

The Valgrind Quick Start Guide

Release 3.2.1 16 September 2006

Copyright © 2000-2006 Valgrind Developers

Email: valgrind@valgrind.org

The Valgrind Quick Start Guide

1. Introduction

The Valgrind distribution has multiple tools. The most popular is the memory checking tool (called Memcheck) which can detect many common memory errors such as:

-

It's worth fixing errors in the order they are reported, as later errors can be caused by earlier errors.

Memory leak messages look like this:

```
==19182== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
```

Valgrind User Manual

Release 3.2.1 16 September 2006
Copyright ©

3.5. Details of Memcheck's checking machinery

1. Introduction

1.1. An Overview of Valgrind

3.

2. Using and understanding the Valgrind core

- Next line: a small number of suppression types have extra information after the second line (eg. the Param suppression for Memcheck)
-

`--trace-children=<yes|no> [default t: no]`

When enabled, Valgrind will trace into child processes. This can be confusing and isn't usually what you want, so it is disabled by default.

`--track-fds=<yes|no> [default t: no]`

When enabled, Valgrind will print out a list of open file descriptors on exit. Along with each file descriptor is printed a stack backtrace of where the file was opened and any details relating to the file descriptor such as the file name or socket details.

`--time-stamp=<yes|no> [default t: no]`

When enabled, each message is preceded with an indication of the elapsed wallclock time since startup, expressed as days, hours, minutes, seconds and milliseconds.

`--log-fd=<number> [default t: 2, stderr]`

`--input-fd=<number> [default: 0, stdin]`

When using `--db-attach=yes` and `--gen-suppressions=yes`, Valgrind will stop so as to read keyboard input from you, when each error occurs. By default it reads from the standard input (stdin), which is problematic for programs which close stdin. This option allows you to specify an alternative file descriptor from which to read input.

`--max-stackframe 0, stdin]`

The maximum size of a stack frame - if the stack pointer moves by more than this amount then Valgrind will assume that the program is switching to a different stack.

You may use this option (if your program has a large stack) to specify a file descriptor from which to read input. This option allows you to specify an alternative file descriptor from which to read input.

stac(.,)-068(this)-960heuristic will, and will is of i stack

This option allows you to the tpreholdn to a different


```
#include <stdio.h>
#include "valgrind.h"
```


Read the [Valgrind FAQ](#) for more advice about common problems, crashes, etc.

2.13. Limitations

The following list of limitations seems long. However, most programs actually work fine.

Valgrind will run Linux ELF binaries, on a kernel 2.4.X or 2.6.X system, on the x86, amd64, ppc32 and ppc64 architectures, subject to the following constraints:

- On x86 and amd64, there is no support for 3DNow! instructions. If the translator encounters these, Valgrind

- As of version 3.2.0, Valgrind has the following limitations in its implementation of PPC32 and PPC64 floating point arithmetic, relative to IEEE754.

Scalar (non-AltiVec): Valgrind provides a bit-exact emulation of all floating point instructions, except for "fre" and "fres", which are done more precisely than required by the PowerPC architecture specification. All floating point operations observe the current rounding mode.

However, fpscr[FPRF] is not set after each operation. That could be done but would give measurable performance overheads, and so far no need for it has been found.

- Warning: client switching stacks?

Valgrind spotted such a large change in the stack pointer, `%esp`, that it guesses the client is switching to a different stack. At this point it makes a kludgy guess where the base of the new stack is, and sets memory permissions

If it says . . . no, your `mpi cc` has failed to compile and link a test MPI2 program.

If the configure test succeeds, continue in the usual way with `make` and `make install`. The final install tree should then contain `libmpi.wrap.so`.

The wrappers should reduce Memcheck's false-error rate on MPI applications. Because the wrapping is done at the MPI interface, there will still potentially be a large number of errors reported in the MPI implementation below the interface. The best you can do is try to suppress them.

You may also find that the input-side (buffer length/definedness) checks find errors in your MPI use, for example passing too short a buffer to `MPI_Recv`.

Functions which are not wrapped may increase the false error rate. A possible approach is to run with

--undef-value-errors=<yes|no> [default: yes]

Controls whether memcheck detects dangerous uses of undefined value errors. When yes, Memcheck behaves like Addrcheck, a lightweight memory-checking tool that used to be part of Valgrind, which didn't detect undefined value errors. Use this option if you don't like seeing undefined value errors.

3.3. Explanation of error messages from Memcheck

Despite considerable sophistication under the hood, Memcheck can only really detect two kinds of errors: use of illegal addresses, and use of undefined values. Nevertheless, this is enough to help you discover all sorts of memory-

Memcheck: a heavyweight memory checker

3.3.4. When a block is freed with an inappropriate deallocation function

In the following example, a block allocated with `new[]` has wrongly been deallocated with `free`:

```
Mismatched free() / delete / delete []
at 0x40043249: free (vg_clientfuncs.c: 171)
by 0x4102BB4E: QGArray::~QGArray(void) (tools/qgarray.cpp: 149)
by 0x4C261C41: PptDoc::~PptDoc(void) (include/qmemarray.h: 60)
by 0x4C261F0E: PptXml::~PptXml(void) (pptxml.cc: 44)
Address 0x4BB292A8 is 0 bytes inside a block of size 64 alloc'd
at 0x4004318C: __builtin_vec_new (vg_clientfuncs.c: 152)
by 0x4C21BC15: KLaola::readSbStream(int) const (klaola.cc: 314)
by 0x4C21C155: KLaola::stream(KLaola::OLENo8.4879-o7e0-11.9551Td[by]-426(0x4C261Cm009(0x
```

After the system call, Memcheck updates its tracked information to precisely reflect any changes in memory permissions caused by the system call.

