Subject: Re: Ideas on U++ as library
Posted by cbpporter on Sun, 20 Dec 2015 16:43:46 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 20 December 2015 12:40Well, I suggest starting with allocator....

It is the least controversial thing and relatively simple to 'extract'.

Technically, there is one problem though to be resolved: When thread exits, heap cleanup function has to be called.

That was easy to do in U++, where we have our own Thread class, but will need to be resolved for standalone allocator.

It looks like pthread_key_create resolves issue in Posix (maybe...), but I do not see equivalent in Win32 :(Worst scenario - it has to be documented.... :)

Other than that, the endproduct that I would like to produce here is single 'amalgamated' .c file (like sqlite3). You include it in .cpp or add to project and you get your application running faster, using less memory. Or include in .c and get 'super_malloc', 'super_free' functions. With something #define JUST_DECLARATIONS, you can also include it in header.

So the task would be mostly about

- convert current code in C++ to be .c compilable. Tedious, but possible
- in the process, add more comments, including algorithm overview
- create 'amalgamating export' utility that will take current Core and export amalgamated allocator code as single .c file (or maybe some other extension)
- create website section for it, publish :)

Mirek
Maybe it should be clearly set in stone what we are trying to achieve with this. There where multiple reasons proposed in the larger thread for making libs. One goal was to cut out small useful parts of U++ and offer them as libs to people who don't have this functionality. If this is the goal, then anything goes, like C, but this is mostly useful for non U++ users. Another proposed reason was to increase the modularity and accessibility of parts of U++ for random environments, and that is all about having the maximum power done in the most U++ way without forcing people to use a full package and TheIDE. This is useful for all people, maybe even more to U++ users. If this is the case, the allocator that will be created from this attempt should be C++, with the best and leanest OOP design and be well documented. And created with the expectation that while not all people will use Thread, most will, since outside of boost there are no threads in C++.

So which approach shall we take?

And I also think that before we get started we answer both the previous question and decide on what libs to create and how? While U++ is not popular, it is a good product and it would be a shame to ruin it or increase its complexity with some ad-hoc decisions meant to attract new people.