Subject: Core 2019
Posted by mirek on Fri, 07 Jun 2019 11:56:26 GMT
View Forum Message <> Reply to Message

I have made some substantial changes to Core memory allocator and index, improving performance of some synthetic benchmarks.

Allocator now much better handles big blocks, which improves e.g. performance of adding ~20000 elements to Vector<int> 3 times. Also, memory pages of most categories can be now reused in another category. We have now 3 categories of blocks <1KB, <64KB and <32MB/220MB (32 bit cpu/64 bit cpu). MemoryTryRealloc is now properly implemented and used in library. mingw performance is improved with TLS workaround.

sizeof(Index) is now 40 (was ~90). Adding elements to Index is now faster.

Frankly, in retrospective it was all mostly a lot of work for really small gains as all low-hanging fruits were already picked years ago. But large blocks handling in allocator is quite nice improvement...

Subject: Re: Core 2019
Posted by Novo on Fri, 07 Jun 2019 15:51:40 GMT
View Forum Message <> Reply to Message

Thanks a lot!

One of my data-intensive MT apps is running ~20% faster now.
It looks like it is using 4 to 6 times more RAM.
And I'm getting a timing report, which, probably, should be disabled:
TIMING Large Alloc 2 : 808.40 ms - 1.27 us (825.00 ms / 636928), min: 0.00 ns, max: 17.00 ms, nesting: 0 - 636928
TIMING Large Alloc : 1.97 s - 167.81 ns (2.27 s / 11734322), min: 0.00 ns, max: 28.00 ms, nesting: 0 - 11734325

Subject: Re: Core 2019
Posted by mirek on Fri, 07 Jun 2019 16:01:39 GMT
View Forum Message <> Reply to Message

Novo wrote on Fri, 07 June 2019 17:51Thanks a lot!

One of my data-intensive MT apps is running ~20% faster now.
It looks like it is using 4 to 6 times more RAM.

How do you measure it?

The new thing is that we now allocate 224MB chunks of _address space_. So virtual memory is way up, but that is not what physical memory use is....

Quote:
And I'm getting a timing report, which, probably, should be disabled:
TIMING Large Alloc 2 : 808.40 ms - 1.27 us (825.00 ms / 636928), min: 0.00 ns, max: 17.00 ms, nesting: 0 - 636928
TIMING Large Alloc   : 1.97 s - 167.81 ns (2.27 s / 11734322), min: 0.00 ns, max: 28.00 ms, nesting: 0 - 11734325

Thanks!

Mirek

Subject: Re: Core 2019
Posted by Novo on Fri, 07 Jun 2019 21:00:17 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 07 June 2019 12:01Novo wrote on Fri, 07 June 2019 17:51Thanks a lot!

One of my data-intensive MT apps is running ~20% faster now.
It looks like it is using 4 to 6 times more RAM.

How do you measure it?

The new thing is that we now allocate 224MB chunks of _address space_. So virtual memory is way up, but that is not what physical memory use is....

I'm using old-fashioned top (a Linux tool). I was looking at %MEM and at RES.
To be precise, the difference is ~2.75 times and not 4 or 6 times as I mentioned before.
I measured the same app compiled against git:40cd0fd5e (svn://ultimatepp.org/upp/trunk@13354) and git: 8e0f32d6262 (svn://ultimatepp.org/upp/trunk@13368)
With the old allocator I was getting 0.8% RAM max (~260Mb). The app was running for 292 s.
With the new one I got 2.2% RAM max (~714Mb). Now it takes 230 s. to run it. This is one minute less, and that is cool.

A singly-threaded version of the same app has improved a little bit as well: 2428.33 s. vs 2491.37 s.
The difference is ~2.5%

Subject: Re: Core 2019
Posted by Novo on Sat, 08 Jun 2019 16:30:29 GMT
View Forum Message <> Reply to Message

I couldn't compile code with the flag .USEMALLOC defined. I'm getting this:

error: use of undeclared identifier 'MemoryTryRealloc'
I just wanted to compare the new U++ allocator with the standard one ...

Subject: Re: Core 2019
Posted by mirek on Sat, 08 Jun 2019 16:31:04 GMT
View Forum Message <> Reply to Message

This is definitely something to investigate...

The working hypothesis is that you are allocating some really large blocks (>10MB) and in previous allocator, these were immediately unmapped back (and it was fast peak, so unnoticed while watching top), while the new allocator keeps them for reuse and system has not swapped them out yet. My experience is that the cuprit is usually a big StringStream.

We can test this. In HeapImp.h, there is HPAGE constant. This is the size of "master chunk" (in 4KB units) and also maximum size of block that allocator keeps for reuse. Try to change that to something smaller, like 256 and retest...

Mirek

Subject: Re: Core 2019
Posted by mirek on Sat, 08 Jun 2019 16:40:45 GMT
View Forum Message <> Reply to Message

USEMALLOC fixed

Subject: Re: Core 2019
Posted by Novo on Sat, 08 Jun 2019 17:44:30 GMT
View Forum Message <> Reply to Message

mirek wrote on Sat, 08 June 2019 12:40USEMALLOC fixed
For some weird reason I'm getting a linker error (full rebuild)

in function `Upp::sProfile(Upp::MemoryProfile const&)':
/home/ssg/dvlp/cpp/upp/git/uppsrc/CtrlLib/CtrlUtil.cpp:368: undefined reference to `Upp::AsString(Upp::MemoryProfile const&)'
Configuration: Debug (Release is fine)
Flags: GUI .USEMALLOC

I'm not getting any problems with linking when flags are "MT .USEMALLOC".
BLITZ is used in all cases.
This is weird.

Subject: Re: Core 2019
Posted by mirek on Sat, 08 Jun 2019 18:38:58 GMT
View Forum Message <> Reply to Message

Hopefully fixed.

Mirek

Subject: Re: Core 2019
Posted by Novo on Sat, 08 Jun 2019 19:42:43 GMT
View Forum Message <> Reply to Message

Novo wrote on Sat, 08 June 2019 12:30
I just wanted to compare the new U++ allocator with the standard one ...
So, StdAlloc-based MT version runs for 233 s. and it is using 1.6% RAM max (~541Mb).
It is somewhere in-between the new and the old U++ allocator.

Subject: Re: Core 2019
Posted by Novo on Sat, 08 Jun 2019 19:45:01 GMT
View Forum Message <> Reply to Message

mirek wrote on Sat, 08 June 2019 14:38Hopefully fixed.

Mirek
Everything is fine now.

Thank you!

Subject: Re: Core 2019
Posted by Novo on Sat, 08 Jun 2019 19:54:08 GMT
View Forum Message <> Reply to Message

mirek wrote on Sat, 08 June 2019 12:31
We can test this. In HeapImp.h, there is HPAGE constant. This is the size of "master chunk" (in 4KB units) and also maximum size of block that allocator keeps for reuse. Try to change that to something smaller, like 256 and retest...

Mirek

In case of HPAGE = 256 it is starting to use tens of gigabytes in just a few seconds ...

Subject: Re: Core 2019
Posted by Novo on Sat, 08 Jun 2019 20:06:49 GMT
View Forum Message <> Reply to Message

Novo wrote on Sat, 08 June 2019 15:54
In case of HPAGE = 256 it is starting to use tens of gigabytes in just a few seconds ...
In case of HPAGE = 8192 it is using 2.0% RAM max (~646Mb) one one (some data is read from disc into memory) run
and 2.2% RAM max (~714Mb) on another run (all data is cashed in memory).
Well, "top" is not the best tool to check memory usage.

