Subject: [PATCH 1/2] iptables 32bit compat layer
Posted by Mishin Dmitry on Mon, 20 Feb 2006 08:10:38 GMT
View Forum Message <> Reply to Message

Hello,

This patch set extends current iptables compatibility layer in order to get
32bit iptables to work on 64bit kernel. Current layer is insufficient
due to alignment checks both in kernel and user space tools.

This patch introduces base compatibility interface for other ip_tables modules

--
Thanks,
Dmitry.

```
--- ./include/linux/netfilter/x_tables.h.iptcompat 2006-02-15 16:16:02.000000000 +0300
+++ ./include/linux/netfilter/x_tables.h 2006-02-15 18:53:09.000000000 +0300
@@ -80,12 +80,19 @@ struct xt_counters_info

 #ifdef __KERNEL__

+#include <linux/config.h>
 #include <linux/netdevice.h>

 #define ASSERT_READ_LOCK(x)
 #define ASSERT_WRITE_LOCK(x)
 #include <linux/netfilter_ipv4/listhelp.h>

+#ifdef CONFIG_COMPAT
+#define COMPAT_TO_USER  1
+#define COMPAT_FROM_USER -1
+#define COMPAT_CALC_SIZE 0
+#endif
+
 struct xt_match
 {
  struct list_head list;
@@ -118,6 +125,10 @@ struct xt_match
  /* Called when entry of this type deleted. */
  void (*destroy)(void *matchinfo, unsigned int matchinfosize);

+#ifdef CONFIG_COMPAT
+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *match, void **dstptr, int *size, int convert);
+#endif
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;
```

```
 };
@@ -154,6 +165,10 @@ struct xt_target
  /* Called when entry of this type deleted. */
  void (*destroy)(void *targinfo, unsigned int targinfosize);

+#ifdef CONFIG_COMPAT
+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *target, void **dstptr, int *size, int convert);
+#endif
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;
 };
@@ -233,6 +248,34 @@ extern void xt_proto_fini(int af);
 extern struct xt_table_info *xt_alloc_table_info(unsigned int size);
 extern void xt_free_table_info(struct xt_table_info *info);

+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
+
+/* FIXME: this works only on 32 bit tasks
+ * need to change whole approach in order to calculate align as function of
+ * current task alignment */
+
+struct compat_xt_counters
+{
+ u_int32_t cnt[4];
+};
+
+struct compat_xt_counters_info
+{
+ char name[XT_TABLE_MAXNAMELEN];
+ compat_uint_t num_counters;
+ struct compat_xt_counters counters[0];
+};
+
+#define COMPAT_XT_ALIGN(s) (((s) + (__alignof__(struct compat_xt_counters)-1)) \
+  & ~(__alignof__(struct compat_xt_counters)-1))
+
+extern int ipt_match_align_compat(void *match, void **dstptr,
+  int *size, int off, int convert);
+extern int ipt_target_align_compat(void *target, void **dstptr,
+  int *size, int off, int convert);
+
+#endif /* CONFIG_COMPAT */
 #endif /* __KERNEL__ */

 #endif /* _X_TABLES_H */
--- ./include/linux/netfilter_ipv4/ip_tables.h.iptcompat 2006-02-15 16:06:41.000000000 +0300
```

```
+++ ./include/linux/netfilter_ipv4/ip_tables.h 2006-02-15 16:37:12.000000000 +0300
@@ -16,6 +16,7 @@
 #define _IPTABLES_H

 #ifdef __KERNEL__
+#include <linux/config.h>
 #include <linux/if.h>
 #include <linux/types.h>
 #include <linux/in.h>
@@ -364,5 +365,62 @@ extern unsigned int ipt_do_table(struct
      void *userdata);

 #define IPT_ALIGN(s) XT_ALIGN(s)
+
+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
+
+struct compat_ipt_getinfo
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ compat_uint_t valid_hooks;
+ compat_uint_t hook_entry[NF_IP_NUMHOOKS];
+ compat_uint_t underflow[NF_IP_NUMHOOKS];
+ compat_uint_t num_entries;
+ compat_uint_t size;
+};
+
+struct compat_ipt_entry
+{
+ struct ipt_ip ip;
+ compat_uint_t nfcache;
+ u_int16_t target_offset;
+ u_int16_t next_offset;
+ compat_uint_t comefrom;
+ struct compat_xt_counters counters;
+ unsigned char elems[0];
+};
+
+struct compat_ipt_entry_match
+{
+ union {
+  struct {
+   u_int16_t match_size;
+   char name[IPT_FUNCTION_MAXNAMELEN];
+  } user;
+  u_int16_t match_size;
+ } u;
+ unsigned char data[0];
```

```
+};
+
+struct compat_ipt_entry_target
+{
+ union {
+  struct {
+   u_int16_t target_size;
+   char name[IPT_FUNCTION_MAXNAMELEN];
+  } user;
+  u_int16_t target_size;
+ } u;
+ unsigned char data[0];
+};
+
+#define COMPAT_IPT_ALIGN(s)  COMPAT_XT_ALIGN(s)
+
+extern int ipt_match_align_compat(void *match, void **dstptr,
+  int *size, int off, int convert);
+extern int ipt_target_align_compat(void *target, void **dstptr,
+  int *size, int off, int convert);
+
+#endif /* CONFIG_COMPAT */
 #endif /*__KERNEL__*/
 #endif /* _IPTABLES_H */
--- ./include/net/compat.h.iptcompat 2006-01-03 06:21:10.000000000 +0300
+++ ./include/net/compat.h 2006-02-15 18:45:49.000000000 +0300
@@ -23,6 +23,14 @@ struct compat_cmsghdr {
  compat_int_t cmsg_type;
 };

+#if defined(CONFIG_X86_64)
+#define is_current_32bits() (current_thread_info()->flags & _TIF_IA32)
+#elif defined(CONFIG_IA64)
+#define is_current_32bits() (IS_IA32_PROCESS(ia64_task_regs(current)))
+#else
+#define is_current_32bits() 0
+#endif
+
 #else /* defined(CONFIG_COMPAT) */
 #define compat_msghdr msghdr  /* to avoid compiler warnings */
 #endif /* defined(CONFIG_COMPAT) */
--- ./net/compat.c.iptcompat 2006-01-03 06:21:10.000000000 +0300
+++ ./net/compat.c 2006-02-15 16:38:45.000000000 +0300
@@ -308,107 +308,6 @@ void scm_detach_fds_compat(struct msghdr
 }

 /*
- * For now, we assume that the compatibility and native version
```

```
- * of struct ipt_entry are the same - sfr.  FIXME
- */
-struct compat_ipt_replace {
- char   name[IPT_TABLE_MAXNAMELEN];
- u32   valid_hooks;
- u32   num_entries;
- u32   size;
- u32   hook_entry[NF_IP_NUMHOOKS];
- u32   underflow[NF_IP_NUMHOOKS];
- u32   num_counters;
- compat_uptr_t  counters; /* struct ipt_counters * */
- struct ipt_entry entries[0];
-};
-
-static int do_netfilter_replace(int fd, int level, int optname,
-    char __user *optval, int optlen)
-{
- struct compat_ipt_replace __user *urepl;
- struct ipt_replace __user *repl_nat;
- char name[IPT_TABLE_MAXNAMELEN];
- u32 origsize, tmp32, num_counters;
- unsigned int repl_nat_size;
- int ret;
- int i;
- compat_uptr_t ucntrs;
-
- urepl = (struct compat_ipt_replace __user *)optval;
- if (get_user(origsize, &urepl->size))
-  return -EFAULT;
-
- /* Hack: Causes ipchains to give correct error msg --RR */
- if (optlen != sizeof(*urepl) + origsize)
-  return -ENOPROTOOPT;
-
- /* XXX Assumes that size of ipt_entry is the same both in
-  *     native and compat environments.
