Valgrind vs. KVM

Christian Bornträger

IBM Deutschland Research & Development GmbH
borntraeger@de.ibm.com

Co-maintainer KVM and
QEMU/KVM for s390x
(aka System z, zEnterprise,
IBM mainframe)

Valgrind vs. KVM based QEMU

http://wiki.qemu.org/Debugging_with_Valgrind says:

“valgrind really doesn't function well when using KVM so it's advised to use TCG”

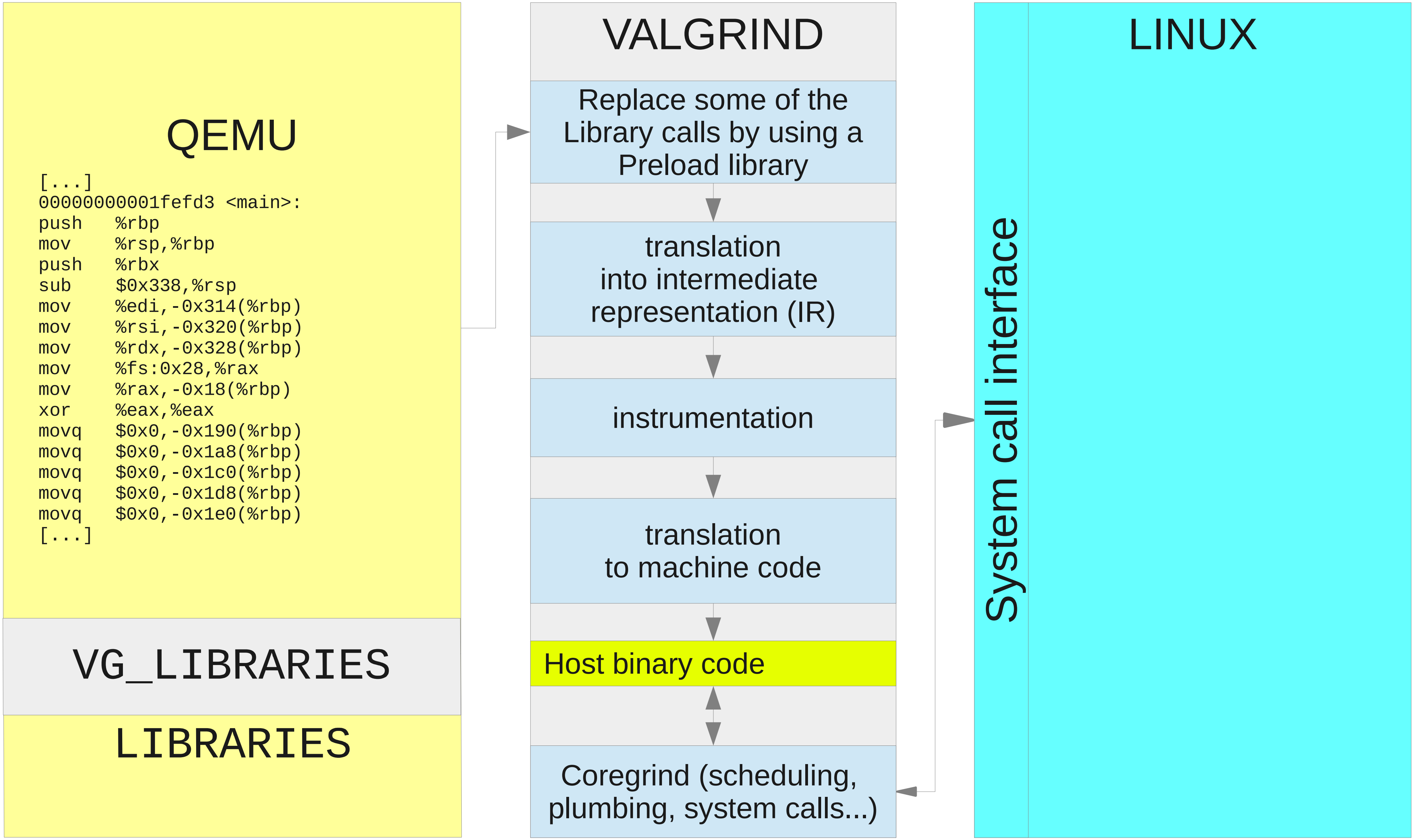
- So: my presentation ends here....
- Really?

