Subject: [PATCH] Add kernel/notifier.c
Posted by Alexey Dobriyan on Thu, 19 Jul 2007 16:46:11 GMT
View Forum Message <> Reply to Message

There is separate notifier header, but no separate notifier .c file.

Extract notifier code out of kernel/sys.c which will remain for
misc syscalls I hope. Merge kernel/die_notifier.c into kernel/notifier.c.

Signed-off-by: Alexey Dobriyan <adobriyan@sw.ru>

```
 include/linux/notifier.h |    2
 kernel/Makefile          |    4
 kernel/die_notifier.c    |   38 ---
 kernel/notifier.c        |  569 ++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/sys.c             |  531 ----------------------------------------
 5 files changed, 573 insertions(+), 571 deletions(-)

--- a/include/linux/notifier.h
+++ b/include/linux/notifier.h
@@ -212,5 +212,7 @@ extern int __srcu_notifier_call_chain(struct srcu_notifier_head *nh,
 #define CPU_DEAD_FROZEN  (CPU_DEAD | CPU_TASKS_FROZEN)
 #define CPU_DYING_FROZEN (CPU_DYING | CPU_TASKS_FROZEN)

+extern struct blocking_notifier_head reboot_notifier_list;
+
 #endif /* __KERNEL__ */
 #endif /* _LINUX_NOTIFIER_H */
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -8,8 +8,8 @@ obj-y     = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
       signal.o sys.o kmod.o workqueue.o pid.o \
       rcupdate.o extable.o params.o posix-timers.o \
       kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
-      hrtimer.o rwsem.o latency.o nsproxy.o srcu.o die_notifier.o \
-      utsname.o
+      hrtimer.o rwsem.o latency.o nsproxy.o srcu.o \
+      utsname.o notifier.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
deleted file mode 100644
--- a/kernel/die_notifier.c
+++ /dev/null
@@ -1,38 +0,0 @@
-
-#include <linux/module.h>
```

```
-#include <linux/notifier.h>
-#include <linux/vmalloc.h>
-#include <linux/kdebug.h>
-
-
-static ATOMIC_NOTIFIER_HEAD(die_chain);
-
-int notify_die(enum die_val val, const char *str,
-	       struct pt_regs *regs, long err, int trap, int sig)
-{
- struct die_args args = {
-  .regs  = regs,
-  .str  = str,
-  .err  = err,
-  .trapnr  = trap,
-  .signr  = sig,
-
- };
-
- return atomic_notifier_call_chain(&die_chain, val, &args);
-}
-
-int register_die_notifier(struct notifier_block *nb)
-{
- vmalloc_sync_all();
- return atomic_notifier_chain_register(&die_chain, nb);
-}
-EXPORT_SYMBOL_GPL(register_die_notifier);
-
-int unregister_die_notifier(struct notifier_block *nb)
-{
- return atomic_notifier_chain_unregister(&die_chain, nb);
-}
-EXPORT_SYMBOL_GPL(unregister_die_notifier);
-
-
--- /dev/null
+++ b/kernel/notifier.c
@@ -0,0 +1,569 @@
+#include <linux/kdebug.h>
+#include <linux/kprobes.h>
+#include <linux/module.h>
+#include <linux/notifier.h>
+#include <linux/rcupdate.h>
+#include <linux/vmalloc.h>
+
+/*
+ * Notifier list for kernel code which wants to be called
```

```
+ * at shutdown. This is used to stop any idling DMA operations
+ * and the like.
+ */
+
+BLOCKING_NOTIFIER_HEAD(reboot_notifier_list);
+
+/*
+ * Notifier chain core routines.  The exported routines below
+ * are layered on top of these, with appropriate locking added.
+ */
+
+static int notifier_chain_register(struct notifier_block **nl,
+  struct notifier_block *n)
+{
+ while ((*nl) != NULL) {
+  if (n->priority > (*nl)->priority)
+   break;
+  nl = &((*nl)->next);
+ }
+ n->next = *nl;
+ rcu_assign_pointer(*nl, n);
+ return 0;
+}
+
+static int notifier_chain_unregister(struct notifier_block **nl,
+  struct notifier_block *n)
+{
+ while ((*nl) != NULL) {
+  if ((*nl) == n) {
+   rcu_assign_pointer(*nl, n->next);
+   return 0;
+  }
+  nl = &((*nl)->next);
+ }
+ return -ENOENT;
+}
+
+/**
+ * notifier_call_chain - Informs the registered notifiers about an event.
+ * @nl:  Pointer to head of the blocking notifier chain
+ * @val:  Value passed unmodified to notifier function
+ * @v:  Pointer passed unmodified to notifier function
+ * @nr_to_call: Number of notifier functions to be called. Don't care
+ *        value of this parameter is -1.
+ * @nr_calls: Records the number of notifications sent. Don't care
+ *      value of this field is NULL.
+ *  @returns: notifier_call_chain returns the value returned by the
+ *   last notifier function called.
```

```
+ */
+
+static int __kprobes notifier_call_chain(struct notifier_block **nl,
+     unsigned long val, void *v,
+     int nr_to_call, int *nr_calls)
+{
+ int ret = NOTIFY_DONE;
+ struct notifier_block *nb, *next_nb;
+
+ nb = rcu_dereference(*nl);
+
+ while (nb && nr_to_call) {
+  next_nb = rcu_dereference(nb->next);
+  ret = nb->notifier_call(nb, val, v);
+
+  if (nr_calls)
+   (*nr_calls)++;
+
+  if ((ret & NOTIFY_STOP_MASK) == NOTIFY_STOP_MASK)
+   break;
+  nb = next_nb;
+  nr_to_call--;
+ }
+ return ret;
+}
+
+/*
+ * Atomic notifier chain routines.  Registration and unregistration
+ * use a spinlock, and call_chain is synchronized by RCU (no locks).
+ */
+
+/**
+ * atomic_notifier_chain_register - Add notifier to an atomic notifier chain
+ * @nh: Pointer to head of the atomic notifier chain
+ * @n: New entry in notifier chain
+ *
+ * Adds a notifier to an atomic notifier chain.
+ *
+ * Currently always returns zero.
+ */
+
+int atomic_notifier_chain_register(struct atomic_notifier_head *nh,
+  struct notifier_block *n)
+{
+ unsigned long flags;
+ int ret;
+
+ spin_lock_irqsave(&nh->lock, flags);
```

```
+ ret = notifier_chain_register(&nh->head, n);
+ spin_unlock_irqrestore(&nh->lock, flags);
+ return ret;
+}
+
+EXPORT_SYMBOL_GPL(atomic_notifier_chain_register);
+
+/**
+ * atomic_notifier_chain_unregister - Remove notifier from an atomic notifier chain
+ * @nh: Pointer to head of the atomic notifier chain
+ * @n: Entry to remove from notifier chain
+ *
+ * Removes a notifier from an atomic notifier chain.
+ *
+ * Returns zero on success or %-ENOENT on failure.
+ */
+int atomic_notifier_chain_unregister(struct atomic_notifier_head *nh,
+  struct notifier_block *n)
+{
+ unsigned long flags;
+ int ret;
+