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 08:03:29 GMT
View Forum Message <> Reply to Message

Novo wrote on Sat, 08 June 2019 21:54mirek wrote on Sat, 08 June 2019 12:31
We can test this. In HeapImp.h, there is HPAGE constant. This is the size of "master chunk" (in 4KB units) and also maximum size of block that allocator keeps for reuse. Try to change that to something smaller, like 256 and retest...

Mirek
In case of HPAGE = 256 it is starting to use tens of gigabytes in just a few seconds ...

Now that is an excelent clue :)

Found and fixed a bug (stupid one really). Can you test now please?

Mirek

Subject: Re: Core 2019
Posted by Novo on Sun, 09 Jun 2019 13:20:26 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 09 June 2019 04:03Novo wrote on Sat, 08 June 2019 21:54mirek wrote on Sat, 08 June 2019 12:31
We can test this. In HeapImp.h, there is HPAGE constant. This is the size of "master chunk" (in 4KB units) and also maximum size of block that allocator keeps for reuse. Try to change that to something smaller, like 256 and retest...

Mirek
In case of HPAGE = 256 it is starting to use tens of gigabytes in just a few seconds ...

Now that is an excelent clue :)

Found and fixed a bug (stupid one really). Can you test now please?

Mirek

HPAGE = 256
ram: 308 Mb, time: 253 s.

HPAGE = 7 * 8192
ram: 714 Mb, time: 232 s.

StdAlloc still remains the best choice for MT ...

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 14:33:37 GMT
View Forum Message <> Reply to Message

Novo wrote on Sun, 09 June 2019 15:20mirek wrote on Sun, 09 June 2019 04:03Novo wrote on Sat, 08 June 2019 21:54mirek wrote on Sat, 08 June 2019 12:31
We can test this. In HeapImp.h, there is HPAGE constant. This is the size of "master chunk" (in 4KB units) and also maximum size of block that allocator keeps for reuse. Try to change that to something smaller, like 256 and retest...

Mirek
In case of HPAGE = 256 it is starting to use tens of gigabytes in just a few seconds ...

Now that is an excelent clue :)

Found and fixed a bug (stupid one really). Can you test now please?

Mirek

HPAGE = 256
ram: 308 Mb, time: 253 s.

OK, at least the bug was fixed... :)

Quote:

HPAGE = 7 * 8192
ram: 714 Mb, time: 232 s.

StdAlloc still remains the best choice for MT ...

Can you try some other value, like 4096 or 8192...

Anyway, maybe this is really only misinterpreted reporting. The idea was that if I allocate a lot of address space, it is not really in physical memory unless written to.

Mirek

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 14:43:08 GMT
View Forum Message <> Reply to Message

Would it be possible to get peak memory profile?

Basically, you call PeakMemoryProfile at the start to activate it, then RDUMP(PeakMemoryProfile()) at the end of app. (Slows down the allocator).

Mirek

Subject: Re: Core 2019
Posted by Novo on Sun, 09 Jun 2019 15:02:08 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 09 June 2019 10:33
Can you try some other value, like 4096 or 8192...

Anyway, maybe this is really only misinterpreted reporting. The idea was that if I allocate a lot of address space, it is not really in physical memory unless written to.

Mirek

HPAGE = 4096
mem: 680 Mb, time: 232 s.

HPAGE = 8192
mem: 777 Mb, time: 232 s.

If I remember correctly, some of the system allocation routines initialize allocated memory with zeros even if you do not write there anything ...

Subject: Re: Core 2019
Posted by Novo on Sun, 09 Jun 2019 15:15:19 GMT
View Forum Message <> Reply to Message

I hacked your TIMING macro and made a similar RMEMUSE one:

```
namespace Upp {

struct MemInspector {
protected:
 static bool active;

 const char *name;
 int        call_count;
 int        min_mem;
 int        max_mem;
 int        max_nesting;
 int        all_count;
 StaticMutex mutex;

public:
 MemInspector(const char *name = NULL); // Not String !!!
 ~MemInspector();

 void Add(int mem, int nesting);

 String Dump();

 class Routine {
 public:
  Routine(MemInspector& stat, int& nesting)
  : nesting(nesting), stat(stat) {
   ++nesting;
  }

  ~Routine() {
   --nesting;
   int mem = MemoryUsedKb();
   stat.Add(mem, nesting);
  }

 protected:
  int& nesting;
  MemInspector& stat;
 };

 static void Activate(bool b) { active = b; }
};

bool MemInspector::active = true;

MemInspector::MemInspector(const char *_name) {
 name = _name ? _name : "";
```

```
 all_count = call_count = max_nesting = min_mem = max_mem = 0;
}

MemInspector::~MemInspector() {
 Mutex::Lock __(mutex);
 StdLog() << Dump() << "\r\n";
}

void MemInspector::Add(int mem, int nesting)
{
 // mem = MemoryUsedKb() - mem;
 Mutex::Lock __(mutex);
 if(!active) return;
 all_count++;
 if(nesting > max_nesting)
  max_nesting = nesting;
 if(nesting == 0) {
  if(call_count++ == 0)
   min_mem = max_mem = mem;
  else {
   if(mem < min_mem)
    min_mem = mem;
   if(mem > max_mem)
    max_mem = mem;
  }
 }
}

String MemInspector::Dump() {
 Mutex::Lock __(mutex);
 String s = Sprintf("MEMUSE %-15s: ", name);
 if(call_count == 0)
  return s + "No active hit";
 return s
     << "min: " << min_mem
     << ", max: " << max_mem
     << Sprintf(", nesting: %d - %d", max_nesting, all_count);
}


}

#define RMEMUSE(x) \
 static UPP::MemInspector COMBINE(sMemStat, __LINE__)(x); \
 static thread_local int COMBINE(sMemStatNesting, __LINE__); \
 UPP::MemInspector::Routine COMBINE(sMemStatR, __LINE__)(COMBINE(sMemStat,
__LINE__), COMBINE(sMemStatNesting, __LINE__))
```

What I'm getting in case of HPAGE = 7 * 8192 is
TIMING Chunk : 4108.80 s - 22.66 ms (4108.80 s / 181363), min: 1.00 ms, max: 1.24 s , nesting: 0 - 181363
MEMUSE Chunk : min: 30844, max: 341052, nesting: 0 - 181363
TIMING Read Data : 228.28 s - 228.28 s (228.28 s / 1), min: 228.28 s , max: 228.28 s , nesting: 0 - 1

top is saying max used memory (RES) is ~771 Mb.

Subject: Re: Core 2019
Posted by Novo on Sun, 09 Jun 2019 15:34:24 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 09 June 2019 10:43Would it be possible to get peak memory profile?

Basically, you call PeakMemoryProfile at the start to activate it, then RDUMP(PeakMemoryProfile()) at the end of app. (Slows down the allocator).