Valgrind overview (1/4)

- “Valgrind is a tool for finding memory leaks”

  ?
- Valgrind is an instrumentation framework for building dynamic analysis tools
  - works on compiled binary code - no source checker
  - “understands” most instructions and most system calls
- Used as debugging tool

Valgrind overview (2/4)


QEMU
VG_LIBRARIES
LIBRARIES
VALGRIND
Replace some of the Library calls by using a Preload library
translation into intermediate representation (IR)
instrumentation
translation to machine code
Host binary code
Coregrind (scheduling, plumbing, system calls...)
LINUX
System call interface

```
[...]
000000001fefd3 <main>:
push    %rbp
mov     %rsp,%rbp
push    %rbx
sub     $0x338,%rsp
mov     %edi,-0x314(%rbp)
mov     %rsi,-0x320(%rbp)
mov     %rdx,-0x328(%rbp)
mov     %fs:0x28,%rax
mov     %rax,-0x18(%rbp)
xor     %eax,%eax
movq    $0x0,-0x190(%rbp)
movq    $0x0,-0x1a8(%rbp)
movq    $0x0,-0x1c0(%rbp)
movq    $0x0,-0x1d8(%rbp)
movq    $0x0,-0x1e0(%rbp)
[...]
```

Valgrind overview (3/4)

- The instrumentation is done with tools
  - Memcheck (default): detects memory-management problems
  - Cachegrind: cache profiler
  - massif: Heap profiler
  - Helgrind/DRD: Thread race debugger
  - ….
- Usage is simple:

```
# valgrind [valgrind parameters] <program> [program parameters]
e.g.
# valgrind qemu-system-x86_64 -drive file=image,if=virtio -enable-kvm
or
# valgrind –tool=helgrind qemu-system-x86_64 -drive file=image,if=virtio -enable-kvm
```

Valgrind overview (4/4)

- Valgrind detects (depending on the tool)
  - Memory leaks
  - Usage of undefined memory
  - Heap buffer overflows
  - Undefined parameters in system calls
  - Misuse of library calls
  - Threading errors
  - […]
  - See http://valgrind.org/docs/manual/manual.html for details

Valgrinds view on system calls (1/3)

- Valgrind’s memcheck does several things:
  - [..]
  - Leak detection
  - Definedness checking
    - All side effects of system calls need to be considered
    - Long list of system call pre and post handlers

```
VALGRIND: coregrind/m_syswrap/syswrap-amd64-linux.c:
[...]
   PLAX_(__NR_rt_sigreturn,      sys_rt_sigreturn),   // 15
   LINXY(__NR_ioctl,             sys_ioctl),          // 16
   GENXY(__NR_pread64,           sys_pread64),        // 17
[...]
```

- Platform, linux and generic pre (x) and post(y) handler
- All contain annotations about side effects

Valgrinds view on system calls (2/3)

- Long list of special ioctls
- Default handler, that considers the IORW macros
  - Usually fine, but
    - Wrong ioctl annotations:

```
LINUX: include/uapi/linux/kvm.h:
#define KVM_GET_PIT    _IOWR(KVMIO, 0x65, struct kvm_pit_state) get is logically a read (*)
#define KVM_SET_PIT    _IOR(KVMIO,  0x66, struct kvm_pit_state) This is a write → _IOW
```

- Ioctl numbers are ABI: Needs to be “fixed” in valgrind
- (In this case we have KVM_[G|S]ET_PIT2....)

