TotalView New Features

Etnus

June 2004

version 6.5

Contents

New Features

New Platforms and Compilers 1

New and Changed GUI Features 2

Tools > Memory Debugging Command Added 2
Node Display in the Variable Window 4
STL String data types Transformed 4
Type Transformations 4

New Features

This booklet contains information about changes made to TotalView for version 6.5.

The information in this document is to let you know what changes have occurred. You'll find descriptions for all changes within the *TotalView Users Guide*.

TotalView has many features and it gives you a great number of tools for finding your program's problems. An easy way to get acquainted with these features is to subscribe to the "Tip of the Week". If you subscribe to this mailing list, you'll receive an email message every week that tells you something about TotalView.

- All of the tips are archived on our web site at **http://www.etnus.com/Support/Tips/index.html**.
- If you like what you see, you can subscribe at **http://www.etnus.com/mojo/mojo.cgi**.

New Platforms and Compilers

TotalView now supports the following operating system versions:

- Red Hat Fedora Core 1 on x86 architectures.
- SuSE Linux Profession 9.0 and SuSe Linux Personal on x86 and x86-64 architectures.

TotalView now supports the following compilers:

- gcc 3.4.0 for C and C++ on most platforms.
- gcc 3.4.0 for Fortran 77 on x86, x86-64, and ia64 Linux.
- Intel C and C++ 8.0.066 on x86 and ia64 Linux.
- Intel Fortran 8.0.046 on x86 and ia64 Linux
- Portland Group C and C++ 5.1 on x86 and x86-64 Linux.

For complete information, see the *TotalView Platforms Guide*.

New and Changed GUI Features

Tools > Memory Debugging Command Added

This release of TotalView adds to the memory debugging features that previously existed within TotalView. It also consolidates memory debugging interactions within one window.

The TotalView Memory Debugger can help you locate many of your program's memory problems. For example, you can:

- Stop execution when **free()**, **realloc()**, and other heap API problems occur.

  If your program tries to free memory that it can't or shouldn't free, the Memory Debugger can stop execution. This lets you can identify the statement that caused the problem.
- List leaks.

  The Memory Debugger can display your program's leaks. (*Leaks* are memory blocks that are allocated, but which are no longer referenced.)

  When your program allocates a memory block, the Memory Debugger creates a backtrace. When it makes a list of your leaks, it includes this backtrace in the list. This lets you see the place where your program allocated the memory block.
- Paint allocated and deallocated blocks.

  When your program's memory manager allocates or deallocates memory, the Memory Debugger can write a bit pattern into it. Writing this bit pattern is called *painting*.

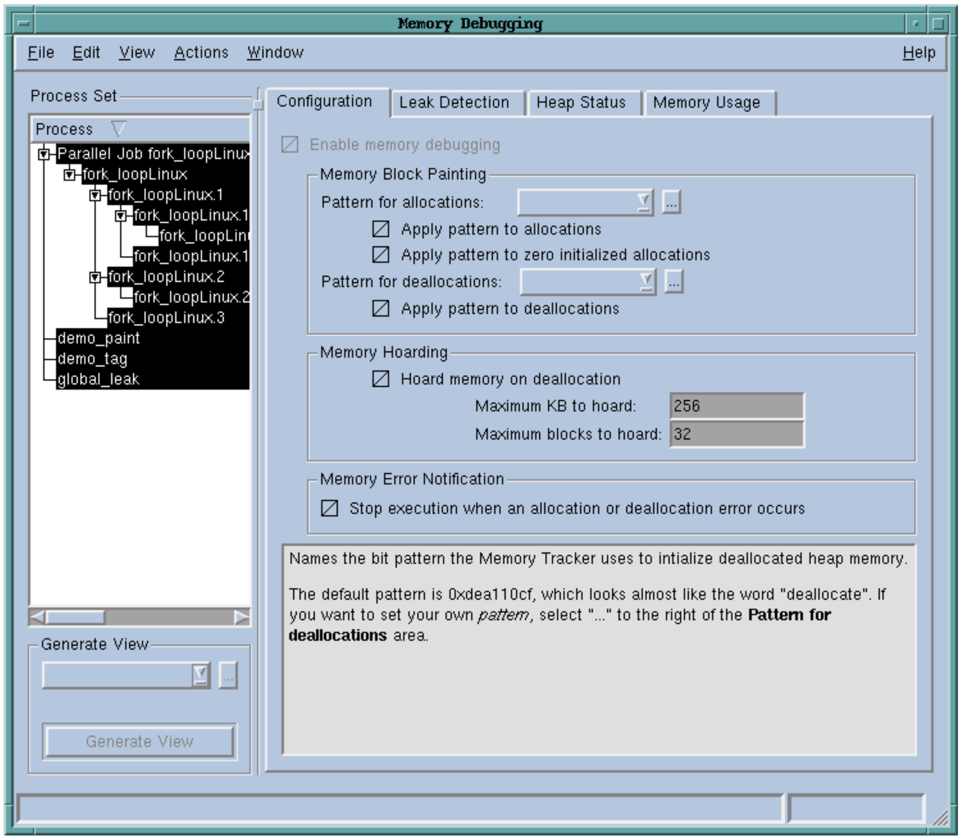
  When you see this bit pattern in a Variable or Expression List Window, you can tell that you are using memory before your program initializes it or after your program deallocates it. Depending upon the architecture, you might even be able to force an exception when your program accesses this memory.
- Identify dangling pointers.

  A dangling pointer is a pointer that points into deallocated memory. If the pointer being displayed in a Variable is dangling, TotalView adds information to the data element so that you know about the problem.
- Hold onto deallocated memory.

  When trying to identify memory problems, holding on to memory after your program releases it can sometimes help locate problems. Holding onto freed memory is called *hoarding*.

