Linux内存调试工具—Valgrind

larkguo@gmail.com

2007-5-19

目录

1 简介

Valgrind 的主要作者 Julian Seward 获得了 2005 年的 Google-O'Reilly 开源大奖之一——Best Tool Maker。Valgrind 是一个 GPL 的软件，用于 Linux 程序的调试和压型（profiling)。使用 Valgrind 的工具包，你可以自动的检测许多内存管理和线程的 bug，避免花费太多的时间在 bug 寻找上，使得你的程序更加稳固。Valgrind 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。

Valgrind 工具主要用于 C 和 C++写的程序，因为用这些语言写的程序倾向于有更多的 bug！但是它也能用于调试和压型（profiling）混合语言写的程序。Valgrind 已经被部分地或者完全的用于 C，C++，Java，Perl，Python，汇编，Fortran，Ada 和许多其他语言写的程序。

Valgrind可以和其他工具一起使用。Valgrind可以绑定GDB并附加到程序检测出错误的地方，因此你可以执行并且指出当前运行到什么地方了。

另外基于 valgrind 的分析工具前端 KCacheGrind 作为一个单独包也已可用。

2 安装

从http://valgrind.org/downloads/ 下载valgrind-3.2.1.tar.bz2 后:

```
tar -jvxf valgrind-3.2.1.tar.bz2
cd  valgrind-3.2.1
./configure
make
make install
```

3 主要功能

Valgrind 支持很多工具:Memcheck，Addrcheck，Cachegrind，Massif，Helgrind 和 Callgrind 等。在运行 Valgrind 时，你必须指明想用的工具,如果省略工具名，默认运行 memcheck。

Memcheck 是最常用的工具，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，下面通过例子看一下它的具体使用：

3. 1 内存泄漏

构造一个存在内存泄漏的 C 程序 test.c：

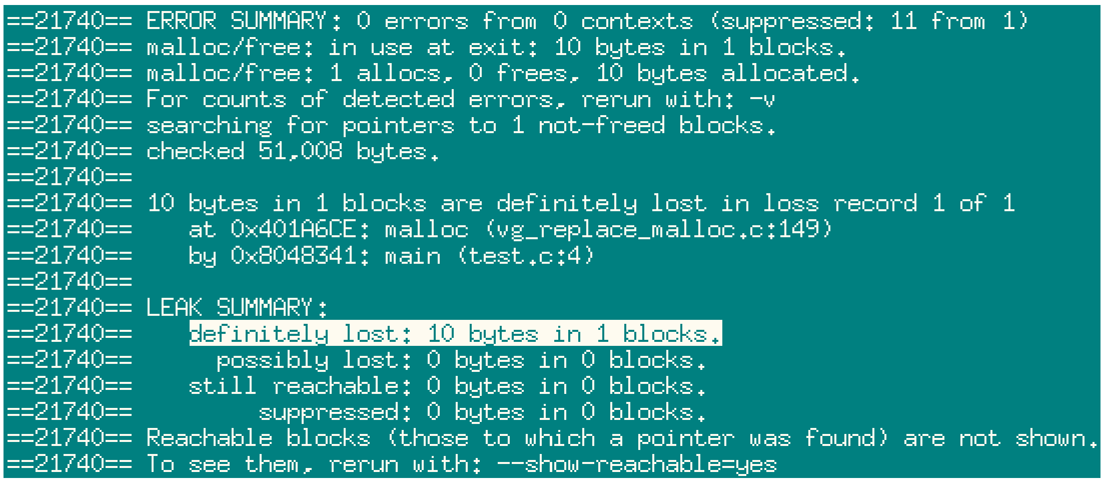
```
#include <stdlib.h>
int main()
{
        char *x = malloc(10);
        return 0;
}
```

编译:gcc -g test.c -o test

注：使用 -g 选项，您就有可能获得一些信息来直接指出相关的代码行.