Here's an example of two system calls with invalid parameters:

```
#include <stdlib.h>
#include <unistd.h>
int main(void)
{
    char* arr = malloc(10);
    int* arr2 = malloc(sizeof(int));
    write(1/ * stdout */, arr, 10);
    exit(arr2[0]);
}
```


Memcheck: a heavyweight memory checker

3.6. Client Requests

The following client requests are defined in `memcheck.h`. See `memcheck.h` for exact details of their arguments.

- `VALGRIND_MAKE_MEM_NOACCESS`, `VALGRIND_MAKE_MEM_UNDEFINED` and `VALGRIND_MAKE_MEM_DEFINED`. These mark address ranges as completely inaccessible, accessible but containing undefined data, and accessible and containing defined data, respectively. Subsequent errors may have their faulting addresses described in terms of these blocks. Returns a "block handle". Returns zero when not run on Valgrind.
- `VALGRIND_MAKE_MEM_DEFINED_IF_ADDRESSABLE`. This is just like `VALGRIND_MAKE_MEM_DEFINED`

4. Cachegrind: a cache profiler

Detailed technical documentation on how Cachegrind works is available in [How Cachegrind works](#). If you only want to know how to

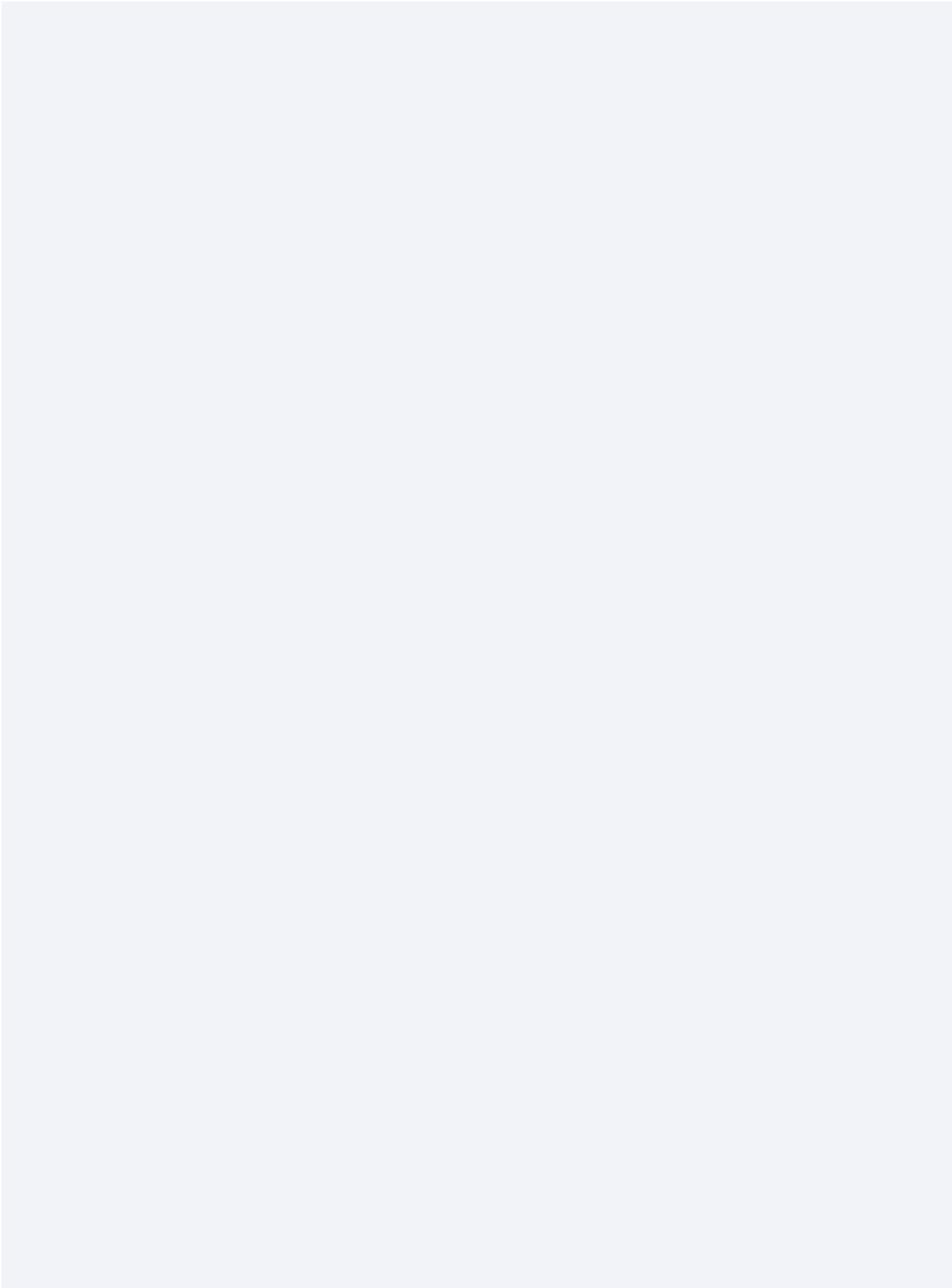
The `.pid` suffix on the output file name serves two purposes. Firstly, it means you don't have to rename old log files that you don't want to overwrite. Secondly, and more importantly, it allows correct profiling with the `--trace-children=yes` option of programs that spawn child processes.