  For example, retaining a block can sometimes force a memory error to occur. Or, when coupled with painting, you'll be able to tell when your program is trying to access deallocated memory.

After you select the **Tools > Memory Debugging** command, TotalView displays the following window:



Enable memory debugging

Memory Block Painting

Pattern for allocations:

Apply pattern to allocations

Apply pattern to zero initialized allocations

Pattern for deallocations:

Apply pattern to deallocations

Memory Hoarding

Hoard memory on deallocation

Maximum KB to hoard: 256

Maximum blocks to hoard: 32

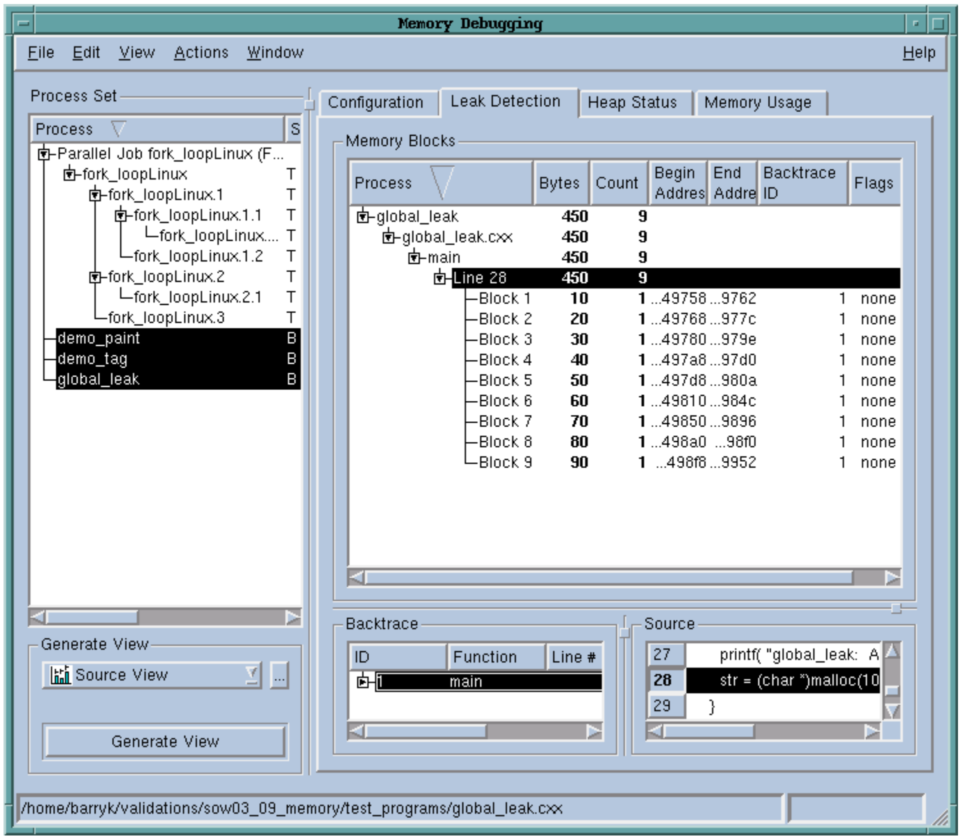
Memory Error Notification

Stop execution when an allocation or deallocation error occurs

Names the bit pattern the Memory Tracker uses to intialize deallocated heap memory.

The default pattern is 0xdea110cf, which looks almost like the word "deallocate". If you want to set your own *pattern*, select "..." to the right of the **Pattern for deallocations** area.

If memory debugging is enabled, you can tell the Memory Debugger to display information whenever execution stops. For example, here is a window showing leak information:



Process	S
Parallel Job fork_loopLinux (F...	
fork_loopLinux	T
fork_loopLinux.1	T
fork_loopLinux.1.1	T
fork_loopLinux....	T
fork_loopLinux.1.2	T
fork_loopLinux.2	T
fork_loopLinux.2.1	T
fork_loopLinux.3	T
demo_paint	B
demo_tag	B
global_leak	B

Process	Bytes	Count	Begin Addres	End Addre	Backtrace ID	Flags
global_leak	450	9				
global_leak.cxx	450	9				
main	450	9				
Line 28	450	9				
Block 1	10	1	...49758	...9762	1	none
Block 2	20	1	...49768	...977c	1	none
Block 3	30	1	...49780	...979e	1	none
Block 4	40	1	...497a8	...97d0	1	none
Block 5	50	1	...497d8	...980a	1	none
Block 6	60	1	...49810	...984c	1	none
Block 7	70	1	...49850	...9896	1	none
Block 8	80	1	...498a0	...98f0	1	none
Block 9	90	1	...498f8	...9952	1	none

ID	Function	Line #
1	main	

```
27     printf( "global_leak:  A
28     str = (char *)malloc(10
29   }
```

/home/barryk/validations/sow03_09_memory/test_programs/global_leak.cxx

The Backtrace Pane shows the stack frames that existed when your program allocated a memory block. The Source Pane shows the line where it made the allocation.

For more information, see the *Debugging Memory Problems Using TotalView* document.

Node Display in the Variable Window

The View > Nodes command was removed. This command was only used when viewing UPC variables. You can see the nodes upon which a variable resides by right-clicking on the column headers and selecting **Node**.

STL String data types Transformed

STLView now transforms String data types.

Type Transformations

The way in which you create type transformations has been simplified. While older methods still work, the new methods are more direct. For information, see the "*Creating Type Transformations*" chapter of the *TotalView Reference Guide*.

The *Type Transformations Guide* has been archived on our web site. It is will no longer be updated. However, it may be useful if you are attempting to transform a very difficult data structure or class.