应用 Valgrind 发现 Linux 程序的内存问题

作 / BIG PHOENIX 分享会主持兼讲师
腾讯技术工程师 - 王宇

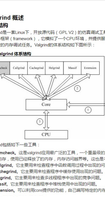
PHOENIX CLUB

↑ 回目录

Valgrind 概述

Valgrind 是一款用于构建动态分析工具的工具集。Valgrind 的作用（core）以及基于其构建的各种工具（tool）。内核类似于一个框架（framework），它模拟了一个CPU环境，并提供很多其他的工具，用来在工具实现的子进程（plug-in）。利用内核提供的调用来检查和处理代码的指令流。Valgrind的体系结构如下图所示：

图 1 Valgrind 体系结构



Valgrind的核心功能工作主要：

1. **Memcheck**，这是最常用的工具，主要用来检测以下几种常见的内存问题：
   - **非法内存访问**：当一个应用程序访问它不该访问的内存区域时，这种情况可能发生在：访问已经被释放的内存区域；访问一个缓冲区的末尾；或在一个应用程序的栈帧内读写内存。
   - **使用未初始化的值**：这种错误通常是由于应用程序使用栈上的内存，或调用malloc()分配内存，或调用new操作符来分配内存，但是在使用这些内存之前，没有对其进行初始化。
   - **内存泄漏**：这种错误通常是由于应用程序没有释放掉动态分配内存，例如忘记了申请内存的指针的内容。
   - **不匹配的内存释放**：这种错误通常是由于应用程序使用malloc()或new来分配内存，但是随后调用free()或delete来释放内存。
2. **Cachegrind**，它主要用来检测高速缓存的使用情况。
3. **Helgrind**，它主要用来检查多线程程序中的竞争条件。
4. **Massif**，它主要用来检查程序中堆栈使用情况的信息。
5. **Lackey**，可以用来做一些用户自定义的检查，由工具的作者来决定工具。

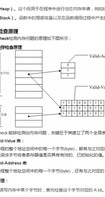
Valgrind的核心功能工作原理：

1. **Memcheck**，这是最常用的工具，主要用来检测以下几种常见的内存问题：
   - **非法内存访问**：当一个应用程序访问它不该访问的内存区域时，这种情况可能发生在：访问已经被释放的内存区域；访问一个缓冲区的末尾；或在一个应用程序的栈帧内读写内存。
   - **使用未初始化的值**：这种错误通常是由于应用程序使用栈上的内存，或调用malloc()分配内存，或调用new操作符来分配内存之后，没有对其进行初始化。
   - **内存泄漏**：这种错误通常是由于应用程序没有释放掉动态分配内存，例如忘记了申请内存的指针的内容。
   - **不匹配的内存释放**：这种错误通常是由于应用程序使用malloc()或new来分配内存，但是随后调用free()或delete来释放内存。
2. **线程（Thread）**，这个线程存在程序中的子进程的内存映射，例如栈和堆的区域。一个线程可能会创建一个或多个线程。
3. **核（Core）**，这个核存在程序中的子进程的内存映射，例如栈和堆的区域。core是用户自定义的核心。
4. **代码块（Block）**，这种内存映射表示连续的基本程序指令流的内存映射区域。

内存检查原理

Memcheck检测内存问题的原理如下图所示：

图 2 内存检查原理



Memcheck 最新的版本是3.6.0版本，本章节所讲解的内容，主要是基于该版本。

1. **Valid-Value 位**

对于通过操作内存数据的寄存器或内存的每一个字节(Byte)，都有与之对应的 8 个 bits，对于 CPU 的每个寄存器，也有一个与之对应的 32 位 bits。这些 bits 被用来计算在这个字节或者寄存器值是否有意义的情况。

2. **Valid-Address 位**

对于通过每一个字节内存数据的内存地址，还有与之对应的 1 个 bit，也表示该字节是否可以被读写。

检测原理：

- 当要读写内存中某个字节时，首先检查这个字节对应的 A 位。如果该 A 位显示此位是无效的话，memcheck 则报告该错误。
- 对这一位（bit）的检查是由底层的合成指令来完成的，这些合成指令被插入到由 Valgrind 核心翻译成等效指令的代码流中。这些新插入的指令，能够保证程序的执行速度，并且不会导致程序崩溃。

这个程序调用 memcheck，于是它以这样开始：**valgrind +其他参数**。

第三步：同样，这个程序也会崩溃，但是这次会有一些有用的输出。

以下是这样之后会发生的情况：

```
[bin]$valgrind ./memtest
==31337== Memcheck, a memory error detector
==31337== Copyright (C) 2002-2006, and GNU GPL'd, by Julian Seward et al.
==31337== Using valgrind-3.2.0, a dynamic binary instrumentation framework.
==31337== Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
==31337== For more details, rerun with: -v
==31337==
==31337== Invalid write of size 4
==31337==    at 0x8048394: main (memtest.c:5)
==31337==  Address 0x1bd48050 is 0 bytes after a block of size 40 alloc'd
==31337==    at 0x1B900DEC: malloc (vg_replace_malloc.c:149)
==31337==    by 0x80483F1: main (memtest.c:3)
==31337==
==31337== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 3 from 1)
==31337== malloc/free: in use at exit: 40 bytes in 1 blocks.
==31337== malloc/free: 1 allocs, 0 frees, 40 bytes allocated.
==31337== For counts of detected errors, rerun with: -v
==31337== searching for pointers to 1 bytes at 0x1BE99F7B.
==31337== checked 76,708 bytes.
==31337==
==31337== LEAK SUMMARY:
==31337==    definitely lost: 40 bytes in 1 blocks.
==31337==    indirectly lost: 0 bytes in 0 blocks.
==31337==      possibly lost: 0 bytes in 0 blocks.
==31337==    still reachable: 0 bytes in 0 blocks.
==31337==         suppressed: 3,084 bytes in 1 blocks.
==31337== Rerun with --leak-check=full to see details of leaked memory.
```