4.2.2. Cachegrind options

Manually specifies the I1/D1/L2 cache configuration, where `size` and `line_size` are measured in bytes. The three items must be comma-separated, but with no spaces, eg:

```
valgrind --tool=cachegrind --l1=65535,2,64
```

You can specify one, two or three of the I1/D1/L2 caches. Any `l1` must



- -h, --hel p
- v, --versi on

Help and version, as usual.

- --sort=A, B, C [default: order in cachegri nd. out. pi d]

4.3.2. Things to watch out for

Some odd things that can occur during annotation:

Page 1 of 1

if you profile with `--skip-plt=no`. If a call is ignored, cost events happening will be attached to the enclosing function.

If you have a recursive function, you can distinguish the first 10 recursion levels by specifying `--fn-recursion10=funcprefix`. Or for all functions with `--fn-recursion=10`, but this will give you much bigger profile data files. In the profile data, you will see the recursion levels of "func" as the different functions with names "func", "func'2", "func'3" and so on.

If you have call chains "A > B > C" and "A > C > B" in your program, you usually get a "false" cycle "B <> C". Use

--compress-strings=<no|yes> [default: yes]

This option influences the output format of the profile data. It specifies whether strings (file and function names)

`--collect-atstart=<yes|no> [default: yes]`
Specify whether event collection is switched on at beginning of the profile run.

To only look at parts of your program, you have two possibilities:

- 1.

6.2.3. Spacetime Graphs

At this level, we can see all the places from which `_nl_load_local_e_from_archive()` was called such that it allocated memory at `0x401767D0`. (We can click on the top 22.1% link to go back to the parent entry.) At this

--heap-admin=<number> [default: 8]

The number of admin bytes per block to use. This can only be an estimate of the average, since it may vary. The allocator used by

7. Helgrind: a data-race detector

Valgrind FAQ

Release 3.2.1 16 September 2006

Copyright © 2000-2006 Valgrind Developers

Email: valgrind@valgrind.org

3. Valgrind aborts10-Expectedlys

4. Valgrind behaves unexpectedly

- 4.1. My program uses the C++ STL and string classes. Valgrind reports 'still reachable' memory leaks involving these classes at the exit of the program, but there should be none.

First of all: relax, it's probably not a bug, but a feature. Many implementations of the C++ standard libraries use their own memory pool allocators. Memory for quite a number of destructed objects is not immediately freed and given back to the OS, but kept in the pool(s) for later re-use. The fact that the pools are not freed at the exit() of the program cause Valgrind to report this memory as still reachable. The behaviour not to free pools at the exit() could be called a bug of the library though.

Using gcc, you can force the STL to use malloc and to free memory as soon as possible by globally disabling

Valgrind Technical Documentation

Release 3.2.1 16 September 2006

Copyright © 2000-2006 Valgrind Developers

Email: valgrind@valgrind.org

becomes "trapped" in `valgrind.so` and the translations it generates. The synthetic CPU provided by Valgrind does, however, return from this initialisation function. So the normal startup actions, orchestrated by the dynamic linker `ld.so`, continue as usual, except on the synthetic CPU, not the real one. Eventually

- Valgrind runs in the same namespace as the client, at least from `ld.so`'s point of view, and it therefore absolutely

- The **addresses** of various helper routines called from generated code: `VG_(helper_val ue_check4_fail)`, `VG_(helper_val ue_check0_fail)`, which register V-check failures, `VG_(helper_perc_STOREV4)`, `VG_(helper_perc_STOREV1)`, `VG_(helper_perc_LOADV4)`, `VG_(helper_perc_LOADV1)`, which do stores and loads of V bits to/from the sparse array which keeps track of V bits in memory, and `VGM_(handle_esp_assignment)`

UCode is, more or less, a simple two-address RISC-like code. In keeping with the x86 AT&T assembly syntax, generally speaking the first operand is the source operand, and the second is the destination operand, which is modified when the uinstr is notionally executed.

UCode instructions have up to three operand fields, each of which has a corresponding Tag describing it. Possible values for the tag are:

-

•

- (UNARY) Pessimising casts:

The effect of these transformations on our short block is rather unexciting, and shown below. On longer basic blocks they can dramatically improve code quality.

