

Linux on System z debugging with Valgrind

Christian Bornträger <borntraeger@de.ibm.com>



Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

AIX*	IBM*	PowerVM	System z10	z/OS*
BladeCenter*	IBM eServer	PR/SM	WebSphere*	zSeries*
DataPower*	IBM (logo)*	Smarter Planet	z9*	z/VM*
DB2*	InfiniBand*	System x*	z10 BC	z/VSE
FICON*	Parallel Sysplex*	System z*	z10 EC	
GDPS*	POWER*	System z9*	zEnterprise	
HiperSockets	POWER7*			

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license there from.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Windows Server and the Windows logo are trademarks of the Microsoft group of countries.

InfiniBand is a trademark and service mark of the InfiniBand Trade Association.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

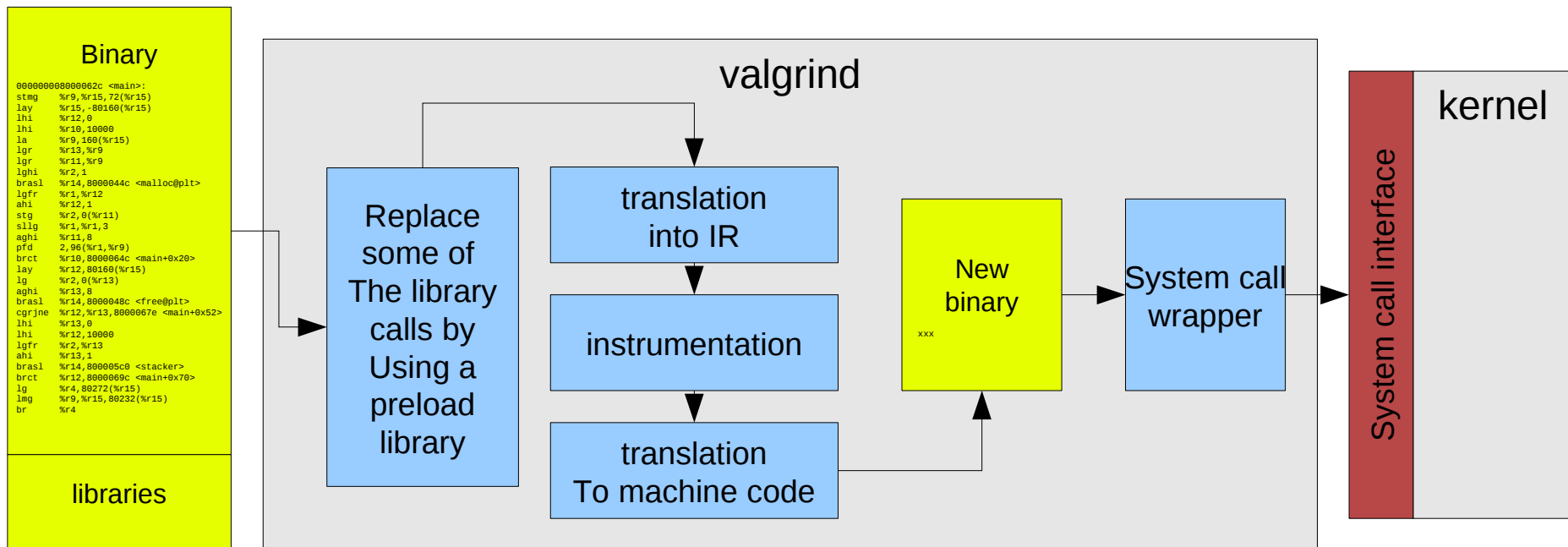
Overview (1)



- Valgrind is one of the most prominent debugging tools for Linux
- Finds long-standing and hard to find problems
- Works on binary code
 - no source code necessary (e.g. if linked against a closed library)
 - No need to recompile
- Plenty of users: Firefox, OpenOffice, AbiWord, Opera, KDE, GNOME, Qt, libstdc++, MySQL, PostgreSQL, Perl, Python, PHP, Samba, RenderMan, Nasa Mars Lander software, SAS, The GIMP, Ogg Vorbis, Unreal Tournament, Medal of Honour...
- Also supports x86, power, arm and mips
- Modular system: translation framework + tools
- award-winning
 - May 2008: TrollTech's „inaugural Qt Open Source Development Award for the best open source development tool“
 - July 2006: Julian Seward won a Google-O'Reilly Open Source Award for "Best Toolmaker" für his work on valgrind

