TopMemory v3.55

High Performance

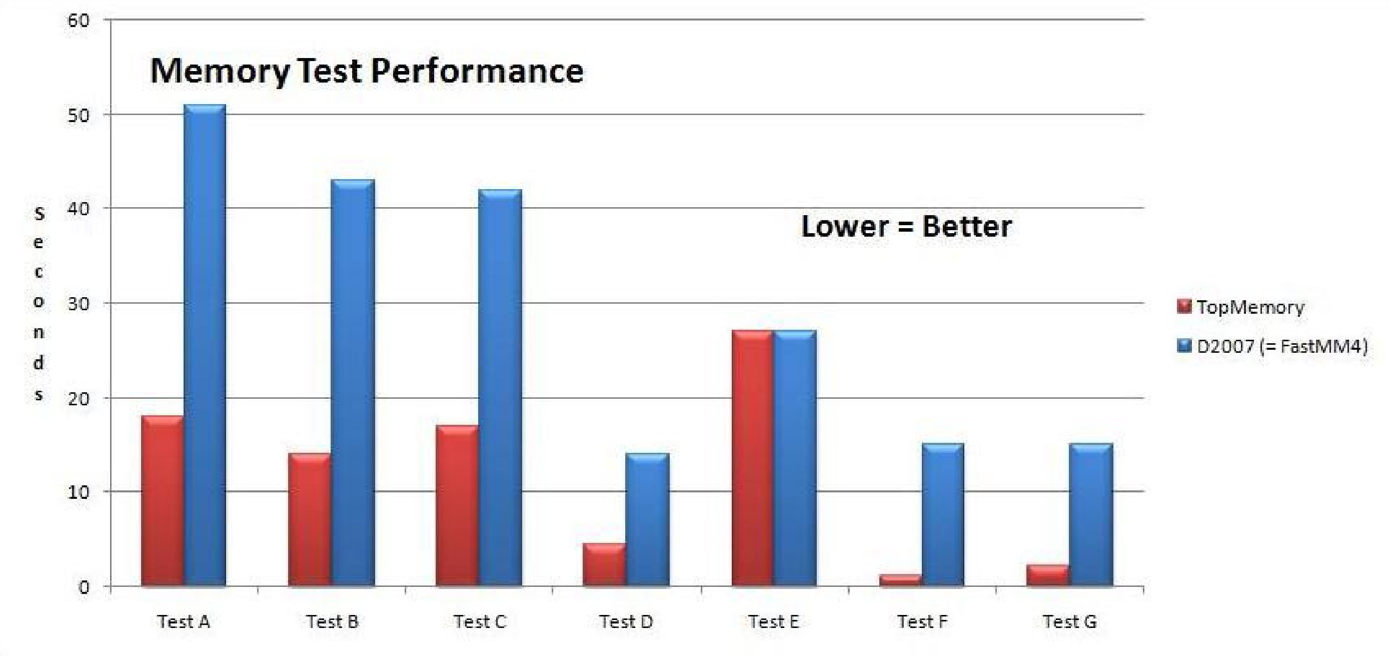
Fully Scalable

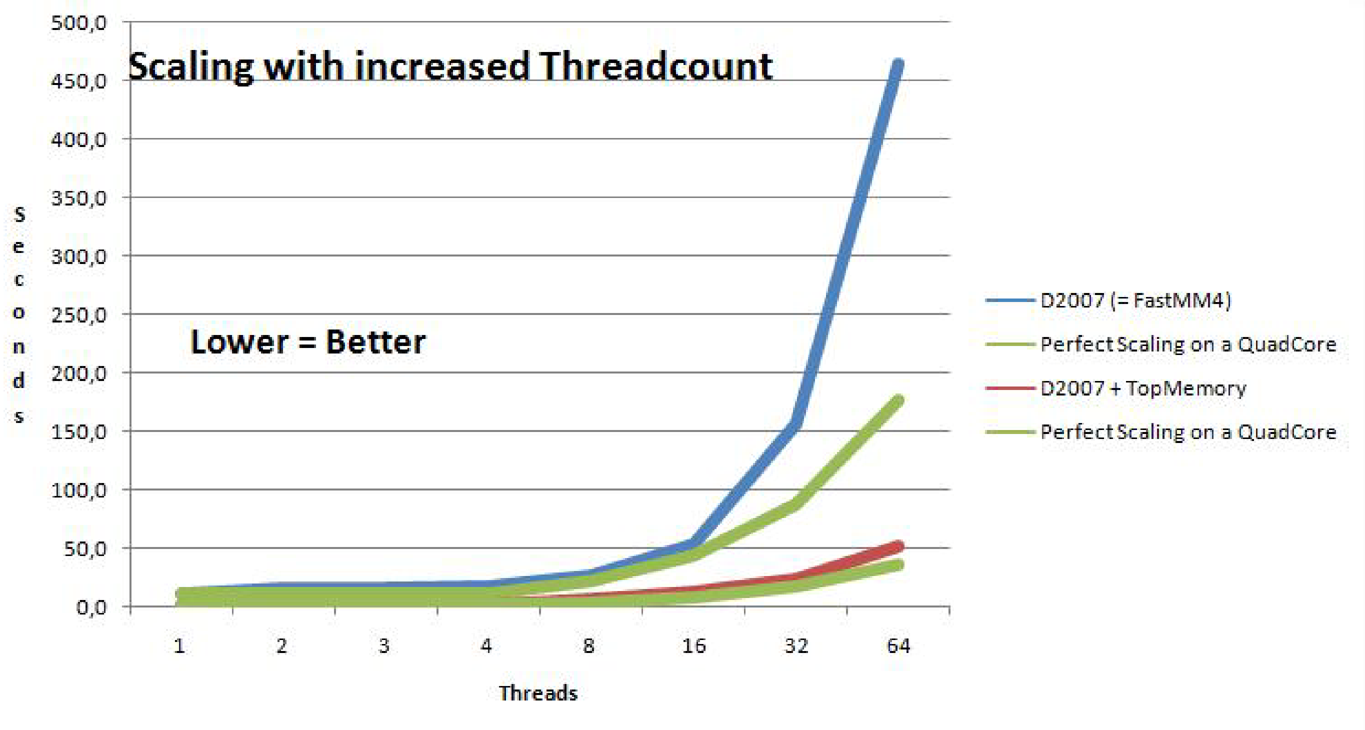
Free

Memory Manager

for

Delphi


Memory Test Performance
Lower = Better
Seconds
60
50
40
30
20
10
0
Test A
Test B
Test C
Test D
Test E
Test F
Test G
TopMemory
D2007 (= FastMM4)


Scaling with increased Threadcount
Lower = Better
Seconds
500,0
450,0
400,0
350,0
300,0
250,0
200,0
150,0
100,0
50,0
0,0
1
2
3
4
8
16
32
64
Threads
D2007 (= FastMM4)
Perfect Scaling on a QuadCore
D2007 + TopMemory
Perfect Scaling on a QuadCore

Introduction

TopMemory is a Memory Manager replacement for Delphi. It was written to improve the performance of Delphi applications. For those that wonder why you would want a new memory manager I have included benchmarks to show that a new memory manager can really make a very big difference, especially in multithreaded applications. *Even if you are already using FastMM4 or the new 2006+ memory manager!*

TopMemory does not cost money, you can use it for free. All sourcecode is included. Please do email if you have questions, remarks etc. TopMemory has been extensively tested with Delphi 2007 and should work with Delphi 5 and higher.

For support;

Website	->	www.topsoftwaresite.nl	
Email	->	support@topsoftwaresite.nl	(Ivo Tops)

© Notices and Thanks go to;

- Robert Lee
  For MultiMM of which I have used both some concepts and the trick to hook the threads.

