



What Symbian OS Panics Cause Trouble When Creating a Phone?

Version 1.0
April 23, 2008

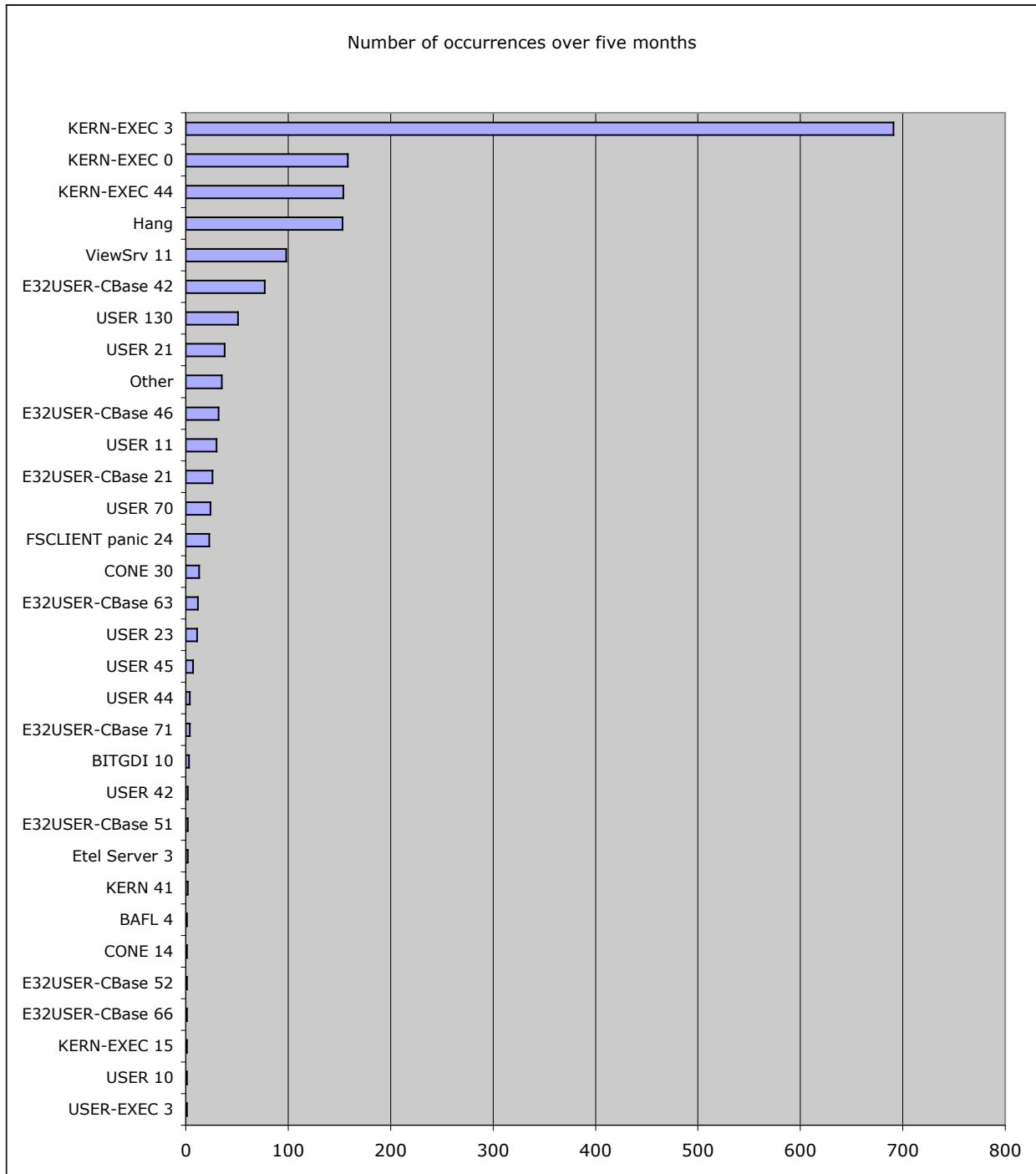
1. Contents

1. Contents	1
2. Introduction	1
3. Panic frequency	2
4. The usual suspects	3
5. Effort required to solve	3
6. The scary suspects	4
7. How to tackle the toughest offenders	6
8. Further information	7
8.1. Revision history	7

2. Introduction

This white paper presents statistics on which types of Symbian OS panics cause most trouble when producing a phone, and how best to avoid those problems.