+ spin_lock_irqsave(&nh->lock, flags);
+ ret = notifier_chain_unregister(&nh->head, n);
+ spin_unlock_irqrestore(&nh->lock, flags);
+ synchronize_rcu();
+ return ret;
+}
+
+EXPORT_SYMBOL_GPL(atomic_notifier_chain_unregister);
+
+/**
+ * __atomic_notifier_call_chain - Call functions in an atomic notifier chain
+ * @nh: Pointer to head of the atomic notifier chain
+ * @val: Value passed unmodified to notifier function
+ * @v: Pointer passed unmodified to notifier function
+ * @nr_to_call: See the comment for notifier_call_chain.
+ * @nr_calls: See the comment for notifier_call_chain.
+ *
+ * Calls each function in a notifier chain in turn.  The functions
+ * run in an atomic context, so they must not block.
+ * This routine uses RCU to synchronize with changes to the chain.
+ *
+ * If the return value of the notifier can be and'ed
+ * with %NOTIFY_STOP_MASK then atomic_notifier_call_chain()
+ * will return immediately, with the return value of
+ * the notifier function which halted execution.
+ * Otherwise the return value is the return value
```

```
+ * of the last notifier function called.
+ */
+
+int __kprobes __atomic_notifier_call_chain(struct atomic_notifier_head *nh,
+     unsigned long val, void *v,
+     int nr_to_call, int *nr_calls)
+{
+ int ret;
+
+ rcu_read_lock();
+ ret = notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
+ rcu_read_unlock();
+ return ret;
+}
+
+EXPORT_SYMBOL_GPL(__atomic_notifier_call_chain);
+
+int __kprobes atomic_notifier_call_chain(struct atomic_notifier_head *nh,
+  unsigned long val, void *v)
+{
+ return __atomic_notifier_call_chain(nh, val, v, -1, NULL);
+}
+
+EXPORT_SYMBOL_GPL(atomic_notifier_call_chain);
+/*
+ * Blocking notifier chain routines.  All access to the chain is
+ * synchronized by an rwsem.
+ */
+
+/**
+ * blocking_notifier_chain_register - Add notifier to a blocking notifier chain
+ * @nh: Pointer to head of the blocking notifier chain
+ * @n: New entry in notifier chain
+ *
+ * Adds a notifier to a blocking notifier chain.
+ * Must be called in process context.
+ *
+ * Currently always returns zero.
+ */
+
+int blocking_notifier_chain_register(struct blocking_notifier_head *nh,
+  struct notifier_block *n)
+{
+ int ret;
+
+ /*
+  * This code gets used during boot-up, when task switching is
+  * not yet working and interrupts must remain disabled.  At
```

```
+  * such times we must not call down_write().
+  */
+ if (unlikely(system_state == SYSTEM_BOOTING))
+  return notifier_chain_register(&nh->head, n);
+
+ down_write(&nh->rwsem);
+ ret = notifier_chain_register(&nh->head, n);
+ up_write(&nh->rwsem);
+ return ret;
+}
+
+EXPORT_SYMBOL_GPL(blocking_notifier_chain_register);
+
+/**
+ * blocking_notifier_chain_unregister - Remove notifier from a blocking notifier chain
+ * @nh: Pointer to head of the blocking notifier chain
+ * @n: Entry to remove from notifier chain
+ *
+ * Removes a notifier from a blocking notifier chain.
+ * Must be called from process context.
+ *
+ * Returns zero on success or %-ENOENT on failure.
+ */
+int blocking_notifier_chain_unregister(struct blocking_notifier_head *nh,
+  struct notifier_block *n)
+{
+ int ret;
+
+ /*
+  * This code gets used during boot-up, when task switching is
+  * not yet working and interrupts must remain disabled.  At
+  * such times we must not call down_write().
+  */
+ if (unlikely(system_state == SYSTEM_BOOTING))
+  return notifier_chain_unregister(&nh->head, n);
+
+ down_write(&nh->rwsem);
+ ret = notifier_chain_unregister(&nh->head, n);
+ up_write(&nh->rwsem);
+ return ret;
+}
+
+EXPORT_SYMBOL_GPL(blocking_notifier_chain_unregister);
+
+/**
+ * __blocking_notifier_call_chain - Call functions in a blocking notifier chain
+ * @nh: Pointer to head of the blocking notifier chain
+ * @val: Value passed unmodified to notifier function
```

```
+ * @v: Pointer passed unmodified to notifier function
+ * @nr_to_call: See comment for notifier_call_chain.
+ * @nr_calls: See comment for notifier_call_chain.
+ *
+ * Calls each function in a notifier chain in turn.  The functions
+ * run in a process context, so they are allowed to block.
+ *
+ * If the return value of the notifier can be and'ed
+ * with %NOTIFY_STOP_MASK then blocking_notifier_call_chain()
+ * will return immediately, with the return value of
+ * the notifier function which halted execution.
+ * Otherwise the return value is the return value
+ * of the last notifier function called.
+ */
+
+int __blocking_notifier_call_chain(struct blocking_notifier_head *nh,
+       unsigned long val, void *v,
+       int nr_to_call, int *nr_calls)
+{
+ int ret = NOTIFY_DONE;
+
+ /*
+  * We check the head outside the lock, but if this access is
+  * racy then it does not matter what the result of the test
+  * is, we re-check the list after having taken the lock anyway:
+  */
+ if (rcu_dereference(nh->head)) {
+  down_read(&nh->rwsem);
+  ret = notifier_call_chain(&nh->head, val, v, nr_to_call,
+      nr_calls);
+  up_read(&nh->rwsem);
+ }
+ return ret;
+}
+EXPORT_SYMBOL_GPL(__blocking_notifier_call_chain);
+
+int blocking_notifier_call_chain(struct blocking_notifier_head *nh,
+  unsigned long val, void *v)
+{
+ return __blocking_notifier_call_chain(nh, val, v, -1, NULL);
+}
+EXPORT_SYMBOL_GPL(blocking_notifier_call_chain);
+
+/*
+ * Raw notifier chain routines.  There is no protection;
+ * the caller must provide it.  Use at your own risk!
+ */
+
```

```
+/**
+ * raw_notifier_chain_register - Add notifier to a raw notifier chain
+ * @nh: Pointer to head of the raw notifier chain
+ * @n: New entry in notifier chain
+ *
+ * Adds a notifier to a raw notifier chain.
+ * All locking must be provided by the caller.
+ *
+ * Currently always returns zero.
+ */
+
+int raw_notifier_chain_register(struct raw_notifier_head *nh,
+  struct notifier_block *n)
+{
+ return notifier_chain_register(&nh->head, n);
+}
+
+EXPORT_SYMBOL_GPL(raw_notifier_chain_register);
+
+/**
+ * raw_notifier_chain_unregister - Remove notifier from a raw notifier chain