-  */
- repl_nat_size = sizeof(*repl_nat) + origsize;
- repl_nat = compat_alloc_user_space(repl_nat_size);
-
- ret = -EFAULT;
- if (put_user(origsize, &repl_nat->size))
-  goto out;
-
- if (!access_ok(VERIFY_READ, urepl, optlen) ||
-     !access_ok(VERIFY_WRITE, repl_nat, optlen))
-  goto out;
-
```

```
- if (__copy_from_user(name, urepl->name, sizeof(urepl->name)) ||
-     __copy_to_user(repl_nat->name, name, sizeof(repl_nat->name)))
-  goto out;
-
- if (__get_user(tmp32, &urepl->valid_hooks) ||
-     __put_user(tmp32, &repl_nat->valid_hooks))
-  goto out;
-
- if (__get_user(tmp32, &urepl->num_entries) ||
-     __put_user(tmp32, &repl_nat->num_entries))
-  goto out;
-
- if (__get_user(num_counters, &urepl->num_counters) ||
-     __put_user(num_counters, &repl_nat->num_counters))
-  goto out;
-
- if (__get_user(ucntrs, &urepl->counters) ||
-     __put_user(compat_ptr(ucntrs), &repl_nat->counters))
-  goto out;
-
- if (__copy_in_user(&repl_nat->entries[0],
-      &urepl->entries[0],
-      origsize))
-  goto out;
-
- for (i = 0; i < NF_IP_NUMHOOKS; i++) {
-  if (__get_user(tmp32, &urepl->hook_entry[i]) ||
-      __put_user(tmp32, &repl_nat->hook_entry[i]) ||
-      __get_user(tmp32, &urepl->underflow[i]) ||
-      __put_user(tmp32, &repl_nat->underflow[i]))
-   goto out;
- }
-
- /*
-  * Since struct ipt_counters just contains two u_int64_t members
-  * we can just do the access_ok check here and pass the (converted)
-  * pointer into the standard syscall.  We hope that the pointer is
-  * not misaligned ...
-  */
- if (!access_ok(VERIFY_WRITE, compat_ptr(ucntrs),
-         num_counters * sizeof(struct ipt_counters)))
-  goto out;
-
-
- ret = sys_setsockopt(fd, level, optname,
-        (char __user *)repl_nat, repl_nat_size);
-
-out:
```

```
- return ret;
-}
-
-/*
  * A struct sock_filter is architecture independent.
  */
 struct compat_sock_fprog {
@@ -460,10 +359,6 @@ static int do_set_sock_timeout(int fd, i
 asmlinkage long compat_sys_setsockopt(int fd, int level, int optname,
     char __user *optval, int optlen)
 {
- /* SO_SET_REPLACE seems to be the same in all levels */
- if (optname == IPT_SO_SET_REPLACE)
-  return do_netfilter_replace(fd, level, optname,
-         optval, optlen);
  if (level == SOL_SOCKET && optname == SO_ATTACH_FILTER)
   return do_set_attach_filter(fd, level, optname,
          optval, optlen);
--- ./net/ipv4/netfilter/ip_tables.c.iptcompat 2006-02-15 16:06:42.000000000 +0300
+++ ./net/ipv4/netfilter/ip_tables.c 2006-02-17 19:38:05.000000000 +0300
@@ -24,6 +24,7 @@
 #include <linux/module.h>
 #include <linux/icmp.h>
 #include <net/ip.h>
+#include <net/compat.h>
 #include <asm/uaccess.h>
 #include <asm/semaphore.h>
 #include <linux/proc_fs.h>
@@ -480,7 +481,7 @@ standard_check(const struct ipt_entry_ta
  if (t->u.target_size
      != IPT_ALIGN(sizeof(struct ipt_standard_target))) {
   duprintf("standard_check: target size %u != %u\n",
-    t->u.target_size,
+    t->u.target_size, (unsigned int)
     IPT_ALIGN(sizeof(struct ipt_standard_target)));
   return 0;
  }
@@ -790,17 +791,11 @@ get_counters(const struct xt_table_info
  }
 }

-static int
-copy_entries_to_user(unsigned int total_size,
-       struct ipt_table *table,
-       void __user *userptr)
+static inline struct xt_counters * alloc_counters(struct ipt_table *table)
 {
- unsigned int off, num, countersize;
```

```
- struct ipt_entry *e;
+ unsigned int countersize;
  struct xt_counters *counters;
  struct xt_table_info *private = table->private;
- int ret = 0;
- void *loc_cpu_entry;

  /* We need atomic snapshot of counters: rest doesn't change
     (other than comefrom, which userspace doesn't care
@@ -809,13 +804,32 @@ copy_entries_to_user(unsigned int total_
  counters = vmalloc_node(countersize, numa_node_id());

  if (counters == NULL)
-  return -ENOMEM;
+  return ERR_PTR(-ENOMEM);

  /* First, sum counters... */
  write_lock_bh(&table->lock);
  get_counters(private, counters);
  write_unlock_bh(&table->lock);

+ return counters;
+}
+
+static int
+copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table,
+       void __user *userptr)
+{
+ unsigned int off, num;
+ struct ipt_entry *e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ int ret = 0;
+ void *loc_cpu_entry;
+
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
  /* choose the copy that is on our node/cpu, ...