Instrumented code:

```

0: GETVL    %EDX, q0
1: GETL     %EDX, t0

2: TAG1o    q0 = Left4 ( q0 )
3: INCL     t0

4: PUTVL    q0, %EDX
5: PUTL     t0, %EDX

6: TESTVL   q0
7: SETVL    q0
8: LOADVB   (t0), q0
9: LDB      (t0), t0

10: TAG1o    q0 = SWiden14 ( q0 )
11: WIDENL_Bs t0

12: PUTVL    q0, %EAX
13: PUTL     t0, %EAX

14: GETVL    %ECX, q8
15: GETL     %ECX, t8

16: MOVL     q0, q4
17: SHLL     $0x1, q4
18: TAG2o    q4 = UfU4 ( q8, q4 )
19: TAG1o    q4 = Left4 ( q4 )
20: LEA2L    1(t8, t0, 2), t4

21: TESTVL   q4
22: SETVL    q4
23: LOADVB   (t4), q10
24: LDB      (t4), t10

25: SETVB    q12
26: MOVB     $0x20, t12

27: MOVL     q10, q14
28: TAG2o    q14 = ImproveAND1_T0 ( t10, q14 )
29: TAG2o    q10 = UfU1 ( q12, q10 )
30: TAG2o    q10 = DiFD1 ( q14, q10 )
31: MOVL     q12, q14
32: TAG2o    q14 = ImproveAND1_T0 ( t12, q14 )
33: TAG2o    q10 = DiFD1 ( q14, q10 )
34: MOVL     q10, q16
35: TAG1o    q16 = PCast10 ( q16 )
36: PUTVFO   q16
37: ANDB     t12, t10 (-wOSZACP)

38: INCEIPo  $9

39: GETVFO   q18
40: TESTVo   q18
41: SETVo    q18
42: Jnzo     $0x40435A50 (-rOSZACP)

43: JMPo     $0x40435A5B

```

1.2.9. UCode post-instrumentation cleanup

This pass, coordinated by `vg_cleanup()`


```

at 29: del ete Ui fU1 due to defd arg1
at 32: change ImproveAND1_TQ to MOV due to defd arg2
at 41: del ete SETV
at 31: del ete MOV
at 25: del ete SETV
at 22: del ete SETV
at 7: del ete SETV

```

```

0: GETVL    %EDX, q0
1: GETL     %EDX, t0

```

```

2: TAG1o    q0 = Left4 ( q0 )
3: INCL     t0

```

```

4e %ED1
    tX, %ED1

```

```

ESETVL
q0 = ( q0 )
1: t0

```

```

%AX1
    tX, %AX0

```

```

GETVL     %CDX, 81
GETL      %CDX,

```

```

7:
82: TA21o    40 = Ui ft4 ( 8X, )
    TAG1o    40 = Left4 ( 40 )

```

```

t8, (tX2(X, )-426(40)]TJ0-23. 9154Td[212(X,-426(40)]t

```

```

62:

```

```

    TA21o    140 = ImproveAND1_TQ ( )
02: TA21o    1q0 = ( 14X, 1q0 )
22:
    TA21o    1q0 = ( 14X, 1q0 )
42:
52: TAG1o    161 = , )

```

```

wOSZACP()) ]TJ0-23. 9103Td[383: I NEI P$21

```

```

90: GETFoL
ESEToL

```

```

rOSZACP()) ]TJ0-23. 9154Td[432:

```

1.2.10. Translation from UCode

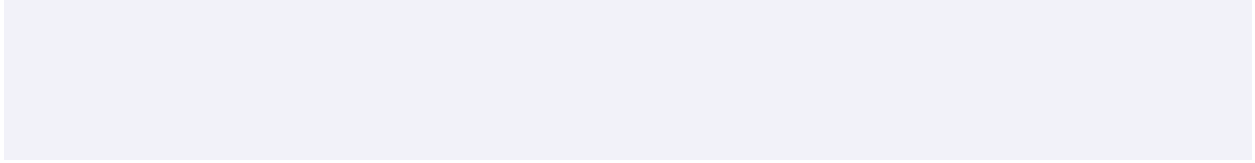
This is all very simple, even though `vg_from_ucose.c` is a big file. Position-independent x86 code is generated into a dynamically allocated array `emitted_code`; this is doubled in size when it overflows. Eventually `th-269(si4rg0.T-240.22526`

4. Errors, error contexts, error reporting, suppressions
5. Client malloc/free
6. Low-level memory management
7. A and V bitmaps
8. Symbol table management
9. Dealing with system calls

It would be great if Valgrind was ported to FreeBSD and x86 NetBSD, and to x86 OpenBSD, if it's possible (doesn't OpenBSD use a.out-style executables, not ELF ?)

The main difficulties, for an x86-ELF platform, seem to be:

- You'd need to rewrite the `/proc/self/maps` parser (`vg_procsel_fmaps.c`). Easy.
- You'd need to rewrite `vg_syscall_mem.c`, or, more specifically, provide one for your OS. This is tedious, but you can implement syscalls on demand, and the Linux kernel interface is, for the most part, going to look very similar to the *BSD interfaces, so it's really a copy-paste-and-modify-on-demand job. As part of this, you'd need supply a new



```
void fooble ( void )
{
    int a[10];
    int b[10];
    a[10] = 99;
}
```

Now imagine rewriting this as

```
void fooble ( void )
{
    int spacer0;
    int a[10];
    int spacer1;
    int b[10];
    int spacer2;
    VALGRIND_MAKE_NOACCESS(&spacer0, sizeof(int));
    VALGRIND_MAKE_NOACCESS(&spacer1, sizeof(int));
    VALGRIND_MAKE_NOACCESS(&spacer2, sizeof(int));
    a[10] = 99;
}
```

Now the invalid write is certain to hit `spacer0` or `spacer1`, so Valgrind will spot the error.