- No ioctl annotations:

```
LINUX: include/uapi/linux/kvm.h:
#define KVM_GET_API_VERSION       _IO(KVMIO,   0x00)
#define KVM_CREATE_VM             _IO(KVMIO,   0x01) /* returns a VM fd */
#define KVM_S390_ENABLE_SIE       _IO(KVMIO,   0x06)
#define KVM_CHECK_EXTENSION       _IO(KVMIO,   0x03)
[...]
```

- old ioctl, just ignoring the scheme?
- Really no parameters? Is arg checked for defaults?

(*) the kernel reads, replaces and writes struct kvm_pit_state (WTF?)

Valgrinds view on system calls (3/3)

- Extensible or flexible data structures:

```
LINUX: arch/x86/include/uapi/asm/kvm.h:
struct kvm_pit_state2 {
        struct kvm_pit_channel_state channels[3];
        __u32 flags;
        __u32 reserved[9]; Reserved bytes are not initialized
};
```

```
==23019== Syscall param ioctl(generic) points to uninitialised byte(s)
==23019==    at 0x6EFD837: ioctl (in /lib64/libc-2.12.so)
==23019==    by 0x1F8AC3: kvm_vm_ioctl (kvm-all.c:1851)
==23019==    by 0x26E3D7: kvm_pit_put (i8254.c:171)
==23019==    by 0x26E63E: kvm_pit_irq_control (i8254.c:233)
==23019==    by 0x3808D8: qemu_set_irq (irq.c:43)
[...]
```

- 2 Options
  - Special case handler in valgrind
  - Zero-initialization in QEMU

```
QEMU: hw/i386/kvm/i8254.c:
static void kvm_pit_put(PITCommonState *pit)
 {
     KVMPITState *s = KVM_PIT(pit);
-    struct kvm_pit_state2 kpit;
+    struct kvm_pit_state2 kpit = {};
     struct kvm_pit_channel_state *kchan;
     struct PITChannelState *sc;
```

Valgrind related QEMU changes (1/3)

- Valgrind is already used by several people
- A quick git log --grep valgrind
  - 4 Enablement patches (+83/-22)
  - 7 Patches to reduce noise (+82/-7)
  - 20 real bug fixes with valgrind findings (+118/-111)

Valgrind related QEMU changes (2/3)

```
configure:
if test "$valgrind_h" = "yes" ; then
  echo "CONFIG_VALGRIND_H=y" >> $config_host_mak
fi
```

```
coroutine-ucontext.c:
Coroutine *qemu_coroutine_new(void)
[...]
#ifdef CONFIG_VALGRIND_H
    co->valgrind_stack_id =
        VALGRIND_STACK_REGISTER(co->stack, co->stack + stack_size);
#endif
[...]
static inline void valgrind_stack_deregister(CoroutineUContext *co)
{
    VALGRIND_STACK_DEREGISTER(co->valgrind_stack_id);
}
```

```
kvm-all.c:
void kvm_setup_guest_memory(void *start, size_t size)
{
#ifdef CONFIG_VALGRIND_H
    VALGRIND_MAKE_MEM_DEFINED(start, size);
#endif
```

NO LONGER NECESSARY
gone with 2.2-rc

Valgrind related QEMU changes (3/3)

- So how does this work?
- Valgrind allows annotations
  - “system calls into valgrind”
  - Puts parameters on stack
  - Special NOP code sequence detected by valgrind
    - small performance cost without valgrind
  - Install valgrind-dev[el] or similar to have /usr/include/valgrind*
- QEMU has workarounds for valgrind
  - Tree with additional fixes and workarounds available at

```
git://github.com/borntraeger/qemu.git valgrind
```

(will be rebased)

Threading errors

- So what about helgrind and friends?

```
[...]
==22556== More than 10000000 total errors detected.  I'm not reporting any more.
==22556== Final error counts will be inaccurate.  Go fix your program!
==22556== Rerun with --error-limit=no to disable this cutoff.  Note
==22556== that errors may occur in your program without prior warning from
==22556== Valgrind, because errors are no longer being displayed.
==22556==
```