   * This choice is lazy (because current thread is
   * allowed to migrate to another cpu)
@@ -875,25 +889,391 @@ copy_entries_to_user(unsigned int total_
  return ret;
 }

+#ifdef CONFIG_COMPAT
```

```
+static DECLARE_MUTEX(compat_ipt_mutex);
+
+struct compat_delta {
+ struct compat_delta *next;
+ u_int16_t offset;
+ short delta;
+};
+
+static struct compat_delta *compat_offsets = NULL;
+
+static int compat_add_offset(u_int16_t offset, short delta)
+{
+ struct compat_delta *tmp;
+
+ tmp = kmalloc(sizeof(struct compat_delta), GFP_KERNEL);
+ if (!tmp)
+  return -ENOMEM;
+ tmp->offset = offset;
+ tmp->delta = delta;
+ if (compat_offsets) {
+  tmp->next = compat_offsets->next;
+  compat_offsets->next = tmp;
+ } else {
+  compat_offsets = tmp;
+  tmp->next = NULL;
+ }
+ return 0;
+}
+
+static void compat_flush_offsets(void)
+{
+ struct compat_delta *tmp, *next;
+
+ if (compat_offsets) {
+  for(tmp = compat_offsets; tmp; tmp = next) {
+   next = tmp->next;
+   kfree(tmp);
+  }
+  compat_offsets = NULL;
+ }
+}
+
+static short compat_calc_jump(u_int16_t offset)
+{
+ struct compat_delta *tmp;
+ short delta;
+
+ for(tmp = compat_offsets, delta = 0; tmp; tmp = tmp->next)
```

```
+  if (tmp->offset < offset)
+   delta += tmp->delta;
+ return delta;
+}
+
+struct compat_ipt_standard_target
+{
+ struct compat_ipt_entry_target target;
+ compat_int_t verdict;
+};
+
+#define IPT_ST_OFFSET (sizeof(struct ipt_standard_target) - \
+    sizeof(struct compat_ipt_standard_target))
+
+struct compat_ipt_standard
+{
+ struct compat_ipt_entry entry;
+ struct compat_ipt_standard_target target;
+};
+
+static int compat_ipt_standard_fn(void *target,
+  void **dstptr, int *size, int convert)
+{
+ struct compat_ipt_standard_target compat_st, *pcompat_st;
+ struct ipt_standard_target st, *pst;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pst = (struct ipt_standard_target *)target;
+   memcpy(&compat_st.target, &pst->target,
+     sizeof(struct ipt_entry_target));
+   compat_st.verdict = pst->verdict;
+   if (compat_st.verdict > 0)
+    compat_st.verdict -=
+     compat_calc_jump(compat_st.verdict);
+   compat_st.target.u.user.target_size =
+   sizeof(struct compat_ipt_standard_target);
+   if (__copy_to_user(*dstptr, &compat_st,
+    sizeof(struct compat_ipt_standard_target)))
+    ret = -EFAULT;
+   *size -= IPT_ST_OFFSET;
+   *dstptr += sizeof(struct compat_ipt_standard_target);
+   break;
+  case COMPAT_FROM_USER:
+   pcompat_st =
+    (struct compat_ipt_standard_target *)target;
```

```
+   memcpy(&st.target, &pcompat_st->target,
+     sizeof(struct ipt_entry_target));
+   st.verdict = pcompat_st->verdict;
+   if (st.verdict > 0)
+    st.verdict += compat_calc_jump(st.verdict);
+   st.target.u.user.target_size =
+   sizeof(struct ipt_standard_target);
+   memcpy(*dstptr, &st,
+     sizeof(struct ipt_standard_target));
+   *size += IPT_ST_OFFSET;
+   *dstptr += sizeof(struct ipt_standard_target);
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += IPT_ST_OFFSET;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+
+int ipt_target_align_compat(void *target, void **dstptr,
+  int *size, int off, int convert)
+{
+ struct compat_ipt_entry_target *pcompat;
+ struct ipt_entry_target *pt;
+ u_int16_t tsize;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pt = (struct ipt_entry_target *)target;
+   tsize = pt->u.user.target_size;
+   if (__copy_to_user(*dstptr, pt, tsize)) {
+    ret = -EFAULT;
+    break;
+   }
+   tsize -= off;
+   if (put_user(tsize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += tsize;
+   break;
+  case COMPAT_FROM_USER:
+   pcompat = (struct compat_ipt_entry_target *)target;
+   pt = (struct ipt_entry_target *)*dstptr;
```

```
+   tsize = pcompat->u.user.target_size;
+   memcpy(pt, pcompat, tsize);
+   tsize += off;
+   pt->u.user.target_size = tsize;
+   *size += off;
+   *dstptr += tsize;
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+
+int ipt_match_align_compat(void *match, void **dstptr,
+  int *size, int off, int convert)
+{
+ struct compat_ipt_entry_match *pcompat_m;
+ struct ipt_entry_match *pm;
+ u_int16_t msize;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pm = (struct ipt_entry_match *)match;
+   msize = pm->u.user.match_size;
+   if (__copy_to_user(*dstptr, pm, msize)) {
+    ret = -EFAULT;
+    break;
+   }
+   msize -= off;
+   if (put_user(msize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += msize;
+   break;
+  case COMPAT_FROM_USER:
+   pcompat_m = (struct compat_ipt_entry_match *)match;
+   pm = (struct ipt_entry_match *)*dstptr;
+   msize = pcompat_m->u.user.match_size;
+   memcpy(pm, pcompat_m, msize);
+   msize += off;
+   pm->u.user.match_size = msize;
+   *size += off;
```

```
+   *dstptr += msize;
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+
+static int icmp_compat(void *match,
+  void **dstptr, int *size, int convert)
+{
+ int off;
+
+ off = IPT_ALIGN(sizeof(struct ipt_icmp)) -
+  COMPAT_IPT_ALIGN(sizeof(struct ipt_icmp));
+ return ipt_match_align_compat(match, dstptr, size, off, convert);
+}
+
+static inline int
+compat_calc_match(struct ipt_entry_match *m, int * size)
+{
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+ return 0;
+}
+
+static int compat_calc_entry(struct ipt_entry *e, struct xt_table_info *info,
+  void *base, struct xt_table_info *newinfo)
+{
+ struct ipt_entry_target *t;
+ u_int16_t entry_offset;
+ int off, i, ret;
+
+ off = 0;
+ entry_offset = (void *)e - base;
+ IPT_MATCH_ITERATE(e, compat_calc_match, &off);
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ newinfo->size -= off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  return ret;
