Valgrinding complete KDE sessions

or
Post-crash bug detection considered harmful

Julian Seward

julian@valgrind.org

http://www.valgrind.org

What and why?

What?

- run complete process tree resulting from startkde on Valgrind
- use Valgrind's Memcheck tool to find invalid reads, writes and uninitialised-value uses
- monitor all components of KDE, all the time, under a wide range of uses

Why?

- to do pre-crash bug detection (normal way is post-crash :-)
- end goal: continuous monitoring of a number of user desktops with realistic workloads
- also ... to stress the Valgrind toolchain, just to see if it is possible

How? Is it practical?

How?

- use a recent valgrind -- preferably 3.2.1
- get all diagnostic output out of the way: --log-socket=127.0.0.1:1500
- trace all children: --trace-children=yes
- use a custom suppression file: --suppressions=kde354.supp
- run KDE inside a VNC server

Practical?

- does it work? yes, on {x86/amd64/ppc32}-linux
- is it fast enough? (note: build KDE with "-g -O")
  - no: 1.7 GHz P4, 1GB memory
  - marginal: 2.2 GHz Athlon64, 1GB memory
  - yes: 2 x 2.5 GHz PPC970, 4GB memory

Is it useful?

Does it find real bugs?

```
#134094 uninitialised value, KAccessApp::x11EventFilter(_XEvent*)
#134099 uninitialised value, Client::Client
#134100 uninitialised value, KServiceGroup::entries
#134118 uninitialised value, KStartupInfo::startups_cleanup_internal
#134120 reading freed mem,   KDialogBase::slotCancel
#134188 uninitialised value, RecipientLine::setComboWidth(int)
#134200 uninitialised value, RecipientsView::addLine()
```

Ah, but are they really real bugs?

```
#134099, #134100, #134188, #134200: object field not initialised
                                    (1 already fixed)

#134094: Don't try to QApplication::exit() from a QApplication
         subclass constructor

#134118: associated with startup notification bug? (already fixed, r573914)

#134120: KCMultiDialog deletion ordering problem
```

Pragmatics

- requires a decent machine
  - minimum 1GB mem, 1GB swap
  - PM/Core/Core2/Athlon/PPC970/POWER5 at 2GHz or P4 at 3GHz
  - xcpustate and xload are useful
- Uninteresting applications give errors
  - bash, xterm, perl
  - suppress them in your custom suppressions file
- What about false errors?
  - undefined value errors: valgrind-3.2.1 is the most accurate yet
  - almost all false errors are now due to structure padding at syscalls:

```
Syscall param writev(vector[...]) points to uninitialised byte(s)
   at 0x5656986: do_writev (in /lib/tls/libc-2.3.5.so)
   by 0x539F2FD: (within /usr/X11R6/lib/libX11.so.6.2)
   by 0x539F60E: _X11TransWritev (in /usr/X11R6/lib/libX11.so.6.2)
   by 0x5383394: _XSend (in /usr/X11R6/lib/libX11.so.6.2)
```

  - --gen-suppressions=all is your friend

Tracking down uninitialised-value errors

Often difficult – requires following data arbitrarily far round the application

A typical example:

```
void Class::method ( int x )
{
   if (x == m_fooBar)          <--- Memcheck complains here
      ...
}
```

A possible approach:

```
#include <valgrind/memcheck.h>

ty Class::method ( int x )
{
   VALGRIND_CHECK_DEFINED(x);
   VALGRIND_CHECK_DEFINED(m_fooBar);
   if (x == m_fooBar)
      ...
}
```

One of the checks should get you a complaint. Examine data sources accordingly.

Conclusions

"zero Memcheck errors for complete KDE session" is

- achieveable on reasonable hardware
- achieveable on current KDE code base
- useful (finds real code defects)
- semi-automatable - set up test machines and users, collect errors remotely
- applicable to other monster-sized projects (OpenOffice.org, Gnome)

regarding the errors reported:

- 7 out of 12 reported errors were real code defects
- other 5 padding errors suppressed and never seen again
- all this circa 12 months into a stable branch (3.5.4)
- my 3.5.4 + fixes now starts up and runs konq and kmail with zero errors

Think of it like this:

It's easy. In one day's work I found 5 previously unknown code defects in KDE stable.

They are now fixed (thanks kling@impul.se),and I know nothing about KDE internals.

If you only remember one thing ...

Using Valgrind increases code quality and speed, and saves you time and effort

C/C++/Fortran programmers: you should be using Valgrind!

http://www.valgrind.org