Kjellkod's Blog

KjellKod.cc : Always strive for excellence

Is it possibly to get a stacktrace to file from a signal handler?

Posted on January 21, 2015

I got a great question from a reader of the g2log article on CodeProject regarding g2log's implementation of a crashhandler.

> *[…]new, delete and IO operations are not safe in signal handlers and could cause unpredictable behavior. I checked your implementation and maybe I am wrong but it seems this is exactly what you did (write to file/ empty queue within the signal handler). Is that by design, am I wrong or you missed this.*

Of course the reader was completely right. It is **not safe**, it is one of those "*please don't try this at home*" kind of things. With "safe" I here mean, "Posix guaranteed safe".

Is there **any** "safe" way at all of getting the stacktrace to file after a fatal signal has happened?

Below you can read my answer and rationale behind the crash handling in g2log and g3log. If you know of better ways of doing this then please email me, send me a pull request to BitBucket (g2log or g3log) or just tell me here.

Until then I will stick to the rationale that since the process is crashing it is worthwhile taking the very, very slim chance of triggering a crash from an *unsafe* function call in order to get the stacktrace. A post-mortem reading of the stack trace if worth more than the tiny chance that we would crash-within-the-crash due the implementation of the signal handler.

Long answer to reader / Copied from CodeProject

> *When a fatal signal comes, you are screwed anyway, a post-mortem stacktrace goes a long way of making sure it doesn't happen again. It is worthwhile attempting to get the stacktrace even if it means using functions that technically are not considered kosher.*
>
> *I've outlined my rationale below with some useful links if you want to read up some more on it.*
>
> *Getting a stack trace in a signal handler is not considered* safe *because functions such as new, delete, malloc, free are commonly used. The* worst case scenario *is when malloc itself is triggering a fatal signal. Then the signal handler would be called, which in the* `backtrace()` *call would use malloc.*
>
> *The **good news** is that fatal signals from* `new/delete/free/malloc` *are in my experience fairly unusual. At least compared to all the other reasons why fatal events happen. I.e. bugs.*
>
> *And if we don't get a fatal signal from* `malloc`, `new` *et.al then we are on fairly safe ground.*
>
> *It is nothing* `magical` *why most function calls in a signal handler are discouraged. Non-reentrant functions, global state etc are all things that must be avoided in case of a new fatal signal would happen, while within a signal handler.*

***calculated risk** we are taking.*

As far as I know there is no open-source or closed-source way (yet) that can get a stacktrace in a 100% posix "safe" way. Last time I checked even catching a signal in a multi-threaded application is undefined behaviour. Still we do it! Just because the standard doesn't declare something as "safe" or "defined" doesn't mean it can't be done 99.99% of the time.

Here are some good links with explanations that I found very useful

1. Very good description of signal handler usage, with one example of when using the `signalhandler unsafe printf`, but doing it in a completely safe manner.

http://www.ibm.com/developerworks/library/l-reent/

2. Blog from a Chromium developer, who is making a stack dump without (much) dynamic allocations.
http://phajdan-jr.blogspot.com/2013/01/signal-handler-safety-re-entering-malloc.html

This is awesome work and I could go this way but frankly unless I hear complaints from the users that this is an issue I will leave it alone. If I go down this path it would probably mean making my own implementation of `backtrace()`,. it would be great for the software community at large but take many, many hours of my spare time.

Of course, as always you are more than welcome to help out with improvements either through suggestions here, email me or by doing a pull request to github.com/KjellKod/g3log

Rate this:

Rate This

Like

Be the first to like this.

Related

G3log: Asynchronous logging the easy way
In "g3log"

Kjellkod's g2log vs Google's glog: Are asynchronous loggers taking over?
In "C++"

The world's fastest logger vs G3log
In "g3log"