+
```

```
+ for (i = 0; i< NF_IP_NUMHOOKS; i++) {
+  if (info->hook_entry[i] && (e < (struct ipt_entry *)
+    (base + info->hook_entry[i])))
+   newinfo->hook_entry[i] -= off;
+  if (info->underflow[i] && (e < (struct ipt_entry *)
+    (base + info->underflow[i])))
+   newinfo->underflow[i] -= off;
+ }
+ return 0;
+}
+
+static int compat_table_info(struct xt_table_info *info,
+  struct xt_table_info *newinfo)
+{
+ void *loc_cpu_entry;
+ int i;
+
+ if (!newinfo || !info)
+  return -EINVAL;
+
+ memset(newinfo, 0, sizeof(struct xt_table_info));
+ newinfo->size = info->size;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ loc_cpu_entry = info->entries[raw_smp_processor_id()];
+ return IPT_ENTRY_ITERATE(loc_cpu_entry, info->size,
+   compat_calc_entry, info, loc_cpu_entry, newinfo);
+}
+#endif
+
+static int get_info(void __user *user, int *len)
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ struct ipt_table *t;
+ int ret, size;
+
+#ifdef CONFIG_COMPAT
+ if (is_current_32bits())
+  size = sizeof(struct compat_ipt_getinfo);
+ else
+#endif
+  size = sizeof(struct ipt_getinfo);
+
+ if (*len != size) {
+  duprintf("length %u != %u\n", *len,
+   (unsigned int)sizeof(struct ipt_getinfo));
```

```
+  return -EINVAL;
+ }
+
+ if (copy_from_user(name, user, sizeof(name)) != 0)
+  return -EFAULT;
+
+ name[IPT_TABLE_MAXNAMELEN-1] = '\0';
+#ifdef CONFIG_COMPAT
+ down(&compat_ipt_mutex);
+#endif
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+   "iptable_%s", name);
+ if (t && !IS_ERR(t)) {
+  struct ipt_getinfo info;
+  struct xt_table_info *private = t->private;
+#ifdef CONFIG_COMPAT
+  struct compat_ipt_getinfo compat_info;
+#endif
+  void *pinfo;
+
+#ifdef CONFIG_COMPAT
+  if (is_current_32bits()) {
+   struct xt_table_info tmp;
+   ret = compat_table_info(private, &tmp);
+   compat_flush_offsets();
+   memcpy(compat_info.hook_entry, tmp.hook_entry,
+     sizeof(compat_info.hook_entry));
+   memcpy(compat_info.underflow, tmp.underflow,
+     sizeof(compat_info.underflow));
+   compat_info.valid_hooks = t->valid_hooks;
+   compat_info.num_entries = private->number;
+   compat_info.size = tmp.size;
+   strcpy(compat_info.name, name);
+   pinfo = (void *)&compat_info;
+  } else
+#endif
+  {
+   info.valid_hooks = t->valid_hooks;
+   memcpy(info.hook_entry, private->hook_entry,
+     sizeof(info.hook_entry));
+   memcpy(info.underflow, private->underflow,
+     sizeof(info.underflow));
+   info.num_entries = private->number;
+   info.size = private->size;
+   strcpy(info.name, name);
+   pinfo = (void *)&info;
+  }
+
```

```
+  if (copy_to_user(user, pinfo, *len) != 0)
+   ret = -EFAULT;
+  else
+   ret = 0;
+
+  xt_table_unlock(t);
+  module_put(t->me);
+ } else
+  ret = t ? PTR_ERR(t) : -ENOENT;
+#ifdef CONFIG_COMPAT
+  up(&compat_ipt_mutex);
+#endif
+ return ret;
+}
+
 static int
-get_entries(const struct ipt_get_entries *entries,
-     struct ipt_get_entries __user *uptr)
+get_entries(struct ipt_get_entries __user *uptr, int *len)
 {
  int ret;
+ struct ipt_get_entries get;
  struct ipt_table *t;

- t = xt_find_table_lock(AF_INET, entries->name);
+ if (*len < sizeof(get)) {
+  duprintf("get_entries: %u < %d\n", *len,
+    (unsigned int)sizeof(get));
+  return -EINVAL;
+ }
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;
+ if (*len != sizeof(struct ipt_get_entries) + get.size) {
+  duprintf("get_entries: %u != %u\n", *len,
+    (unsigned int)(sizeof(struct ipt_get_entries) +
+    get.size));
+  return -EINVAL;
+ }
+
+ t = xt_find_table_lock(AF_INET, get.name);
  if (t && !IS_ERR(t)) {
   struct xt_table_info *private = t->private;
   duprintf("t->private->number = %u\n",
     private->number);
-  if (entries->size == private->size)
+  if (get.size == private->size)
    ret = copy_entries_to_user(private->size,
         t, uptr->entrytable);
```

```
   else {
    duprintf("get_entries: I've got %u not %u!\n",
      private->size,
-     entries->size);
+     get.size);
    ret = -EINVAL;
   }
   module_put(t->me);
@@ -905,71 +1285,39 @@ get_entries(const struct ipt_get_entries
 }

 static int
-do_replace(void __user *user, unsigned int len)
+__do_replace(const char *name, unsigned int valid_hooks,
+  struct xt_table_info *newinfo, unsigned int num_counters,
+  void __user *counters_ptr)
 {
  int ret;
- struct ipt_replace tmp;
  struct ipt_table *t;
- struct xt_table_info *newinfo, *oldinfo;
+ struct xt_table_info *oldinfo;
  struct xt_counters *counters;
- void *loc_cpu_entry, *loc_cpu_old_entry;
-
- if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
-  return -EFAULT;
-
- /* Hack: Causes ipchains to give correct error msg --RR */
- if (len != sizeof(tmp) + tmp.size)
-  return -ENOPROTOOPT;
-
- /* overflow check */
- if (tmp.size >= (INT_MAX - sizeof(struct xt_table_info)) / NR_CPUS -
-   SMP_CACHE_BYTES)
-  return -ENOMEM;
- if (tmp.num_counters >= INT_MAX / sizeof(struct xt_counters))
-  return -ENOMEM;
-
- newinfo = xt_alloc_table_info(tmp.size);
- if (!newinfo)
-  return -ENOMEM;
-
- /* choose the copy that is our node/cpu */
- loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
- if (copy_from_user(loc_cpu_entry, user + sizeof(tmp),
-      tmp.size) != 0) {
-  ret = -EFAULT;
```

```
-  goto free_newinfo;
- }
+ void *loc_cpu_old_entry;

- counters = vmalloc(tmp.num_counters * sizeof(struct xt_counters));
+ ret = 0;
+ counters = vmalloc(num_counters * sizeof(struct xt_counters));
  if (!counters) {
   ret = -ENOMEM;
-  goto free_newinfo;
+  goto out;
  }

- ret = translate_table(tmp.name, tmp.valid_hooks,