+ * @nh: Pointer to head of the raw notifier chain
+ * @n: Entry to remove from notifier chain
+ *
+ * Removes a notifier from a raw notifier chain.
+ * All locking must be provided by the caller.
+ *
+ * Returns zero on success or %-ENOENT on failure.
+ */
+int raw_notifier_chain_unregister(struct raw_notifier_head *nh,
+  struct notifier_block *n)
+{
+ return notifier_chain_unregister(&nh->head, n);
+}
+
+EXPORT_SYMBOL_GPL(raw_notifier_chain_unregister);
+
+/**
+ * __raw_notifier_call_chain - Call functions in a raw notifier chain
+ * @nh: Pointer to head of the raw notifier chain
+ * @val: Value passed unmodified to notifier function
+ * @v: Pointer passed unmodified to notifier function
+ * @nr_to_call: See comment for notifier_call_chain.
+ * @nr_calls: See comment for notifier_call_chain
+ *
+ * Calls each function in a notifier chain in turn.  The functions
+ * run in an undefined context.
+ * All locking must be provided by the caller.
```

```
+ *
+ * If the return value of the notifier can be and'ed
+ * with %NOTIFY_STOP_MASK then raw_notifier_call_chain()
+ * will return immediately, with the return value of
+ * the notifier function which halted execution.
+ * Otherwise the return value is the return value
+ * of the last notifier function called.
+ */
+
+int __raw_notifier_call_chain(struct raw_notifier_head *nh,
+        unsigned long val, void *v,
+        int nr_to_call, int *nr_calls)
+{
+ return notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
+}
+
+EXPORT_SYMBOL_GPL(__raw_notifier_call_chain);
+
+int raw_notifier_call_chain(struct raw_notifier_head *nh,
+  unsigned long val, void *v)
+{
+ return __raw_notifier_call_chain(nh, val, v, -1, NULL);
+}
+
+EXPORT_SYMBOL_GPL(raw_notifier_call_chain);
+
+/*
+ * SRCU notifier chain routines.    Registration and unregistration
+ * use a mutex, and call_chain is synchronized by SRCU (no locks).
+ */
+
+/**
+ * srcu_notifier_chain_register - Add notifier to an SRCU notifier chain
+ * @nh: Pointer to head of the SRCU notifier chain
+ * @n: New entry in notifier chain
+ *
+ * Adds a notifier to an SRCU notifier chain.
+ * Must be called in process context.
+ *
+ * Currently always returns zero.
+ */
+
+int srcu_notifier_chain_register(struct srcu_notifier_head *nh,
+  struct notifier_block *n)
+{
+ int ret;
+
+ /*
```

```
+  * This code gets used during boot-up, when task switching is
+  * not yet working and interrupts must remain disabled.  At
+  * such times we must not call mutex_lock().
+  */
+ if (unlikely(system_state == SYSTEM_BOOTING))
+  return notifier_chain_register(&nh->head, n);
+
+ mutex_lock(&nh->mutex);
+ ret = notifier_chain_register(&nh->head, n);
+ mutex_unlock(&nh->mutex);
+ return ret;
+}
+
+EXPORT_SYMBOL_GPL(srcu_notifier_chain_register);
+
+/**
+ * srcu_notifier_chain_unregister - Remove notifier from an SRCU notifier chain
+ * @nh: Pointer to head of the SRCU notifier chain
+ * @n: Entry to remove from notifier chain
+ *
+ * Removes a notifier from an SRCU notifier chain.
+ * Must be called from process context.
+ *
+ * Returns zero on success or %-ENOENT on failure.
+ */
+int srcu_notifier_chain_unregister(struct srcu_notifier_head *nh,
+  struct notifier_block *n)
+{
+ int ret;
+
+ /*
+  * This code gets used during boot-up, when task switching is
+  * not yet working and interrupts must remain disabled.  At
+  * such times we must not call mutex_lock().
+  */
+ if (unlikely(system_state == SYSTEM_BOOTING))
+  return notifier_chain_unregister(&nh->head, n);
+
+ mutex_lock(&nh->mutex);
+ ret = notifier_chain_unregister(&nh->head, n);
+ mutex_unlock(&nh->mutex);
+ synchronize_srcu(&nh->srcu);
+ return ret;
+}
+
+EXPORT_SYMBOL_GPL(srcu_notifier_chain_unregister);
+
+/**
```

```
+ * __srcu_notifier_call_chain - Call functions in an SRCU notifier chain
+ * @nh: Pointer to head of the SRCU notifier chain
+ * @val: Value passed unmodified to notifier function
+ * @v: Pointer passed unmodified to notifier function
+ * @nr_to_call: See comment for notifier_call_chain.
+ * @nr_calls: See comment for notifier_call_chain
+ *
+ * Calls each function in a notifier chain in turn.  The functions
+ * run in a process context, so they are allowed to block.
+ *
+ * If the return value of the notifier can be and'ed
+ * with %NOTIFY_STOP_MASK then srcu_notifier_call_chain()
+ * will return immediately, with the return value of
+ * the notifier function which halted execution.
+ * Otherwise the return value is the return value
+ * of the last notifier function called.
+ */
+
+int __srcu_notifier_call_chain(struct srcu_notifier_head *nh,
+          unsigned long val, void *v,
+          int nr_to_call, int *nr_calls)
+{
+ int ret;
+ int idx;
+
+ idx = srcu_read_lock(&nh->srcu);
+ ret = notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
+ srcu_read_unlock(&nh->srcu, idx);
+ return ret;
+}
+EXPORT_SYMBOL_GPL(__srcu_notifier_call_chain);
+
+int srcu_notifier_call_chain(struct srcu_notifier_head *nh,
+  unsigned long val, void *v)
+{
+ return __srcu_notifier_call_chain(nh, val, v, -1, NULL);
+}
+EXPORT_SYMBOL_GPL(srcu_notifier_call_chain);
+
+/**
+ * srcu_init_notifier_head - Initialize an SRCU notifier head
+ * @nh: Pointer to head of the srcu notifier chain
+ *
+ * Unlike other sorts of notifier heads, SRCU notifier heads require
+ * dynamic initialization.  Be sure to call this routine before
+ * calling any of the other SRCU notifier routines for this head.
+ *
+ * If an SRCU notifier head is deallocated, it must first be cleaned
```

```
+ * up by calling srcu_cleanup_notifier_head().  Otherwise the head's
+ * per-cpu data (used by the SRCU mechanism) will leak.
+ */
+
+void srcu_init_notifier_head(struct srcu_notifier_head *nh)
+{
+ mutex_init(&nh->mutex);
+ if (init_srcu_struct(&nh->srcu) < 0)
+  BUG();
+ nh->head = NULL;
+}
+
+EXPORT_SYMBOL_GPL(srcu_init_notifier_head);
+
+/**
+ * register_reboot_notifier - Register function to be called at reboot time