Mirek
I'm calling PeakMemoryProfile(); before CoWork is created and RDUMP(*PeakMemoryProfile()); after it is destroyed.
*PeakMemoryProfile() = Memory peak 328920

32 B,	13 allocated (0 KB),	113 fragments (3 KB)
64 B,	8 allocated (0 KB),	55 fragments (3 KB)
96 B,	6 allocated (0 KB),	36 fragments (3 KB)
128 B,	3 allocated (0 KB),	28 fragments (3 KB)
160 B,	3 allocated (0 KB),	22 fragments (3 KB)
192 B,	2 allocated (0 KB),	19 fragments (3 KB)
224 B,	3 allocated (0 KB),	15 fragments (3 KB)
256 B,	2 allocated (0 KB),	13 fragments (3 KB)
288 B,	2 allocated (0 KB),	12 fragments (3 KB)
320 B,	2 allocated (0 KB),	10 fragments (3 KB)
352 B,	1 allocated (0 KB),	10 fragments (3 KB)
384 B,	2 allocated (0 KB),	8 fragments (3 KB)
448 B,	2 allocated (0 KB),	7 fragments (3 KB)
576 B,	4 allocated (2 KB),	3 fragments (1 KB)
672 B,	3 allocated (1 KB),	3 fragments (1 KB)
800 B,	2 allocated (1 KB),	3 fragments (2 KB)
992 B,	3 allocated (2 KB),	1 fragments (0 KB)
TOTAL,	61 allocated (15 KB),	358 fragments (50 KB)

Empty 4KB pages 0 (0 KB)
Large block count 9, total size 119 KB
Large fragments count 5, total size 71 KB
Huge block count 80, total size 1779376 KB
Sys block count 0, total size 0 KB
224MB master blocks 4

Large fragments:
1 KB: 1
8 KB: 1
17.25 KB: 1
22 KB: 1
23.5 KB: 1

Huge fragments:
8 KB: 1
16 KB: 1
20 KB: 3
32 KB: 5
36 KB: 2
40 KB: 1
44 KB: 1
52 KB: 1
64 KB: 20
68 KB: 1
80 KB: 6
92 KB: 2
120 KB: 1
128 KB: 1
144 KB: 1
156 KB: 1
164 KB: 1
180 KB: 2
188 KB: 2
192 KB: 3
196 KB: 1
204 KB: 1
248 KB: 1
252 KB: 1
272 KB: 2
276 KB: 1
284 KB: 1
288 KB: 1
296 KB: 2
304 KB: 1
320 KB: 1
328 KB: 1
348 KB: 1
364 KB: 1
384 KB: 1
396 KB: 2
412 KB: 1
440 KB: 1
464 KB: 1
468 KB: 1

484 KB: 1
500 KB: 1
504 KB: 1
512 KB: 1
520 KB: 1
560 KB: 2
564 KB: 1
568 KB: 1
576 KB: 1
580 KB: 1
612 KB: 1
616 KB: 1
620 KB: 1
640 KB: 1
652 KB: 2
696 KB: 1
700 KB: 1
708 KB: 1
740 KB: 1
760 KB: 1
780 KB: 1
784 KB: 1
796 KB: 1
916 KB: 1
944 KB: 1
972 KB: 1
1044 KB: 1
1084 KB: 1
1088 KB: 1
1148 KB: 1
1184 KB: 1
1200 KB: 1
1212 KB: 1
1216 KB: 1
1272 KB: 1
1280 KB: 1
1300 KB: 1
1364 KB: 1
1464 KB: 1
1512 KB: 1
1616 KB: 1
1716 KB: 1
1720 KB: 1
1920 KB: 1
1996 KB: 1
2220 KB: 1
2280 KB: 1
2552 KB: 1

2576 KB: 1
2596 KB: 1
2804 KB: 1
2864 KB: 1
3080 KB: 1
3324 KB: 1
3420 KB: 1
3516 KB: 3
3580 KB: 6
3596 KB: 1
3644 KB: 1
3648 KB: 1
3916 KB: 1
4408 KB: 1
4452 KB: 1
4720 KB: 1
5564 KB: 1
5632 KB: 1
6996 KB: 1
7036 KB: 1
7100 KB: 1
7280 KB: 2
7632 KB: 1
7848 KB: 1
7864 KB: 1
8344 KB: 1
8448 KB: 1
8632 KB: 1
8820 KB: 1
8968 KB: 1
9124 KB: 1
9296 KB: 1
9440 KB: 1
9880 KB: 1
10612 KB: 1
10768 KB: 1
11136 KB: 1
11188 KB: 1
11420 KB: 1
13572 KB: 1
14304 KB: 1
14988 KB: 1
15168 KB: 1
15576 KB: 1
15924 KB: 1
16040 KB: 1
18012 KB: 1
19204 KB: 1

20108 KB: 1
20396 KB: 1
55160 KB: 1

top is saying that app is using 855 Mb max ...

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 16:51:22 GMT
View Forum Message <> Reply to Message

Novo wrote on Sun, 09 June 2019 17:02
If I remember correctly, some of the system allocation routines initialize allocated memory with zeros even if you do not write there anything ...

They can delay that to the moment the page is allocated in physical memory.

Mirek

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 16:54:01 GMT
View Forum Message <> Reply to Message

Looking at peak profile, it looks like there are very little "small" blocks and most of memory is in those 80 "huge" (that means >64KB) blocks.

Can that be correct?

Mirek

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 18:56:03 GMT
View Forum Message <> Reply to Message

Novo wrote on Sun, 09 June 2019 17:15I hacked your TIMING macro and made a similar RMEMUSE one:

There is also

int MemoryUsedKbMax();

anyway, both MemoryUsedKb and this one have one disadvantage - they only count active blocks, so if fragmentation is high, it is not accounted for.

That said, it looks like the fragmentation is the real culprit here. It looks like we have 300MB of active memory and 500MB in memory fragments. Looks like stdalloc fights with that too, with little bit better success.

I would like to get a list of allocations your code is doing so that I can hopefully replicate it and investigate whether there can be anything done to reduce the fragmentation.... I will post temporary changes to get the log tomorrow, if you are willing to help.

Mirek

Subject: Re: Core 2019
Posted by Novo on Sun, 09 Jun 2019 19:39:32 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 09 June 2019 12:54Looking at peak profile, it looks like there are very little "small" blocks and most of memory is in those 80 "huge" (that means >64KB) blocks.

Can that be correct?

Mirek
It is hard to tell. I'm not controlling that.
Another problem is that all allocations/deallocations happen in CoWork's threads. I cannot call RDUMP(*PeakMemoryProfile()) inside of CoWork because it will be called at least 181363 times ...

The app is parsing Wikipedia XML dump. It is decompressing a bz2 archive and parsing chunks of XML. After that my own parser is parsing Mediawiki text.
As a first pass my parser is building a list of tokens organized as a Vector<> (I'm not inserting in the middle :))
My parser is avoiding memory allocation at all possible costs. I'm calling Vector::SetCountR and reusing these vectors. When I need to deal with String I'm using my own not owning data string class.
Unfortunately, I cannot control memory allocation with XmlParser. I have to relay on the default allocator.