3. Panic frequency



This chart shows the number of panics¹ which occurred during five months of the development cycle of a Symbian OS phone. It includes all panics which were produced by any software on the phone, whether belonging to Symbian, the device manufacturer, or any other party.

(It includes only panics with recognised, documented, Symbian panic codes. There were also a number of other panics, most of which had panic codes specific to the device in question).

It also includes device hangs which prompted the device to be rebooted by its own watchdog timer.

¹ The number of panics, not the number of reasons for each panic (which will be lower, as some panics will recur before they have been fixed. This is more likely for common panics such as KERN-EXEC 3).

4. The usual suspects

The top ten panics were:

Proportion	Panic	Notes
42%	KERN-EXEC 3	This indicates some hardware exception: usually a NULL pointer dereference, or some other invalid pointer.
10%	KERN-EXEC 0	This indicates that a Symbian OS kernel handle is being used without it being valid.
9%	KERN-EXEC 44	A bad message handle was passed to the Symbian OS kernel.
9%	Hang	The device may hang and become unresponsive for many reasons. Usually it means an infinite loop or deadlock in an interrupt or a DFC thread.
6%	ViewSrv 11	The view server panics an application with this code when it doesn't respond quickly enough to a request. It usually means a particular application has hung.
5%	E32USER-CBase 42	Multiple requests have been made on the same active object.
3%	USER 130	Array index out of bounds.
2%	USER 21	Descriptor (string) index out of bounds.
2%	E32USER-CBase 46	"Stray event". Some thread has been sent a reply to a request, without being ready to receive that reply.
2%	USER 11	Descriptor (string) overflow.

This is a fairly common distribution for a Symbian OS phone development project.

5. Effort required to solve

The raw percentages of panics masks the fact that some are much easier to fix than others.

A KERN-EXEC 3, for example, is easy. Even if you can't reproduce the crash, it's possible to identify from a Symbian OS crash dump exactly which variable was NULL, and from that it's usually easy to identify the fix.

On the other hand, a hang requires much more effort – because you have to work out why the device is not responsive. But, even this is not too bad, as it's usually possible to get a crash dump showing which threads are locked in some sort of loop or deadlock.

The grand-daddy of all crashes is the Stray Event. Every Symbian OS device creation engineer quakes in their boots at the mention. A stray event panic means some thread has received an event it wasn't expecting. But why wasn't it expecting it? Who knows. The mistake was probably made seconds, minutes or days ago, and the code might now be in a completely different place. There's very little evidence to work out what went wrong. The only options are to put in place extra monitoring tools and struggle to reproduce the problem, or to read reams of source code to try to spot the mistake.