运行:valgrind --tool=memcheck --leak-check=yes ./test

运行结果：



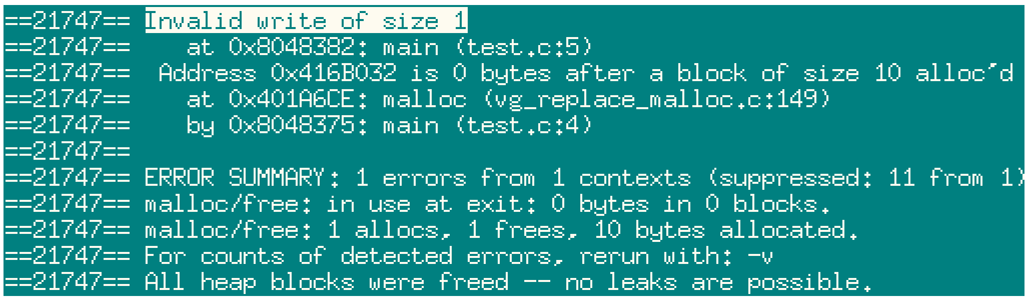
```
==21740== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 11 from 1)
==21740== malloc/free: in use at exit: 10 bytes in 1 blocks.
==21740== malloc/free: 1 allocs, 0 frees, 10 bytes allocated.
==21740== For counts of detected errors, rerun with: -v
==21740== searching for pointers to 1 not-freed blocks.
==21740== checked 51,008 bytes.
==21740==
==21740== 10 bytes in 1 blocks are definitely lost in loss record 1 of 1
==21740==    at 0x401A6CE: malloc (vg_replace_malloc.c:149)
==21740==    by 0x8048341: main (test.c:4)
==21740==
==21740== LEAK SUMMARY:
==21740==    definitely lost: 10 bytes in 1 blocks.
==21740==      possibly lost: 0 bytes in 0 blocks.
==21740==    still reachable: 0 bytes in 0 blocks.
==21740==         suppressed: 0 bytes in 0 blocks.
==21740== Reachable blocks (those to which a pointer was found) are not shown.
==21740== To see them, rerun with: --show-reachable=yes
```

前面的 21740 是程序运行时的进程号,上面告诉我们错误发生的位置，在 main()函数调用 malloc 造成了内存泄漏.

3. 2 无效指针

Valgrind 也可以找出无效堆内存使用。比如，如果你用 malloc 或 new 分配了一个数组，并访问数组末端后面的内存：

```
#include <stdlib.h>
int main()
{
        char *x = malloc(10);
        x[10] = 'a';
        free(x);
        return 0;
}
```



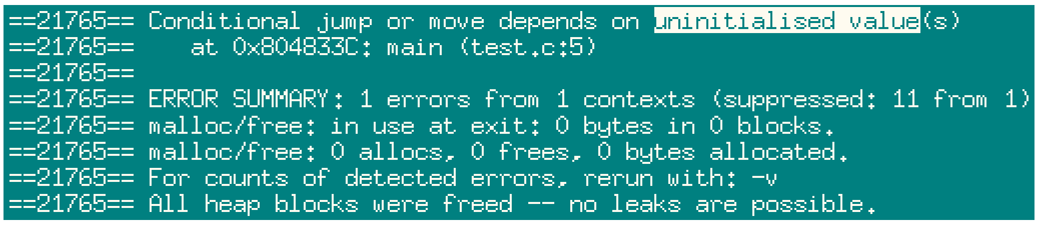
```
==21747== Invalid write of size 1
==21747==    at 0x8048382: main (test.c:5)
==21747==  Address 0x416B032 is 0 bytes after a block of size 10 alloc'd
==21747==    at 0x401A6CE: malloc (vg_replace_malloc.c:149)
==21747==    by 0x8048375: main (test.c:4)
==21747==
==21747== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 1)
==21747== malloc/free: in use at exit: 0 bytes in 0 blocks.
==21747== malloc/free: 1 allocs, 1 frees, 10 bytes allocated.
==21747== For counts of detected errors, rerun with: -v
==21747== All heap blocks were freed -- no leaks are possible.
```

这个信息表明我们分配了 10 字节的内存，但是访问了超出范围的内存，因此，我们就进行了一个´非法写´操作。如果试图从那块内存读取数据，我们就会得到´Invalid read of size X´的警告(X 是试图读取数据的大小，char 是一个字节，而 int 根据系统的不同可能是 2 个字节或 4 个字节)。通常，Valgrind 显示出函数调用栈信息以方便我们准确定位错误。

3.3 使用未初始化变量

Valgrind 甚至可以知道如果一个变量被赋予一个未初始化的变量，这个变量仍然处于"未初始化"状态.