-         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
-         tmp.hook_entry, tmp.underflow);
- if (ret != 0)
-  goto free_newinfo_counters;
-
- duprintf("ip_tables: Translated table\n");
-
- t = try_then_request_module(xt_find_table_lock(AF_INET, tmp.name),
-         "iptable_%s", tmp.name);
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+         "iptable_%s", name);
  if (!t || IS_ERR(t)) {
   ret = t ? PTR_ERR(t) : -ENOENT;
   goto free_newinfo_counters_untrans;
  }

  /* You lied! */
- if (tmp.valid_hooks != t->valid_hooks) {
+ if (valid_hooks != t->valid_hooks) {
   duprintf("Valid hook crap: %08X vs %08X\n",
-    tmp.valid_hooks, t->valid_hooks);
+    valid_hooks, t->valid_hooks);
   ret = -EINVAL;
   goto put_module;
  }

- oldinfo = xt_replace_table(t, tmp.num_counters, newinfo, &ret);
+ oldinfo = xt_replace_table(t, num_counters, newinfo, &ret);
  if (!oldinfo)
   goto put_module;

@@ -989,8 +1337,8 @@ do_replace(void __user *user, unsigned i
  loc_cpu_old_entry = oldinfo->entries[raw_smp_processor_id()];
  IPT_ENTRY_ITERATE(loc_cpu_old_entry, oldinfo->size, cleanup_entry,NULL);
```

```
  xt_free_table_info(oldinfo);
- if (copy_to_user(tmp.counters, counters,
-    sizeof(struct xt_counters) * tmp.num_counters) != 0)
+ if (copy_to_user(counters_ptr, counters,
+     sizeof(struct xt_counters) * num_counters) != 0)
    ret = -EFAULT;
  vfree(counters);
  xt_table_unlock(t);
@@ -1000,9 +1348,62 @@ do_replace(void __user *user, unsigned i
  module_put(t->me);
  xt_table_unlock(t);
  free_newinfo_counters_untrans:
- IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
- free_newinfo_counters:
  vfree(counters);
+ out:
+ return ret;
+}
+
+static int
+do_replace(void __user *user, unsigned int len)
+{
+ int ret;
+ struct ipt_replace tmp;
+ struct xt_table_info *newinfo;
+ void *loc_cpu_entry;
+
+ if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
+  return -EFAULT;
+
+ /* Hack: Causes ipchains to give correct error msg --RR */
+ if (len != sizeof(tmp) + tmp.size)
+  return -ENOPROTOOPT;
+
+ /* overflow check */
+ if (tmp.size >= (INT_MAX - sizeof(struct xt_table_info)) / NR_CPUS -
+   SMP_CACHE_BYTES)
+  return -ENOMEM;
+ if (tmp.num_counters >= INT_MAX / sizeof(struct xt_counters))
+  return -ENOMEM;
+
+ newinfo = xt_alloc_table_info(tmp.size);
+ if (!newinfo)
+  return -ENOMEM;
+
+ /* choose the copy that is our node/cpu */
+ loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
+ if (copy_from_user(loc_cpu_entry, user + sizeof(tmp),
```

```
+      tmp.size) != 0) {
+  ret = -EFAULT;
+  goto free_newinfo;
+ }
+
+ ret = translate_table(tmp.name, tmp.valid_hooks,
+         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
+         tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
+  goto free_newinfo;
+
+ duprintf("ip_tables: Translated table\n");
+
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         tmp.counters);
+ if (ret)
+  goto free_newinfo_untrans;
+ return 0;
+
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
  free_newinfo:
  xt_free_table_info(newinfo);
  return ret;
@@ -1034,28 +1435,56 @@ static int
 do_add_counters(void __user *user, unsigned int len)
 {
  unsigned int i;
- struct xt_counters_info tmp, *paddc;
+ struct xt_counters_info tmp;
+ struct xt_counters *paddc;
+ unsigned int num_counters;
+ char *name;
+ int size;
+ void *ptmp;
  struct ipt_table *t;
  struct xt_table_info *private;
  int ret = 0;
  void *loc_cpu_entry;
+#ifdef CONFIG_COMPAT
+ struct compat_xt_counters_info compat_tmp;

- if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
+ if (is_current_32bits()) {
+  ptmp = &compat_tmp;
+  size = sizeof(struct compat_xt_counters_info);
+ } else
```

```
+#endif
+ {
+  ptmp = &tmp;
+  size = sizeof(struct xt_counters_info);
+ }
+
+ if (copy_from_user(ptmp, user, size) != 0)
   return -EFAULT;

- if (len != sizeof(tmp) + tmp.num_counters*sizeof(struct xt_counters))
+#ifdef CONFIG_COMPAT
+ if (is_current_32bits()) {
+  num_counters = compat_tmp.num_counters;
+  name = compat_tmp.name;
+ } else
+#endif
+ {
+  num_counters = tmp.num_counters;
+  name = tmp.name;
+ }
+
+ if (len != size + num_counters * sizeof(struct xt_counters))
   return -EINVAL;

- paddc = vmalloc_node(len, numa_node_id());
+ paddc = vmalloc_node(len - size, numa_node_id());
  if (!paddc)
   return -ENOMEM;

- if (copy_from_user(paddc, user, len) != 0) {
+ if (copy_from_user(paddc, user + size, len - size) != 0) {
   ret = -EFAULT;
   goto free;
  }

- t = xt_find_table_lock(AF_INET, tmp.name);
+ t = xt_find_table_lock(AF_INET, name);
  if (!t || IS_ERR(t)) {
   ret = t ? PTR_ERR(t) : -ENOENT;
   goto free;
@@ -1063,7 +1492,7 @@ do_add_counters(void __user *user, unsig

  write_lock_bh(&t->lock);
  private = t->private;
- if (private->number != paddc->num_counters) {
+ if (private->number != num_counters) {
   ret = -EINVAL;
   goto unlock_up_free;
```

```
 }
@@ -1074,7 +1503,7 @@ do_add_counters(void __user *user, unsig
  IPT_ENTRY_ITERATE(loc_cpu_entry,
      private->size,
      add_counter_to_entry,
-     paddc->counters,
+     paddc,
      &i);
  unlock_up_free:
  write_unlock_bh(&t->lock);
@@ -1086,6 +1515,577 @@ do_add_counters(void __user *user, unsig
  return ret;
 }

+#ifdef CONFIG_COMPAT
+struct compat_ipt_replace {
+ char   name[IPT_TABLE_MAXNAMELEN];
+ u32   valid_hooks;
+ u32   num_entries;
+ u32   size;
+ u32   hook_entry[NF_IP_NUMHOOKS];
+ u32   underflow[NF_IP_NUMHOOKS];
+ u32   num_counters;
+ compat_uptr_t  counters; /* struct ipt_counters * */
+ struct compat_ipt_entry entries[0];
+};
+