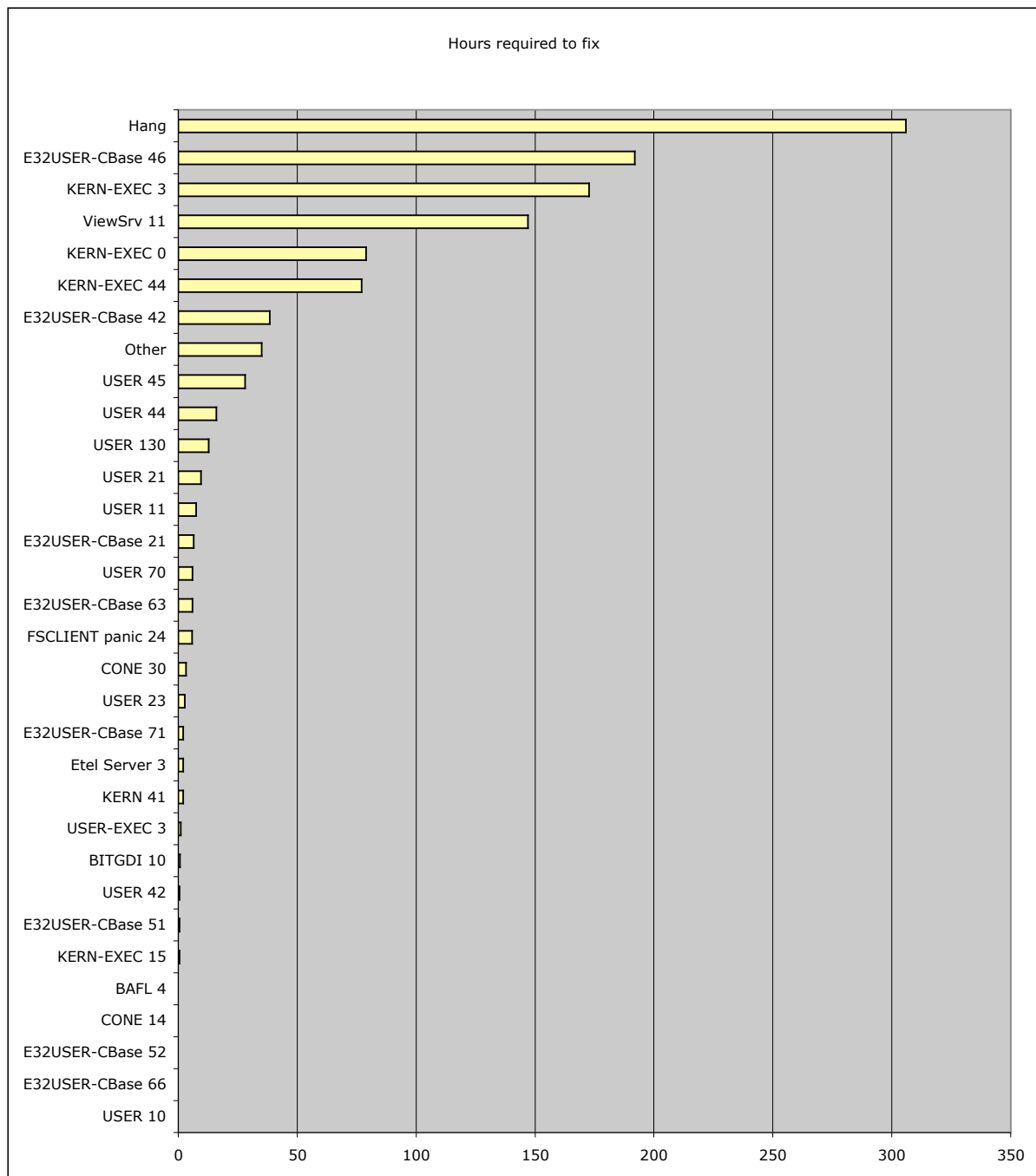
Each of these panics can be categorised as:

- **Call stack provided:** a crash log will contain a call stack identifying the exact line of code which went wrong. Estimated diagnostic time: 15 minutes².
- **Call stack provided, but investigation required:** a call stack shows where the problem occurred, but it may not be immediately obvious why an object is in the state it's in. Estimated diagnostic time: 30 minutes.
- **Application hang:** a ViewSrv 11. A call stack is provided of the hung application, which may or may not make it easy to work out why it has hung. Estimated diagnostic time: 90 minutes.

² These time estimates are solely based on the recollection of a Macrobug engineer who was tasked with analysing around a thousand crash logs in a device creation project a few years ago.

- **System hang:** it may be possible to capture a call stack for each thread, but it takes time to read the call stacks to work out which threads are causing trouble. Estimated diagnostic time: 2 hours.
- **Invalid heap accesses.** Sometimes this is easy to diagnose, because an invalid pointer has been passed to the heap (and again, a full callstack is provided). Sometimes it indicates some terrifying memory corruption problem. Estimated diagnostic time: 4 hours.
- **Stray event:** who's the daddy? The reason for the crash is lost in history. Estimated diagnostic time: 6 hours (if you're lucky).

