Bugfixing is fun

Tips and tricks for debugging KDE applications

David Faure

KDAB Qt NOKIA

July 6th 2009

Outline

What is a bug?

A rampant insect, but also...

A Bug's Life

Defect ⇒ Infection(s) ⇒ Failure

- Defect: the "bug" in the code
- Infection: the effect of the "bug" on program state (variables)
- Failure: the observable result of the "bug" (e.g. crash)

Example

```
int a = computeValue();
int b = a - 1;
int c = 1 / b;
```

Outline

Reproducing bugs

one is not enough

Deterministic bugs

No problem, go to next slide

Non-deterministic bugs

- Run the program in valgrind (memcheck), to detect use of uninitialized data
- Threading: run the program in helgrind, to detect races
- Write automated tests covering as many cases as possible
- Simulate random input until bug occurs
- Postpone until reproduceable :-)

Outline

Simplifying the problem

Because size matters

Proceed by binary search, reducing input size by half every time. Manually or automatically (`ddmin` algorithm).

Example

- Large HTML page crashes konqueror
- Large mail crashes kontact
- Error when calling program with 20 arguments
- LaTeX error while writing this presentation
- Many user actions $\Rightarrow$ find minimum set
- Guilty commit: svn-bisect, git-bisect

Goal: to lead us to the actual failure cause.

Outline

Scientific debugging

It's called computer science for a reason

Procedure

- Observe failure
- Make hypothesis (cause + effect)
- Use hypothesis to make predictions
- Test hypothesis using experiments
  - Experiment where the cause does not occur
  - Or verification of prediction with debugger
- Observe experiment result
  - True: refine hypothesis, if possible, repeat
  - False: find alternate hypothesis, repeat

Scientific debugging: example

Cosmic rays are rarely a good hypothesis

Example

```
a = computeValue();
printf("a = %d\n", a); // shows a = 0! Bug!
```

1. Hypothesis: a being 0 is the cause for a = 0 being printed
2. Prediction: If a was not 0, the value of a would be printed
3. Experiment:

Example

```
a = 1;
printf("a = %d\n", a);
```

4. Result: a = 0 is still printed. Hypothesis rejected.

Scientific debugging: example

Cosmic rays are rarely a good hypothesis

Example

```
a = computeValue();
printf("a = %d\n", a); // shows a = 0! Bug!
```

1. Hypothesis: a being 0 is the cause for a = 0 being printed
2. Prediction: If a was not 0, the value of a would be printed
3. Experiment:

Example

```
a = 1;
printf("a = %d\n", a);
```

4. Result: a = 0 is still printed. Hypothesis rejected.

Scientific debugging: example

Cosmic rays are rarely a good hypothesis

Example

```
a = computeValue();
printf("a = %d\n", a); // shows a = 0! Bug!
```

1. Hypothesis: a being 0 is the cause for a = 0 being printed
2. Prediction: If a was not 0, the value of a would be printed
3. Experiment:

Example

```
a = 1;
printf("a = %d\n", a);
```

4. Result: a = 0 is still printed. Hypothesis rejected.

Scientific debugging: example

Cosmic rays are rarely a good hypothesis

Example

```
a = computeValue();
printf("a = %d\n", a); // shows a = 0! Bug!
```

1. Hypothesis: a being 0 is the cause for a = 0 being printed
2. Prediction: If a was not 0, the value of a would be printed
3. Experiment:

Example

```
a = 1;
printf("a = %d\n", a);
```

4. Result: a = 0 is still printed. Hypothesis rejected.

Scientific debugging: example

Cosmic rays are rarely a good hypothesis

Example

```
a = computeValue();
printf("a = %d\n", a); // shows a = 0! Bug!
```

1. Hypothesis: a being 0 is the cause for a = 0 being printed
2. Prediction: If a was not 0, the value of a would be printed
3. Experiment:

Example

```
a = 1;
printf("a = %d\n", a);
```

4. Result: a = 0 is still printed. Hypothesis rejected.

Scientific debugging: example

kDebug rocks, printf not so much

Example

```
double a;
a = computeValue();
printf("a = %d\n", a);
```

1. Hypothesis: the format %d is the cause for a = 0 being printed
2. Prediction: using %f, the value of a is actually printed.
3. Experiment: printf(“a = %f\n”, a);
4. Result: works. Hypothesis verified. And in this case, fix found.

