Subject: Proposal: optional int64 size/indexes
Posted by Mindtraveller on Sun, 16 Aug 2015 08:08:34 GMT
View Forum Message <> Reply to Message

Hi,

Looking into U++ sources I've found many places where indexes of size variables are represented by int type members.
As we all know, 64-bit hardware is more and more widely used, so x64 support is crucial for U++. Yes, U++ is succesfully built with all latest C++ x64 compilers including MSVC, GCC and CLANG/LLVM. But 64-bit compatibility is more than just compilation. We need support for large indexes, large addresses and large sizes inside U++ containers and classes.

For example, FileMapping maps file using int64 as size (which is perfect), but FileMapping offset routines use int variable as index. Obviously, it doesn't support int64 indexes out-of-the-box.
The same is for U++ containers and other classes.

It would be too easy to say "OK, let's just switch to int64 everywhere we can". But it is not the best way.
Just because efficiency is a target. And more of that, U++ is used on embedded systems where simple int is preferrable.

What I propose is two possible options:

1) To add more int64-based member functions for U++ containers and classes.
For example: operator[](int64), etc.

2) To use something like INDEX typedef for internal U++ indexes and sizes to switch between int and int64. The switching is done with compilation flag like INT64SIZE.

Thanks
Pavel

Subject: Re: Proposal: optional int64 size/indexes
Posted by mirek on Mon, 17 Aug 2015 07:39:09 GMT
View Forum Message <> Reply to Message

Well, I spent a lot of time thinking about this in the past...

For now, my position is that yes, FileMapping should have these members fixed (probably using size_t instead of dword there). If there are any similar issues in the U++, please report them as bugs.

Anyway, situation is much less clear w.r.t. containers. For starters, consider this: Containers now support up to 2G of items. Now if you are going to store just pointer in the Vector, sizeof(void *) is 8 bytes, which results in 16GB of flat memory block. My current machines (desktop, notebook) have 8GB of physical memory... And that is just Vector, e.g. Index has inherent memory

consumption of about 24 bytes per item. (-> 2G of items is now about 64G of memory).

On the other side, increasing index size from 4 to 8 bytes is not as innocent as it might sound. For example, those 24 bytes per item in Index (and e.g. VectorMap) is now 48 bytes. That is pretty high price to just have a good feeling about "hey, we are really 64-bit now!".

Also, quite minor issue is that while AMD64 ISA is fully 64-bit and int64 is as fast as int32 in most operations that count (64 bit division is slower than 32bit though, which can have impact on some hashing), their opcodes are one byte longer.

IMO, in reality, 64 bit is in fact about being able to access much more memory directly than to have arrays larger than 2G items. Which is what we do now just fine.

All that said, I understand that in some circumstances (e.g. big iron server with 256G memory doing some really serious shit...) it would be helpful to have containers without 2G restriction. My conclusion was that we in fact would need separate set of containers here, "LVector, LArray, LIndex"... etc.

Subject: Re: Proposal: optional int64 size/indexes
Posted by koldo on Mon, 17 Aug 2015 12:31:16 GMT
View Forum Message <> Reply to Message

I agree with Mirek. In this case do not forget that size_t is unsigned, but to handle pointer arithmetics, ptrdiff_t is the right signed type.

Subject: Re: Proposal: optional int64 size/indexes
Posted by Mindtraveller on Sat, 29 Aug 2015 16:37:28 GMT
View Forum Message <> Reply to Message

Good point! I agree with you.