Overview (2)

- Technology is based on a JIT (Just-in-Time Compiler)
- Intermediate language allows debugging instrumentation



Overview (3)

•Usage: valgrind <program>

```
# valgrind buggy_program
==2799== Memcheck, a memory error detector
==2799== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2799== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2799== Command: buggy_program
==2799==
==2799== HEAP SUMMARY:
==2799==    in use at exit: 200 bytes in 2 blocks
==2799== total heap usage: 2 allocs, 0 frees, 200 bytes allocated
==2799==
==2799== LEAK SUMMARY:
==2799==    definitely lost: 100 bytes in 1 blocks
==2799==    indirectly lost: 0 bytes in 0 blocks
==2799==    possibly lost: 0 bytes in 0 blocks
==2799==    still reachable: 100 bytes in 1 blocks
==2799==    suppressed: 0 bytes in 0 blocks
==2799== Rerun with --leak-check=full to see details of leaked memory
[...]
```

Tools

- Several tools

- Memcheck (default): detects memory and data flow problems

- Cachegrind: cache profiling

- Massif: heap profiling

- Helgrind: thread debugging

- DRD: thread debugging

- None: no debugging (for valgrind JIT testing)

- Callgrind: codeflow and profiling

- Tool can be selected with `--tool=none|memcheck|cachegrind...`

Memcheck

• memcheck detects

–Invalid accesses

- Non-allocated memory
- use-after-free
- Array overruns on heap
- Invalid stack areas

–Use of uninitialized data

–Memory leaks

–Invalid free

- Double free
- Wrong free function (free vs. delete vs. delete[] etc.)

–Invalid overlap (memcpy)

–Unfortunately no array overrun on stack

• Important parameters:

--leak-check=full

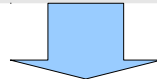
--track-origins=yes

Memcheck – memory leaks

```

01: #include <stdio.h>
02: #include <stdlib.h>
03: #include <string.h>
04:
05: int main()
06: {
07:     int i;
08:
09:     for (i=0; i < 5; i++)
10:         malloc(1);
11: }

```



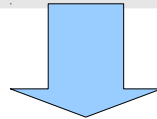
```

cbornta@r1745045:~/valgrind-demo> valgrind --leak-check=full ./memcheck-leak
[...]
==4390==
==4390== HEAP SUMMARY:
==4390==    in use at exit: 5 bytes in 5 blocks
==4390== total heap usage: 5 allocs, 0 frees, 5 bytes allocated
==4390==
==4390== 4 bytes in 4 blocks are definitely lost in loss record 2 of 2
==4390==    at 0x4026B26: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-s390x-
linux.so)
==4390==    by 0x800005F1: main (memcheck-leak.c:10)
==4390==
==4390== LEAK SUMMARY:
==4390==    definitely lost: 4 bytes in 4 blocks
==4390==    indirectly lost: 0 bytes in 0 blocks
==4390==    possibly lost: 0 bytes in 0 blocks
==4390==    still reachable: 1 bytes in 1 blocks
==4390==    suppressed: 0 bytes in 0 blocks
==4390== Reachable blocks (those to which a pointer was found) are not shown.
==4390== To see them, rerun with: --leak-check=full --show-reachable=yes
[...]

```


Memcheck – missing initialisation

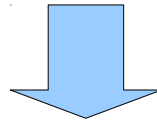
```
01: #include <stdlib.h>
02: int main()
03: {
04:     int a,b,c;
05:
06:     a = 1;
07:     c = a + b;
08:     exit(c);
09: }
```



```
cborntra@r1745045:~/valgrind-demo> valgrind --track-origins=yes ./memcheck-unit
[...]
==4471== Syscall param exit_group(status) contains uninitialised byte(s)
==4471==    at 0x41008A2: _Exit (in /lib64/libc-2.4.so)
==4471==    by 0x4093B43: exit (in /lib64/libc-2.4.so)
==4471==    by 0x800005FF: main (memcheck-unit.c:8)
==4474== Uninitialised value was created by a stack allocation
==4474==    at 0x800005D2: main (memcheck-unit.c:3)
==4471==
==4471==
==4471== HEAP SUMMARY:
==4471==    in use at exit: 0 bytes in 0 blocks
==4471== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4471==
==4471== All heap blocks were freed -- no leaks are possible
==4471==
==4471== For counts of detected and suppressed errors, rerun with: -v
==4471== Use --track-origins=yes to see where uninitialised values come from
==4471== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
```