Ideally, I'd love to see U++ allocator designed like this.
Related papers:
https://people.cs.umass.edu/~emery/pubs/berger-pldi2001.pdf
https://erdani.com/publications/cuj-2005-12.pdf
https://accu.org/content/conf2008/Alexandrescu-memory-alloca tion.screen.pdf

It doesn't have to be a complete implementation of everything. I just would like to be able plug into U++'s allocator in a similar fashion and extend/tune it.

mirek wrote on Sun, 09 June 2019 14:56I would like to get a list of allocations your code is doing so that I can hopefully replicate it and investigate whether there can be anything done to reduce

the fragmentation.... I will post temporary changes to get the log tomorrow, if you are willing to help.

Yes, I'm willing to help. I even willing to implement this policy-based allocator. I just need the ability to integrate it into U++. It doesn't have to be a part of U++.

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 21:11:41 GMT
View Forum Message <> Reply to Message

Novo wrote on Sun, 09 June 2019 21:39mirek wrote on Sun, 09 June 2019 12:54Looking at peak profile, it looks like there are very little "small" blocks and most of memory is in those 80 "huge" (that means >64KB) blocks.

Can that be correct?

Mirek
It is hard to tell. I'm not controlling that.
Another problem is that all allocations/deallocations happen in CoWork's threads. I cannot call RDUMP(*PeakMemoryProfile()) inside of CoWork because it will be called at least 181363 times ...

Why would you want to? Peak is really peak, it is profile at the moment when there is maximum memory use.

One caveat about profile is that it is only profile of current thread for small and large blocks. But our problem is with huge blocks anyway.

Quote:
The app is parsing Wikipedia XML dump. It is decompressing a bz2 archive and parsing chunks of XML. After that my own parser is parsing Mediawiki text.
As a first pass my parser is building a list of tokens organized as a Vector<> (I'm not inserting in the middle :))
My parser is avoiding memory allocation at all possible costs. I'm calling Vector::SetCountR and reusing these vectors. When I need to deal with String I'm using my own not owning data string class.

Well, maybe there can also be an interference with MemoryTryRealloc (as those Vectors grow). Perhaps you can test what happens if

```
bool  MemoryTryRealloc(void *ptr, size_t& newsize) {
 return false; // (((dword)(uintptr_t)ptr) & 16) && MemoryTryRealloc__(ptr, newsize);
}
```

Quote:
Unfortunately, I cannot control memory allocation with XmlParser. I have to relay on the default allocator.

There are not many... BTW, are you parsing memory - XmlParser(const char *), or streams - XmlParser(Stream& in) ?

Mirek

Subject: Re: Core 2019
Posted by mirek on Sun, 09 Jun 2019 21:25:20 GMT
View Forum Message <> Reply to Message

Here is the code for logging all huge allocations (replace in Core/hheap.cpp):

```
void *Heap::HugeAlloc(size_t count) // count in 4kb pages
{
 ASSERT(count);

#ifdef LSTAT
 if(count < 65536)
  hstat[count]++;
#endif

 huge_4KB_count += count;

 if(huge_4KB_count > huge_4KB_count_max) {
  huge_4KB_count_max = huge_4KB_count;
  if(MemoryUsedKb() > sKBLimit)
   Panic("MemoryLimitKb breached!");
  if(sPeak)
   Make(*sPeak);
 }

 if(!D::freelist[0]->next) { // initialization
  for(int i = 0; i < 2; i++)
   Dbl_Self(D::freelist[i]);
 }

 if(count > HPAGE) { // we are wasting 4KB to store just 4 bytes here, but this is >32MB after all..
  LTIMING("SysAlloc");
  byte *sysblk = (byte *)SysAllocRaw((count + 1) * 4096, 0);
```

```
  BlkHeader *h = (BlkHeader *)(sysblk + 4096);
  h->size = 0;
  *((size_t *)sysblk) = count;
  sys_count++;
  sys_size += 4096 * count;
  return h;
 }

 LTIMING("Huge Alloc");

 word wcount = (word)count;

 if(16 * free_4KB > huge_4KB_count) // keep number of free 4KB blocks in check
  FreeSmallEmpty(INT_MAX, int(free_4KB - huge_4KB_count / 32));

 for(int pass = 0; pass < 2; pass++) {
  for(int i = count >= 16; i < 2; i++) {
   BlkHeader *l = D::freelist[i];
   BlkHeader *h = l->next;
   while(h != l) {
    word sz = h->GetSize();
    if(sz >= count) {
     void *ptr = MakeAlloc(h, wcount);
     if(count > 16)
      RLOG("HugeAlloc " << asString(ptr) << ", size: " << asString(count));
     return ptr;
    }
    h = h->next;
   }
  }

  if(!FreeSmallEmpty(wcount, INT_MAX)) { // try to coalesce 4KB small free blocks back to huge
storage
   void *ptr = SysAllocRaw(HPAGE * 4096, 0);
   HugePage *pg = (HugePage *)MemoryAllocPermanent(sizeof(HugePage));
   pg->page = ptr;
   pg->next = huge_pages;
   huge_pages = pg;
   AddChunk((BlkHeader *)ptr, HPAGE); // failed, add 32MB from the system
   huge_chunks++;
  }
 }
 Panic("Out of memory");
 return NULL;
}

int Heap::HugeFree(void *ptr)
{
```

```
 BlkHeader *h = (BlkHeader *)ptr;
 if(h->size == 0) {
  LTIMING("Sys Free");
  byte *sysblk = (byte *)h - 4096;
  size_t count = *((size_t *)sysblk);
  SysFreeRaw(sysblk, (count + 1) * 4096);
  huge_4KB_count -= count;
  sys_count--;
  sys_size -= 4096 * count;
  return 0;
 }
 LTIMING("Huge Free");
 if(h->GetSize() > 16)
  RLOG("HugeFree " << asString(ptr) << ", size: " << asString(h->GetSize()));
 huge_4KB_count -= h->GetSize();
 return BlkHeap::Free(h)->GetSize();
}

bool Heap::HugeTryRealloc(void *ptr, size_t count)
{
 bool b = count <= HPAGE && BlkHeap::TryRealloc(ptr, count, huge_4KB_count);
 if(b)
  RLOG("HugeRealloc " << asString(ptr) << ", size: " << asString(count));
 return b;
}
```

(please test with active MemoryTryRealloc)

Subject: Re: Core 2019
Posted by mirek on Mon, 10 Jun 2019 15:27:57 GMT
View Forum Message <> Reply to Message

I have tried to improve fragmentation using approximate best fit, hopefully this will help a bit... (in trunk)

Subject: Re: Core 2019
Posted by Novo on Mon, 10 Jun 2019 16:01:33 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 10 June 2019 11:27I have tried to improve fragmentation using approximate best fit, hopefully this will help a bit... (in trunk)
Thanks!
mem: 400 Mb, time: 230 s.
This is a huge improvement.

Subject: Re: Core 2019
Posted by mirek on Mon, 10 Jun 2019 16:17:38 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 10 June 2019 18:01mirek wrote on Mon, 10 June 2019 11:27I have tried to improve fragmentation using approximate best fit, hopefully this will help a bit... (in trunk)
Thanks!
mem: 400 Mb, time: 230 s.
This is a huge improvement.

Cool. So I guess issue solved and we do not need to worry about other tests?

Mirek

Subject: Re: Core 2019
Posted by Novo on Mon, 10 Jun 2019 16:18:02 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 09 June 2019 17:11BTW, are you parsing memory - XmlParser(const char *), or streams - XmlParser(Stream& in) ?

