Subject: Re: C++ FQA
Posted by exolon on Tue, 13 Nov 2007 17:58:55 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Tue, 13 November 2007 17:24As I said, GC *can* globally be more performant, and *can* be in the very short time very performant too. What i can't is assure the performance *all* the time.
If I do a benchmark with a lot of allocations/deallocations, GC may behave wonderful. But then, if you stress benchmark beyond a threshold, you'll have your app stopping for maybe 2-3 seconds collecting stuffs. If you go forth, global app time *can* be shorter than with manual allocation, but in the middle you had the infamous stop.

Besides real-time tasks (on wich that's not acceptable), there are also many apps on which you couldn't accept such a behaviour, like multimedia, but even in a GUI a 3 second stop makes a bad feeling.
I find it hard to believe that modern GC implementations would cause a lag of 3 seconds in a normal GUI or even multimedia app.
You talk about a stress test, but surely this isn't really the general case?

So it looks to me like you're talking about special, extreme cases; the worst-case scenarios for a stopping GC (there are incremental GC schemes too, but I don't see that they exist in the real world).

Do you really prefer handling all the memory (de)allocation, when objects are being passed around and you don't know how many times a function will necessarily be called... in every case? In complex programs?
I respect your choice to favour whatever mechanism, manual or GC (and I certainly don't think you're stupid if your choice is different to mine - I make stupid choices all the time) but I really don't think the overhead is as bad as people make out.
If it turns out that GC is as good or nearly as good as manual management in 90% of cases, then I would definitely be going that route, unless we're dealing with cases where absolutely no unexpected latency can be introduced whatsoever... i.e. real-time with hard deadlines, in which case we'd probably be using something like C in uCOS or even assembly, anyway.

That said, I do understand that there's a non-performance-related problem with calling destructors during GC - namely that of two objects about to be collected, say FileReader and FileHandle, FileReader has a reference to FileHandle, and in its destructor calls handle->Close() or something, which will die if the FileHandle happens to be destroyed first (since the only reference to it is another 'dead' object), and I guess this might be what Luzr alluded to.
So I'm not sure what the best solution there is, without explicitly calling some "finalise" type method which would be just as annoying as deleting it anyway.

[edit]
Afterthought... then again, why not just have Close() in FileHandle's destructor, rather than having the client object do the cleanup?
Anyway, I'm sure there are cases out there that are more valid than my crappy example.