+ * @nb: Info about notifier function to be called
+ *
+ * Registers a function with the list of functions
+ * to be called at reboot time.
+ *
+ * Currently always returns zero, as blocking_notifier_chain_register()
+ * always returns zero.
+ */
+
+int register_reboot_notifier(struct notifier_block * nb)
+{
+ return blocking_notifier_chain_register(&reboot_notifier_list, nb);
+}
+
+EXPORT_SYMBOL(register_reboot_notifier);
+
+/**
+ * unregister_reboot_notifier - Unregister previously registered reboot notifier
+ * @nb: Hook to be unregistered
+ *
+ * Unregisters a previously registered reboot
+ * notifier function.
+ *
+ * Returns zero on success, or %-ENOENT on failure.
+ */
+
+int unregister_reboot_notifier(struct notifier_block * nb)
+{
+ return  blocking_notifier_chain_unregister(&reboot_notifier_list , nb);
+}
+
+EXPORT_SYMBOL(unregister_reboot_notifier);
```

```
+
+
+
+static ATOMIC_NOTIFIER_HEAD(die_chain);
+
+int notify_die(enum die_val val, const char *str,
+	       struct pt_regs *regs, long err, int trap, int sig)
+{
+ struct die_args args = {
+  .regs  = regs,
+  .str  = str,
+  .err  = err,
+  .trapnr  = trap,
+  .signr  = sig,
+
+ };
+
+ return atomic_notifier_call_chain(&die_chain, val, &args);
+}
+
+int register_die_notifier(struct notifier_block *nb)
+{
+ vmalloc_sync_all();
+ return atomic_notifier_chain_register(&die_chain, nb);
+}
+EXPORT_SYMBOL_GPL(register_die_notifier);
+
+int unregister_die_notifier(struct notifier_block *nb)
+{
+ return atomic_notifier_chain_unregister(&die_chain, nb);
+}
+EXPORT_SYMBOL_GPL(unregister_die_notifier);
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -99,537 +99,6 @@ int C_A_D = 1;
 struct pid *cad_pid;
 EXPORT_SYMBOL(cad_pid);

-/*
- * Notifier list for kernel code which wants to be called
- * at shutdown. This is used to stop any idling DMA operations
- * and the like.
- */
-
-static BLOCKING_NOTIFIER_HEAD(reboot_notifier_list);
-
-/*
- * Notifier chain core routines.  The exported routines below
```

```
- * are layered on top of these, with appropriate locking added.
- */
-
-static int notifier_chain_register(struct notifier_block **nl,
-  struct notifier_block *n)
-{
- while ((*nl) != NULL) {
-  if (n->priority > (*nl)->priority)
-   break;
-  nl = &((*nl)->next);
- }
- n->next = *nl;
- rcu_assign_pointer(*nl, n);
- return 0;
-}
-
-static int notifier_chain_unregister(struct notifier_block **nl,
-  struct notifier_block *n)
-{
- while ((*nl) != NULL) {
-  if ((*nl) == n) {
-   rcu_assign_pointer(*nl, n->next);
-   return 0;
-  }
-  nl = &((*nl)->next);
- }
- return -ENOENT;
-}
-
-/**
- * notifier_call_chain - Informs the registered notifiers about an event.
- * @nl:  Pointer to head of the blocking notifier chain
- * @val:  Value passed unmodified to notifier function
- * @v:  Pointer passed unmodified to notifier function
- * @nr_to_call: Number of notifier functions to be called. Don't care
- *        value of this parameter is -1.
- * @nr_calls: Records the number of notifications sent. Don't care
- *      value of this field is NULL.
- *  @returns: notifier_call_chain returns the value returned by the
- *   last notifier function called.
- */
-
-static int __kprobes notifier_call_chain(struct notifier_block **nl,
-     unsigned long val, void *v,
-     int nr_to_call, int *nr_calls)
-{
- int ret = NOTIFY_DONE;
- struct notifier_block *nb, *next_nb;
```

```
-
- nb = rcu_dereference(*nl);
-
- while (nb && nr_to_call) {
-  next_nb = rcu_dereference(nb->next);
-  ret = nb->notifier_call(nb, val, v);
-
-  if (nr_calls)
-   (*nr_calls)++;
-
-  if ((ret & NOTIFY_STOP_MASK) == NOTIFY_STOP_MASK)
-   break;
-  nb = next_nb;
-  nr_to_call--;
- }
- return ret;
-}
-
-/*
- * Atomic notifier chain routines.  Registration and unregistration
- * use a spinlock, and call_chain is synchronized by RCU (no locks).
- */
-
-/**
- * atomic_notifier_chain_register - Add notifier to an atomic notifier chain
- * @nh: Pointer to head of the atomic notifier chain
- * @n: New entry in notifier chain
- *
- * Adds a notifier to an atomic notifier chain.
- *
- * Currently always returns zero.
- */
-
-int atomic_notifier_chain_register(struct atomic_notifier_head *nh,
-  struct notifier_block *n)
-{
- unsigned long flags;
- int ret;
-
- spin_lock_irqsave(&nh->lock, flags);
- ret = notifier_chain_register(&nh->head, n);
- spin_unlock_irqrestore(&nh->lock, flags);
- return ret;
-}
-
-EXPORT_SYMBOL_GPL(atomic_notifier_chain_register);
-
-/**
```

```
- * atomic_notifier_chain_unregister - Remove notifier from an atomic notifier chain
- * @nh: Pointer to head of the atomic notifier chain
- * @n: Entry to remove from notifier chain
- *
- * Removes a notifier from an atomic notifier chain.
- *
- * Returns zero on success or %-ENOENT on failure.
- */
-int atomic_notifier_chain_unregister(struct atomic_notifier_head *nh,
-  struct notifier_block *n)
-{
- unsigned long flags;
- int ret;
-
- spin_lock_irqsave(&nh->lock, flags);
- ret = notifier_chain_unregister(&nh->head, n);
- spin_unlock_irqrestore(&nh->lock, flags);
- synchronize_rcu();
- return ret;
-}
-
-EXPORT_SYMBOL_GPL(atomic_notifier_chain_unregister);
-
-/**
- * __atomic_notifier_call_chain - Call functions in an atomic notifier chain
- * @nh: Pointer to head of the atomic notifier chain
- * @val: Value passed unmodified to notifier function
- * @v: Pointer passed unmodified to notifier function
- * @nr_to_call: See the comment for notifier_call_chain.
- * @nr_calls: See the comment for notifier_call_chain.
- *
- * Calls each function in a notifier chain in turn.  The functions
- * run in an atomic context, so they must not block.
- * This routine uses RCU to synchronize with changes to the chain.
- *
- * If the return value of the notifier can be and'ed
- * with %NOTIFY_STOP_MASK then atomic_notifier_call_chain()
- * will return immediately, with the return value of
- * the notifier function which halted execution.