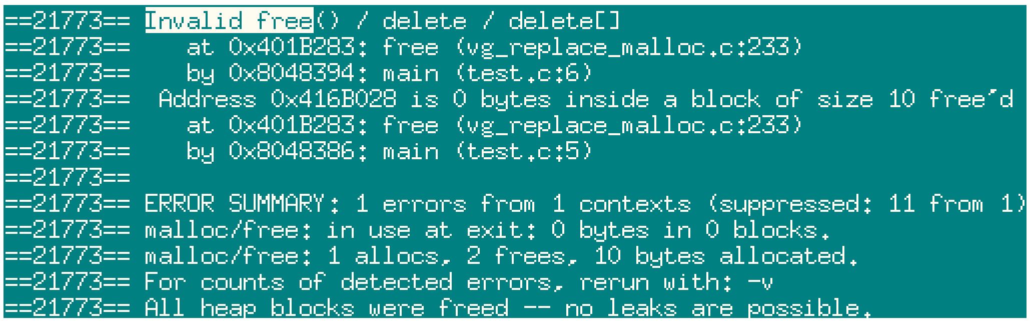
```
#include <stdio.h>
int main()
{
    int x;
    if(x == 0)
    {
        printf("X is zero"); /* replace with cout and include
        iostream for C++ */
    }
    return 0;
}
```



```
==21765== Conditional jump or move depends on uninitialised value(s)
==21765==    at 0x804833C: main (test.c:5)
==21765==
==21765== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 1)
==21765== malloc/free: in use at exit: 0 bytes in 0 blocks.
==21765== malloc/free: 0 allocs, 0 frees, 0 bytes allocated.
==21765== For counts of detected errors, rerun with: -v
==21765== All heap blocks were freed -- no leaks are possible.
```

3.4 重复释放

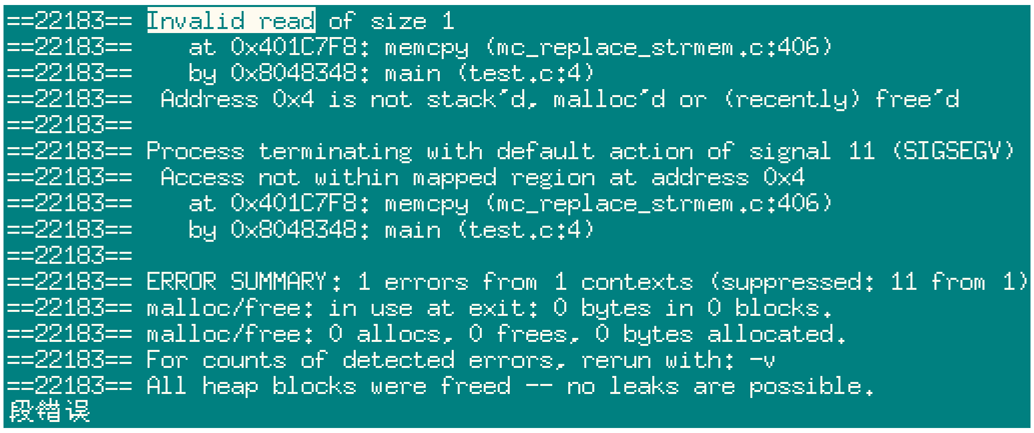
```
#include <stdlib.h>
int main()
{
    char *x = malloc(10);
    free(x);
    free(x);
    return 0;
}
```



```
==21773== Invalid free() / delete / delete[]
==21773==    at 0x401B283: free (vg_replace_malloc.c:233)
==21773==    by 0x8048394: main (test.c:6)
==21773==  Address 0x416B028 is 0 bytes inside a block of size 10 free'd
==21773==    at 0x401B283: free (vg_replace_malloc.c:233)
==21773==    by 0x8048386: main (test.c:5)
==21773==
==21773== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 1)
==21773== malloc/free: in use at exit: 0 bytes in 0 blocks.
==21773== malloc/free: 1 allocs, 2 frees, 10 bytes allocated.
==21773== For counts of detected errors, rerun with: -v
==21773== All heap blocks were freed -- no leaks are possible.
```

