Subject: Crazy(?) idea about debugging
Posted by Mindtraveller on Wed, 18 Mar 2009 14:19:41 GMT
View Forum Message <> Reply to Message

Sometimes you have crash and have no idea what really led program to the exception. Let`s imagine we have a debug version of our program running with debug version of memory mamanger, which "knows" where and which variables are situated. Let`s imagine we have a background thread which is continiously passing all the variables making their snapshots i.e. once per 50 milliseconds. This thread keeps "history" with something about 100 snapshots per variable. So the idea is, when exception is executed, memory manager does some analysis. Analysis covers all the variables. If variable isn`t changed through it`s history, it is thrown out from list. More variable is changed through it`s history, closer to top it is. In the end we will have something like "crash story" with top variables from list, which most likely led to the exception.

Subject: Re: Crazy(?) idea about debugging
Posted by mr_ped on Wed, 18 Mar 2009 15:01:52 GMT
View Forum Message <> Reply to Message

Interesting.
But things to keep in mind:
- a good old full memory dump would lead to the cause of exception in ordinary case faster, at least the exact exception instruction is easier to fetch, the true cause may be more hidden. (and it may be lot more hidden than history of 100 values anyway)
- with non-atomic variables you may get half way updated values in history, it would be next to impossible to keep those data consistent unless you make cooperative call to the history keeper to let him know when it's safe to store history (at that point the whole concept becomes too cumbersome, you can do that on your own with your own data history watcher and calling it periodically).
- lot of C/C++ code works with local variables at heap, they will be very difficult to follow+store in the history manager.

I *believe* unit testing with QA level tests (i.e. not just 100% code coverage for TDD, but additional quality tests with wider range of data) is better and cheaper safety net, although it's far from universal solution, because some problems are difficult to unit test. Also QA level of tests gets into the way of TDD, so it makes sense to use just minimal set of tests to do TDD and after the module does reach some "1.0" version, put it under QA tests with additional range of data. After that minor refactoring should change all the tests (a bit cumbersome, but usually tests don't change as much as code, because usually you want to keep the output same, so it's workable), or in case of huge refactoring it's probably better to drop QA part, work with just original minimal set, and then introduce new QA tests for "2.0".

This should bring the need of such idea to minimum, because such well tested code usually doesn't crash at all, or does crash from such complex conditions and hidden faults, that data history with 100 last values may be completely useless to figure out where it went wrong.

... sorry if I sound too negative, maybe I need more time to let the idea sink down to appreciate it more.

Subject: Re: Crazy(?) idea about debugging
Posted by Mindtraveller on Wed, 18 Mar 2009 15:17:37 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Wed, 18 March 2009 18:01... sorry if I sound too negative, maybe I need more time to let the idea sink down to appreciate it more. Good critics is the most invaluable thing, thank you! I agree that the best way no to have such difficult errors is to change the approach to

should be supported on language level. At worst, such things should be supported by IDEs. May be we could think of some innovations on that level.

Subject: Re: Crazy(?) idea about debugging
Posted by mr_ped on Wed, 18 Mar 2009 19:51:28 GMT
View Forum Message <> Reply to Message

supported on language level. At worst, such things should be supported by IDEs. May be we could think of some innovations on that level.

Well, I'm quite happy with cooperation of U++ with Bazaar/UnitTest++, although I can imagine some improvements to make it "production quality" and integral part of TheIDE.
But for my particular needs it works quite ok. Especially I like it in conjunction with U++ nest/packages philosophy, because it allows very easily to have test suites separated from actual code. Whenever I start new TDD package, I simply create two new packages, one is the real new package containing the code itself, the other one uses this real package, and adds all the Unit tests. When I do TDD, I work inside the test package (with the real package easily available for edit, like if it would be selected as main package) (so far so good), after it's finished I simply switch to "real" package and there we go, no tests at all without any UnitTest++ include (this is where the power of packages really shines). There are some minor details to solve like main in both packages, etc, but those are easy to solve.

Still as I said, I can imagine some improvements to it and better IDE integration. Also maybe I should finally bump the UT++ version to current SVN, but recently I work on many different things and this is of low priority to me. The key word is your "stimulate", you absolutely hit the nail at the head with this.

Anyone: if you are using this Bazaar package and you would like to get current SVN, send me note so I know somebody else does care, if I'm the only one using it, I don't care right now too much.

Subject: Re: Crazy(?) idea about debugging

Posted by mr_ped on Wed, 18 Mar 2009 20:05:12 GMT
View Forum Message <> Reply to Message

To get back on topic, at least partially... John Carmack (ID Software) once said that during work at Quake 1 (if I recall correctly) he was tampering with the game replays improvements over old DOOM (which already did allow player to record gameplay movie and then replay it). It was already based on deterministic game engine behavior, so only user inputs had to be stored. I think DOOM does use fixed rate (60Hz?) user input sampling and that's what goes into replay file too.

So while he was on it, he decided to treat timer results just like another user input, i.e. the game timing was part of input data structure and processed centrally in the same routine as user keys and mouse. According to his own words, this improved reproducibility of game crashes from replays a lot, because he was now not only able to simulate player's movements, but also his framerate and the graphical effects become deterministic too, not just gameplay physics events.

This is IMHO very familiar to your idea, but works only for strictly deterministic application which react to limited amount of input data. If your app fits this description, you should probably think about central input processing routine, making everything what matters in it's run part of input, and then save a "replay" in every (debug) run, so in case of crash you can run a replay with the same input and see if it leads to crash again.

As UPP is also GUI library handling user clicks and keys, maybe having some integrated package to create "replays" of user would be possible. But it would need to cover many aspects of "input", including timers and maybe something more. It would be quite a challenge to make it universal enough. But it would help both with debugging, and with GUI automated testing.

Subject: Re: Crazy(?) idea about debugging
Posted by mirek on Wed, 18 Mar 2009 20:38:15 GMT
View Forum Message <> Reply to Message

Well, indeed. If I can reproduce the error, fix is usually matter of minutes (ok, sometimes hours, but rarely weeks

mr_ped wrote on Wed, 18 March 2009 16:05
As UPP is also GUI library handling user clicks and keys, maybe having some integrated package to create "replays" of user would be possible. But it would need to cover many aspects of "input", including timers and maybe something more. It would be quite a challenge to make it universal enough. But it would help both with debugging, and with GUI automated testing.

Well, we have something. It is not complete replay, but it has helped me to crack some bugs.

ActivateUsrLog();

will start dumping various informations about user GUI actions in text format. Sometimes you can restore the path leading to the error that way.

Mirek