Memcheck – overlap detection

```
01: #include <string.h>
02:
03: int main()
04: {
05:     char buf[1000];
06:
07:     memcpy(buf, buf+10, 200);
08: }
```



```
cborntra@r1745045:~/valgrind-demo> valgrind ./memcheck-memcpy
==4194== Memcheck, a memory error detector
==4194== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==4194== Using Valgrind-3.6.0 and LibVEX; rerun with -h for copyright info
==4194== Command: ./memcheck-memcpy
==4194==
==4194== Source and destination overlap in memcpy(0x7feffffb58, 0x7feffffb62, 200)
==4194==   at 0x402A1F0: memcpy (in /usr/lib64/valgrind/vgpreload_memcheck-s390x-
linux.so)
==4194==   by 0x80000601: main (memcheck-memcpy.c:7)
==4194==
==4194== HEAP SUMMARY:
==4194==   in use at exit: 0 bytes in 0 blocks
==4194==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4194==
==4194== All heap blocks were freed -- no leaks are possible
==4194==
==4194== For counts of detected and suppressed errors, rerun with: -v
==4194== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
```

drd/helgrind

- Detects potential races between threads
 - read/write without lock
 - Lock contention
- Mis-use of pthread library
 - Invalid pointer (e.g. mutex instead of condition variable)
 - Unlock without lock
 - Unlock of a lock that was taken by a different thread
 - Deletion of a taken lock
 - ...
- DRD and Helgrind cover similar cases
 - Both tools have strong and weak points → use both

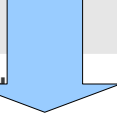
Helgrind

Data races

```

01: #include <pthread.h>
02:
03: static char mem;
04:
05: static void* thread_func(void* arg)
06: {
07:     mem = (char) (unsigned long) arg;
08:     return NULL;
09: }
10:
11: int main(int argc, char** argv)
12: {
13:     pthread_t tid;
14:     pthread_create(&tid, NULL, thread_func, (void *) 1);
15:     pthread_create(&tid, NULL, thread_func, (void *) 2);
16:     return 0;
17: }

```



```

cborntra@r1745045:~/valgrind-demo> valgrind --tool=helgrind ./helgrind
[...]
==7027== Thread #3 was created
==7027==   at 0x4137FEC: clone (in /lib64/libc-2.4.so)
==7027==   by 0x40473BF: do_clone (in /lib64/libpthread-2.4.so)
==7027==   by 0x40478C9: pthread_create@@GLIBC_2.2 (in /lib64/libpthread-2.4.so)
==7027==   by 0x402C977: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-s390x-linux.so)
==7027==   by 0x8000068B: main (helgrind-race.c:15)
==7027==
==7027== Thread #2 was created
==7027==   at 0x4137FEC: clone (in /lib64/libc-2.4.so)
==7027==   by 0x40473BF: do_clone (in /lib64/libpthread-2.4.so)
==7027==   by 0x40478C9: pthread_create@@GLIBC_2.2 (in /lib64/libpthread-2.4.so)
==7027==   by 0x402C977: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-s390x-linux.so)
==7027==   by 0x8000066B: main (helgrind-race.c:14)
==7027==
==7027== Possible data race during write of size 1 at 0x80001acc by thread #3
==7027==   at 0x80000616: thread_func (helgrind-race.c:7)
==7027==   by 0x402CB2D: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-s390x-linux.so)
==7027==   by 0x40470C1: start_thread (in /lib64/libpthread-2.4.so)
==7027==   by 0x413806D: ??? (in /lib64/libc-2.4.so)
==7027== This conflicts with a previous write of size 1 by thread #2
==7027==   at 0x80000616: thread_func (helgrind-race.c:7)
==7027==   by 0x402CB2D: ??? (in /usr/lib64/valgrind/vgpreload_helgrind-s390x-linux.so)
==7027==   by 0x40470C1: start_thread (in /lib64/libpthread-2.4.so)
==7027==   by 0x413806D: ??? (in /lib64/libc-2.4.so)
[...]