Stream. bz2::DecompressStream.
I guess that XmlParser is responsible for fragmentation.

Subject: Re: Core 2019
Posted by Novo on Mon, 10 Jun 2019 16:21:44 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 10 June 2019 12:17
Cool. So I guess issue solved and we do not need to worry about other tests?

I'll try to run other tests and see what happens ...

Subject: Re: Core 2019
Posted by Novo on Mon, 10 Jun 2019 16:34:58 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 09 June 2019 17:11
Well, maybe there can also be an interference with MemoryTryRealloc (as those Vectors grow).
Perhaps you can test what happens if

```
bool  MemoryTryRealloc(void *ptr, size_t& newsize) {
 return false; // (((dword)(uintptr_t)ptr) & 16) && MemoryTryRealloc__(ptr, newsize);
}
```

This doesn't affect anything.

Subject: Re: Core 2019
Posted by Novo on Mon, 10 Jun 2019 16:45:35 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 09 June 2019 17:25Here is the code for logging all huge allocations (replace in Core/hheap.cpp):

This code is crashing with the latest trunk.
I guess we can stop at this point.

Subject: Re: Core 2019
Posted by Novo on Fri, 21 Jun 2019 03:43:00 GMT
View Forum Message <> Reply to Message

Update.

I changed my app. Now it is doing a bunch of string manipulations (mostly concatenations)

Results:
U++:
time: 234s, mem is growing to 4.4Gb and it is not going down till the very end.

glibc:
Default settings (8-core CPU, CoWork pool has 18 threads):
time: 239s, mem max is 6.5Gb down to 3.6Gb

export MALLOC_ARENA_MAX=16
time: 239s, mem max is 4.2Gb down to 2.8Gb

export MALLOC_ARENA_MAX=8
time: 244s, mem max is 4.0Gb down to 1.3Gb

Conclusion:
glibc allocator is more efficient with a little bit of manual tuning. Difference in performance is not that signifficant. Cannot tell anything about Windows.

Subject: Re: Core 2019
Posted by Novo on Fri, 21 Jun 2019 04:14:59 GMT
View Forum Message <> Reply to Message

It looks like performance of the glibc allocator depends on a state of the system. Got new interesting results:

export MALLOC_ARENA_MAX=20
time: 239s, mem max is 3.9Gb down to 0.7Gb

Subject: Re: Core 2019
Posted by mirek on Fri, 21 Jun 2019 07:16:15 GMT
View Forum Message <> Reply to Message

I think you cannot deduce too much about efficiency looking at "down" number. That will depend a lot on overall load of system, more the load, less this number as system will page out unused pages from your apps address space. So the current philosophy of U++ allocator is that this does not matter.

Further explanation. The function that any allocator is using to obtain address space from system is mmap and there is munmap that returns address space to system. Normally there is a threshold - if block is too big, it allocation is simply handled by mmap / munmap calls, meaning it is returned to the system at MemoryFree. If it is less than threshold, bigger chung is mmaped from the system and then divided to smaller chunks (somehow).

Now what is different is that standard GCC allocator has thershold at 4MB. U++ allocatar at 224MB. In practive, this means that if you alloc / free 5MB block in std, it gets released back to system immediately. With U++, blocks up to 224 MB are not returned to the system immediately. If they are really unused, this just means that system will retrieve them when there is a need for more physical memory.

Mirek

Subject: Re: Core 2019
Posted by mirek on Fri, 21 Jun 2019 07:23:53 GMT
View Forum Message <> Reply to Message

Anyway, peak and final memory profiles would be nice to know... :)

(Although one problem is that only calling thread's memory is in the profile).

Subject: Re: Core 2019

Posted by Novo on Fri, 21 Jun 2019 15:59:14 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 21 June 2019 03:16
Now what is different is that standard GCC allocator has thershold at 4MB. U++ allocatar at 224MB. In practive, this means that if you alloc / free 5MB block in std, it gets released back to system immediately. With U++, blocks up to 224 MB are not returned to the system immediately. If they are really unused, this just means that system will retrieve them when there is a need for more physical memory.

Mirek

This info doesn't match what I'm reading in the docs I posted above.

> The lower limit for this parameter is 0. The upper limit is
> DEFAULT_MMAP_THRESHOLD_MAX: 512*1024 on 32-bit systems or
> 4*1024*1024*sizeof(long) on 64-bit systems.
>
> Note: Nowadays, glibc uses a dynamic mmap threshold by
> default. The initial value of the threshold is 128*1024, but
> when blocks larger than the current threshold and less than or
> equal to DEFAULT_MMAP_THRESHOLD_MAX are freed, the threshold
> is adjusted upward to the size of the freed block. When
> dynamic mmap thresholding is in effect, the threshold for
> trimming the heap is also dynamically adjusted to be twice the
> dynamic mmap threshold. Dynamic adjustment of the mmap
> threshold is disabled if any of the M_TRIM_THRESHOLD,
> M_TOP_PAD, M_MMAP_THRESHOLD, or M_MMAP_MAX parameters is set.

So, the old allocator on 64-bit systems had threshold 32Mb (4*1024*1024*sizeof(long)).
In the new one it is determined dynamically. Initial value is 128Kb.

I traced my app with a tool which intercepts all malloc/free calls. I know exactly what is stressing the allocator :roll:
Memory block sizes:

File Attachments
1) Screenshot_2019-06-21_11-51-00.png, downloaded 715 times

Subject: Re: Core 2019
Posted by Novo on Fri, 21 Jun 2019 16:01:35 GMT
View Forum Message <> Reply to Message

Memory consumption (actual, total of all malloc/free):

File Attachments
1) Screenshot_2019-06-21_11-58-58.png, downloaded 621 times

Subject: Re: Core 2019
Posted by Novo on Fri, 21 Jun 2019 16:10:34 GMT
View Forum Message <> Reply to Message

Unfortunately, I couldn't find a decent tool to track real amount of system memory (mmaped) used by an app.
A chart similar to one above would be very helpful, otherwise I can just compare most notable values.

Subject: Re: Core 2019
Posted by Novo on Fri, 21 Jun 2019 17:58:55 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 21 June 2019 03:23Anyway, peak and final memory profiles would be nice to know... :)

(Although one problem is that only calling thread's memory is in the profile).
I've attached a profile ...

File Attachments
1) wiki_infobox_mt.log, downloaded 285 times

Subject: Re: Core 2019
Posted by Novo on Fri, 21 Jun 2019 18:28:14 GMT
View Forum Message <> Reply to Message

My version of libc:

$ /lib/x86_64-linux-gnu/libc.so.6
GNU C Library (Ubuntu GLIBC 2.29-0ubuntu2) stable release version 2.29.

Subject: Re: Core 2019
Posted by Novo on Sat, 22 Jun 2019 00:51:03 GMT
View Forum Message <> Reply to Message

U++, RSS, collected as "top -d 0.5".

File Attachments
1) upp.png, downloaded 825 times

Subject: Re: Core 2019
Posted by Novo on Sat, 22 Jun 2019 00:53:47 GMT
View Forum Message <> Reply to Message

glibc, RSS, collected the same way.

File Attachments
1) glibc.png, downloaded 720 times

Subject: Re: Core 2019
Posted by mirek on Sat, 22 Jun 2019 08:23:59 GMT
View Forum Message <> Reply to Message

Well, from what I see, the real difference is that U++ "keeps" the address space...

