Subject: Value question (memory consumption)
Posted by Factor on Mon, 29 Nov 2010 10:31:15 GMT
View Forum Message <> Reply to Message

I've made a simple test with Vector<Value>:

```
#include <Core/Core.h>

using namespace UPP;

#define ITEM_COUNT 1000000

CONSOLE_APP_MAIN {

 Vector<Value> v;

 getchar();

 for(int i=0;i<ITEM_COUNT;i++) v.Add((int)i);

 getchar();

 v.Clear(); v.Shrink();

 getchar();
}
```

1. Adding ITEM_COUNT items (ints for simplicity) to Vector v raises the memory consumption of the process to ~20MB (Win XP - Task manager).
2. After clearing and shrinking Vector v the memory usage drops to 17MB.
As I saw the reference counting works as it should and the "delete *this;" (Void::Release) part of the inner Void class is called, still the memory usage of the process remains high.

I've tried to directly call the ~Value() destructor for all the Vector items before clear, but the result was the same.

Is this the normal behavior?

Subject: Re: Value question (memory consumption)
Posted by cbpporter on Mon, 29 Nov 2010 11:51:54 GMT
View Forum Message <> Reply to Message

Well, a second allocation with the loop will get you to the same memory consumption as the first time. So the memory is reused.

Subject: Re: Value question (memory consumption)
Posted by dolik.rce on Mon, 29 Nov 2010 20:23:06 GMT
View Forum Message <> Reply to Message

Hi Factor,

This is actually not related to Value at all. If you try the same with Vector<int>, you will notice the same behavior.

I think it is caused by the way Vector allocates memory when it grows. If you do v.SetCount(ITEM_COUNT,0), instead of series of Add() calls and then call Clear() and Shrink, you will get back at the memory usage you started with.

I am not sure if this is a bug. I have a strange feeling that this might be actually result of how OS manages the memory. If you ask it for a big chunk at once and then return it later, it might work different than when you are asking for chunks of size increasing as powers of two (which is what vector does).

Maybe someone knowing the memory handling better than me might shed some more light into this. But from what I've seen, I would recommend to use SetCount() every time you know how many items will be necessary.

Best regards,
Honza

Subject: Re: Value question (memory consumption)
Posted by Anonymous on Mon, 29 Nov 2010 21:57:05 GMT
View Forum Message <> Reply to Message

Memory consumption is higher with U++ allocator. I've tested with USEMALLOC flag and the memory consumption was half, but has same behavior, only part of the memory is released.

I've modified example to do new allocations and seems that memory get back to OS, at least here on Linux, if i don't use U++ allocator.

```
#include <Core/Core.h>

using namespace UPP;

#define ITEM_COUNT 1000000

CONSOLE_APP_MAIN {
 Vector<Value> v;
 getchar();
 for(int i=0;i<ITEM_COUNT;i++) v.Add((int)i);
 getchar();
 v.Clear(); v.Shrink();
 getchar();
```

```
 for(int i=0;i<ITEM_COUNT;i++) v.Add((int)i);
 getchar();
 v.Clear(); v.Shrink();
 getchar();
}
```

without U++ allocator:
292 KiB
38.4 MiB
30.8 MiB
38.8 MiB
420 KiB

with U++ allocator:
412 KiB
34.5 MiB (for a sec.) 70.2 MiB maybe Vector double amount of memory it need
62.5 MiB
70.2 MiB
62.5 MiB

Linux 2.6.35-23-generic #40-Ubuntu SMP Wed Nov 17 22:14:33 UTC 2010 x86_64 GNU/Linux

Andrei

Subject: Re: Value question (memory consumption)
Posted by mr_ped on Tue, 30 Nov 2010 08:13:45 GMT
View Forum Message <> Reply to Message

IIRC U++ allocator keeps most of the memory for reuse with later allocation.

Subject: Re: Value question (memory consumption)
Posted by mirek on Tue, 30 Nov 2010 12:21:57 GMT
View Forum Message <> Reply to Message

andreincx wrote on Mon, 29 November 2010 16:57Memory consumption is higher with U++ allocator. I've tested with USEMALLOC flag and the memory consumption was half, but has same behavior, only part of the memory is released.

Have you tested in debug or release (Optimal)? Those numbers look quite high to me (for U++ allocator). In debug mode, U++ inserts a LOT of debugging info into the heap, so the consumption is higher.

(Will have to test myself...