- What is going on here?
  - Several sophisticated schemes in QEMU
  - rfifolock
  - real problems in QEMU
  - Deficiencies in valgrind

Clever schemes

- Several variables rely on
  - access <= word size
  - memory barriers

```
QEMU: async.c
[...]
    /* Make sure that the members are ready before putting bh into list */
    smp_wmb();
[...]
```

- Quick fix: ignore specific things(in init function)

```
QEMU: async.c
+    VALGRIND_HG_DISABLE_CHECKING(&bh->scheduled, sizeof(bh->scheduled));
+    VALGRIND_HG_DISABLE_CHECKING(&bh->idle, sizeof(bh->idle));
+    VALGRIND_HG_DISABLE_CHECKING(&ctx->dispatching, sizeof(ctx->dispatching));

QEMU: thread-pool.c
+    VALGRIND_HG_DISABLE_CHECKING(&req->state, sizeof(req->state));
+    VALGRIND_HG_DISABLE_CHECKING(&req->ret, sizeof(req->ret));
```

- Proper Fix? full “happens before” annotations

rfifolock

- Rfifolock is a nested and fair locking scheme
- Valgrind needs annotations for self-made locking structures
  - Annotations based on similar pthread functions
  - rfifolock is tricky to be announced via these methods
  - Quick hack available at

```
git://github.com/borntraeger/qemu.git rfifolock
```

On threading bug messages

```
[...]
==10551==  Lock at 0xA47720 was first observed
==10551==    at 0x4A10A53: pthread_mutex_init (hg_intercepts.c:518)
==10551==    by 0x5727D4: qemu_mutex_init (qemu-thread-posix.c:57)
[…]
==10551== Possible data race during read of size 1 at 0x8C6C040 by thread #1
==10551== Locks held: 1, at address 0xA47720
==10551==    at 0x4C38885: inflate (in /lib64/libz.so.1.2.3)
==10551==    by 0x51670B: decompress_buffer (qcow2-cluster.c:1336)
[…]
==10551== This conflicts with a previous write of size 8 by thread #2
==10551== Locks held: none
==10551==    at 0x5EAD063: ??? (in /lib64/libpthread-2.12.so)
==10551==    by 0x528358: handle_aiocb_rw_linear (raw-posix.c:747)
[…]
```

- Valgrind cannot prove/disprove all cases
- Several places needs to be audited
- Possible outcome
  - Bugfix
  - Annotation
  - Suppression

Valgrind and KVM based QEMU

http://wiki.qemu.org/Debugging_with_Valgrind says:

“valgrind really doesn't function well when using KVM so it's advised to use TCG”

- In fact: it can give a lot of benefit for KVM/QEMU
  - All KVM-based guest operations are hidden from valgrind
  - BUT: the same is true for QEMU
  - We want to use valgrind to check QEMU-code not KVM
  - Valgrind does see all activities of QEMU code
    - Valgrind tracks all mallocs/frees and stack activities
    - Valgrind tracks all memory operations by QEMU
      - Valgrind tracks definedness and source
    - Valgrind tracks all system calls by QEMU
    - This will work, as long as valgrind understands the KVM ioctls and its side effects
    - Valgrind does need some help here and there, though

Hints (1/2)

- For TCG, you can use --smc-check=all-non-file
- -g, unstripped binaries or debuginfo packages improve stacktraces
- Compiler optimizations can prevent warnings (or make them appear....)
- killall qemu-system-<arch> won’t work
  - Use killall memcheck-x86_64-linux
- Performance will be a lot slower
  - virtioblk is a lot faster under valgrind than ATA
  - serial console is faster

Hints (2/2)

