Subject: Re: [PATCH 2/5] slab_id: Generic slab ID infrastructure
Posted by Glauber Costa on Fri, 07 Oct 2011 08:27:46 GMT
View Forum Message <> Reply to Message

Hi Pavel,

On 10/06/2011 08:23 PM, Pavel Emelyanov wrote:
> The idea of how to generate and ID for an arbitrary slab object is simple:
>
> - The ID is 128 bits
> - The upper 64 bits are slab ID
> - The lower 64 bits are object index withing a slab (yes, it's too many,
> but is done for simplicity - not to deal with 96-bit numbers)
> - The slab ID is the 48-bit per-cpu monotonic counter mixed with 16-bit
> cpuid. Even if being incremented 1M times per second the first part
> will stay uniqe for 200+ years. The cpuid is required to make values
> picked on two cpus differ.

So why can't we just use tighter numbers, and leave some reserved fields
instead ?

Having ids in the objects of the slab may prove useful in the future for
other uses as well.

For instance, concurrent to that, we're trying to figure out ways to
have per-cgroup pages/objects accounted in the memory controller.

The most up2date proposals create an entire kmem cache for each cgroup,
thus trivially guaranteeing uniqueness. It however, leads to fragmentation.
Having the objects to be IDed and being cgroup part of this id, could
help us achieve the same goal with less fragmentation.

>
> Signed-off-by: Pavel Emelyanov<xemul@parallels.com>
>
> ---
> include/linux/slab.h | 17 ++++++++++++++++
> init/Kconfig | 9 +++++++++
> mm/Makefile | 1 +
> mm/slab_obj_ids.c | 25 +++++++++++++++++++++++++
> 4 files changed, 52 insertions(+), 0 deletions(-)
> create mode 100644 mm/slab_obj_ids.c
>
> diff --git a/include/linux/slab.h b/include/linux/slab.h
> index 573c809..ae9c735 100644
> --- a/include/linux/slab.h
> +++ b/include/linux/slab.h
> @@ -23,6 +23,7 @@

> #define SLAB_CACHE_DMA 0x00004000UL /* Use GFP_DMA memory */
> #define SLAB_STORE_USER 0x00010000UL /* DEBUG: Store the last owner for bug hunting */
> #define SLAB_PANIC 0x00040000UL /* Panic if kmem_cache_create() fails */
> +#define SLAB_WANT_OBJIDS 0x00080000UL /* Want GENERIC_OBJECT_IDS-friendly slabs */
> /*
>   * SLAB_DESTROY_BY_RCU - **WARNING** READ THIS!
>   *
> @@ -162,6 +163,22 @@ void kfree(const void *);
> void kzfree(const void *);
> size_t ksize(const void *);
>
> +#ifdef CONFIG_SLAB_OBJECT_IDS
> +void __slab_pick_id(u64 *s_id);
> +static inline void __slab_get_id(u64 *id, u64 s_id, u64 o_id)
> +{
> + id[0] = o_id;
> + id[1] = s_id;
> +}
> +
> +void k_object_id(const void *, u64 *id);
> +#else
> +static inline void k_object_id(const void *x, u64 *id)
> +{
> + id[0] = id[1] = 0;
> +}
> +#endif
> +
> /*
>   * Allocator specific definitions. These are mainly used to establish optimized
>   * ways to convert kmalloc() calls to kmem_cache_alloc() invocations by
> diff --git a/init/Kconfig b/init/Kconfig
> index d627783..4c1c0e6 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -1200,6 +1200,15 @@ config SLUB_DEBUG
> SLUB sysfs support. /sys/slab will not exist and there will be
> no support for cache validation etc.
>
> +config SLAB_OBJECT_IDS
> + default y
> + bool "Enable slab kernel object ID infrastructure"
> + depends on !SLOB
> + help
> + This option provides an infrastructure for calculating ID-s of
> + slab/slub objects. These ID-s are not based on the object location
> + in memory and thus can be shown to the userspace.

```
> +
>   config COMPAT_BRK
>    bool "Disable heap randomization"
>    default y
> diff --git a/mm/Makefile b/mm/Makefile
> index 836e416..fb65080 100644
> --- a/mm/Makefile
> +++ b/mm/Makefile
> @@ -50,3 +50,4 @@ obj-$(CONFIG_HWPOISON_INJECT) += hwpoison-inject.o
>   obj-$(CONFIG_DEBUG_KMEMLEAK) += kmemleak.o
>   obj-$(CONFIG_DEBUG_KMEMLEAK_TEST) += kmemleak-test.o
>   obj-$(CONFIG_CLEANCACHE) += cleancache.o
> +obj-$(CONFIG_SLAB_OBJECT_IDS) += slab_obj_ids.o
> diff --git a/mm/slab_obj_ids.c b/mm/slab_obj_ids.c
> new file mode 100644
> index 0000000..87d1693
> --- /dev/null
> +++ b/mm/slab_obj_ids.c
> @@ -0,0 +1,25 @@
> +#include<linux/percpu.h>
> +
> +#define SLUB_ID_CPU_SHIFT 16
> +static DEFINE_PER_CPU(u64, slub_ids);
> +
> +void __slab_pick_id(u64 *s_id)
> +{
> + int cpu;
> + u64 id;
> +
> + /*
> +  * The idea behind this all is very simple:
> +  *
> +  * The ID is the 48-bit per-cpu monotonic counter mixed with 16-bit cpuid.
> +  * Even if being incremented 1M times per second the first part will stay
> +  * uniqe for 200+ years. The cpuid is required to make values picked on
> +  * two cpus differ.
> +  */
> +
> + cpu = get_cpu();
> + id = ++per_cpu(slub_ids, cpu);
> + WARN_ON_ONCE(id>>  (64 - SLUB_ID_CPU_SHIFT) != 0);
> + *s_id = (id<<  SLUB_ID_CPU_SHIFT) | cpu;
> + put_cpu();
> +}
```