Maybe you can strace both allocators and grep for mmap / munmap?

Also, profile after the CoWork would be nice to know.

Mirek

Subject: Re: Core 2019
Posted by Novo on Sat, 22 Jun 2019 15:49:55 GMT
View Forum Message <> Reply to Message

mirek wrote on Sat, 22 June 2019 04:23Well, from what I see, the real difference is that U++ "keeps" the address space...

Maybe you can strace both allocators and grep for mmap / munmap?

Also, profile after the CoWork would be nice to know.

Mirek

Attached.
Just mmap and munmap do not tell much.

glibc calls brk a lot as well.
Profile after the CoWork was posted in this message.

File Attachments
1) strace.01.zip, downloaded 267 times

Subject: Re: Core 2019
Posted by mirek on Sat, 22 Jun 2019 20:22:11 GMT
View Forum Message <> Reply to Message

Novo wrote on Sat, 22 June 2019 17:49
Profile after the CoWork was posted in this message.

I believe that is just peak profile, but I might be looking at it wrong.

strace logs seem weird a bit - I do not see enough allocations (either way) for 4GB.

Subject: Re: Core 2019
Posted by Novo on Sat, 22 Jun 2019 21:44:33 GMT
View Forum Message <> Reply to Message

mirek wrote on Sat, 22 June 2019 16:22
I believe that is just peak profile, but I might be looking at it wrong.

What is "Profile after the CoWork"? I do know only about MemoryUsedKb(), MemoryUsedKbMax(), PeakMemoryProfile() ...
In this log-file all three of them are called after CoWork.

mirek wrote on Sat, 22 June 2019 16:22
strace logs seem weird a bit - I do not see enough allocations (either way) for 4GB.

My bad. I didn't follow the threads ...
New logs are attached.

File Attachments
1) 02.zip, downloaded 276 times

Subject: Re: Core 2019
Posted by mirek on Sun, 23 Jun 2019 06:21:13 GMT
View Forum Message <> Reply to Message

Novo wrote on Sat, 22 June 2019 23:44mirek wrote on Sat, 22 June 2019 16:22
I believe that is just peak profile, but I might be looking at it wrong.

What is "Profile after the CoWork"? I do know only about MemoryUsedKb(), MemoryUsedKbMax(), PeakMemoryProfile() ...
In this log-file all three of them are called after CoWork.

"MemoryProfile()" :)

The differnce is that PeakMemoryProfile returns snapshot at point where maximum memory is allocated, while "MemoryProfile" returns current status.

Mirek

Subject: Re: Core 2019
Posted by mirek on Sun, 23 Jun 2019 07:55:55 GMT
View Forum Message <> Reply to Message

Novo wrote on Sat, 22 June 2019 23:44
My bad. I didn't follow the threads ...
New logs are attached.

Well, GCC definitely unmaps regions before mapping them back again, so the original hypothesis holds.

Now the interesting question is "are we doing something wrong?". Perhaps we do... Maybe 224MB is too big chunk and it is true that it will waste swap space....

Mirek

Subject: Re: Core 2019
Posted by mirek on Sun, 23 Jun 2019 08:19:53 GMT
View Forum Message <> Reply to Message

It would probably be worth to experiment with HPAGE constant... Can be any number >16.

Mirek

Subject: Re: Core 2019
Posted by Novo on Sun, 23 Jun 2019 19:26:26 GMT

View Forum Message <> Reply to Message

mirek wrote on Sun, 23 June 2019 02:21
"MemoryProfile()" :)

Attached :blush:

File Attachments
1) wiki_infobox_mt.log, downloaded 263 times

Subject: Re: Core 2019
Posted by Novo on Mon, 24 Jun 2019 04:13:11 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 23 June 2019 04:19It would probably be worth to experiment with HPAGE constant... Can be any number >16.

I'm afraid I cannot afford to spend time on U++'s allocator anymore.
I'm switching to the glibc's one for the time being.
It would be great to see jemalloc and tcmalloc integrated into U++ because they are supposed to be better at avoiding fragmentation.

Subject: Re: Core 2019
Posted by mirek on Mon, 24 Jun 2019 07:22:47 GMT
View Forum Message <> Reply to Message

Novo wrote on Sun, 23 June 2019 21:26mirek wrote on Sun, 23 June 2019 02:21
"MemoryProfile()" :)

Attached :blush:

Huge block count 4, total size 1951 KB
Huge fragments count 424, total size 4810724 KB

Basically means that all the memory is really freed, as expected. Fragments list also reveals that fragmentation is not really too bad (a lot of big blocks).

Based on this, I do not see any defect or deficiency, except the choosen one (keep the memory).

Mirek

Subject: Re: Core 2019
Posted by Novo on Mon, 24 Jun 2019 15:36:02 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 24 June 2019 03:22
Based on this, I do not see any defect or deficiency, except the choosen one (keep the memory).

Well, I do not think that taking from the system 4.6Gb when the app is allocating ~400Mb is acceptable. 8)
Default behavior of glibc's allocator is optimized for huge enterprise-level apps, but with some manual tweaking it performs fine with small apps. jemalloc would perform even better, I believe.

Subject: Re: Core 2019
Posted by mirek on Mon, 24 Jun 2019 16:42:23 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 24 June 2019 17:36mirek wrote on Mon, 24 June 2019 03:22
Based on this, I do not see any defect or deficiency, except the choosen one (keep the memory).

Well, I do not think that taking from the system 4.6Gb when the app is allocating ~400Mb is acceptable. 8)
Default behavior of glibc's allocator is optimized for huge enterprise-level apps, but with some manual tweaking it performs fine with small apps. jemalloc would perform even better, I believe.

Well, app actually IS allocating ~4.5 GB at the peak, that is what all indicators show - or have I got that wrong?

So it is really a question of priorities. Do we want to unmap that memory or keep it for the future use as mmap/munmap are quite expensive calls? This was the question I have asked and at that time the answer was: "we want to keep it". Now I am not so sure... :)

Looks like we need MemoryOptions and/or MemoryShrink...

Anyway, back to drawing board...

Mirek

Subject: Re: Core 2019
Posted by Novo on Mon, 24 Jun 2019 17:36:24 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 24 June 2019 12:42
Well, app actually IS allocating ~4.5 GB at the peak, that is what all indicators show - or have I got that wrong?

Top with default options gives me 4.6GB, 4.5GB or 4.4GB. It depends on a run. :roll:

Subject: Re: Core 2019
Posted by mirek on Mon, 24 Jun 2019 18:15:46 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 24 June 2019 19:36mirek wrote on Mon, 24 June 2019 12:42
Well, app actually IS allocating ~4.5 GB at the peak, that is what all indicators show - or have I got that wrong?

Top with default options gives me 4.6GB, 4.5GB or 4.4GB. It depends on a run. :roll:

Of course, as it is MT and allocation/deallocation order is not fixed...

Subject: Re: Core 2019
Posted by mirek on Wed, 26 Jun 2019 06:52:36 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 24 June 2019 06:13mirek wrote on Sun, 23 June 2019 04:19It would probably be worth to experiment with HPAGE constant... Can be any number >16.

I'm afraid I cannot afford to spend time on U++'s allocator anymore.
I'm switching to the glibc's one for the time being.