Subject: Re: Value question (memory consumption)
Posted by mirek on Tue, 30 Nov 2010 12:25:25 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Tue, 30 November 2010 03:13IIRC U++ allocator keeps most of the memory for reuse with later allocation.

Yes, only "very large" (> app. 64KB) memory blocks are returned to OS. Rest is kept for later reuse. Anyway, this behaviour is pretty typical for most allocators, only maybe some of them have different threshold.

Subject: Re: Value question (memory consumption)
Posted by mirek on Tue, 30 Nov 2010 12:36:13 GMT
View Forum Message <> Reply to Message

andreincx wrote on Mon, 29 November 2010 16:57Memory consumption is higher with U++ allocator. I've tested with USEMALLOC flag and the memory consumption was half, but has same behavior, only part of the memory is released.

I've modified example to do new allocations and seems that memory get back to OS, at least here on Linux, if i don't use U++ allocator.

```
#include <Core/Core.h>

using namespace UPP;

#define ITEM_COUNT 1000000

CONSOLE_APP_MAIN {
 Vector<Value> v;
 getchar();
 for(int i=0;i<ITEM_COUNT;i++) v.Add((int)i);
 getchar();
 v.Clear(); v.Shrink();
 getchar();
 for(int i=0;i<ITEM_COUNT;i++) v.Add((int)i);
 getchar();
 v.Clear(); v.Shrink();
 getchar();
}
```

without U++ allocator:
292 KiB
38.4 MiB
30.8 MiB
38.8 MiB

420 KiB

with U++ allocator:
412 KiB
34.5 MiB (for a sec.) 70.2 MiB maybe Vector double amount of memory it need
62.5 MiB
70.2 MiB
62.5 MiB

Linux 2.6.35-23-generic #40-Ubuntu SMP Wed Nov 17 22:14:33 UTC 2010 x86_64 GNU/Linux

Andrei

In Win32, memory consumption estimated using task manager, it goes like:

USEMALLOC:
724KB
28.2MB
1.1MB
28.2MB
1.2MB

With U++ allocator:
624KB
20.4MB
16.5MB
20.4MB
16.5MB

So the maximum usage is significantly less than for M$ allocator..

I would like to know why Linux is so much higher.

If it is not because of Debug release, we should check the correctnes of core allocation routines in POSIX:

```
void *SysAllocRaw(size_t size)
{
 sKB += int(((size + 4095) & ~4095) >> 10);
#ifdef PLATFORM_WIN32
 void *ptr = VirtualAlloc(NULL, size, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
#else
#ifdef PLATFORM_LINUX
 void *ptr =  mmap(0, size, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0);
#else
 void *ptr =  mmap(0, size, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANON, -1, 0);
```

```
#endif
 if(ptr == MAP_FAILED)
  ptr = NULL;
#endif
 if(!ptr)
  Panic("Out of memory!");
 DoPeakProfile();
 return ptr;
}

void  SysFreeRaw(void *ptr, size_t size)
{
 sKB -= int(((size + 4095) & ~4095) >> 10);
#ifdef PLATFORM_WIN32
 VirtualFree(ptr, 0, MEM_RELEASE);
#else
 munmap(ptr, size);
#endif
}

int s4kb__;
int s64kb__;

void *AllocRaw4KB()
{
 static int   left;
 static byte *ptr;
 static int   n = 32;
 if(left == 0) {
  left = n >> 5;
  ptr = (byte *)SysAllocRaw(left * 4096);
 }
 n = n + 1;
 if(n > 4096) n = 4096;
 void *p = ptr;
 ptr += 4096;
 left--;
 s4kb__++;
 DoPeakProfile();
 return p;
}

void *AllocRaw64KB()
{
 static int   left;
 static byte *ptr;
 static int   n = 32;
 if(left == 0) {
```

```
  left = n >> 5;
  ptr = (byte *)SysAllocRaw(left * 65536);
 }
 n = n + 1;
 if(n > 256) n = 256;
 void *p = ptr;
 ptr += 65536;
 left--;
 s64kb__++;
 DoPeakProfile();
 return p;
}
```

(the rest is the same for Win32, POSIX):

Mirek

Subject: Re: Value question (memory consumption)
Posted by dolik.rce on Tue, 30 Nov 2010 12:51:30 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 30 November 2010 13:21andreincx wrote on Mon, 29 November 2010 16:57Memory consumption is higher with U++ allocator. I've tested with USEMALLOC flag and the memory consumption was half, but has same behavior, only part of the memory is released.