3.5 读/写不适当的栈中内存块

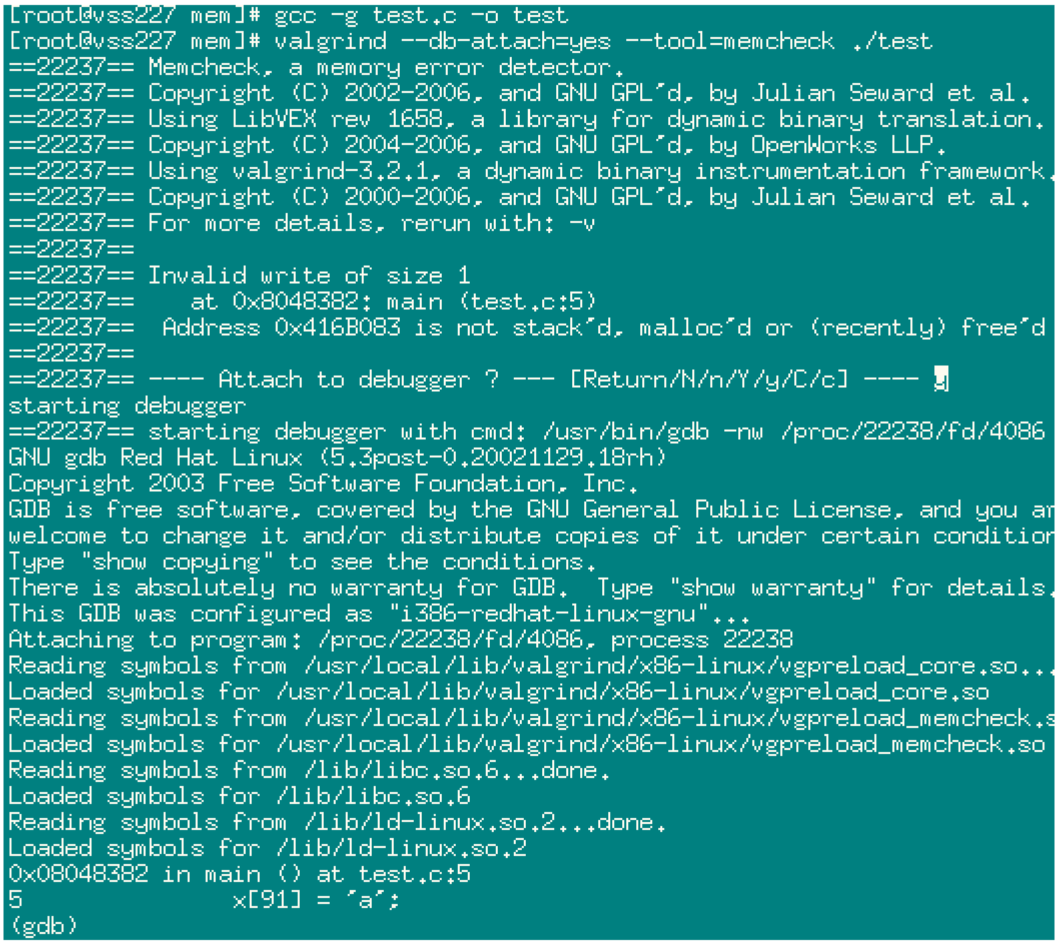
```
#include <stdlib.h>
int main()
{
        memcpy(0x8000,0,5);
        return 0;
}
```



```
==22183== Invalid read of size 1
==22183==    at 0x401C7F8: memcpy (mc_replace_strmem.c:406)
==22183==    by 0x8048348: main (test.c:4)
==22183==  Address 0x4 is not stack'd, malloc'd or (recently) free'd
==22183==
==22183== Process terminating with default action of signal 11 (SIGSEGV)
==22183==  Access not within mapped region at address 0x4
==22183==    at 0x401C7F8: memcpy (mc_replace_strmem.c:406)
==22183==    by 0x8048348: main (test.c:4)
==22183==
==22183== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 1)
==22183== malloc/free: in use at exit: 0 bytes in 0 blocks.
==22183== malloc/free: 0 allocs, 0 frees, 0 bytes allocated.
==22183== For counts of detected errors, rerun with: -v
==22183== All heap blocks were freed -- no leaks are possible.
段错误
```