Scientific debugging: example

kDebug rocks, printf not so much

Example

```
double a;
a = computeValue();
printf("a = %d\n", a);
```

1. Hypothesis: the format %d is the cause for a = 0 being printed
2. Prediction: using %f, the value of a is actually printed.
3. Experiment: printf(“a = %f\n”, a);
4. Result: works. Hypothesis verified. And in this case, fix found.

Scientific debugging: example

kDebug rocks, printf not so much

Example

```
double a;
a = computeValue();
printf("a = %d\n", a);
```

1. Hypothesis: the format %d is the cause for a = 0 being printed
2. Prediction: using %f, the value of a is actually printed.
3. Experiment: printf("a = %f\n", a);
4. Result: works. Hypothesis verified. And in this case, fix found.

Scientific debugging: example

kDebug rocks, printf not so much

Example

```
double a;
a = computeValue();
printf("a = %d\n", a);
```

1. Hypothesis: the format %d is the cause for a = 0 being printed
2. Prediction: using %f, the value of a is actually printed.
3. Experiment: printf(“a = %f\n”, a);
4. Result: works. Hypothesis verified. And in this case, fix found.

Scientific debugging: example

kDebug rocks, printf not so much

Example

```
double a;
a = computeValue();
printf("a = %d\n", a);
```

1. Hypothesis: the format %d is the cause for a = 0 being printed
2. Prediction: using %f, the value of a is actually printed.
3. Experiment: printf(“a = %f\n”, a);
4. Result: works. Hypothesis verified. And in this case, fix found.

Mental check-lists

It must be one of these

A useful basis for making hypothesis is mental check-lists.

Slot not called. Why?

- signal not emitted
- receiver deleted
- emitter deleted
- connect() not done (yet?)
- connect() failed (e.g. wrong syntax; check stderr)
- connect() done on other instances
- disconnect() called

Deduce errors

Elementary, my dear Watson

Dependencies

To isolate value origins, follow back the dependencies from the statement in question.

- Data dependencies (V2 is calculated from V1)
- Control dependencies (statements executed conditionnally)

Debuggers can’t go back...

- Multiple runs (in debugger, or after adding debug output)
- Enough debug output for a comprehensive log
- Backtrace

Outline

Observing facts: kDebug

the reason for all the noise

- Make output clear and complete.

Confusing message

```
if (!findItem(name, flags))
  kDebug() << "Item named" << name << "not found";
```

- Where am I called from?

```
kDebug() << kBacktrace();
At runtime:
qdbus org.kde.foo /KDebug printBacktrace
```

  - Not very useful by default, due to –hidden-visibility.
  - Less useful than “bt” in gdb (which shows values).

debug statements in Qt itself

the door was open, I came in and made changes

Modify Qt to:

- Insert printf, qDebug/qWarning or even hand-made qBacktrace, to see all invocations of a given method.
- Add abort() to catch warnings from bad Qt API usage, like
  - “postEvent: unexpected null receiver” (often due to NULL->deleteLater())
  - “Calling appendChild() on a null node does nothing.” (kontact startup)
- Getting more info from Qt. “QAction::eventFilter: Ambiguous shortcut overload: Del”
  (impossible in gdb, better use qDebug patch from maelcum)

Observing facts: gdb

Genuine Debugging Beast

Demo: gdb ./ktoolbar_unittest

Additional tips

- fs (finish and step)
- set print object
- Don't "break qWarning".
  Use "break qt_message_output".

gdb konqueror

b 'KXmlGuiWindow::applyMainWindowSettings'
qs4 config.d.d->sOwner.d.d_ptr.fileName
Associating commands with breakpoint