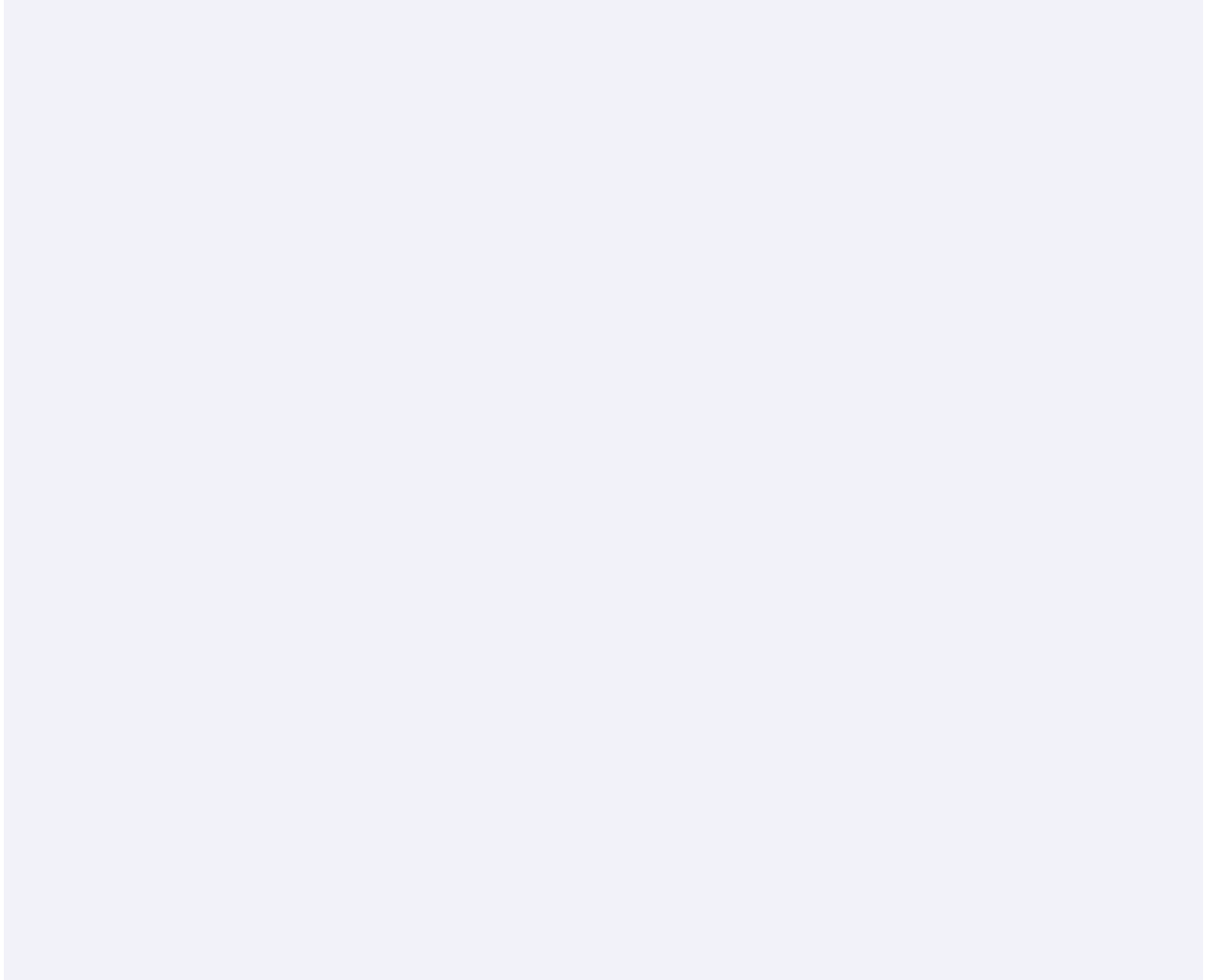
There are two complications.

1. The first is that we don't want to annotate sources by hand, so the Right Thing to do is to write a C/C++

2. How Cachegrind works

2.1. Cache profiling

[Note: this document is now very old, and a lot of its contents are out of date, and misleading.]



Function "func1" is located in "file1.c", the same as "main". Therefore, a "cfl=" specification for the call to "func1" is not needed. The function "func1" only consists of code at line 51 of "file1.c", where "func2" is called.

3.1.5. Name Compression

With the introduction of association specifications like calls it is needed to specify the same function or same file name multiple times. As absolute filenames or symbol names in C++ can be quite long, it is advantageous to be able to specify integer IDs for position specifications.

To support name compression, a position specification can be not only of the format "spec=name", but also "spec=(ID) name" to specify a mapping of an integer ID to a name, and "spec=(ID)" to reference a previously defined ID mapping. There is a separate ID mapping for each position specification, i.e. you can use ID 1 for both a file name and a symbol name.