```

Callgrind

- Profiling tool
 - Call graphs
 - Function dependencies
 - Cache accesses
 - Branch prediction
- creates callgrind.out.<pid> files
- Use GUIs to read data
- Additional information for the out files with
 - --ump-instr=yes: assembler code
 - --trace-jump=yes : branches
- Contains a simplified branch prediction

Profiling -GUIs (e.g. kcachegrind)

valgrind --tool=callgrind --dump-instr=yes --trace-jump=yes ./5-callgrind-deep

```

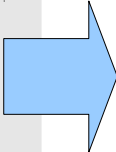
int t3(void)
{
    return 1;
}

int t2(void)
{
    return t3();
}

int t1(void)
{
    return t2();
}

int main()
{
    int i, sum;

    sum = 0;
    for (i=0; i<100000; i++)
        sum+=t1();
    return sum;
}
    
```



The screenshot shows the kcachegrind GUI with the following data:

Incl.	Self	Called	Function	Location
96.44	0.00	(0)	0x00000000000000cc0	ld-2.4.so
93.32	0.00	1	(below main)	libc-2.4.so
93.20	25.19	1	main	deep: deep.c
68.01	25.19	100 000	t1	deep: deep.c
42.82	25.19	100 000	t2	deep: deep.c
17.63	17.63	100 000	t3	deep: deep.c

No.	Ir	Hex	Assembly Instructions	Source Position
8000 05F2	0.00	c0 00 00 00 00 00 07 1ari	%r13,80000700 <_IO_stain_usea+0x10>	deep.c:17
8000 05F8	0.00	a7 fb ff 58	aghi %r15,-168	deep.c:17
8000 05FC	0.00	b9 04 00 bf	lgr %r11,%r15	deep.c:17
8000 0600	0.00	a7 18 00 00	lhi %r1,0	deep.c:20
8000 0604	0.00	50 10 b0 a4	st %r1,164(%r11)	deep.c:20
8000 0608	0.00	a7 18 00 00	lhi %r1,0	deep.c:21
8000 060C	0.00	50 10 b0 a0	st %r1,160(%r11)	deep.c:21
8000 0610	0.00	a7 f4 00 11	j 80000632 <main+0x46>	deep.c:21
Jump 1 times to 0x80000632				
8000 0614	2.52	c0 e5 ff ff d4	brsl %r14,800005bc <t1>	deep.c:22
100000 call(s) to 't1' (deep: deep.c)				
8000 061A	2.52	b9 04 00 12	lgr %r1,%r2	deep.c:22
8000 061E	2.52	5a 10 b0 a4	a %r1,164(%r11)	deep.c:22
8000 0622	2.52	50 10 b0 a4	st %r1,164(%r11)	deep.c:22
8000 0626	2.52	58 10 b0 a0	l %r1,160(%r11)	deep.c:21

Cachegrind

- Shows simulated cache behaviour

- Default based on z10 cache values
- Parameter available, e.g. to use z9 or z196 values
- Handles 2 levels: 1st level und last level for instruction and data
- Also creates cachegrind.out.<pid> files

```
r1745045:~ # valgrind --tool=cachegrind ls
==21487== Cachegrind, a cache and branch-prediction profiler
==21487== Copyright (C) 2002-2010, and GNU GPL'd, by Nicholas Nethercote et al.
==21487== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==21487== Command: ls
==21487==
--21487-- Warning: Cannot auto-detect cache config on s390x, using one or more defaults
bin inst-sys repos testtools
==21487==
==21487== I   refs:          656,270
==21487== I1 misses:           792
==21487== LLi misses:         656
==21487== I1 miss rate:    0.12%
==21487== LLi miss rate:  0.09%
==21487==
==21487== D   refs:          453,124 (361,066 rd + 92,058 wr)
==21487== D1 misses:         1,869 ( 1,589 rd +   280 wr)
==21487== LLd misses:         1,313 ( 1,061 rd +   252 wr)
==21487== D1 miss rate:    0.4% (  0.4% +  0.3% )
==21487== LLd miss rate:  0.2% (  0.2% +  0.2% )
==21487==
==21487== LL refs:           2,661 ( 2,381 rd +   280 wr)
==21487== LL misses:         1,969 ( 1,717 rd +   252 wr)
==21487== LL miss rate:    0.1% (  0.1% +  0.2% )
```

Availability

- System z support since version 3.7
 - SLES11SP2
- Backports into 3.6
 - SLES10
 - Since SP4 on SDK
 - RHEL6
 - Since U1 on main DVD
 - RHEL5
 - No valgrind for z

Installation

- **RHEL6:**

- `yum install valgrind valgrind-devel`
- Kcachegrind: install kde-sdk (+plus X and fonts)