- Alexey A. Dynnikov
  For his CPU measurement unit which I have put to use in the testtool

- Victor van Uitert, Andre Mussche and Thaddy de Koning for testing and feedback

- The Fastcode project and it's participants for a fun coding competition

Table of Contents

If you use TopMM mail me what you would like me to improve. I’m thinking of;

- Free Pascal Support
- Checking whether we can use some Vista features (new locking etc.)
- 64 bit support (for FPC first and Delphi when it get’s 64 bit support itself)

Releasenotes

V3.55

- Removed Math en System unit usage
- Fixed AV when closing applications using Delphi7 on some systems

V3.54

- Performance improved significantly
- Enabled 4GB support so your applications can use up to 4 GB
  (read how to enable this on page 8)
- Fixed 64k Aliasing slowdowns on P4+HT (thanx Andre Mussche!)
- Cleaned code to use Cardinal instead of Integer for 4GB support
- Fixed rare bug with large not inplace re-allocations
- Updated charts and documentation

V2.20

- Fixed version detection so Delphi 2007 and higher will use new features
- Improved shutdown report with logging to Disk and IDE, easily configurable from code
- Detection and cleanup memory used by non-delphi threads that used memory but are done. Delphi 2006 and higher only

V2.10

- Added memory Leaks ShutDown report (through Debug Event Log window in IDE)
  - o Reports threads not returning memory upon application exit
  - o Reports all memory unexpectedly leaked

  See new manual section on this topic
- TopMaintenance thread is now properly shut down and waited shortly for upon closing

V2.00

- Added SSE2 support for improved memory copy performance
- Allocations of 16 bytes or more are now always 16 byte aligned (useful for SSE2)
- Allocations smaller as 16 bytes are now always 4 byte aligned
- Removed TOPSPEED compiler directive (memory used was too big for performancegain)
- Added spinlockcount to critical sections
- Successfully and Extensively Tested with Delphi 2006
- For Delphi 2006 VMT endthread patch is no longer needed (hooking a fine callback now)
- Tweaked some of the strategies for extra performance
- TopMemory units will now only show up while debugging if you have
  Specified the TOPDEBUG compiler directive
- Updated benchmarks
- Added TOP_BLOCK to the Compiler Directives documentation
- Fixed bug where reallocating a small amount of memory to a massive amount that gives out of memory will lock up the memory manager

V1.52

- Removed unused FPUxx routines (that did not compile in Delphi5)

v1.51

- Pool was not always being used even if memory in it. Fixed

v1.50

- First Public Release

Benchmarks

See the included Benchmarks spreadsheet for my findings. Especially multicore machines benefit from using TopMemory. Gains of up too 300% (for memory intensive apps) are possible when compared to the standard Delphi 2007 memory manager!

The results were obtained using my own benchmarking and validation tool. But benchmarks are synthetic. The best way to see if TopMemory helps your application is to try it and see your runtime improve!

How do You use TopMemory?

- Make sure your application can find the TopMemory units. Add the path to the Delphi Library paths or to your applications search path.

- Add TopMemory as the first used unit to your project file (.dpr)

```
program Project1;

uses
 TopMemory,
 Forms,
 Main in 'Main.pas',
 ..;
```