+static inline int compat_copy_match_to_user(struct ipt_entry_match *m,
+  void __user **dstptr, compat_uint_t *size)
+{
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, dstptr, size, COMPAT_TO_USER);
+ else {
+  if (__copy_to_user(*dstptr, m, m->u.match_size))
+   return -EFAULT;
+  *dstptr += m->u.match_size;
+ }
+ return 0;
+}
+
+static int compat_copy_entry_to_user(struct ipt_entry *e,
+  void __user **dstptr, compat_uint_t *size)
+{
+ struct ipt_entry_target __user *t;
+ struct compat_ipt_entry __user *ce;
+ u_int16_t target_offset, next_offset;
+ compat_uint_t origsize;
+ int ret;
```

```
+
+ ret = -EFAULT;
+ origsize = *size;
+ ce = (struct compat_ipt_entry __user *)*dstptr;
+ if (__copy_to_user(ce, e, sizeof(struct ipt_entry)))
+  goto out;
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_to_user, dstptr, size);
+ target_offset = e->target_offset - (origsize - *size);
+ if (ret)
+  goto out;
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat) {
+  ret = t->u.kernel.target->compat(t,
+    dstptr, size, COMPAT_TO_USER);
+  if (ret)
+   goto out;
+ } else {
+  ret = -EFAULT;
+  if (__copy_to_user(*dstptr, t, t->u.target_size))
+   goto out;
+  *dstptr += t->u.target_size;
+ }
+ ret = -EFAULT;
+ next_offset = e->next_offset - (origsize - *size);
+ if (__put_user(target_offset, &ce->target_offset))
+  goto out;
+ if (__put_user(next_offset, &ce->next_offset))
+  goto out;
+ return 0;
+out:
+ return ret;
+}
+
+static inline int
+compat_check_calc_match(struct ipt_entry_match *m,
+     const char *name,
+     const struct ipt_ip *ip,
+     unsigned int hookmask,
+     int *size, int *i)
+{
+ struct ipt_match *match;
+
+ match = try_then_request_module(xt_find_match(AF_INET, m->u.user.name,
+         m->u.user.revision),
+     "ipt_%s", m->u.user.name);
+ if (IS_ERR(match) || !match) {
```

```
+  duprintf("compat_check_calc_match: `%s' not found\n",
+    m->u.user.name);
+  return match ? PTR_ERR(match) : -ENOENT;
+ }
+ m->u.kernel.match = match;
+
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+
+ (*i)++;
+ return 0;
+}
+
+static inline int
+check_compat_entry_size_and_hooks(struct ipt_entry *e,
+      struct xt_table_info *newinfo,
+      unsigned int *size,
+      unsigned char *base,
+      unsigned char *limit,
+      unsigned int *hook_entries,
+      unsigned int *underflows,
+      unsigned int *i,
+      const char *name)
+{
+ struct ipt_entry_target *t;
+ struct ipt_target *target;
+ u_int16_t entry_offset;
+ int ret, off, h, j;
+
+ duprintf("check_compat_entry_size_and_hooks %p\n", e);
+ if ((unsigned long)e % __alignof__(struct compat_ipt_entry) != 0
+     || (unsigned char *)e + sizeof(struct compat_ipt_entry) >= limit) {
+  duprintf("Bad offset %p, limit = %p\n", e, limit);
+  return -EINVAL;
+ }
+
+ if (e->next_offset < sizeof(struct compat_ipt_entry) +
+   sizeof(struct compat_ipt_entry_target)) {
+  duprintf("checking: element %p size %u\n",
+    e, e->next_offset);
+  return -EINVAL;
+ }
+
+ if (!ip_checkentry(&e->ip)) {
+  duprintf("ip_tables: ip check failed %p %s.\n", e, name);
+  return -EINVAL;
+ }
+
```

```
+ off = 0;
+ entry_offset = (void *)e - (void *)base;
+ j = 0;
+ ret = IPT_MATCH_ITERATE(e, compat_check_calc_match, name, &e->ip,
+   e->comefrom, &off, &j);
+ if (ret != 0)
+  goto out;
+
+ t = ipt_get_target(e);
+ target = try_then_request_module(xt_find_target(AF_INET,
+           t->u.user.name,
+           t->u.user.revision),
+      "ipt_%s", t->u.user.name);
+ if (IS_ERR(target) || !target) {
+  duprintf("check_entry: `%s' not found\n", t->u.user.name);
+  ret = target ? PTR_ERR(target) : -ENOENT;
+  goto out;
+ }
+ t->u.kernel.target = target;
+
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ *size += off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  goto out;
+
+ /* Check hooks & underflows */
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)e - base == hook_entries[h])
+   newinfo->hook_entry[h] = hook_entries[h];
+  if ((unsigned char *)e - base == underflows[h])
+   newinfo->underflow[h] = underflows[h];
+ }
+
+ /* Clear counters and comefrom */
+ e->counters = ((struct ipt_counters) { 0, 0 });
+ e->comefrom = 0;
+
+ (*i)++;
+ return 0;
+out:
+ IPT_MATCH_ITERATE(e, cleanup_match, &j);
+ return ret;
+}
+
+static inline int compat_copy_match_from_user(struct ipt_entry_match *m,
+ void **dstptr, compat_uint_t *size, const char *name,
```

```
+ const struct ipt_ip *ip, unsigned int hookmask)
+{
+ struct ipt_entry_match *dm;
+
+ dm = (struct ipt_entry_match *)*dstptr;
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, dstptr, size, COMPAT_FROM_USER);
+ else {
+  memcpy(*dstptr, m, m->u.match_size);
+  *dstptr += m->u.match_size;
+ }
+
+ if (dm->u.kernel.match->checkentry
+     && !dm->u.kernel.match->checkentry(name, ip, dm->data,
+           dm->u.match_size - sizeof(*dm),
+           hookmask)) {
+  module_put(dm->u.kernel.match->me);
+  duprintf("ip_tables: check failed for `%s'.\n",
+    dm->u.kernel.match->name);
+  return -EINVAL;
+ }
+
+ return 0;
+}
+
+static int compat_copy_entry_from_user(struct ipt_entry *e, void **dstptr,
+ unsigned int *size, const char *name,
+ struct xt_table_info *newinfo, unsigned char *base)
+{
+ struct ipt_entry_target *t;
+ struct ipt_entry *de;
+ unsigned int origsize;
+ int ret, h;
+
+ ret = 0;
+ origsize = *size;
+ de = (struct ipt_entry *)*dstptr;
+ memcpy(de, e, sizeof(struct ipt_entry));
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_from_user, dstptr, size,
+   name, &de->ip, de->comefrom);
+ if (ret)
+  goto out;
+ de->target_offset = e->target_offset - (origsize - *size);
