Subject: Ideas on U++ as library
Posted by dolik.rce on Fri, 18 Dec 2015 13:40:58 GMT
View Forum Message <> Reply to Message

Moving this discussion here, so I don't hijack original discussion about PR.

mirek wrote on Fri, 18 December 2015 11:01dolik.rce wrote on Thu, 17 December 2015 21:08
I remember some experiments with converting upp files to Visual Studio projects, waf based builds and of course the universal makefile, but frankly, all of those are quite difficult to use without previous knowledge of U++. Making a library that could be just linked against would help a lot, but I know this has already been tried several times before, with mixed results.

Well, I am afraid this is still mostly the question of INITBLOCK.

Should we drop INITBLOCK and require manual registration of modules used?

Mirek

Do you mean that INITBLOCKs cause trouble with building icpp files or that they are not called in library? I believe the first issue is more or less solved, so I'll assume the second is what you meant. Correct me if I'm mistaken.

I haven't really think any of this through, but here are some ideas:

1) Some script during library build goes through all code and gathers INITBLOCKs, then generates code that executes them in init function/DllMain. It might require some changes to how INITBLOCK macro is defined when building as library, but that should be simple.

2) Ignore INITBLOCKs when building library and allow manual registration of everything the user needs. This would require good documentation of what can registered  When using static linking, everything could still work as of now.

3) Putting all INITBLOCKs content into separate header files. It would then be users responsibility to include necessary headers exactly once into an application. This would make the manual initialization quite simple (one include per package) and it would also simplify compilation - you'd just need to special handle one file that includes all the initialization headers, instead of all icpp files.

Any other ideas are welcome :) If I have some spare time during holidays, I'll try to experiment with this a bit.

Having a dynamic library would lower the barrier for newbies significantly. I believe many leave U++ before trying it because a) it seems to be slow, when building everything for the first time or b) it requires static linking.

Honza

Subject: Re: Ideas on U++ as library
Posted by mirek on Sat, 19 Dec 2015 07:55:35 GMT
View Forum Message <> Reply to Message

Well, there is one equally pressing issue:

What should be in the library?

Subject: Re: Ideas on U++ as library
Posted by dolik.rce on Sat, 19 Dec 2015 12:26:16 GMT
View Forum Message <> Reply to Message

mirek wrote on Sat, 19 December 2015 08:55Well, there is one equally pressing issue:

What should be in the library?

Well, I was thinking along the lines of one library per package. For simple experiments, linking to "libUppCore" would be enough, for GUI, you'd have to link "libUppCtrlLib". The dependencies between the libs would follow the same pattern as U++ packages, that would make the most sense to me.

It doesn't really feel natural to take away all the modularity of U++ by compiling it all into single huge library, so I'd try to avoid that if possible. In any case, I think "libuppcore" would be a good place to start experimenting. We could figure out the rest when we get there :)

Honza

Subject: Re: Ideas on U++ as library
Posted by mirek on Sun, 20 Dec 2015 10:40:13 GMT
View Forum Message <> Reply to Message

Well, 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

---

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.

---

Subject: Re: Ideas on U++ as library
Posted by dolik.rce on Sun, 20 Dec 2015 19:28:54 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Sun, 20 December 2015 17:43Maybe 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.
This is very good point. When I started this thread, I was thinking mainly about the second case you mention, that is to increase the accessibility, while keeping things U++ way - fast, simple and modular.

Concerning the allocator, I believe it could simply provide both C and C++ API. Both is easily possible from C++ library. My original idea was something like jemalloc, which can be simply preloaded before loading any other program. It overrides the calls to malloc/free, new/delete and other memmory handling functions and provides better performance than the standard version without even recompiling the application. It is super easy to test (for jemalloc it is just calling "LD_PRELOAD=libjemalloc.so.1 MyApplication") so  people can see the performance boost quickly and may even start digging into the underlaying U++ technology.

cbpporter wrote on Sun, 20 December 2015 17:43And created with the expectation that while not all people will use Thread, most will, since outside of boost there are no threads in C++.Actually there is std::thread in C++11 ;)

cbpporter wrote on Sun, 20 December 2015 17:43So 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.

I think those two approaches might go in parallel. Mirek is probably the right person to separate the allocator (I quickly tried today and it seems to be quite tagled with other Core code), while I, or anyone else can try to turn the U++ packages into libraries. Both task are IMHO important.

Honza

---

Subject: Re: Ideas on U++ as library
Posted by cbpporter on Sun, 20 Dec 2015 20:39:16 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Sun, 20 December 2015 21:28
cbpporter wrote on Sun, 20 December 2015 17:43And created with the expectation that while not all people will use Thread, most will, since outside of boost there are no threads in C++.Actually there is std::thread in C++11 ;)

There is? Sorry, I'm really behind with my C++11. The thing is I have pretty much given up hope on C++. By the time the comity gets the balls to turn C++ into an elegant productivity language, I've moved on hopefully to something better.

---

Subject: Re: Ideas on U++ as library
Posted by mirek on Mon, 21 Dec 2015 09:05:47 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Sun, 20 December 2015 17:43
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.

If there is something I can guarantee, any efforts like this will NOT affect U++ usability.

The only thing I plan here is to remove C++ constructs from heap allocator, which will lead to slight increase in code complixity (I estimate 10-20%), but nothing serious. Mostly, it is just about removing nested structs and replacing methods with functions (with _this parameter :). This will make possible to reuse the code for C easily. About the worth thing I might do is to add some comments meant as markers for 'export' utility.

---

Subject: Re: Ideas on U++ as library
Posted by cbpporter on Mon, 21 Dec 2015 16:51:40 GMT
View Forum Message <> Reply to Message

Well I did spend the better half of my Sunday night googling allocators, including jemalloc and watching this: https://www.youtube.com/watch?v=RcWp5vwGlYU

I'm afraid if the first step is the allocator, it needs to be accompanied by a benchmark too :).

Subject: Re: Ideas on U++ as library
Posted by mirek on Mon, 21 Dec 2015 19:52:23 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Mon, 21 December 2015 17:51Well I did spend the better half of my Sunday night googling allocators, including jemalloc and watching this: https://www.youtube.com/watch?v=RcWp5vwGlYU

I'm afraid if the first step is the allocator, it needs to be accompanied by a benchmark too :).

No problem. But you might help, it is fun.

Starting point might be benchmarks/SimpleAlloc.

I yet have to invent some MT benchmarks.

Subject: Re: Ideas on U++ as library
Posted by Novo on Tue, 22 Dec 2015 02:39:22 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 21 December 2015 14:52
I yet have to invent some MT benchmarks.

This code can be useful, IMHO.