- * Otherwise the return value is the return value
- * of the last notifier function called.
- */
-
-int __kprobes __atomic_notifier_call_chain(struct atomic_notifier_head *nh,
-     unsigned long val, void *v,
-     int nr_to_call, int *nr_calls)
-{
- int ret;
```

```
-
- rcu_read_lock();
- ret = notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
- rcu_read_unlock();
- return ret;
-}
-
-EXPORT_SYMBOL_GPL(__atomic_notifier_call_chain);
-
-int __kprobes atomic_notifier_call_chain(struct atomic_notifier_head *nh,
-  unsigned long val, void *v)
-{
- return __atomic_notifier_call_chain(nh, val, v, -1, NULL);
-}
-
-EXPORT_SYMBOL_GPL(atomic_notifier_call_chain);
-/*
- * Blocking notifier chain routines.  All access to the chain is
- * synchronized by an rwsem.
- */
-
-/**
- * blocking_notifier_chain_register - Add notifier to a blocking notifier chain
- * @nh: Pointer to head of the blocking notifier chain
- * @n: New entry in notifier chain
- *
- * Adds a notifier to a blocking notifier chain.
- * Must be called in process context.
- *
- * Currently always returns zero.
- */
-
-int blocking_notifier_chain_register(struct blocking_notifier_head *nh,
-  struct notifier_block *n)
-{
- int ret;
-
- /*
-  * This code gets used during boot-up, when task switching is
-  * not yet working and interrupts must remain disabled.  At
-  * such times we must not call down_write().
-  */
- if (unlikely(system_state == SYSTEM_BOOTING))
-  return notifier_chain_register(&nh->head, n);
-
- down_write(&nh->rwsem);
- ret = notifier_chain_register(&nh->head, n);
- up_write(&nh->rwsem);
```

```
- return ret;
-}
-
-EXPORT_SYMBOL_GPL(blocking_notifier_chain_register);
-
-/**
- * blocking_notifier_chain_unregister - Remove notifier from a blocking notifier chain
- * @nh: Pointer to head of the blocking notifier chain
- * @n: Entry to remove from notifier chain
- *
- * Removes a notifier from a blocking notifier chain.
- * Must be called from process context.
- *
- * Returns zero on success or %-ENOENT on failure.
- */
-int blocking_notifier_chain_unregister(struct blocking_notifier_head *nh,
-  struct notifier_block *n)
-{
- int ret;
-
- /*
-  * This code gets used during boot-up, when task switching is
-  * not yet working and interrupts must remain disabled.  At
-  * such times we must not call down_write().
-  */
- if (unlikely(system_state == SYSTEM_BOOTING))
-  return notifier_chain_unregister(&nh->head, n);
-
- down_write(&nh->rwsem);
- ret = notifier_chain_unregister(&nh->head, n);
- up_write(&nh->rwsem);
- return ret;
-}
-
-EXPORT_SYMBOL_GPL(blocking_notifier_chain_unregister);
-
-/**
- * __blocking_notifier_call_chain - Call functions in a blocking notifier chain
- * @nh: Pointer to head of the blocking notifier chain
- * @val: Value passed unmodified to notifier function
- * @v: Pointer passed unmodified to notifier function
- * @nr_to_call: See comment for notifier_call_chain.
- * @nr_calls: See comment for notifier_call_chain.
- *
- * Calls each function in a notifier chain in turn.  The functions
- * run in a process context, so they are allowed to block.
- *
- * If the return value of the notifier can be and'ed
```

```
- * with %NOTIFY_STOP_MASK then blocking_notifier_call_chain()
- * will return immediately, with the return value of
- * the notifier function which halted execution.
- * Otherwise the return value is the return value
- * of the last notifier function called.
- */
-
-int __blocking_notifier_call_chain(struct blocking_notifier_head *nh,
-       unsigned long val, void *v,
-       int nr_to_call, int *nr_calls)
-{
- int ret = NOTIFY_DONE;
-
- /*
-  * We check the head outside the lock, but if this access is
-  * racy then it does not matter what the result of the test
-  * is, we re-check the list after having taken the lock anyway:
-  */
- if (rcu_dereference(nh->head)) {
-  down_read(&nh->rwsem);
-  ret = notifier_call_chain(&nh->head, val, v, nr_to_call,
-      nr_calls);
-  up_read(&nh->rwsem);
- }
- return ret;
-}
-EXPORT_SYMBOL_GPL(__blocking_notifier_call_chain);
-
-int blocking_notifier_call_chain(struct blocking_notifier_head *nh,
-  unsigned long val, void *v)
-{
- return __blocking_notifier_call_chain(nh, val, v, -1, NULL);
-}
-EXPORT_SYMBOL_GPL(blocking_notifier_call_chain);
-
-/*
- * Raw notifier chain routines.  There is no protection;
- * the caller must provide it.  Use at your own risk!
- */
-
-/**
- * raw_notifier_chain_register - Add notifier to a raw notifier chain
- * @nh: Pointer to head of the raw notifier chain
- * @n: New entry in notifier chain
- *
- * Adds a notifier to a raw notifier chain.
- * All locking must be provided by the caller.
- *
```

```
- * Currently always returns zero.
- */
-
-int raw_notifier_chain_register(struct raw_notifier_head *nh,
-  struct notifier_block *n)
-{
- return notifier_chain_register(&nh->head, n);
-}
-
-EXPORT_SYMBOL_GPL(raw_notifier_chain_register);
-
-/**
- * raw_notifier_chain_unregister - Remove notifier from a raw notifier chain
- * @nh: Pointer to head of the raw notifier chain
- * @n: Entry to remove from notifier chain
- *
- * Removes a notifier from a raw notifier chain.
- * All locking must be provided by the caller.
- *
- * Returns zero on success or %-ENOENT on failure.
- */
-int raw_notifier_chain_unregister(struct raw_notifier_head *nh,
-  struct notifier_block *n)
-{
- return notifier_chain_unregister(&nh->head, n);
-}
-
-EXPORT_SYMBOL_GPL(raw_notifier_chain_unregister);
-
-/**
- * __raw_notifier_call_chain - Call functions in a raw notifier chain
- * @nh: Pointer to head of the raw notifier chain
- * @val: Value passed unmodified to notifier function
- * @v: Pointer passed unmodified to notifier function
- * @nr_to_call: See comment for notifier_call_chain.
- * @nr_calls: See comment for notifier_call_chain
- *
- * Calls each function in a notifier chain in turn.  The functions
- * run in an undefined context.
- * All locking must be provided by the caller.
- *