With string compression, the example from 1.4 looks like this:

events: Instructions

```
fl=(1) file1.c
fn=(1) main
16 20
cfn=(2) func1
call s=1 50
16 400
cfl=(2) file2.c
cfn=(3) func2
call s=3 20
16 400

fn=(2)
51 100
cfl=(2)
cfn=(3)
call s=2 20
51 300

fl=(2)
fn=(3)
20 700
```

events: Instructions

```
# define file ID mapping
fl=(1) file1.c
fl=(2) file2.c
# define function ID mapping
fn=(1) main
```



```
TargetCommand : = "cmd: " Space* NoNewLineChar*
```


3.2.3. Description of Body Lines

There exist lines `spec=position`. The values for position specifications are arbitrary strings. When starting with "(" and a digit, it's a string in compressed format. Otherwise it's the real position string. This allows for file and symbol names as position strings, as these never start with "(" + *digit*. The compressed format is either "(" *number*

- **coverage tool:** Cachegrind can already be used for doing test coverage, but it's massive overkill to use it just for that.

It would be easy to write a coverage tool that records how many times each basic block was recorded. Again, the `cg_annotate`

For example, if a tool wants the core's help in recording and reporting errors, it must call `VG_(needs_tool_errors)` and provide definitions of eight functions for comparing errors, printing out errors, reading suppressions from a suppressions file, etc. While writing these functions requires some work, it's much less than doing error handling from scratch because the core is doing most of the work. See the function `VG_(needs_tool_errors)` in `include/pub_tool_tooliface.h` for full details of all the needs.

Third, the tool can indicate which events in core it wants to be notified about, using the functions `VG_(track_*)()`.

GDB may be able to give you useful information.
-fomit-frame-pointer

Note that by default most of the system is built with

The interface version number has the form X.Y. Changes in Y indicate binary compatible changes. Changes in X indicate binary incompatible changes. If the core and tool has the same major version number X they should work together. If X doesn't match, Valgrind will abort execution with an explanation of the problem.

This approach was chosen so that if the interface changes in the future, old tools won't work and the reason will be

Valgrind Distribution Documents

1. ACKNOWLEDGEMENTS

Cerion Armour-Brown, cerion@open-works.co.uk

Cerion worked on PowerPC instruction set support using the Vex dynamic-translation framework.

Jeremy Fitzhardinge, jeremy@valgrind.org

Jeremy wrote Helgrind and totally overhauled low-level syscall/signal and address space layout stuff, among many other improvements.

Tom Hughes, tom@valgrind.org

Tom did a vast number of bug fixes, and helped out with support for more recent Linux/glibc versions.

Frederic Gobry helped with autoconf and automake. Daniel Berlin modified readelf's dwarf2 source line reader, written by Nick Clifton, for use in Valgrind. Michael Matz and Simon Hausmann modified the GNU binutils demangler(s) for use in Valgrind.

And lots and lots of other people sent bug reports, patches, and very helpful feedback.

'--cache-file=FILE'

Use and save the results of the tests in FILE instead of
'./config.cache'. Set FILE to '/dev/null' to disable caching, for
debugging 'configure'.

'--help'

Print a summary of the options to 'configure', and exit.

'--quiet'

'--silent'

'-q'

Do not print messages saying which checks are being made. To
suppress all normal output, redirect it to '/dev/null' (any error
messages will still be shown).

'--srcdir=DIR'

Look for the package's source code in directory DIR. Usually
'configure' can determine that directory automatically.

'--version'

Print the version of Autoconf used to generate the 'configure'
script, and exit.

'configure' also accepts some other, not widely useful, options.

4. NEWS

Release

~~~~~  
u.2.1        xcept  
and    s  
platforms,  
Fedora white) es unch ugs        ed  
bugs ge gf  
--tool=cachegrind,

In w        y  
wet,        ve  
yatch unehy v ed        unch bua-450memollo03Tds-450inNo-450err450fv ed  
3,n-i-bz pgra-450e-45ork  
3,n-i-bzAMESuncho w  
106852bzunch  
117172bz veuse

127521bzu0xF0bu0x48bu0xFbu0xC7  
128917bzu0x66bu0xFbu0xF6bu0xC4  
129246bzuppc32/ppc64  
129358bzunch  
12986k  
3,n-iBartVmd64->IDo]Cadoesn'xcept450memo.X ugs  
3,n-i450x86/amd50nsn'



- Addrcheck has been removed. It has not worked since version 2.4.0,

- MPI support: partial support for debugging distributed applications using the MPI support library specification has been added. Valgrind is aware of the MPI state caused by a subset of the MPI functions, and will carefully MPI interface.
- A new `--error-exitcode=`, has been added. This allows MPI the MPI exit code in runs MPI Valgrind reported errors, which is useful MPI using Valgrind as part of an automated test suite.
- Various segfaults reading old-style MPI debug information have been fixed.
- A simple MPI evaluation suite has been added. See



which is like MAKE\_MEM\_DEFINED but only affects a byte if the byte is already addressable.

- The way client requests are encoded in the instruction stream has changed. Unfortunately, this means 3.2.0 will not honour client requests compiled into binaries using headers from earlier versions of Valgrind. We will try to keep the client request encodings more stable in future.

#### BUGS FIXED:

108258 NPTL pthread cleanup handlers not called  
117290 valgrind is sigKILL'd on startup  
117295 == 117290

118703  
118703



~~~~~

- Valgrind can now run itself (see README_DEVELOPERS for how). This is not much use to you, but it means the developers can now profile Valgrind using Cachegrind. As a result a couple of performance bad cases have been fixed.

