kmemtrace.txt

kmemtrace - Kernel Memory Tracer

by Eduard - Gabriel Munteanu
<eduard.munteanu@linux360.ro>

I. Introduction

kmemtrace helps kernel developers figure out two things:
1) how different allocators (SLAB, SLUB etc.) perform
2) how kernel code allocates memory and how much

To do this, we trace every allocation and export information to the userspace through the relay interface. We export things such as the number of requested bytes, the number of bytes actually allocated (i.e. including internal fragmentation), whether this is a slab allocation or a plain kmalloc() and so on.

The actual analysis is performed by a userspace tool (see section III for details on where to get it from). It logs the data exported by the kernel, processes it and (as of writing this) can provide the following information:
- the total amount of memory allocated and fragmentation per call-site
- the amount of memory allocated and fragmentation per allocation
- total memory allocated and fragmentation in the collected dataset
- number of cross-CPU allocation and frees (makes sense in NUMA environments)

Moreover, it can potentially find inconsistent and erroneous behavior in kernel code, such as using slab free functions on kmalloc'ed memory or allocating less memory than requested (but not truly failed allocations).

kmemtrace also makes provisions for tracing on some arch and analysing the data on another.

II. Design and goals

kmemtrace was designed to handle rather large amounts of data. Thus, it uses the relay interface to export whatever is logged to userspace, which then stores it. Analysis and reporting is done asynchronously, that is, after the data is collected and stored. By design, it allows one to log and analyse on different machines and different arches.

As of writing this, the ABI is not considered stable, though it might not change much. However, no guarantees are made about compatibility yet. When deemed stable, the ABI should still allow easy extension while maintaining backward compatibility. This is described further in Documentation/ABI.

Summary of design goals:
- allow logging and analysis to be done across different machines
- be fast and anticipate usage in high-load environments (*)
- be reasonably extensible
- make it possible for GNU/Linux distributions to have kmemtrace included in their repositories

(*) - one of the reasons Pekka Enberg's original userspace data analysis tool's code was rewritten from Perl to C (although this is more than a

simple conversion)

III. Quick usage guide
======================

1) Get a kernel that supports kmemtrace and build it accordingly (i.e. enable CONFIG_KMEMTRACE).

2) Get the userspace tool and build it:
```
$ git clone git://repo.or.cz/kmemtrace-user.git         # current repository
$ cd kmemtrace-user/
$ ./autogen.sh
$ ./configure
$ make
```

3) Boot the kmemtrace-enabled kernel if you haven't, preferably in the 'single' runlevel (so that relay buffers don't fill up easily), and run kmemtrace:
```
# '$' does not mean user, but root here.
$ mount -t debugfs none /sys/kernel/debug
$ mount -t proc none /proc
$ cd path/to/kmemtrace-user/
$ ./kmemtraced
Wait a bit, then stop it with CTRL+C.
$ cat /sys/kernel/debug/kmemtrace/total_overruns        # Check if we didn't
                                                        # overrun, should
                                                        # be zero.
$ (Optionally) [Run kmemtrace_check separately on each cpu[0-9]*.out file to
                check its correctness]
$ ./kmemtrace-report
```

Now you should have a nice and short summary of how the allocator performs.

IV. FAQ and known issues
========================

Q: 'cat /sys/kernel/debug/kmemtrace/total_overruns' is non-zero, how do I fix this? Should I worry?
A: If it's non-zero, this affects kmemtrace's accuracy, depending on how large the number is. You can fix it by supplying a higher 'kmemtrace.subbufs=N' kernel parameter.

Q: kmemtrace_check reports errors, how do I fix this? Should I worry?
A: This is a bug and should be reported. It can occur for a variety of reasons:
- possible bugs in relay code
- possible misuse of relay by kmemtrace
- timestamps being collected unorderly

Or you may fix it yourself and send us a patch.

Q: kmemtrace_report shows many errors, how do I fix this? Should I worry?
A: This is a known issue and I'm working on it. These might be true errors in kernel code, which may have inconsistent behavior (e.g. allocating memory

with kmem_cache_alloc() and freeing it with kfree()). Pekka Enberg pointed out this behavior may work with SLAB, but may fail with other allocators.

It may also be due to lack of tracing in some unusual allocator functions.

We don't want bug reports regarding this issue yet.

V. See also
==========

Documentation/kernel-parameters.txt
Documentation/ABI/testing/debugfs-kmemtrace