- * If the return value of the notifier can be and'ed
- * with %NOTIFY_STOP_MASK then raw_notifier_call_chain()
- * will return immediately, with the return value of
- * the notifier function which halted execution.
- * Otherwise the return value is the return value
- * of the last notifier function called.
- */
```

```
-
-int __raw_notifier_call_chain(struct raw_notifier_head *nh,
-        unsigned long val, void *v,
-        int nr_to_call, int *nr_calls)
-{
- return notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
-}
-
-EXPORT_SYMBOL_GPL(__raw_notifier_call_chain);
-
-int raw_notifier_call_chain(struct raw_notifier_head *nh,
-  unsigned long val, void *v)
-{
- return __raw_notifier_call_chain(nh, val, v, -1, NULL);
-}
-
-EXPORT_SYMBOL_GPL(raw_notifier_call_chain);
-
-/*
- * SRCU notifier chain routines.    Registration and unregistration
- * use a mutex, and call_chain is synchronized by SRCU (no locks).
- */
-
-/**
- * srcu_notifier_chain_register - Add notifier to an SRCU notifier chain
- * @nh: Pointer to head of the SRCU notifier chain
- * @n: New entry in notifier chain
- *
- * Adds a notifier to an SRCU notifier chain.
- * Must be called in process context.
- *
- * Currently always returns zero.
- */
-
-int srcu_notifier_chain_register(struct srcu_notifier_head *nh,
-  struct notifier_block *n)
-{
- int ret;
-
- /*
-  * This code gets used during boot-up, when task switching is
-  * not yet working and interrupts must remain disabled.  At
-  * such times we must not call mutex_lock().
-  */
- if (unlikely(system_state == SYSTEM_BOOTING))
-  return notifier_chain_register(&nh->head, n);
-
- mutex_lock(&nh->mutex);
```

```
- ret = notifier_chain_register(&nh->head, n);
- mutex_unlock(&nh->mutex);
- return ret;
-}
-
-EXPORT_SYMBOL_GPL(srcu_notifier_chain_register);
-
-/**
- * srcu_notifier_chain_unregister - Remove notifier from an SRCU notifier chain
- * @nh: Pointer to head of the SRCU notifier chain
- * @n: Entry to remove from notifier chain
- *
- * Removes a notifier from an SRCU notifier chain.
- * Must be called from process context.
- *
- * Returns zero on success or %-ENOENT on failure.
- */
-int srcu_notifier_chain_unregister(struct srcu_notifier_head *nh,
-  struct notifier_block *n)
-{
- int ret;
-
- /*
-  * This code gets used during boot-up, when task switching is
-  * not yet working and interrupts must remain disabled.  At
-  * such times we must not call mutex_lock().
-  */
- if (unlikely(system_state == SYSTEM_BOOTING))
-  return notifier_chain_unregister(&nh->head, n);
-
- mutex_lock(&nh->mutex);
- ret = notifier_chain_unregister(&nh->head, n);
- mutex_unlock(&nh->mutex);
- synchronize_srcu(&nh->srcu);
- return ret;
-}
-
-EXPORT_SYMBOL_GPL(srcu_notifier_chain_unregister);
-
-/**
- * __srcu_notifier_call_chain - Call functions in an SRCU notifier chain
- * @nh: Pointer to head of the SRCU notifier chain
- * @val: Value passed unmodified to notifier function
- * @v: Pointer passed unmodified to notifier function
- * @nr_to_call: See comment for notifier_call_chain.
- * @nr_calls: See comment for notifier_call_chain
- *
- * Calls each function in a notifier chain in turn.  The functions
```

```
- * run in a process context, so they are allowed to block.
- *
- * If the return value of the notifier can be and'ed
- * with %NOTIFY_STOP_MASK then srcu_notifier_call_chain()
- * will return immediately, with the return value of
- * the notifier function which halted execution.
- * Otherwise the return value is the return value
- * of the last notifier function called.
- */
-
-int __srcu_notifier_call_chain(struct srcu_notifier_head *nh,
-          unsigned long val, void *v,
-          int nr_to_call, int *nr_calls)
-{
- int ret;
- int idx;
-
- idx = srcu_read_lock(&nh->srcu);
- ret = notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
- srcu_read_unlock(&nh->srcu, idx);
- return ret;
-}
-EXPORT_SYMBOL_GPL(__srcu_notifier_call_chain);
-
-int srcu_notifier_call_chain(struct srcu_notifier_head *nh,
-  unsigned long val, void *v)
-{
- return __srcu_notifier_call_chain(nh, val, v, -1, NULL);
-}
-EXPORT_SYMBOL_GPL(srcu_notifier_call_chain);
-
-/**
- * srcu_init_notifier_head - Initialize an SRCU notifier head
- * @nh: Pointer to head of the srcu notifier chain
- *
- * Unlike other sorts of notifier heads, SRCU notifier heads require
- * dynamic initialization.  Be sure to call this routine before
- * calling any of the other SRCU notifier routines for this head.
- *
- * If an SRCU notifier head is deallocated, it must first be cleaned
- * up by calling srcu_cleanup_notifier_head().  Otherwise the head's
- * per-cpu data (used by the SRCU mechanism) will leak.
- */
-
-void srcu_init_notifier_head(struct srcu_notifier_head *nh)
-{
- mutex_init(&nh->mutex);
- if (init_srcu_struct(&nh->srcu) < 0)
```

```
-  BUG();
- nh->head = NULL;
-}
-
-EXPORT_SYMBOL_GPL(srcu_init_notifier_head);
-
-/**
- * register_reboot_notifier - Register function to be called at reboot time
- * @nb: Info about notifier function to be called
- *
- * Registers a function with the list of functions
- * to be called at reboot time.
- *
- * Currently always returns zero, as blocking_notifier_chain_register()
- * always returns zero.
- */
-
-int register_reboot_notifier(struct notifier_block * nb)
-{
- return blocking_notifier_chain_register(&reboot_notifier_list, nb);
-}
-
-EXPORT_SYMBOL(register_reboot_notifier);
-
-/**
- * unregister_reboot_notifier - Unregister previously registered reboot notifier
- * @nb: Hook to be unregistered
- *
- * Unregisters a previously registered reboot
- * notifier function.
- *
- * Returns zero on success, or %-ENOENT on failure.
- */
-
-int unregister_reboot_notifier(struct notifier_block * nb)
-{
- return  blocking_notifier_chain_unregister(&reboot_notifier_list , nb);
-}
-
-EXPORT_SYMBOL(unregister_reboot_notifier);
-
 static int set_one_prio(struct task_struct *p, int niceval, int error)
 {
  int no_nice;
```