3.6 绑定调试器

```
#include <stdlib.h>
int main()
{
        char *x = malloc(10);
        x[91] = 'a';
        free(x);
        return 0;
}
```



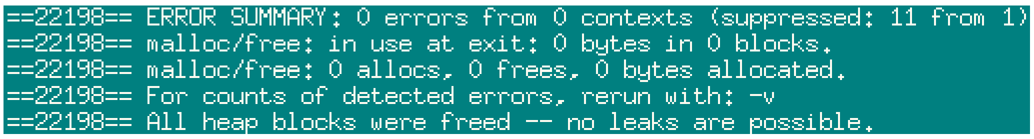
```
[root@vss227 mem]# gcc -g test.c -o test
[root@vss227 mem]# valgrind --db-attach=yes --tool=memcheck ./test
==22237== Memcheck, a memory error detector.
==22237== Copyright (C) 2002-2006, and GNU GPL'd, by Julian Seward et al.
==22237== Using LibVEX rev 1658, a library for dynamic binary translation.
==22237== Copyright (C) 2004-2006, and GNU GPL'd, by OpenWorks LLP.
==22237== Using valgrind-3.2.1, a dynamic binary instrumentation framework.
==22237== Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
==22237== For more details, rerun with: -v
==22237==
==22237== Invalid write of size 1
==22237==    at 0x8048382: main (test.c:5)
==22237==  Address 0x416B083 is not stack'd, malloc'd or (recently) free'd
==22237==
==22237== ---- Attach to debugger ? --- [Return/N/n/Y/y/C/c] ---- y
starting debugger
==22237== starting debugger with cmd: /usr/bin/gdb -nw /proc/22238/fd/4086
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you ar
welcome to change it and/or distribute copies of it under certain conditior
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
Attaching to program: /proc/22238/fd/4086, process 22238
Reading symbols from /usr/local/lib/valgrind/x86-linux/vgpreload_core.so...
Loaded symbols for /usr/local/lib/valgrind/x86-linux/vgpreload_core.so
Reading symbols from /usr/local/lib/valgrind/x86-linux/vgpreload_memcheck.s
Loaded symbols for /usr/local/lib/valgrind/x86-linux/vgpreload_memcheck.so
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
0x08048382 in main () at test.c:5
5               x[91] = 'a';
(gdb)
```

4 Valgrind 不能查出哪些错误

Valgrind 不对静态数组(分配在栈上)进行边界检查。如果在程序中声明了一个数组:

```
int main()
{
    char x[10];
    x[11] = 'a';
}
```

Valgrind 则不会警告你:



```
==22198== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 11 from 1)
==22198== malloc/free: in use at exit: 0 bytes in 0 blocks.
==22198== malloc/free: 0 allocs, 0 frees, 0 bytes allocated.
==22198== For counts of detected errors, rerun with: -v
==22198== All heap blocks were freed -- no leaks are possible.
```