When you multiply the diagnostic time by the frequency of occurrence, you get:



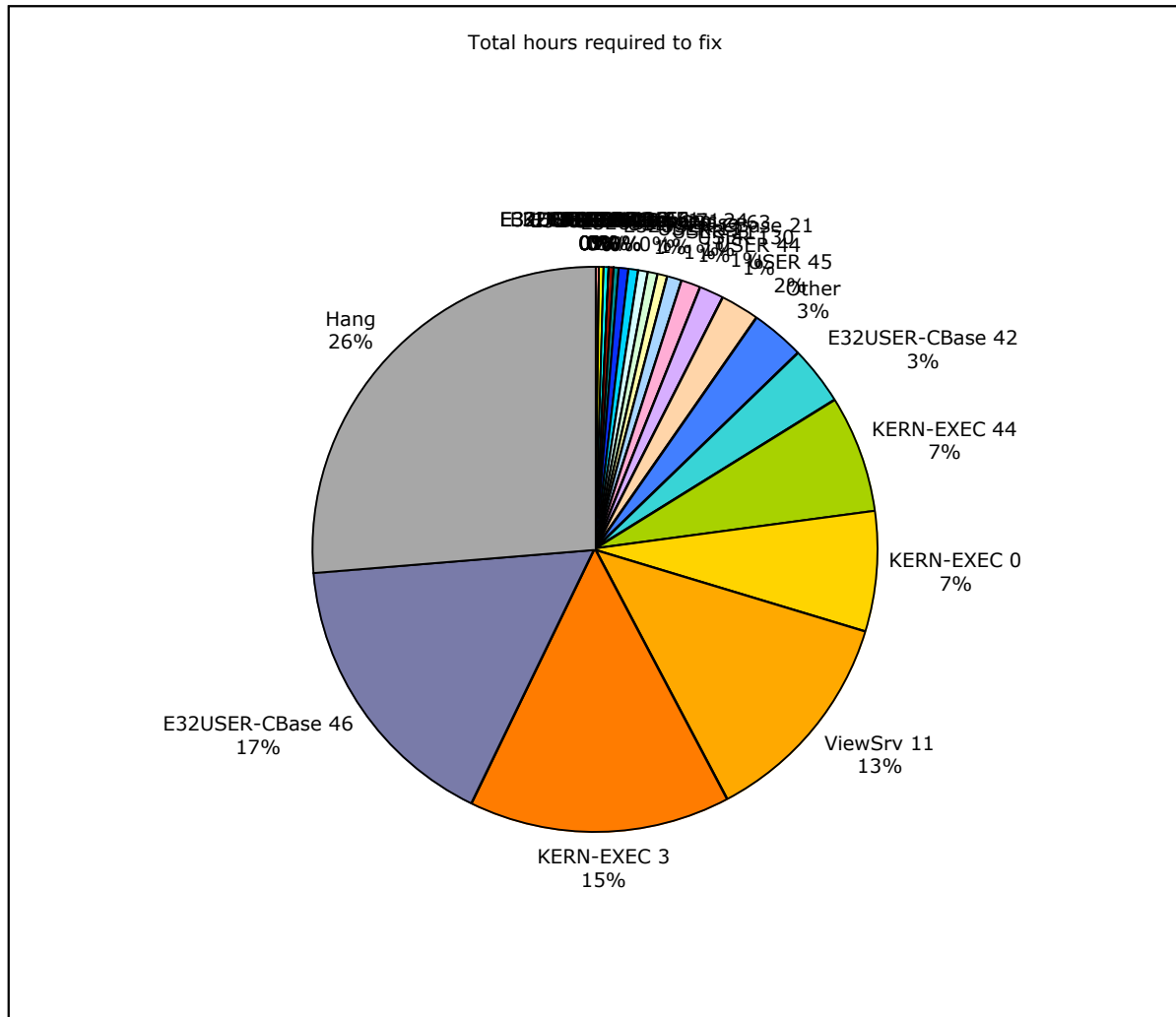
6. The scary suspects

Suddenly, the top ten is different!