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t,
```

```
+    dstptr, size, COMPAT_FROM_USER);
+ else {
+  memcpy(*dstptr, t, t->u.target_size);
+  *dstptr += t->u.target_size;
+ }
+
+ de->next_offset = e->next_offset - (origsize - *size);
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)de - base < newinfo->hook_entry[h])
+   newinfo->hook_entry[h] -= origsize - *size;
+  if ((unsigned char *)de - base < newinfo->underflow[h])
+   newinfo->underflow[h] -= origsize - *size;
+ }
+
+ ret = -EINVAL;
+ t = ipt_get_target(de);
+ if (t->u.kernel.target == &ipt_standard_target) {
+  if (!standard_check(t, *size))
+   goto out;
+ } else if (t->u.kernel.target->checkentry
+     && !t->u.kernel.target->checkentry(name, de, t->data,
+            t->u.target_size
+            - sizeof(*t),
+            de->comefrom)) {
+  module_put(t->u.kernel.target->me);
+  duprintf("ip_tables: compat: check failed for `%s'.\n",
+    t->u.kernel.target->name);
+  goto out;
+ }
+ ret = 0;
+out:
+ return ret;
+}
+
+static int
+translate_compat_table(const char *name,
+  unsigned int valid_hooks,
+  struct xt_table_info **pinfo,
+  void **pentry0,
+  unsigned int total_size,
+  unsigned int number,
+  unsigned int *hook_entries,
+  unsigned int *underflows)
+{
+ unsigned int i;
+ struct xt_table_info *newinfo, *info;
+ void *pos, *entry0, *entry1;
+ unsigned int size;
```

```
+ int ret;
+
+ info = *pinfo;
+ entry0 = *pentry0;
+ size = total_size;
+ info->number = number;
+
+ /* Init all hooks to impossible value. */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  info->hook_entry[i] = 0xFFFFFFFF;
+  info->underflow[i] = 0xFFFFFFFF;
+ }
+
+ duprintf("translate_compat_table: size %u\n", info->size);
+ i = 0;
+ down(&compat_ipt_mutex);
+ /* Walk through entries, checking offsets. */
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+    check_compat_entry_size_and_hooks,
+    info, &size, entry0,
+    entry0 + total_size,
+    hook_entries, underflows, &i, name);
+ if (ret != 0)
+  goto out_unlock;
+
+ ret = -EINVAL;
+ if (i != number) {
+  duprintf("translate_compat_table: %u not %u entries\n",
+    i, number);
+  goto out_unlock;
+ }
+
+ /* Check hooks all assigned */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  /* Only hooks which are valid */
+  if (!(valid_hooks & (1 << i)))
+   continue;
+  if (info->hook_entry[i] == 0xFFFFFFFF) {
+   duprintf("Invalid hook entry %u %u\n",
+     i, hook_entries[i]);
+   goto out_unlock;
+  }
+  if (info->underflow[i] == 0xFFFFFFFF) {
+   duprintf("Invalid underflow %u %u\n",
+     i, underflows[i]);
+   goto out_unlock;
+  }
+ }
```

```
+
+ ret = -ENOMEM;
+ newinfo = xt_alloc_table_info(size);
+ if (!newinfo)
+  goto out_unlock;
+
+ newinfo->number = number;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ entry1 = newinfo->entries[raw_smp_processor_id()];
+ pos = entry1;
+ size =  total_size;
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+   compat_copy_entry_from_user, &pos, &size,
+   name, newinfo, entry1);
+ compat_flush_offsets();
+ up(&compat_ipt_mutex);
+ if (ret)
+  goto free_newinfo;
+
+ ret = -ELOOP;
+ if (!mark_source_chains(newinfo, valid_hooks, entry1))
+  goto free_newinfo;
+
+ /* And one copy for every other CPU */
+ for_each_cpu(i)
+  if (newinfo->entries[i] && newinfo->entries[i] != entry1)
+   memcpy(newinfo->entries[i], entry1, newinfo->size);
+
+ *pinfo = newinfo;
+ *pentry0 = entry1;
+ xt_free_table_info(info);
+ return 0;
+
+free_newinfo:
+ xt_free_table_info(newinfo);
+out:
+ return ret;
+out_unlock:
+ up(&compat_ipt_mutex);
+ goto out;
+}
+
+static int
+compat_do_replace(void __user *user, unsigned int len)
+{
```

```
+ int ret;
+ struct compat_ipt_replace tmp;
+ struct xt_table_info *newinfo;
+ void *loc_cpu_entry;
+
+ if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
+  return -EFAULT;
+
+ /* Hack: Causes ipchains to give correct error msg --RR */
+ if (len != sizeof(tmp) + tmp.size)
+  return -ENOPROTOOPT;
+
+ /* overflow check */
+ if (tmp.size >= (INT_MAX - sizeof(struct xt_table_info)) / NR_CPUS -
+   SMP_CACHE_BYTES)
+  return -ENOMEM;
+ if (tmp.num_counters >= INT_MAX / sizeof(struct xt_counters))
+  return -ENOMEM;
+
+ newinfo = xt_alloc_table_info(tmp.size);
+ if (!newinfo)
+  return -ENOMEM;
+
+ /* choose the copy that is our node/cpu */
+ loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
+ if (copy_from_user(loc_cpu_entry, user + sizeof(tmp),
+      tmp.size) != 0) {
+  ret = -EFAULT;
+  goto free_newinfo;
+ }
+
+ ret = translate_compat_table(tmp.name, tmp.valid_hooks,
+         &newinfo, &loc_cpu_entry, tmp.size,
+         tmp.num_entries, tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
+  goto free_newinfo;
+
+ duprintf("compat_do_replace: Translated table\n");
+
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         compat_ptr(tmp.counters));
+ if (ret)
+  goto free_newinfo_untrans;
+ return 0;
+
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
```

```
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}
+
+struct compat_ipt_get_entries
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ compat_uint_t size;
+ struct compat_ipt_entry entrytable[0];
+};
+
+static int compat_copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table, void __user *userptr)
+{
+ unsigned int off, num;
+ struct compat_ipt_entry e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ void __user *pos;
+ unsigned int size;
+ int ret = 0;
+ void *loc_cpu_entry;
+
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
+ /* choose the copy that is on our node/cpu, ...