- 在这里的这段输出中，以空格开头的信息是 valgrind 的信息。
- 这里以这段输出的第一行（以空格开头的信息）为例，这段输出的含义是：
  - 中间的这个进程号是 valgrind 进程运行时用的进程号，发现的内存问题，通过编译选项选择，可以发现：
  - 这是一个对内存的写操作，发现写操作的内存地址是 0x1bd48050，
  - 发现该操作的长度是 4 个 bytes，
  - 发现该操作的调用栈信息，以及程序的行号信息。
- 这段输出中的其他信息，以及其他的行的含义，可以参考上面的解释。

- 关于更多的工具使用方法的说明，请参考Valgrind的官方网站。

- 对内存泄漏的检查，这种错误一般会通过命令选项的方式来实现，这里以下面的方式进行演示：

```
valgrind --leak-check=yes ./memtest
```

- 对内存泄漏的检查，这种错误一般会通过命令选项的方式来实现，内存泄漏的方式：-l 或 --leak-check 的缩写形式。

- 对内存泄漏的检查原理，请参考Valgrind的官方网站。

利用Memcheck发现常见的内存问题

在Linux平台开发应用程序时，最常遇见的问题是编译时出现内存错误，例如出现了常见下为内存错误类型提醒，并给出了如何用valgrind来检测程序。

内存泄漏

对于这个程序来说，这段程序是正常的，并没有编译不同的问题，也不会出现内存错误。

下面的程序是有一个内存泄漏的问题，并没有编译不同的问题，也不会出现内存错误。

```
[bin]$valgrind ./memtest
==31337== Memcheck, a memory error detector
==31337== Copyright (C) 2002-2006, and GNU GPL'd, by Julian Seward et al.
==31337== Using valgrind-3.2.0, a dynamic binary instrumentation framework.
==31337== Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
==31337== For more details, rerun with: -v
==31337==
==31337== Invalid write of size 4
==31337==    at 0x8048394: main (memtest.c:5)
==31337==  Address 0x1bd48050 is 0 bytes after a block of size 40 alloc'd
==31337==    at 0x1B900DEC: malloc (vg_replace_malloc.c:149)
==31337==    by 0x80483F1: main (memtest.c:3)
==31337==
==31337== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 3 for 1)
==31337== malloc/free: in use at exit: 40 bytes in 1 blocks.
==31337== malloc/free: 1 allocs, 0 frees, 40 bytes allocated.
==31337== For counts of detected errors, rerun with: -v
==31337== searching for pointers to 1 bytes at 0x1BE99F7B.
==31337== checked 76,708 bytes.
==31337==
==31337== LEAK SUMMARY:
==31337==    definitely lost: 40 bytes in 1 blocks.
==31337==    indirectly lost: 0 bytes in 0 blocks.
==31337==      possibly lost: 0 bytes in 0 blocks.
==31337==    still reachable: 0 bytes in 0 blocks.
==31337==         suppressed: 3,084 bytes in 1 blocks.
==31337== Rerun with --leak-check=full to see details of leaked memory.
```

- 在这里的这段输出中，以空格开头的信息（以空格开头的信息）为例，这段输出的含义是：
  - 中间的这个进程号是 valgrind 进程运行时用的进程号，发现的内存泄漏，通过编译选项选择，可以发现：
  - 这是一个对内存的写操作，发现写操作的内存地址是 0x1bd48050，
  - 发现该操作的长度是 4 个 bytes，
  - 发现该操作的调用栈信息，以及程序的行号信息。
- 这段输出中的其他信息，以及其他的行的含义，可以参考上面的解释。

内存读写错误

问题原理：

- 这种错误一般会通过编译选项的方式来实现，这种编译选项的方式，也是为了帮助程序员能够在程序的内存方面，不会出现内存错误。
- 这种编译选项的方式，也是为了帮助程序员能够在程序的内存方面，不会出现内存泄漏的问题。
- 这种编译选项的方式，也是为了帮助程序员能够在程序的内存方面，不会出现内存读写错误的问题。

问题现象：

- 这种错误一般会通过编译选项的方式来实现，这种编译选项的方式，也是为了帮助程序员能够在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题现象：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题原因：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题解决：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题现象：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题原因：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题解决：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

内存泄漏

问题原理：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题现象：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题原因：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

问题解决：

- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。
- 这种编译选项的方式，也是为了帮助程序员能在程序的内存方面，不会出现内存读写错误的问题。

内存泄漏

内存泄漏（Memory leak）指的是一段程序中动态申请的内存，在使用完毕后因为疏忽，又无法被释放掉的情况。这种情况会使内存的总量不断增加，最终会导致内存耗尽，程序崩溃。

下面是一个内存泄漏的程序，每个人的内存泄漏情况都会不同，但是本章节下面的内存泄漏情况，是以下面的情况：

- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。
- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。
- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。

在这一个例子中，每个人的内存泄漏情况都会不同，但是本章节下面的内存泄漏情况，是以下面的情况：

- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。
- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。
- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。

总结

- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。
- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。
- 这段程序是一个内存泄漏的程序，这段程序是一个内存泄漏的程序。