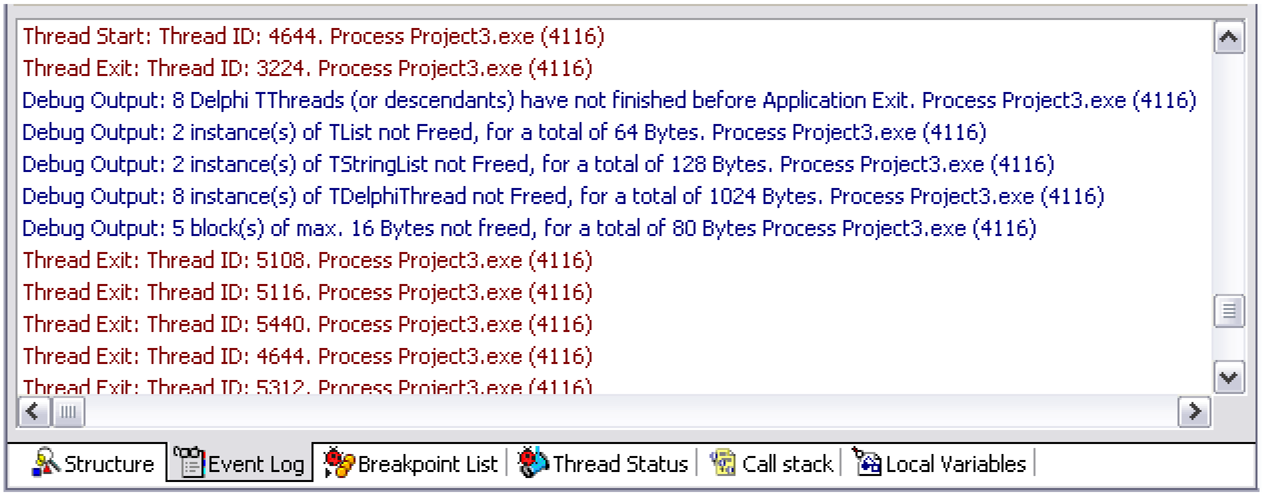
- Perform a full Build of your application.

Detecting Memory leaks

When running from the IDE pay attention to the debug event log window. Upon stopping your application TopMemoryManager will report any unexpected memory leaks and threads that have not stopped. See the screenshot below.

Edit the three Booleans in unit TOPMemory.pas if you want to globally change the way memory leak reporting is done. You can switch logging off, or specify logging to be done only to IDE or only to a logfile. You might also change these variables in your application at any time. Include TOPReporting.pas in your uses clause and set these parameters from code.

Not all memory leaks reported have be a problem. Some might even be from VCL components. Since D2006 you can register expected memory leaks in code with the Memory Manager. These will not be reported. See documentation on RegisterExpectedMemoryLeak in the Delphi documentation.



```
Thread Start: Thread ID: 4644. Process Project3.exe (4116)
Thread Exit: Thread ID: 3224. Process Project3.exe (4116)
Debug Output: 8 Delphi TThreads (or descendants) have not finished before Application Exit. Process Project3.exe (4116)
Debug Output: 2 instance(s) of TList not Freed, for a total of 64 Bytes. Process Project3.exe (4116)
Debug Output: 2 instance(s) of TStringList not Freed, for a total of 128 Bytes. Process Project3.exe (4116)
Debug Output: 8 instance(s) of TDelphiThread not Freed, for a total of 1024 Bytes. Process Project3.exe (4116)
Debug Output: 5 block(s) of max. 16 Bytes not freed, for a total of 80 Bytes Process Project3.exe (4116)
Thread Exit: Thread ID: 5108. Process Project3.exe (4116)
Thread Exit: Thread ID: 5116. Process Project3.exe (4116)
Thread Exit: Thread ID: 5440. Process Project3.exe (4116)
Thread Exit: Thread ID: 4644. Process Project3.exe (4116)
Thread Exit: Thread ID: 5312. Process Project3.exe (4116)
```

Using more then 2 GB

You need to add a compilerdirective in your project file to enable large memory support on windows. With this switch your application can allocate up to 4 GB on 64 bit windows versions. And up to around 3GB on 32 bit windows versions that have been booted with the /3GB switch in boot.ini.

You need to add this line into the project file, for instance right after the uses clause where you added TopMemory.

```
{$SetPEFlags IMAGE_FILE_LARGE_ADDRESS_AWARE}
```

The Constant IMAGE_FILE_LARGE_ADDRESS_AWARE is defined in windows.pas, so you need add that to your uses clause also.

After this your pointer's can become greater than MaxInt. Be aware that if you do arithmetic with pointers in your program you need to make sure you cast them to cardinal's, not integers.