- Do you see ?

```
==24021== Warning: client switching stacks?  SP change: 0xffeffe6d8 --> 0x75f70a8
==24021==          to suppress, use: --max-stackframe=68578997808 or greater
```

  - Install valgrind-devel, rerun QEMU’s configure and recompile QEMU
- Valgrind has a builtin gdb server (check –vgdb-error in manual)
- Valgrind allows to provide suppressions
- --fair-sched=yes might help for thread

Outlook

- Newer valgrind versions have specific annotations for several ioctls:
- `KVM_GET_API_VERSION, KVM_CREATE_VM, KVM_CHECK_EXTENSION, […]`
- See https://bugs.kde.org/show_bug.cgi?id=339424 for a bug tracking ioctl changes in valgrind
- As a developer, don’t fear the compile ;-)

```
svn co svn://svn.valgrind.org/valgrind/trunk valgrind
cd valgrind
./autogen.sh
./configure --prefix=...
make
make install
```

- Use valgrind!
- Consider valgrind for new ioctls
- Remember and fix annotations when changing code!

धन्यवाद

Hindi

多謝

Traditional Chinese

Thai

Спасибо

Russian

Gracias

Spanish

Thank You

English

Dziękuję

Polish

شكراً

Arabic

Obrigado

Portuguese

多谢

Simplified Chinese

Danke

German

Grazie

Italian

Merci

French

நன்றி

Tamil

ありがとうございました

Japanese

감사합니다

Korean

BACKUP

QEMU under libvirt

- Simply use a shell script wrapper as emulator

```
[...]
  <on_crash>preserve</on_crash>
  <devices>
    <emulator>/home/userid/wrapper.sh</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native'/>
[...]
```

```
/home/userid/wrapper.sh:
#!/bin/bash
exec /usr/local/bin/valgrind --trace-children=yes --track-origins=yes --leak-check=full --show-leak-kinds=definite --log-file=/tmp/vallog.$$ /home/userid/qemu/build/s390x-softmmu/qemu-system-s390x "$@"
```

- For illustration I also added some parameters to valgrind

– --trace-children=yes	follow any forks
– --track-origins=yes	tells the original location of undefined values
– --leak-check=full	list with all leaks
– --show-leak-kinds=definite	only show leaks were valgrind is sure
– --log-file=xxx	send debugging output into a file

Examples – QEMU 2.1 as of 2014/09/11

```
==22972== 131,072 bytes in 1 blocks are definitely lost in loss record 3,111 of 3,115
==22972==    at 0x4A073FC: realloc (vg_replace_malloc.c:692)
==22972==    by 0x302C9E: realloc_and_trace (vl.c:2833)
==22972==    by 0x5497BFE: g_realloc (in /lib64/libglib-2.0.so.0.2600.1)
==22972==    by 0x54665DA: ??? (in /lib64/libglib-2.0.so.0.2600.1)
==22972==    by 0x54666A2: g_array_set_size (in /lib64/libglib-2.0.so.0.2600.1)
==22972==    by 0x264520: acpi_align_size (acpi-build.c:492)
==22972==    by 0x267D87: acpi_build (acpi-build.c:1691)
==22972==    by 0x268015: acpi_setup (acpi-build.c:1772)
==22972==    by 0x257ECD: pc_guest_info_machine_done (pc.c:1086)
==22972==    by 0x579DAB: notifier_list_notify (notify.c:39)
==22972==    by 0x302A9B: qemu_run_machine_init_done_notifiers (vl.c:2781)
==22972==    by 0x306FA4: main (vl.c:4532)
```

```
hw/i386/acpi-build.c:
void acpi_setup(PcGuestInfo *guest_info)
{
[…]
    acpi_build(build_state->guest_info, &tables);
    build_state->table_ram = acpi_add_rom_blob(build_state, tables.table_data,
                                               ACPI_BUILD_TABLE_FILE);
[...]
    /* Cleanup tables but don't free the memory: we track it
     * in build_state.
     */
    acpi_build_tables_cleanup(&tables, false);
```

- So, we are clever, and valgrind does not understand !

