
Subject: [PATCH 2/5] Generic notifiers for SLUB events
Posted by [Pavel Emelianov](#) on Tue, 25 Sep 2007 14:19:05 GMT
[View Forum Message](#) <> [Reply to Message](#)

If the user wants more than one listener for SLUB event, it can register them all as notifier_block-s so all of them will be notified.

This costs us about 10% of performance loss, in comparison with static linking.

The selected method of notification is srcu notifier blocks. This is selected because the "call" path, i.e. when the notification is done, is lockless and at the same time the handlers can sleep. Neither regular nor atomic notifiers provide such facilities.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/init/Kconfig b/init/Kconfig
index 684ccfb..e9acc29 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -593,6 +599,16 @@ config SLUB_DEBUG
     SLUB sysfs support. /sys/slab will not exist and there will be
     no support for cache validation etc.
```

```
+config SLUB_NOTIFY
+ default y
+ bool "Enable SLUB events generic notification"
+ depends on SLUB
+ help
+   When Y, this option enables generic notifications of some major
+   slub events. However, if you do know that there will be the
+   only listener for them, you may say N here, so that callbacks
+   will be called directly.
+
+ choice
+   prompt "Choose SLAB allocator"
+   default SLUB
diff --git a/mm/slub.c b/mm/slub.c
index 31d04a3..e066a0e 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1040,6 +1040,89 @@ static inline unsigned long kmem_cache_f
 * notifiers
```

*/

```
+ #ifdef CONFIG_SLUB_NOTIFY
+ static struct srcu_notifier_head slub_nb;
+
+ static inline
+ int __slub_notify_fail(int cmd_alloc, int cmd_free, struct kmem_cache *cachep,
+ void *obj, gfp_t gfp)
+ {
+ int ret, called = 0;
+ struct slub_notify_struct arg;
+
+ arg.cachep = cachep;
+ arg.objp = obj;
+ arg.gfp = gfp;
+
+ ret = __srcu_notifier_call_chain(&slub_nb, cmd_alloc, &arg,
+ -1, &called);
+ ret = notifier_to_errno(ret);
+
+ if (ret < 0 && called)
+ __srcu_notifier_call_chain(&slub_nb, cmd_free, &arg,
+ called - 1, NULL);
+
+ return ret;
+ }
+
+ static inline
+ void __slub_notify(int cmd, struct kmem_cache *cachep, void *obj)
+ {
+ struct slub_notify_struct arg;
+
+ arg.cachep = cachep;
+ arg.objp = obj;
+ arg.gfp = 0;
+
+ srcu_notifier_call_chain(&slub_nb, cmd, &arg);
+ }
+
+ int slub_register_notifier(struct notifier_block *nb)
+ {
+ return srcu_notifier_chain_register(&slub_nb, nb);
+ }
+
+ void slub_unregister_notifier(struct notifier_block *nb)
+ {
+ srcu_notifier_chain_unregister(&slub_nb, nb);
+ }
```

```

+
+static inline
+int slub_alloc_notify(struct kmem_cache *cachep, void *obj, gfp_t gfp)
+{
+ return __slub_notify_fail(SLUB_ALLOC, SLUB_FREE, cachep, obj, gfp);
+}
+
+static inline
+void slub_free_notify(struct kmem_cache *cachep, void *obj)
+{
+ __slub_notify(SLUB_FREE, cachep, obj);
+}
+
+static inline
+int slub_newpage_notify(struct kmem_cache *cachep, struct page *pg, gfp_t gfp)
+{
+ return __slub_notify_fail(SLUB_NEWPAGE, SLUB_FREEPAGE, cachep, pg, gfp);
+}
+
+static inline
+void slub_freepage_notify(struct kmem_cache *cachep, struct page *pg)
+{
+ __slub_notify(SLUB_FREEPAGE, cachep, pg);
+}
+
+static inline
+int slub_on_notify(struct kmem_cache *cachep)
+{
+ return __slub_notify_fail(SLUB_ON, SLUB_OFF, cachep, NULL, GFP_KERNEL);
+}
+
+static inline
+int slub_off_notify(struct kmem_cache *cachep)
+{
+ return __slub_notify_fail(SLUB_OFF, SLUB_ON, cachep, NULL, GFP_KERNEL);
+}
+
+#else
+int __attribute__((weak)) slub_alloc_notify(struct kmem_cache *cachep,
+void *obj, gfp_t gfp)
+{
+@@ -1040,6 +1150,7 @@ static inline unsigned long kmem_cache_f
+{
+ return 0;
+}
+#endif

/*
 * Slab allocation and freeing

```

```
@@ -2785,8 +2937,9 @@ void __init kmem_cache_init(void)
#else
    kmem_size = sizeof(struct kmem_cache);
#endif
-
-
+#ifdef CONFIG_SLUB_NOTIFY
+ srcu_init_notifier_head(&slub_nb);
+#endif
    printk(KERN_INFO "SLUB: Genslabs=%d, HWalign=%d, Order=%d-%d, MinObjects=%d,"
        " CPUs=%d, Nodes=%d\n",
        caches, cache_line_size(),
```