Exercise: reading a bt with an assert

Aki crashes aquí

```
#8  abort () from /lib/libc.so.6
#9  qt_message_output(QtMsgType, char const*) () from /usr/lib/libQtCore.so.4
#10 qFatal(char const*, ...) () from /usr/lib/libQtCore.so.4
#11 qt_assert_x(char const*, char const*, char const*, int) () from /usr/lib/libQtCore.so.4
#12 QList<QString>::operator[] (this=0xbf8a991c, i=0) at /usr/include/QtCore/qlist.h:403
#13 Aki::Irc::Socket::connectToHost (this=0x821b528) at /home/me/akiirc/irc/socket.cpp:141
#14 ServerView::ServerView(struct QWidget *) (this=0x821d9e8, parent=0x80d49b0) at /home/me/aki/serverview
#15 MainWindow::MainWindow(struct QWidget *) (this=0x80d49b0, parent=0x0) at /home/me/aki/mainwindow.cpp:9
#16 AkiApplication::newInstance (this=0xbf8aa630) at /home/me/aki/akiapplication.cpp:76
```

Signals and slots

Where do I go from here?

Example

- Demo: gdb ./kdirlistertest, b 'KJob::emitResult'
- emit result(this);
- Which slot is this going to go into?
- call this->dumpObjectInfo()

Result

```
signal: result(KJob*)
 --> KIO::JobUiDelegate::unnamed _k_result(KJob*)
 --> KDirListerCache::unnamed slotResult(KJob*)
```

The right tool for the right bug

No hammer needed

- Inconsistent behavior

Example

kfiltertest would error on byte 15. Running it again, it errored on byte 18.
⇒ valgrind! Use of free'd data.

- Performance issue ⇒ callgrind+kcachegrind
- Memory leak ⇒ memcheck (valgrind) with –leak-check=yes
- Too much memory use ⇒ massif, see next slide
- Which files does it open? ⇒ strace -e open
- Which dirs does it look into? ⇒ strace -e access
- Which file is it reading/writing right now? ⇒ strace + /proc/PID/fd

Too much memory use

This application is sponsored by RAM makers

Recommended way to check actual memory usage

grep VmData /proc/‘pidof kcomboboxtest‘/status
VmData: 15012 kB

Using massif to see where the allocations are

- alias massif=valgrind –tool=massif –alloc-fn=’qMalloc(unsigned long)’
- massif ./kcomboboxtest
- ms_print massif.out.10355 | less

callgrind can also show how many times each method is executed.

Sockets

Don't put your fingers into the socket

Example: strac'ing "kwrapper4 ksmserver" shows it's doing a blocking read on fd 3.
What is that, and who can write to it?

- `strace -p ‘pidof kwrapper4‘` says `read (3`
  (in case of select, use the numbers in [])
- `/proc/‘pidof kwrapper4‘/fd/3` says
  `socket:[89758]`
- `netstat -pn | less -p 89758 says`

```
... 89759  11844/kdeinit4   $KDETMP/ksocket-me/kdeinit4__0
... 89758  11872/kwrapper4
```

- ⇒ `kdeinit4 is the one writing on that socket.`

Thanks Thiago!

Outline

Assert expectations

The impossible cannot happen, and yet it does

Hypothesis says “the impossible actually happened”
$\Rightarrow$ add assert, recompile, re-run.

- It will validate/invalidate the hypothesis.
- It will prevent such an infection from happening in the future.

Add method to check class invariants, call from all places where state should be sane.
Q_ASSERT(sane()); // idea: print lots of debug before returning false

Outline

Fix the bug!

using crazy glue or a frying pan

Outline

OK the fix works, now think about...

Don’t go home just yet

- how to unit-test the problem (revert!) and the fix
- the initial intent of the code (svn annotate + svn log!) and the opinion of the author
- other cases affecting the same code
- other bugs (that this might not fix, or introduce)
- other places where this might happen
- other people (explain the problem and the fix)
- the users (document fix in bugzilla and in the changelog)
- the kittens (hi Luboš)

Devil's guide to debugging
666

- Finding the defect by guessing
- Fixing without understanding the problem
- Adding workaround after the problematic code

Resources
For more info

- Book “Why programs fail” by Andreas Zeller
- Nove Hrady Presentation on debugging (callgrind example)
  `http://kdab.net/ dfaure/conf/n7y/Debugging/html`

Questions ?

David Faure
faure@kde.org