Alignment

All allocations get aligned according to these rules;

Allocation < 4 bytes – Not aligned
Allocation < 8 bytes – 4 byte aligned
Allocation < 16 bytes – 8 byte aligned
All other (larger) allocations are 16 byte aligned

A tip for people wanting to use **SSE** Assembler and use aligned data;

Beware that open arrays will not be aligned correctly for SSE because Delphi allocates the data and uses the first 8 bytes for some housekeeping. There are QC requests with CodeGear to change this.

If you want to use large aligned structures for SSE then create an arraytype and allocate the data yourself like this;

```
Data := AllocMem(4096);  // Data will be 16 byte aligned by TopMemory
TArrayOfSSE2RecordsOf16Bytes(Data)[0] // First item will be 16 byte aligned
```

If you use

```
Data : Array of SSE2RecordsOf16Bytes;
SetLength(Data, 4096 div 16);//Delphi will alloc 4096 + 8 bytes housekeeping
```

The First Item Data[0] will NOT be aligned because of the housekeeping data. It will start 8 bytes after the actual memory allocation to TopMemory that Delphi internally does.

Compiler directives

For normal operations no compiler directives are required.

There are a couple of directives which might be useful when debugging

TOP_BLOCK For testing purposes. Makes all calls to the memory manager blocking and as such rules out problems as being multithreading issues.

TOP_BLOCK_REPORT
Disables the Memory leaks report on shutdown

TOPDEBUG Enables debugging the memory manager

Memory Usage

Part of the speedup that TopMemory achieves comes from allocating more memory then necessary and grouping allocations that are in the same sizerange. Therefore your application will use more memory when it runs. If you care more about low memory usage then TopMemory is not for you. It is optimized for performance at the cost of memory. Which is not usually a problem with the amounts of memory even regular PC’s have these days.

Debugging

Assertions and Exceptions do not go well with Memory Managers. Both result in confusing runtime errors. Therefore TopMemory code has a different solution. All extra checks are with the TOPDEBUG compiler directive. How to debug something you think is a problem due of TopMemory?

First recompile (full build) without TopMemory and check if all goes well. If it does not, fix the problem in your code, this is not a TopMemory problem.

If Delphi runs your code, but not with TopMemory used, debug along these lines;

1 -> First set the TOPDEBUG compiler directive

Using the Delphi Debugger

2 -> Set a breakpoint on the first line of the DebugError procedure in the TopInstall unit
3 -> Optionally turn Optimization off and StackFrames on so you will Get better stack traces and variable info upon a break

4 -> Run the application

5 -> When you break into DebugError the S variable will contain the error.

Not using the Delphi Debugger

2 -> Run the application

3 -> The newly created file TOPMMDEBUG.ERR will contain the debug error.

Finally search for the error in the units named with Top*.pas. If the comment does not say it is for example due to a double free of the same pointer then please mail the error to support@topsoftwaresite.nl. I will try to fix the problem. Otherwise check your code and try to correct the problem there.

TopMemory Architecture

TopMemory employs several techniques to speed up memory allocations. Without going into to much detail these are the highlights;

- Separate memory manager for each thread allowing for fully non blocking memory operations within the context of the thread.

- Grouping of memory allocations in blocks of a certain size. TopMemory only gives chunks of 4, 8, 16, 32, .. bytes. It also allocates them in groups and the number of chunks in subsequent groups grows. This allows for increased re-use of memory and less frequent OS memory operations (at the expense of using some extra memory).

- Pools for each size to which blocks are moved after being freed. Blocks live in the pool and are only freed to the OS after a period of non-use. All threads that require new memory first consult the pool and only allocate new memory if the pool is empty

- Smaller sizes retain some blocks within each thread so even pool access will not slow them down.

- A maintenance thread that culls the pool of excess blocks that have not been used in recent time.