- The XML output format has changed slightly. See docs/internals/xml-output.txt.

- Core dumping has been reinstated (it was disabled in 3.0.0 and 3.0.1). If your program crashes while running under Valgrind, a core file with the name "vgcore.<pid>" will be created (if your settings allow core file creation). Note that the floating point information is not all there. If Valgrind itself crashes, the OS will create a normal core file.

The following are some user-visible changes that occurred in earlier versions that may not have been announced, or were announced but not widely noticed. So we're mentioning them now.

- The --tool flag is option-450(coronce)-450(ag)5(ain;)-900(if)-450(you50(coromit)-450(it,)-450(Memcheck)]TJ8.9663-11.9552Td[(

- The --log-file option no longer puts "pid" in the filename, eg. the old name "foo.pid12345" is now "foo.12345".

- There are sever-450(cographic-450(corfront-ends)-450(for)-450(V)111(algrind,)-450(such)-450(as)-450(V)111(alk)15(yrie,)]TJ8.9663-11.9552Td[(

111809 Memcheck tool doesn't start java
92071 Reading debugging info uses too much memory

114563 stack tracking module not informed when valgrind switches threads
114564 clone() and stacks
114565 == 114564

the `--max-stackframe` flag for simple cases, and the `VALGRIND_STACK_REGISTER`, `VALGRIND_STACK_DEREGISTER` and `VALGRIND_STACK_CHANGE` client requests for trickier cases.

- Support for programs that use self-modifying code has been improved, in particular programs that put temporary code fragments on the stack. This helps for C programs compiled with GCC that use nested functions, and also Ada programs. This is controlled with the `--smc-check` flag, although the default setting should work in most cases.
- Output can now be written in XML. This should help with tools that process Valgrind output.
- Programs that use temporary code fragments on the stack can now be run with `--smc-check=none` to avoid the overhead of the self-modifying code checks.
- The `valgrind` command now has a `--log-file` option to write the log to a file.
- This is the first release of Valgrind since the introduction of the `--max-stackframe` flag.
- Can be used to debug programs that use nested functions.

* Signal handling is much improved and should be very close to what you get when running natively.

One result of this is that Valgrind observes changes to sigcontexts passed to signal handlers. Such modifications will take effect when `pthread_sigmask(SIG_SETMASK, &res, &oldsigmask)` is called.

91604 rw_lookup clears orig and sends the NULL value to rw_new

2.2.0 is not much different from 2.1.2, released seven weeks ago.

88115 In signal handler for SIGFPE, signinfo->si_addr is wrong under Valgrind

78765 Massif crashes on app exit if FP exceptions are enabled

Additionally there are the following changes, which are not connected to any bug report numbers, AFAICS:

- * Fix scary bug causing mis-identification of SSE stores vs loads and so causing memcheck to sometimes give nonsense results on SSE code.
- * Add support for the POSIX message queue system calls.
- * Fix to allow 32-bit Valgrind to run on AMD64 boxes. Note: this does NOT allow Valgrind to work with 64-bit executables - only with 32-bit executables on an AMD64 box.
- * At configure time, only check whether linux/mii.h can be processed so that we don't generate ugly warnings by trying to compile it.
- * Add support for POSIX clocks and timers.

Developer (cvs head) release 2.1.2(on)-at5ease 04d)

2.1.2(on)-(-cotaions)-450fourer woith of buges and

`--logsocket` --> `--log-socket`

to be consistent with each other and other options (esp. `--input-fd`).

We are now doing automatic overnight build-and-test runs on a variety of distros. As a result, we believe 2.1.1 builds and runs on:
Red Hat 7.2, 7.3, 8.0, 9, Fedora Core 1, SuSE 8.2, SuSE 9.

The following bugs, and probably many more, have been fixed. These are listed at <http://bugs.kde.org>. Reporting a bug for valgrind in the <http://bugs.kde.org> is much more likely to get you a fix than mailing developers directly, so please continue to keep sending bugs there.