5 其他功能

5.1 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。

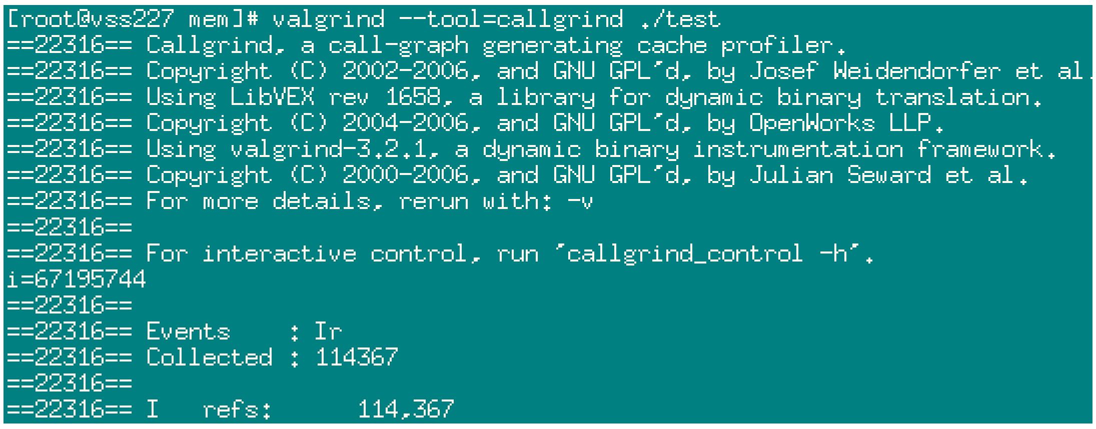
5.2 Callgrind

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。

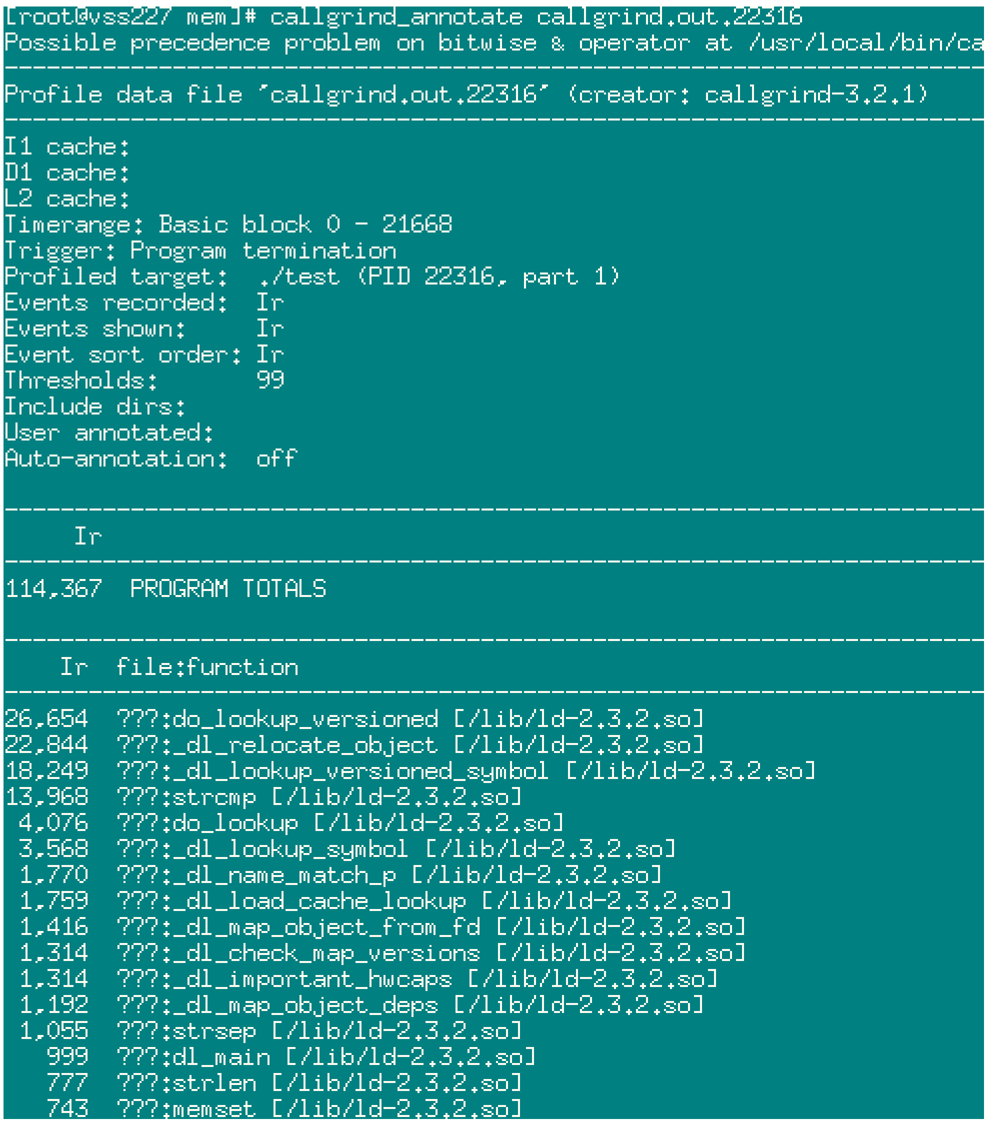
```
#include <stdlib.h>
#include <stdio.h>
void f(void)
{

    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;
}
```



```
[root@vss227 mem]# valgrind --tool=callgrind ./test
==22316== Callgrind, a call-graph generating cache profiler.
==22316== Copyright (C) 2002-2006, and GNU GPL'd, by Josef Weidendorfer et al.
==22316== Using LibVEX rev 1658, a library for dynamic binary translation.
==22316== Copyright (C) 2004-2006, and GNU GPL'd, by OpenWorks LLP.
==22316== Using valgrind-3.2.1, a dynamic binary instrumentation framework.
==22316== Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
==22316== For more details, rerun with: -v
==22316==
==22316== For interactive control, run 'callgrind_control -h'.
i=67195744
==22316==
==22316== Events    : Ir
==22316== Collected : 114367
==22316==
==22316== I   refs:      114,367
```



```
[root@vss227 mem]# callgrind_annotate callgrind.out.22316
Possible precedence problem on bitwise & operator at /usr/local/bin/ca
--------------------------------------------------------------------------------
Profile data file 'callgrind.out.22316' (creator: callgrind-3.2.1)
--------------------------------------------------------------------------------
I1 cache:
D1 cache:
L2 cache:
Timerange: Basic block 0 - 21668