Redesign finished. Address space is now returned to OS when possible.

There is now MemorySetOptions to finetune behaviour. There are 4 parameters, low values should in general result in slower allocator with less memory consumption...

Mirek

Subject: Re: Core 2019
Posted by mirek on Thu, 27 Jun 2019 07:44:22 GMT
View Forum Message <> Reply to Message

In addtion to USEMALLOC there is now HEAPOVERRIDE flag that kicks out all Memory* definitions, allowing you to replace the whole allocator with something else than malloc.

Subject: Re: Core 2019
Posted by mirek on Fri, 28 Jun 2019 07:09:57 GMT
View Forum Message <> Reply to Message

I have got another idea. In order to reproduce allocation pattern, there is now

HEAPLOG

flag. If in main config, heap will produce a log of allocation (to heap.log file at exe dir) so that it can be reproduced and optimized.

heap.log is long, but should be highly compressible.

Subject: Re: Core 2019
Posted by Tom1 on Fri, 28 Jun 2019 14:56:41 GMT
View Forum Message <> Reply to Message

Hi Mirek,

My application started to crash maybe about a week or so ago after updating uppsrc from SVN. Now I got some time to investigate and it seems my app:

- Crashes strangely in different places with current SVN if I compile MSBT19x64 Release
- Works OK with current SVN when compiling with MSBT17x64 Debug or MSBT19x64 Debug
- Works OK with current SVN if I use flag USEMALLOC and compile with MSBT19x64 Release
- Works OK with SVN 13068 compiling with MSBT17x64 Release

Any idea what's wrong? Is it related to the new allocator?

I added some RLOGs and found that it might crash even somewhere in BufferPainter during Stroke... (did not track it all the way down though). Debugger does not help as it does not crash when compiled in debug mode.

Best regards,

Tom

Subject: Re: Core 2019
Posted by mirek on Fri, 28 Jun 2019 15:30:24 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Fri, 28 June 2019 16:56Hi Mirek,

My application started to crash maybe about a week or so ago after updating uppsrc from SVN. Now I got some time to investigate and it seems my app:

- Crashes strangely in different places with current SVN if I compile MSBT19x64 Release
- Works OK with current SVN when compiling with MSBT17x64 Debug or MSBT19x64 Debug
- Works OK with current SVN if I use flag USEMALLOC and compile with MSBT19x64 Release
- Works OK with SVN 13068 compiling with MSBT17x64 Release

Any idea what's wrong? Is it related to the new allocator?

I added some RLOGs and found that it might crash even somewhere in BufferPainter during Stroke... (did not track it all the way down though). Debugger does not help as it does not crash when compiled in debug mode.

Best regards,

Tom

Probably allocator, thanks for reporting. It is under active development. It would be worth quoting the revision tested - even today I have fixed / changed some things...

Mirek

Subject: Re: Core 2019
Posted by mirek on Fri, 28 Jun 2019 15:31:52 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Fri, 28 June 2019 16:56though). Debugger does not help as it does not crash when compiled in debug mode.

You can activate debug info for release mode. Sometimes it is very useful.

Mirek

Subject: Re: Core 2019
Posted by mirek on Fri, 28 Jun 2019 15:40:40 GMT
View Forum Message <> Reply to Message

For what is worth, I have quickly checked PainterExamples and all is working...

Subject: Re: Core 2019
Posted by Tom1 on Fri, 28 Jun 2019 16:38:38 GMT
View Forum Message <> Reply to Message

Hi Mirek,

Thanks for your reply. I will update from SVN again on Monday morning in the office. Then I will check again. I will also try enabling debug info for release mode.

Thanks and best regards,

Tom

Subject: Re: Core 2019
Posted by Novo on Sun, 30 Jun 2019 17:02:08 GMT
View Forum Message <> Reply to Message

I'm getting a crash with a stack trace below when using MemoryAllocPermanent:

Upp::Heap::RemoteFlushRaw (this=<optimized out>) at HeapImp.h:481
Upp::Heap::RemoteFree (this=<optimized out>, ptr=<optimized out>, size=<optimized out>) at HeapImp.h:498
Upp::Heap::Free (this=<optimized out>, ptr=<optimized out>, page=<optimized out>, k=<optimized out>) at sheap.cpp:216
Upp::Heap::Free (this=0x7ffff7a15a10, ptr=<optimized out>) at sheap.cpp:237
Upp::MemoryFree (ptr=<optimized out>) at sheap.cpp:420
judy_close (judy=<optimized out>) at lib/judy.c:147
judy::Map<long long, long long>::~Map (this=<optimized out>) at /home/ssg/dvlp/cpp/sergey/upp/dvlp/plugin/judy/judy.h:48
ConsoleMainFn_ () at test_ht_perf.cpp:98
Upp::AppExecute__ (app=0xfffffffffffffffd) at App.cpp:343
main (argc=-3, argv=0x7ff0b0b5b000, envptr=0x7ffff7a167e8) at test_ht_perf.cpp:8

MemoryAllocPermanent seems to be a replacement for malloc.
The same code using MemoryAlloc works fine.
svn@13460, git@cb77bd58b76a15
Am I doing something wrong or is this a bug?

Subject: Re: Core 2019
Posted by mirek on Sun, 30 Jun 2019 17:11:43 GMT
View Forum Message <> Reply to Message

Novo wrote on Sun, 30 June 2019 19:02I'm getting a crash with a stack trace below when using MemoryAllocPermanent:

Upp::Heap::RemoteFlushRaw (this=<optimized out>) at HeapImp.h:481
Upp::Heap::RemoteFree (this=<optimized out>, ptr=<optimized out>, size=<optimized out>) at HeapImp.h:498
Upp::Heap::Free (this=<optimized out>, ptr=<optimized out>, page=<optimized out>, k=<optimized out>) at sheap.cpp:216
Upp::Heap::Free (this=0x7ffff7a15a10, ptr=<optimized out>) at sheap.cpp:237
Upp::MemoryFree (ptr=<optimized out>) at sheap.cpp:420
judy_close (judy=<optimized out>) at lib/judy.c:147
judy::Map<long long, long long>::~Map (this=<optimized out>) at /home/ssg/dvlp/cpp/sergey/upp/dvlp/plugin/judy/judy.h:48
ConsoleMainFn_ () at test_ht_perf.cpp:98

Upp::AppExecute__ (app=0xfffffffffffffffd) at App.cpp:343
main (argc=-3, argv=0x7ff0b0b5b000, envptr=0x7ffff7a167e8) at test_ht_perf.cpp:8

MemoryAllocPermanent seems to be a replacement for malloc.
The same code using MemoryAlloc works fine.
svn@13460, git@cb77bd58b76a15
Am I doing something wrong or is this a bug?

MemoryAllocPermanent is for allocating memory that is not to be freed (aka is "permanent"). You cannot call MemoryFree on it (as it is "permanent" :).

Mirek

Subject: Re: Core 2019
Posted by Novo on Sun, 30 Jun 2019 18:12:49 GMT
View Forum Message <> Reply to Message

mirek wrote on Sun, 30 June 2019 13:11MemoryAllocPermanent is for allocating memory that is not to be freed (aka is "permanent"). You cannot call MemoryFree on it (as it is "permanent" :).