Subject: Re: [PATCH] Add kernel/notifier.c

On Thu, 19 Jul 2007 20:46:11 +0400
Alexey Dobriyan <adobriyan@sw.ru> wrote:

> There is separate notifier header, but no separate notifier .c file.
>
> Extract notifier code out of kernel/sys.c which will remain for
> misc syscalls I hope. Merge kernel/die_notifier.c into kernel/notifier.c.

If you were running checkpatch (I hope you are) then you'd find that we
copied over a whole pile of cruft. We might as well fix that up while
we're moving the code around.

Also, your patch tried to add some trailing whitespace, but checkpatch
failed to notice that.

(The code motion and the cleanups should really be separate patches, and
indeed they are, but I'll end up joining them before it hits git)

```
diff -puN kernel/notifier.c~add-kernel-notifierc-fix kernel/notifier.c
--- a/kernel/notifier.c~add-kernel-notifierc-fix
+++ a/kernel/notifier.c
@@ -107,7 +107,6 @@ int atomic_notifier_chain_register(struc
  spin_unlock_irqrestore(&nh->lock, flags);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(atomic_notifier_chain_register);

 /**
@@ -131,7 +130,6 @@ int atomic_notifier_chain_unregister(str
  synchronize_rcu();
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(atomic_notifier_chain_unregister);

 /**
@@ -165,7 +163,6 @@ int __kprobes __atomic_notifier_call_cha
  rcu_read_unlock();
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(__atomic_notifier_call_chain);

 int __kprobes atomic_notifier_call_chain(struct atomic_notifier_head *nh,
@@ -173,8 +170,8 @@ int __kprobes atomic_notifier_call_chain
```

```
 {
  return __atomic_notifier_call_chain(nh, val, v, -1, NULL);
 }
-
 EXPORT_SYMBOL_GPL(atomic_notifier_call_chain);
+
 /*
  * Blocking notifier chain routines.  All access to the chain is
  * synchronized by an rwsem.
@@ -209,7 +206,6 @@ int blocking_notifier_chain_register(str
  up_write(&nh->rwsem);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(blocking_notifier_chain_register);

 /**
@@ -240,7 +236,6 @@ int blocking_notifier_chain_unregister(s
  up_write(&nh->rwsem);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(blocking_notifier_chain_unregister);

 /**
@@ -311,7 +306,6 @@ int raw_notifier_chain_register(struct r
 {
  return notifier_chain_register(&nh->head, n);
 }
-
 EXPORT_SYMBOL_GPL(raw_notifier_chain_register);

 /**
@@ -329,7 +323,6 @@ int raw_notifier_chain_unregister(struct
 {
  return notifier_chain_unregister(&nh->head, n);
 }
-
 EXPORT_SYMBOL_GPL(raw_notifier_chain_unregister);

 /**
@@ -358,7 +351,6 @@ int __raw_notifier_call_chain(struct raw
 {
  return notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
 }
-
 EXPORT_SYMBOL_GPL(__raw_notifier_call_chain);
```

```
 int raw_notifier_call_chain(struct raw_notifier_head *nh,
@@ -366,7 +358,6 @@ int raw_notifier_call_chain(struct raw_n
 {
  return __raw_notifier_call_chain(nh, val, v, -1, NULL);
 }
-
 EXPORT_SYMBOL_GPL(raw_notifier_call_chain);

 /*
@@ -403,7 +394,6 @@ int srcu_notifier_chain_register(struct
  mutex_unlock(&nh->mutex);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(srcu_notifier_chain_register);

 /**
@@ -435,7 +425,6 @@ int srcu_notifier_chain_unregister(struc
  synchronize_srcu(&nh->srcu);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(srcu_notifier_chain_unregister);

 /**
@@ -498,7 +487,6 @@ void srcu_init_notifier_head(struct srcu
   BUG();
  nh->head = NULL;
 }
-
 EXPORT_SYMBOL_GPL(srcu_init_notifier_head);

 /**
@@ -512,11 +500,10 @@ EXPORT_SYMBOL_GPL(srcu_init_notifier_hea
  * always returns zero.
  */

-int register_reboot_notifier(struct notifier_block * nb)
+int register_reboot_notifier(struct notifier_block *nb)
 {
  return blocking_notifier_chain_register(&reboot_notifier_list, nb);
 }
-
 EXPORT_SYMBOL(register_reboot_notifier);

 /**
@@ -529,15 +516,13 @@ EXPORT_SYMBOL(register_reboot_notifier);
  * Returns zero on success, or %-ENOENT on failure.
```

```
 */

-int unregister_reboot_notifier(struct notifier_block * nb)
+int unregister_reboot_notifier(struct notifier_block *nb)
 {
  return  blocking_notifier_chain_unregister(&reboot_notifier_list , nb);
 }
-
 EXPORT_SYMBOL(unregister_reboot_notifier);


-
 static ATOMIC_NOTIFIER_HEAD(die_chain);

 int notify_die(enum die_val val, const char *str,
```

Subject: Re: [PATCH] Add kernel/notifier.c
Posted by Alexey Dobriyan on Fri, 20 Jul 2007 07:22:42 GMT
View Forum Message <> Reply to Message

On Thu, Jul 19, 2007 at 02:48:59PM -0700, Andrew Morton wrote:
> On Thu, 19 Jul 2007 20:46:11 +0400
> Alexey Dobriyan <adobriyan@sw.ru> wrote:
>
> > There is separate notifier header, but no separate notifier .c file.
> >
> > Extract notifier code out of kernel/sys.c which will remain for
> > misc syscalls I hope. Merge kernel/die_notifier.c into kernel/notifier.c.
>
> If you were running checkpatch (I hope you are)

:)

> then you'd find that we
> copied over a whole pile of cruft. We might as well fix that up while
> we're moving the code around.
>
> Also, your patch tried to add some trailing whitespace, but checkpatch
> failed to notice that.

It notices here:

```
 $ ./scripts/checkpatch.pl ../notifier.patch | grep ERROR
 ERROR: trailing whitespace
 ERROR: trailing whitespace
 ERROR: trailing whitespace
 ERROR: trailing whitespace
```

```
 ERROR: trailing whitespace
 ERROR: "foo * bar" should be "foo *bar"
 ERROR: trailing whitespace
 ERROR: "foo * bar" should be "foo *bar"
```

> (The code motion and the cleanups should really be separate patches, and
> indeed they are, but I'll end up joining them before it hits git)