Diagnostic time %	Panic	Estimated diagnostic time per incident	Notes
26%	Hang	2 hours	Hangs are always going to be difficult.
17%	E32USER-CBase 46	6 hours	Stray events. Not so common, but a nightmare to solve.
15%	KERN-EXEC 3	15 minutes	NULL pointer dereferences and similar. Usually easy to solve, but their sheer number puts them here.
13%	ViewSrv 11	90 minutes	Application hang.
7%	KERN-EXEC 0	30 minutes	A bad Symbian OS kernel handle. Usually easy to solve but requires investigation. Also very common.
7%	KERN-EXEC 44	30 minutes	Bad message handles passed to the kernel. Usually fairly easy to solve but requires investigation.
3%	E32USER-CBase 42	30 minutes	Multiple requests on the same active object. Usually quite easy to solve, though sometimes you need to go back in history to work out why the first request was made.
2%	USER 45	4 hours	Our first new entrant. Indicates somebody asked to free a invalid heap cell. This is one of those which might be a simple innocent explanation, or might be a
1%	USER 44	4 hours	Very similar to USER 45.
1%	USER 130	15 minutes	Array index out of bounds. Easy but frequent.

Obviously, once the problem is diagnosed, extra time is required to identify a fix.

Here's another view of the same data.



7. How to tackle the toughest offenders

Can any of these problems be prevented? Yes. Some of these issues can be detected using source code analysis tools, before the code even runs on a phone.

Four products will be discussed here:

- **Carbide.c++ CodeScanner.**
CodeScanner has the big advantage that it's provided with the Integrated Development Environment most Symbian OS developers use every day – Carbide.c++.
- **Coverity Prevent.**
Coverity Prevent is a source code analysis tool applicable to many operating systems. It works with Symbian OS³.
- **Klocwork.**
Klocwork⁴ is another static analysis tool which can be made to work with Symbian OS.
- **Macrobug Stray Event Scanner⁵.**
A more specialist tool which only reports potential causes of E32USER-CBase 46 panics.

³ http://www.coverity.com/html/prod_map_dna_c.html

⁴ <http://www.klocwork.com/>

⁵ <http://www.macrobug.com/products/strayscanner/>

Diagnostic time %	Panic	How to detect in advance	Approximate % detectable
26%	Hang	Impossible?	0%
17%	E32USER-CBase 46	Macrobug Stray Event Scanner is the only tool which can detect causes of stray events from source code alone. It reports any situation where a thread makes a request without getting itself ready to receive the reply, so should in theory be able to identify approximately 100% of E32USER-CBase 46 panics from the source code alone. But let's say 95% to be conservative...	95%
15%	KERN-EXEC 3	This is not normally a Symbian-specific problem, so excellent general C++ analysis tools such as Coverity or Klocwork would be good tools for this.	80%
13%	ViewSrv 11	Impossible?	0%
7%	KERN-EXEC 0	No known tool yet.	0%
7%	KERN-EXEC 44	No known tool yet.	0%
3%	E32USER-CBase 42	Some of these are detected as a side-effect of the Macrobug Stray Event Scanner.	25%
2%	USER 45	These two indicate an invalid heap pointer (if you're lucky). If so, they should be detected by Coverity or Klocwork. They may also be detected by CodeScanner if the problem is caused by a failure to follow Symbian OS coding standards.	25% ⁶
1%	USER 44		25% ⁶
1%	USER 130	Should be detected by Coverity or Klocwork if they're good!	100%

So, the diagnostic time savings by each tool are:

Either Coverity or Klocwork	162 hours, or 14% of the total diagnostic time
Macrobug Stray Event Scanner	192 hours, or 17% of the total diagnostic time
Macrobug Stray Event Scanner, <i>and</i> Coverity or Klocwork	354 hours, or 30% of the total diagnostic time

Detecting these bugs earlier through the use of static analysis tools also makes it cheaper to fix them.

Macrobug would therefore strongly recommend that if you're in the Symbian OS device creation business, you apply both Klocwork/Coverity *and* Macrobug's stray event scanner to your code.

8. Further information

All trademarks are the property of their respective owners.

8.1. Revision history

Revision	Changes
1.0	First revision

⁶ Hopefully more than 25% will be detected, but they will probably be the easier ones to fix, so we estimate only 25% of the time will be saved.