- Not quite. add_rom_blob calls rom_add_blob
- rom_add_blob calls malloc, copies and does not free the input buffer. → LEAK

Valgrind on libvirt

- Integrated in test suite
- http://libvirt.org/hacking.html
- See bullets 6 and 7 (make -C tests valgrind)

rfifolock (simplified)

```
--- a/util/rfifolock.c
+++ b/util/rfifolock.c
@@ -15,2 +15,3 @@
 #include "qemu/rfifolock.h"
+#include <valgrind/helgrind.h>

@@ -18,2 +19,3 @@ void rfifolock_init(RFifoLock *r, void (*cb)(void *), void *opaque)
 {
+    ANNOTATE_RWLOCK_CREATE(&r->nesting);
     qemu_mutex_init(&r->lock);
@@ -29,2 +31,3 @@ void rfifolock_destroy(RFifoLock *r)
 {
+    ANNOTATE_RWLOCK_DESTROY(&r->nesting);
     qemu_cond_destroy(&r->cond);
@@ -45,2 +48,3 @@ void rfifolock_lock(RFifoLock *r)
 {
+    bool locked = false;
     qemu_mutex_lock(&r->lock);
@@ -60,2 +64,3 @@ void rfifolock_lock(RFifoLock *r)
         }
+        locked = true;
     }
@@ -65,2 +70,7 @@ void rfifolock_lock(RFifoLock *r)
     qemu_mutex_unlock(&r->lock);
+
+    if (locked) {
+        ANNOTATE_RWLOCK_ACQUIRED(&r->nesting, 1);
+    }
+
 }
@@ -75,2 +85,3 @@ void rfifolock_unlock(RFifoLock *r)
         qemu_cond_broadcast(&r->cond);
+        ANNOTATE_RWLOCK_RELEASED(&r->nesting, 1);
     }
```

Valgrind related QEMU changes

```
7dda5dc migration: initialize RAM to zero
06d71fa configure: Split valgrind test into pragma test and valgrind.h test
2f24e8f qemu-iotests: Valgrind support
c2a8238 Support running QEMU on Valgrind
```

```
62fe833 qemu: Use valgrind annotations to mark kvm guest memory as defined
3f4349d coroutine-ucontext: Help valgrind understand coroutines
7e68075 kvm: fill in padding to help valgrind
160c31f ui/spice-display.c: add missing initialization for valgrind
021730f usb: initialise data element in Linux USB_DISCONNECT ioctl
0873898 tlb flush cleanup
9ed415b initialize struct sigevent before timer_create
```

```
3a1655f vhost-scsi: init backend features earlier
a760715 qemu_opts_append: Play nicely with QemuOptsList's head
f5946db vl.c: Fix memory leak in qemu_register_machine()
4f3ed19 s390x/sclpconsole-lm: Fix and simplify irq setup
b074e62 s390x/sclpconsole: Fix and simplify interrupt injection
7b53f29 s390x/cpu hotplug: Fix memory leak
ef4cbe1 kvm: Fix uninitialized cpuid_data
2c8ebac vga: fix invalid read after free
b432779 virtio: Remove unneeded memcpy
92304bf hw/9pfs: Fix memory leak in error path
e36c876 qapi: Fix memory leak
a5aa842 libcacard: fix soft=... parsing in vcard_emul_options
e332340 Fix NULL alarm_timer pointer at exit
f71903d Make sure to initialize fd_sets in aio.c
68bd348 scsi: Add assertion for use-after-free errors
f156f23 qom: Fix memory leak in function container_get
9cf1f00 hw/pc_sysfw: Fix memory leak
5c87800 qdev: Fix memory leak in function set_pci_devfn
7f84c12 compatfd.c: Don't pass NULL pointer to SYS_signalfd
229609d sdl: Fix memory leakage
```