Trigger: Program termination
Profiled target:  ./test (PID 22316, part 1)
Events recorded:  Ir
Events shown:     Ir
Event sort order: Ir
Thresholds:       99
Include dirs:
User annotated:
Auto-annotation:  off

--------------------------------------------------------------------------------
     Ir
--------------------------------------------------------------------------------
114,367  PROGRAM TOTALS

--------------------------------------------------------------------------------
    Ir  file:function
--------------------------------------------------------------------------------
26,654  ???:do_lookup_versioned [/lib/ld-2.3.2.so]
22,844  ???:_dl_relocate_object [/lib/ld-2.3.2.so]
18,249  ???:_dl_lookup_versioned_symbol [/lib/ld-2.3.2.so]
13,968  ???:strcmp [/lib/ld-2.3.2.so]
 4,076  ???:do_lookup [/lib/ld-2.3.2.so]
 3,568  ???:_dl_lookup_symbol [/lib/ld-2.3.2.so]
 1,770  ???:_dl_name_match_p [/lib/ld-2.3.2.so]
 1,759  ???:_dl_load_cache_lookup [/lib/ld-2.3.2.so]
 1,416  ???:_dl_map_object_from_fd [/lib/ld-2.3.2.so]
 1,314  ???:_dl_check_map_versions [/lib/ld-2.3.2.so]
 1,314  ???:_dl_important_hwcaps [/lib/ld-2.3.2.so]
 1,192  ???:_dl_map_object_deps [/lib/ld-2.3.2.so]
 1,055  ???:strsep [/lib/ld-2.3.2.so]
   999  ???:dl_main [/lib/ld-2.3.2.so]
   777  ???:strlen [/lib/ld-2.3.2.so]
   743  ???:memset [/lib/ld-2.3.2.so]
```

5.3 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。

5.4 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。