69616 glibc 2.3.2 w/NPTL is massively different than what valgrind expects
69856 I don't know how to instrument MMXish stuff (Helgrind)
73892 valgrind segfaults starting with Objective-C debug info
(fix for S-type stabs)
73145 Valgrind complains too much about close(<reserved fd>)
73902 Shadow memory allocation seems to fail on RedHat 8.0
68633 VG_N_SEMAPHORES too low (V itself was leaking semaphores)
75099 impossible to trace multiprocess programs
76839 the 'impossible' happened: disInstr: INT but not 0x80 !
76762 vg_to_ucode.c:3748 (dis_push_segreg): Assertion 'sz == 4' failed.
76747 cannot include valgrind.h in c++ program
76223 parsing B(3,10) gave NULL type => impossible happens
756.t shmdt handling problem
76416 Problems with gcc 3.4 snap 20040225
7561t using -gstabs when building your programs the 'impossible' happened
75787 Patch for some CDROM ioctls CDORM_GET_MCN, CDROM_SEND_PACKET,
7529t gcc 3.4 snapshot's libstdc++ have unsupported instructions.
(REP RET)
73326 vg_syntab2.c:272 (addScopeRange): Assertion 'range->size > 0' failed.
72596 not recognizing __libc_malloc
69489 Would like to attach ddd to running program
72781 Cachegrind crashes with kde programs
73055 Illegal operand at DXTCV11CompressBlockSSE2 (more SSE opcodes)
73026 Descriptor leak check reports port numbers wrongly
71705 README_MISSING_SYSCALL_OR_IOCTL out of date
72643 Improve support for SSE/SSE2 instructions
72484 valgrind leaves it's own signal mask in place when execing
72650 Signal Handling always seems to restart system calls
72006 The mmap system call turns all errors in ENOMEM
71781 gdb attach is pretty useless
71180 unhandled instruction bytes: 0xF 0xAE 0x85 0xE8
69886 writes to zero page cause valgrind to assert on exit
71791 crash when valgrinding gimp 1.3 (stabs reader problem)
69783 unhandled syscall: 218
69782 unhandled instruction bytes: 0x66 0xF 0x2B 0x80
70385 valgrind fails if the soft file descriptor limit is less
than about 828
69529 "rep; nop" should do a yield

Stable release 2.0.0 (5 Nov 2003)

~~~~~

2.0.0 improves SSE/SSE2 support, fixes some minor bugs, and improves support for SuSE 9 and t Red Hat "Severn" beta.

- Furt improvements to SSE/SSE2 support. T entire test suite of t GNU Scientific Library (gsl-1.4) compiled with Intel lcc 7.1 20030307Z '-g -O -xW' now works. I t t gives pretty good coverage of SSE/SSE2 floating point instructions, or at least t subset emitted by lcc.
- Also added support for t following instructions:



- inc/dec %esp
- A whole bunch of SSE/SSE2 instructions









is a known bug which we are looking into.

If you can, your best bet (unfortunately) is to avoid using

# 5. README

Release notes for Valgrind

~~~~~

If you are building a binary package of Valgrind for distribution, please read README_PACKAGERS. It contains some important information.

If you are developing Valgrind, please read README_DEVELOPERS. It contains some useful information.

For instructions on how to build/install, see the end of this file.

Valgrind works on most, reasonably recent Linux setups. If you have problems, consult FAQ.txt to see if there are workarounds.

Executive Summary

~~~~~

Valgrind is an award-winning suite of tools for debugging a-450(is)profiling Linux programs. With the tools that come with Valgrind, you can automatically detect many memory management a-450(is)threading bugs, avoiding hours of frustrating bug-hunting, making your programs more stable. You can also perform detaile450(is)profiling, to spee450(is)up a-450(is)reduce memory use of your programs.

The Valgrind distribution currently includes four tools: a memory error detector, a thread error detector, a cache profiler a-450(is)a heap profiler.

To give you an idea of what Valgrind tools do, when50(is)a program is run



## 6. README\_MISSING\_SYSCALL\_OR\_IOCTL

Dealing with missing system call or ioctl wrappers in Valgrind

~~~~~  
You're probably reading this beca~~~~g2singorValgrind

To compare multiple versions of Valgrind, use the `--vg=` option multiple times. For example, if you have two Valgrinds next to each other, one in `trunk1/` and one in `trunk2/`, from within either `trunk1/` or `trunk2/` do this to compare them on all the performance tests:

```
perl perf/vg_perf --vg=../trunk1 --vg=../trunk2 perf/
```

Debugging Valgrind with GDB

8. README_PACKAGERS

Greetings, packaging person! This information is aimed at people building binary distributions of Valgrind.

Thanks for taking the time and effort to make a binary distribution of Valgrind. The following notes may save you some trouble.

-- (Unfortunate but true) When you configure to build with the `--prefix=/foo/bar/xyzy` option, the prefix `/foo/bar/xyzy` gets baked into valgrind. The consequence is that you must install valgrind at the location specified in the prefix. If you don't, it may appear to work, but will break doing some obscure things, particularly doing `fork()` and `exec()`.

So you caefix.gecxec(blexeRPM / whatever from Valgrind.

-- Doefix.g the de4k(g info offg or libpthead.so.
Valgrind will still work if you do, but it will generate less helpful error messages. Here's an example:

from valgrind.

- Please test the final installation works by running it on something huge. I suggest checking that it can start and exit successfully both Mozilla-1.0 and OpenOffice.org 1.0. I use these as test programs, and I know they fairly thoroughly exercise Valgrind. The command lines to use are:

```
valgrind -v --trace-children=yes --workaround-gcc296-Sugs=yes mozilla
```

```
valgrind -v --trace-children=yes --workaround-gcc296-Sugs=yes soffice
```

If you find any more hints/tips for packaging, please report it as a bugreport. See <http://www.valgrind.org> for details.

1. The GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

patents. We wish to avoid the danger that redistributors of a free

interactive use in the most ordinary way, to print or display an

PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,

modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawing Cov with Front-Co9, w(editors)-450(or)-450(p(or)-450(in)-450n7the)-450(relationship

the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following

given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