Then take this cleanup which is based on yours;
In particular, it fixes [Space][Tab][Space] places and removes lines
between kernel-doc comment and function itself.

```
--- a/kernel/notifier.c
+++ b/kernel/notifier.c
@@ -10,7 +10,6 @@
  * at shutdown. This is used to stop any idling DMA operations
  * and the like.
  */
-
 BLOCKING_NOTIFIER_HEAD(reboot_notifier_list);

 /*
@@ -50,13 +49,12 @@ static int notifier_chain_unregister(struct notifier_block **nl,
  * @val:  Value passed unmodified to notifier function
  * @v:  Pointer passed unmodified to notifier function
  * @nr_to_call: Number of notifier functions to be called. Don't care
- *        value of this parameter is -1.
+ *   value of this parameter is -1.
  * @nr_calls: Records the number of notifications sent. Don't care
- *      value of this field is NULL.
- *  @returns: notifier_call_chain returns the value returned by the
+ *   value of this field is NULL.
+ * @returns: notifier_call_chain returns the value returned by the
  *   last notifier function called.
  */
-
 static int __kprobes notifier_call_chain(struct notifier_block **nl,
      unsigned long val, void *v,
      int nr_to_call, int *nr_calls)
@@ -95,7 +93,6 @@ static int __kprobes notifier_call_chain(struct notifier_block **nl,
  *
  * Currently always returns zero.
  */
-
 int atomic_notifier_chain_register(struct atomic_notifier_head *nh,
   struct notifier_block *n)
 {
```

```
@@ -107,7 +104,6 @@ int atomic_notifier_chain_register(struct atomic_notifier_head *nh,
  spin_unlock_irqrestore(&nh->lock, flags);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(atomic_notifier_chain_register);

 /**
@@ -131,7 +127,6 @@ int atomic_notifier_chain_unregister(struct atomic_notifier_head *nh,
  synchronize_rcu();
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(atomic_notifier_chain_unregister);

 /**
@@ -153,7 +148,6 @@ EXPORT_SYMBOL_GPL(atomic_notifier_chain_unregister);
  * Otherwise the return value is the return value
  * of the last notifier function called.
  */
-
 int __kprobes __atomic_notifier_call_chain(struct atomic_notifier_head *nh,
      unsigned long val, void *v,
      int nr_to_call, int *nr_calls)
@@ -165,7 +159,6 @@ int __kprobes __atomic_notifier_call_chain(struct atomic_notifier_head
*nh,
  rcu_read_unlock();
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(__atomic_notifier_call_chain);

 int __kprobes atomic_notifier_call_chain(struct atomic_notifier_head *nh,
@@ -173,8 +166,8 @@ int __kprobes atomic_notifier_call_chain(struct atomic_notifier_head *nh,
 {
  return __atomic_notifier_call_chain(nh, val, v, -1, NULL);
 }
-
 EXPORT_SYMBOL_GPL(atomic_notifier_call_chain);
+
 /*
  * Blocking notifier chain routines.  All access to the chain is
  * synchronized by an rwsem.
@@ -190,7 +183,6 @@ EXPORT_SYMBOL_GPL(atomic_notifier_call_chain);
  *
  * Currently always returns zero.
  */
-
```

```
 int blocking_notifier_chain_register(struct blocking_notifier_head *nh,
   struct notifier_block *n)
 {
@@ -209,7 +201,6 @@ int blocking_notifier_chain_register(struct blocking_notifier_head *nh,
  up_write(&nh->rwsem);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(blocking_notifier_chain_register);

 /**
@@ -240,7 +231,6 @@ int blocking_notifier_chain_unregister(struct blocking_notifier_head *nh,
  up_write(&nh->rwsem);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(blocking_notifier_chain_unregister);

 /**
@@ -261,7 +251,6 @@ EXPORT_SYMBOL_GPL(blocking_notifier_chain_unregister);
  * Otherwise the return value is the return value
  * of the last notifier function called.
  */
-
 int __blocking_notifier_call_chain(struct blocking_notifier_head *nh,
        unsigned long val, void *v,
        int nr_to_call, int *nr_calls)
@@ -305,13 +294,11 @@ EXPORT_SYMBOL_GPL(blocking_notifier_call_chain);
  *
  * Currently always returns zero.
  */
-
 int raw_notifier_chain_register(struct raw_notifier_head *nh,
   struct notifier_block *n)
 {
  return notifier_chain_register(&nh->head, n);
 }
-
 EXPORT_SYMBOL_GPL(raw_notifier_chain_register);

 /**
@@ -329,7 +316,6 @@ int raw_notifier_chain_unregister(struct raw_notifier_head *nh,
 {
  return notifier_chain_unregister(&nh->head, n);
 }
-
 EXPORT_SYMBOL_GPL(raw_notifier_chain_unregister);
```

```
 /**
@@ -351,14 +337,12 @@ EXPORT_SYMBOL_GPL(raw_notifier_chain_unregister);
  * Otherwise the return value is the return value
  * of the last notifier function called.
  */
-
 int __raw_notifier_call_chain(struct raw_notifier_head *nh,
         unsigned long val, void *v,
         int nr_to_call, int *nr_calls)
 {
  return notifier_call_chain(&nh->head, val, v, nr_to_call, nr_calls);
 }
-
 EXPORT_SYMBOL_GPL(__raw_notifier_call_chain);

 int raw_notifier_call_chain(struct raw_notifier_head *nh,
@@ -366,7 +350,6 @@ int raw_notifier_call_chain(struct raw_notifier_head *nh,
 {
  return __raw_notifier_call_chain(nh, val, v, -1, NULL);
 }
-
 EXPORT_SYMBOL_GPL(raw_notifier_call_chain);

 /*
@@ -384,7 +367,6 @@ EXPORT_SYMBOL_GPL(raw_notifier_call_chain);
  *
  * Currently always returns zero.
  */
-
 int srcu_notifier_chain_register(struct srcu_notifier_head *nh,
   struct notifier_block *n)
 {
@@ -403,7 +385,6 @@ int srcu_notifier_chain_register(struct srcu_notifier_head *nh,
  mutex_unlock(&nh->mutex);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(srcu_notifier_chain_register);

 /**
@@ -435,7 +416,6 @@ int srcu_notifier_chain_unregister(struct srcu_notifier_head *nh,
  synchronize_srcu(&nh->srcu);
  return ret;
 }
-
 EXPORT_SYMBOL_GPL(srcu_notifier_chain_unregister);

 /**
```

```
@@ -456,7 +436,6 @@ EXPORT_SYMBOL_GPL(srcu_notifier_chain_unregister);
  * Otherwise the return value is the return value
  * of the last notifier function called.
  */
-
 int __srcu_notifier_call_chain(struct srcu_notifier_head *nh,
         unsigned long val, void *v,
         int nr_to_call, int *nr_calls)
@@ -490,7 +469,6 @@ EXPORT_SYMBOL_GPL(srcu_notifier_call_chain);
  * up by calling srcu_cleanup_notifier_head().  Otherwise the head's
  * per-cpu data (used by the SRCU mechanism) will leak.
  */
-
 void srcu_init_notifier_head(struct srcu_notifier_head *nh)
 {
  mutex_init(&nh->mutex);
@@ -498,7 +476,6 @@ void srcu_init_notifier_head(struct srcu_notifier_head *nh)
   BUG();
  nh->head = NULL;
 }
-
 EXPORT_SYMBOL_GPL(srcu_init_notifier_head);

 /**
@@ -511,12 +488,10 @@ EXPORT_SYMBOL_GPL(srcu_init_notifier_head);
  * Currently always returns zero, as blocking_notifier_chain_register()
  * always returns zero.
  */
-
-int register_reboot_notifier(struct notifier_block * nb)
+int register_reboot_notifier(struct notifier_block *nb)
 {
  return blocking_notifier_chain_register(&reboot_notifier_list, nb);
 }
-
 EXPORT_SYMBOL(register_reboot_notifier);

 /**
@@ -528,30 +503,25 @@ EXPORT_SYMBOL(register_reboot_notifier);
  *
  * Returns zero on success, or %-ENOENT on failure.
  */
-
-int unregister_reboot_notifier(struct notifier_block * nb)
+int unregister_reboot_notifier(struct notifier_block *nb)
 {
  return  blocking_notifier_chain_unregister(&reboot_notifier_list , nb);
 }
```

```
-
 EXPORT_SYMBOL(unregister_reboot_notifier);

-
-
 static ATOMIC_NOTIFIER_HEAD(die_chain);

 int notify_die(enum die_val val, const char *str,
        struct pt_regs *regs, long err, int trap, int sig)
 {
  struct die_args args = {
-  .regs  = regs,
-  .str  = str,
-  .err  = err,
-  .trapnr  = trap,
-  .signr  = sig,
+  .regs = regs,
+  .str = str,
+  .err = err,
+  .trapnr = trap,
+  .signr = sig,

  };
-
  return atomic_notifier_call_chain(&die_chain, val, &args);
 }
```