析方法(1)

推荐排行榜
1. Valgrind简单用法(7)
2. 关于file_operations结构体(3)
3. 详解linux下的串口通讯开发(3)
4. 用prctl给线程命名(2)
5. gcc 编译器常用的命令行参数一览(2)

starting debugger
==3893== starting debugger with cmd: /usr/bin/gdb -nw /proc/3895/fd/1014 3895
退出gdb后我们又能回到valgrind继续执行程序。
还是用上面的程序，我们使用callgrind来分析一下它的效率：
$ valgrind --tool=callgrind ./memleak
Callgrind会输出很多，而且最后在当前目录下生成一个文件： callgrind.out.pid。用callgrind_annotate来查看它：
$ callgrind_annotate callgrind.out.3949
详细的信息就列出来了。而且，当callgrind运行你的程序时，你还可以使用callgrind_control来观察程序的执行，而且不会干扰它的运行。
再来看一下cachegrind的表现：
$ valgrind --tool=cachegrind ./memleak
得到如下信息：
==4073== I refs: 147,500
==4073== I1 misses: 1,189
==4073== L2i misses: 679
==4073== I1 miss rate: 0.80%
==4073== L2i miss rate: 0.46%
==4073==
==4073== D refs: 61,920 (46,126 rd + 15,794 wr)
==4073== D1 misses: 1,759 (1,545 rd + 214 wr)
==4073== L2d misses: 1,241 (1,062 rd + 179 wr)
==4073== D1 miss rate: 2.8% (3.3% + 1.3%)
==4073== L2d miss rate: 2.0% (2.3% + 1.1%)
==4073==
==4073== L2 refs: 2,948 (2,734 rd + 214 wr)
==4073== L2 misses: 1,920 (1,741 rd + 179 wr)
==4073== L2 miss rate: 0.9% (0.8% + 1.1%)
上面的是指令缓存，I1和L2i缓存，的访问信息，包括总的访问次数，丢失次数，丢失率。
中间的是数据缓存，D1和L2d缓存，的访问的相关信息，下面的L2缓存单独的信息。Cachegrind也生成一个文件，名为cachegrind.out.pid，可以通过cg_annotate来读取。输出是一个更详细的列表。Massif的使用和cachegrind类似，不过它也会生成一个名为massif.pid.ps的PostScript文件，里面只有一幅描述堆栈使用状况的彩图。
以上只是简单的演示了valgrind的使用，更多的信息可以在它附带的文档中得到，也可以访问valgrind的主页：http://www.valgrind.org。学会正确合理地使用valgrind对于调试程序会有很大的帮助

SunBo
关注 - 2
粉丝 - 52
+加关注

« 上一篇：iPhone开发入门
» 下一篇：Linux Samba服务器搭建

posted on 2010-05-05 03:03 SunBo 阅读(50229) 评论(0) 编辑 收藏

最新知识库文章:

· 写给初学前端工程师的一封信
· 实用VPC虚拟私有云设计原则
· 如何阅读计算机科学类的书
· Google 及其云智慧
· 做到这一点，你也可以成为优秀的程序员
» 更多知识库文章...