- **SLES11**

- `zypper install valgrind valgrind-devel`
- Kcachegrind:
 - kdesdk3 source rpm from opensuse factory. Rebuild source rpm (<http://download.opensuse.org/source/factory/repo/oss/suse/src/kdesdk3-.....src.rpm>)
 - Install the resulting kdesdk-profile package
 - Start with `/opt/kde3/bin/kcachegrind`

Hints

- No need to recompile, but
 - Better results with debug info
 - Gcc option `-O0` might result in more findings (the compiler might hide some errors)
 - Gcc option `-fno-builtin` might result in more findings
- `--trace-children=yes` will also debug child processes
- Setuid programs might cause trouble
 - Valgrind is the process container (→ no setuid)
 - Possible solution: remove setuid and start as the right user, check documentation for other ways
- Killall/ps etc.
 - Valgrind starter will be replaced by the tool
 - `none-s390x-linux`
 - `memcheck-s390x-linux`
 - etc.
- The program will be slower
 - 5-30 times slower for memcheck
- Suppression files can be used to block specific errors
- `LANG=C` to avoid `CU**` instruction

Limitations for System z

- No exp-ptrcheck (exp-sgcheck) tool support
- Only 64bit applications
- No decimal floating point
- Current focus is on compiler generated code (gcc)
- Some other instructions are missing in the older backports (translate, stfle)
- Only partial support for gcc option -march=z196

Debugging: gdb attach

```
cborntra@tuxmaker:~/valgrind-demo> valgrind --db-attach=yes ./fault
==13002== Memcheck, a memory error detector
==13002== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==13002== Using Valgrind-3.6.0.SVN and LibVEX; rerun with -h for copyright info
==13002== Command: ./fault
==13002==
==13002== Invalid read of size 1
==13002==    at 0x8000057C: main (fault.c:7)
==13002==   Address 0x0 is not stack'd, malloc'd or (recently) free'd
==13002==
==13002==
==13002== ---- Attach to debugger ? --- [Return/N/n/Y/y/C/c] ---- y
==13002== starting debugger with cmd: /usr/bin/gdb -nw /proc/13033/fd/1014 13033
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "s390x-suse-linux"...
Using host libthread_db library "/lib64/libthread_db.so.1".
Attaching to program: /proc/13033/fd/1014, process 13033
0x000000008000057c in main () at fault.c:7
7         exit(*c);
(gdb)
```

Debugging gdb server (≥ 3.7)

- Since 3.7 valgrind contains a gdb server
- vgdb control program
- `--vgdb-error=0` as quick way to enable

```
[cbornta@r1745045 valgrind]$ valgrind --vgdb-error=0
/home/cbornta/REPOS/valgrind/none/tests/s390x/mvcl
==65367== Memcheck, a memory error detector
==65367== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==65367== Using Valgrind-3.8.0.SVN and LibVEX; rerun with -h for copyright info
==65367== Command: /home/cbornta/REPOS/valgrind/none/tests/s390x/mvcl
==65367==
==65367== (action at startup) vgdb me ...
==65367==
==65367== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==65367==   /path/to/gdb /home/cbornta/REPOS/valgrind/none/tests/s390x/mvcl
==65367== and then give GDB the following command
==65367==   target remote | /usr/local/lib/valgrind/../../bin/vgdb --pid=65367
==65367== --pid is optional if only one valgrind process is running
==65367==
```

Annotations

- Your program can control valgrind
 - Several macros defined in valgrind header files under `/usr/include/valgrind/`
 - Make memory defined/undefined, check if running under valgrind etc.
 - Valgrind-devel package

```
01: int main()
02: {
03:     int a = 3;
04:
05:     printf("No Error here: %d\n", a);
06:     VALGRIND_MAKE_MEM_UNDEFINED(&a, sizeof(a));
07:     printf("Here is an error %d\n", a);
08:     return 0;
09: }
```

Problem reporting

- https://bugs.kde.org/enter_bug.cgi?product=valgrind

- Missing instructions will look like

```
vex s390->IR: unimplemented insn: xxxx xxxx
```

```
valgrind: Unrecognised instruction at address
```

```
0XXXXXXXXX.
```

- Missing system calls

```
WARNING: unhandled syscall: xxx
```

You may be able to write your own handler.

Read the file README_MISSING_SYSCALL_OR_IOCTL.

- Valgrind internal errors

- Might be a valgrind bug
- Might be caused by buggy programs

Thanks