- Memory not allocated through TopMemory is recognized and will automatically be redirected to the old memory manager.

- Each poolmanager for a certain allocation size has a number of sublists with blocks so even when all poolrequests are for the same allocation size they will not instantly be blocking the entire pool.

- Automatic alignment of ALL allocations on 4 byte boundaries (allocation < 16) and on 16 byte boundaries (allocation >= 16 bytes).

- Reporting of unfinished threads and memory leaks upon shutdown

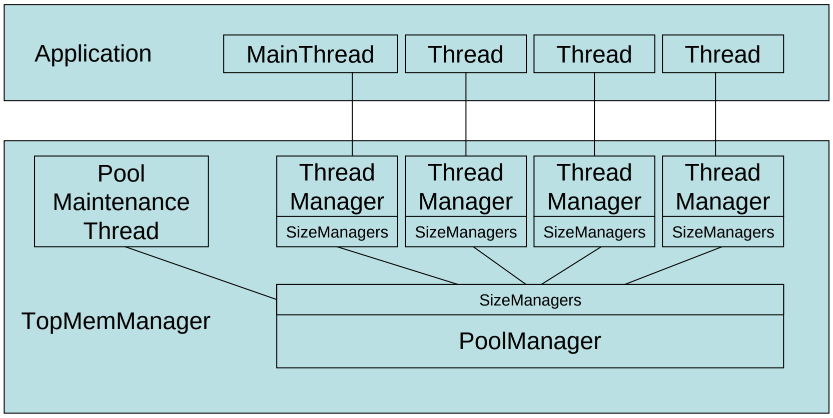
Conceptual overview

TopMemory allocates a threadmanager to each new thread. Within the threadmanager a list of sizemanagers is created that handle allocations of certain sizes. All allocations between x-y bytes end up in the same sizemanager. Blocks of multiple series of y bytes are then allocated and issued to the application.

The bulk of the extra performance is gained in two ways; First there is no critical section that threads need to enter to get memory because they have their own threadmanager. Secondly the pre-allocation in blocks means we have to go a lot less to the OS for memory allocations.

To improve synergy between the threads there is a pool that bundles blocks before releasing them to the OS.

TopMemory Architecture



Cross Thread performance hit

TopMemory is fast because it is lock free and each thread manages it’s own pool of memory. A lock is required however if you free memory in another thread then the thread it was allocated in. If you do this a lot it could degrade your applications performance unnecessarily.. As a rule (of good programming) you should always try to allocate and free data both in the same pieces of code, but also in the same threadcontext.

So instead of allocating data in the main thread and free it in your workerthreads it is better to either also allocate it in the workerthreads or in the main thread and also free it in the main thread.

This design is not only beneficial when you use TopMemory. Newer versions of Windows and Computer architectures try to match processors, memory and threads close together and keep them linked to gain performance benefits. Read up on e.g. the NUMA architecture.

Nitty Gritty Deep Down Stuff

Double Free does not give an error

TopMemory does not behave exactly identical as the regular Memory Manager when your program does ‘bad’ things. Freeing the same pointer twice will raise an exception with the normal manager (Access Violation in GetMem.Inc).

TopMemory might accept it but then re-issue the memory later which was incorrectly freed twice. Or give it back to the OS twice. Using the TOPDEBUG conditional will check for this error and also zero all memory returned immediately. This ofcourse carries a performance penalty. Please consult the debugging section further down in this manual on how to best use this conditional. Use TOPDEBUG only for testing, not for production.

Detecting the end of a thread’s life

Topmemorymanager has a submanager for each thread. Therefore it needs to know when a thread stops.

In versions of Delphi before 2006 TopMM patches the EndThread method. Your code will no longer go through System.EndThread but through TopMemory.EndThread. If you want to debug it set your breakpoint in the TopMemory unit.

In Delphi 2006 TopMemoryManager simply hooks the TSystemThreadEndProc function. This new method is far better and we thank CodeGear for providing the new Hook <smile>.