Have you tested in debug or release (Optimal)? Those numbers look quite high to me (for U++ allocator). In debug mode, U++ inserts a LOT of debugging info into the heap, so the consumption is higher.

(Will have to test myself...

I tested with optimal and U++ allocator.
luzr wrote on Tue, 30 November 2010 13:25Yes, only "very large" (> app. 64KB) memory blocks are returned to OS. Rest is kept for later reuse. Anyway, this behaviour is pretty typical for most allocators, only maybe some of them have different threshold.This explains it. SetCount() allocates all in one big block so it is returned, but Add() causes allocations of 1+2+4+8+...2^16B (that is up to 64KB) which are not returned (any further are bigger than 64KB and therefore returned). That should make ~128KB of non-returned blocks and that agrees exactly with what I measured for Vector<int>. For Vector<Value> it leaves more non-returned blocks probably because there are some additional allocations when constructing the Value...

Thanks for explanation,
Honza

Subject: Re: Value question (memory consumption)
Posted by mirek on Tue, 30 Nov 2010 13:26:36 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Tue, 30 November 2010 07:51I tested with optimal and U++ allocator.

I was responding to 'andreincx' - and more specifically to the claim that U++ allocator eats more memory. If it does, it is a bug and we should solve it...

Subject: Re: Value question (memory consumption)
Posted by mirek on Tue, 30 Nov 2010 13:46:15 GMT
View Forum Message <> Reply to Message

andreincx wrote on Mon, 29 November 2010 16:57Memory consumption is higher with U++ allocator. I've tested with USEMALLOC flag and the memory consumption was half, but has same behavior, only part of the memory is released.

I've modified example to do new allocations and seems that memory get back to OS, at least here on Linux, if i don't use U++ allocator.

```
#include <Core/Core.h>

using namespace UPP;

#define ITEM_COUNT 1000000

CONSOLE_APP_MAIN {
 Vector<Value> v;
 getchar();
 for(int i=0;i<ITEM_COUNT;i++) v.Add((int)i);
 getchar();
 v.Clear(); v.Shrink();
 getchar();
 for(int i=0;i<ITEM_COUNT;i++) v.Add((int)i);
 getchar();
 v.Clear(); v.Shrink();
 getchar();
}
```

without U++ allocator:
292 KiB
38.4 MiB
30.8 MiB
38.8 MiB
420 KiB

with U++ allocator:
412 KiB
34.5 MiB (for a sec.) 70.2 MiB maybe Vector double amount of memory it need
62.5 MiB
70.2 MiB
62.5 MiB

Linux 2.6.35-23-generic #40-Ubuntu SMP Wed Nov 17 22:14:33 UTC 2010 x86_64 GNU/Linux

Andrei

My testing, Ubuntu64:

USEMALLOC:

256KB
38.4MB
30.8MB
38.4MB
384KB

U++ Allocator:

347KB
23.5MB
15.9MB
23.5MB
15.9MB

My bet is that you was testing in DEBUG mode...

Subject: Re: Value question (memory consumption)
Posted by Anonymous on Tue, 30 Nov 2010 15:48:46 GMT
View Forum Message <> Reply to Message

Hello,

Sorry, I haven't had time to respond until now. Yes, i've tested in DEBUG mode only.

On release i've got same results as Mirek with U++ allocator, except for first allocation(364 KiB).
Without U++ allocator, again same results, except first and last allocation (268 KiB, 396 KiB).

Subject: Re: Value question (memory consumption)

Posted by cbpporter on Thu, 09 Dec 2010 13:03:15 GMT
View Forum Message <> Reply to Message

A small question: are there any functions that tell me how much memory is "free" but has not been released back to the OS? To see if I'm wasting memory or is everything alright?

Subject: Re: Value question (memory consumption)
Posted by mirek on Sat, 11 Dec 2010 09:38:30 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 09 December 2010 08:03A small question: are there any functions that tell me how much memory is "free" but has not been released back to the OS? To see if I'm wasting memory or is everything alright?

Actually, there is very comprehensive info about memory behaviour (gets quite handy if someone is proud enough to spend time developing the allocator, right?

For GUI represenation, call MemoryProfileInfo(). Usually, I am putting this somewhere into About box as hidden feature. E.g. in theide, invoke Assist/About.. box and press Alt+M.