Mirek
Thanks! A couple of lines of documentation would be really helpful ... :roll:

Subject: Re: Core 2019
Posted by mirek on Sun, 30 Jun 2019 22:03:00 GMT
View Forum Message <> Reply to Message

Novo wrote on Sun, 30 June 2019 20:12mirek wrote on Sun, 30 June 2019 13:11MemoryAllocPermanent is for allocating memory that is not to be freed (aka is "permanent"). You cannot call MemoryFree on it (as it is "permanent" :).

Mirek
Thanks! A couple of lines of documentation would be really helpful ... :roll:

You are right. Done.

Subject: Re: Core 2019
Posted by Tom1 on Mon, 01 Jul 2019 07:14:01 GMT
View Forum Message <> Reply to Message

Hi Mirek,

I just downloaded SVN 13462 and now it all works OK again... No crashes.

Thanks and best regards,

Tom

Subject: Re: Core 2019
Posted by Novo on Thu, 11 Jul 2019 17:55:45 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 27 June 2019 03:44In addtion to USEMALLOC there is now HEAPOVERRIDE flag that kicks out all Memory* definitions, allowing you to replace the whole allocator with something else than malloc.
Thanks a lot!

P.S. Documentation for MemoryLimitKb has a typo ...

Subject: Re: Core 2019
Posted by mirek on Thu, 11 Jul 2019 18:14:17 GMT
View Forum Message <> Reply to Message

Novo wrote on Thu, 11 July 2019 19:55mirek wrote on Thu, 27 June 2019 03:44In addtion to USEMALLOC there is now HEAPOVERRIDE flag that kicks out all Memory* definitions, allowing you to replace the whole allocator with something else than malloc.
Thanks a lot!

P.S. Documentation for MemoryLimitKb has a typo ...

Must be blind, do not see it....

Subject: Re: Core 2019
Posted by Novo on Thu, 11 Jul 2019 18:53:00 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 11 July 2019 14:14Novo wrote on Thu, 11 July 2019 19:55mirek wrote on Thu, 27 June 2019 03:44In addtion to USEMALLOC there is now HEAPOVERRIDE flag that kicks out all Memory* definitions, allowing you to replace the whole allocator with something else than malloc.
Thanks a lot!

P.S. Documentation for MemoryLimitKb has a typo ...

Must be blind, do not see it....
"defualt values"

Subject: Re: Core 2019
Posted by mirek on Fri, 12 Jul 2019 08:09:42 GMT
View Forum Message <> Reply to Message

Novo wrote on Thu, 11 July 2019 19:55mirek wrote on Thu, 27 June 2019 03:44In addtion to USEMALLOC there is now HEAPOVERRIDE flag that kicks out all Memory* definitions, allowing you to replace the whole allocator with something else than malloc.
Thanks a lot!

Any chance to get allocator retested? If still unsatisfactory, send me allocation log?

Mirek

Subject: Re: Core 2019
Posted by Novo on Fri, 12 Jul 2019 14:50:26 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 12 July 2019 04:09Novo wrote on Thu, 11 July 2019 19:55mirek wrote on Thu, 27 June 2019 03:44In addtion to USEMALLOC there is now HEAPOVERRIDE flag that kicks out all Memory* definitions, allowing you to replace the whole allocator with something else than malloc.
Thanks a lot!

Any chance to get allocator retested? If still unsatisfactory, send me allocation log?

Mirek

Yes, I can do that, but that will take me a few weeks. My apps are currently in broken state.

Could you please add info about HEAPOVERRIDE and MemoryOptions (it is just briefly mentioned in docs) to documentation?

Also linking to / embedding of Resolving memory leaks into Heap docs would be great because otherwise this info is distributed among multiple topics and is hard to find. I personally didn't know about the --memory-breakpoint__ trick.

And, I guess, it is possible to add new allocators to U++ via plugins now ...

Subject: Re: Core 2019
Posted by Novo on Thu, 08 Aug 2019 19:47:17 GMT
View Forum Message <> Reply to Message

I checked "Heap implementation" article from "Help Topics" (which for some reason is not

published on web), and it looks like it doesn't match current state of the allocator.

I'm trying to get minimum value of alignment of allocated memory. From a simple test below
```
size_t sz = 0;
void* m = nullptr;
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
```

I got that min alignment is 32. Is this correct?
And min block size is 28. This is a little bit weird.
This message states that "the smallest allocation has size 32 and is 32 bytes aligned", which doesn't match the help topic.

Is it possible to expose allocator-related info via a public enum?
Knowing min alignment is critical. Info about block sizes is also important.

TIA

Subject: Re: Core 2019
Posted by mirek on Thu, 08 Aug 2019 20:47:23 GMT
View Forum Message <> Reply to Message

Novo wrote on Thu, 08 August 2019 21:47I checked "Heap implementation" article from "Help Topics" (which for some reason is not published on web), and it looks like it doesn't match current state of the allocator.

I'm trying to get minimum value of alignment of allocated memory. From a simple test below
```
size_t sz = 0;
void* m = nullptr;
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
```

```
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
m = MemoryAlloc(1);
sz = GetMemoryBlockSize(m);
```

I got that min alignment is 32. Is this correct?
And min block size is 28. This is a little bit weird.
This message states that "the smallest allocation has size 32 and is 32 bytes aligned", which doesn't match the help topic.

Is it possible to expose allocator-related info via a public enum?
Knowing min alignment is critical. Info about block sizes is also important.

TIA

The problem is that the minimal block size is different in debug, as it adds fences everywhere.

In release, 32 holds true.

Mirek

Subject: Re: Core 2019
Posted by mirek on Thu, 08 Aug 2019 20:53:30 GMT
View Forum Message <> Reply to Message

PS.: Guaranteed alignment is 16.

Subject: Re: Core 2019
Posted by mirek on Thu, 08 Aug 2019 20:59:09 GMT
View Forum Message <> Reply to Message

Commited:

```
enum {
 UPP_HEAP_ALIGNMENT = 16,
 UPP_HEAP_MINBLOCK = 32,
};
```

Subject: Re: Core 2019
Posted by Novo on Thu, 08 Aug 2019 21:30:05 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 08 August 2019 16:59Commited:

```
enum {
 UPP_HEAP_ALIGNMENT = 16,
 UPP_HEAP_MINBLOCK = 32,
};
```

Thanks!
I guess, UPP_HEAP_MINBLOCK should be defined separately for Release and Debug. I initially assumed that it is 32 bytes. I'm glad that I checked that this isn't true ...
It looks kile it is 32 * N - 4 in Debug.

Subject: Re: Core 2019
Posted by mirek on Thu, 08 Aug 2019 22:05:32 GMT
View Forum Message <> Reply to Message

Novo wrote on Thu, 08 August 2019 23:30mirek wrote on Thu, 08 August 2019 16:59Commited:

```
enum {
 UPP_HEAP_ALIGNMENT = 16,
 UPP_HEAP_MINBLOCK = 32,
};
```

Thanks!
I guess, UPP_HEAP_MINBLOCK should be defined separately for Release and Debug. I initially assumed that it is 32 bytes. I'm glad that I checked that this isn't true ...
It looks kile it is 32 * N - 4 in Debug.

True, but really this should only be tuning parameter for optimization, thus only relevant in release... (Alignment on the other hand migh be important elsewhere, but that is stable).