+  * This choice is lazy (because current thread is
+  * allowed to migrate to another cpu)
+  */
+ loc_cpu_entry = private->entries[raw_smp_processor_id()];
+ pos = userptr;
+ size = total_size;
+ ret = IPT_ENTRY_ITERATE(loc_cpu_entry, total_size,
+   compat_copy_entry_to_user, &pos, &size);
+ if (ret)
+  goto free_counters;
+
+ /* ... then go back and fix counters and names */
+ for (off = 0, num = 0; off < size; off += e.next_offset, num++) {
+  unsigned int i;
+  struct ipt_entry_match m;
+  struct ipt_entry_target t;
+
+  ret = -EFAULT;
+  if (copy_from_user(&e, userptr + off,
```

```
+     sizeof(struct compat_ipt_entry)))
+   goto free_counters;
+  if (copy_to_user(userptr + off +
+   offsetof(struct compat_ipt_entry, counters),
+    &counters[num], sizeof(counters[num])))
+   goto free_counters;
+
+  for (i = sizeof(struct compat_ipt_entry);
+    i < e.target_offset; i += m.u.match_size) {
+   if (copy_from_user(&m, userptr + off + i,
+     sizeof(struct ipt_entry_match)))
+    goto free_counters;
+   if (copy_to_user(userptr + off + i +
+    offsetof(struct ipt_entry_match, u.user.name),
+    m.u.kernel.match->name,
+    strlen(m.u.kernel.match->name) + 1))
+    goto free_counters;
+  }
+
+  if (copy_from_user(&t, userptr + off + e.target_offset,
+     sizeof(struct ipt_entry_target)))
+   goto free_counters;
+  if (copy_to_user(userptr + off + e.target_offset +
+   offsetof(struct ipt_entry_target, u.user.name),
+   t.u.kernel.target->name,
+   strlen(t.u.kernel.target->name) + 1))
+   goto free_counters;
+ }
+ ret = 0;
+free_counters:
+ vfree(counters);
+ return ret;
+}
+
+static int
+compat_get_entries(struct compat_ipt_get_entries __user *uptr, int *len)
+{
+ int ret;
+ struct compat_ipt_get_entries get;
+ struct ipt_table *t;
+
+
+ if (*len < sizeof(get)) {
+  duprintf("compat_get_entries: %u < %u\n",
+    *len, (unsigned int)sizeof(get));
+  return -EINVAL;
+ }
+
```

```
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;
+
+ if (*len != sizeof(struct compat_ipt_get_entries) + get.size) {
+  duprintf("compat_get_entries: %u != %u\n", *len,
+   (unsigned int)(sizeof(struct compat_ipt_get_entries) +
+   get.size));
+  return -EINVAL;
+ }
+
+ down(&compat_ipt_mutex);
+ t = xt_find_table_lock(AF_INET, get.name);
+ if (t && !IS_ERR(t)) {
+  struct xt_table_info *private = t->private;
+  struct xt_table_info info;
+  duprintf("t->private->number = %u\n",
+    private->number);
+  ret = compat_table_info(private, &info);
+  if (!ret && get.size == info.size) {
+   ret = compat_copy_entries_to_user(private->size,
+         t, uptr->entrytable);
+  } else if (!ret) {
+   duprintf("compat_get_entries: I've got %u not %u!\n",
+     private->size,
+     get.size);
+   ret = -EINVAL;
+  }
+  compat_flush_offsets();
+  module_put(t->me);
+  xt_table_unlock(t);
+ } else
+  ret = t ? PTR_ERR(t) : -ENOENT;
+
+ up(&compat_ipt_mutex);
+ return ret;
+}
+
+static int
+compat_do_ipt_get_ctl(struct sock *sk, int cmd, void __user *user, int *len)
+{
+ int ret;
+
+ switch (cmd) {
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len);
+  break;
+ case IPT_SO_GET_ENTRIES:
+  ret = compat_get_entries(user, len);
```

```
+  break;
+ default:
+  duprintf("compat_do_ipt_get_ctl: unknown request %i\n", cmd);
+  ret = -EINVAL;
+ }
+ return ret;
+}
+#endif
+
 static int
 do_ipt_set_ctl(struct sock *sk, int cmd, void __user *user, unsigned int len)
 {
@@ -1094,6 +2094,11 @@ do_ipt_set_ctl(struct sock *sk, int cmd,
  if (!capable(CAP_NET_ADMIN))
   return -EPERM;

+#ifdef CONFIG_COMPAT
+ if (is_current_32bits() && (cmd == IPT_SO_SET_REPLACE))
+  return compat_do_replace(user, len);
+#endif
+
  switch (cmd) {
  case IPT_SO_SET_REPLACE:
   ret = do_replace(user, len);
@@ -1119,66 +2124,19 @@ do_ipt_get_ctl(struct sock *sk, int cmd,
  if (!capable(CAP_NET_ADMIN))
   return -EPERM;

- switch (cmd) {
- case IPT_SO_GET_INFO: {
-  char name[IPT_TABLE_MAXNAMELEN];
-  struct ipt_table *t;
-
-  if (*len != sizeof(struct ipt_getinfo)) {
-   duprintf("length %u != %u\n", *len,
-     sizeof(struct ipt_getinfo));
-   ret = -EINVAL;
-   break;
-  }
-
-  if (copy_from_user(name, user, sizeof(name)) != 0) {
-   ret = -EFAULT;
-   break;
-  }
-  name[IPT_TABLE_MAXNAMELEN-1] = '\0';
-
-  t = try_then_request_module(xt_find_table_lock(AF_INET, name),
-         "iptable_%s", name);
```

```
-  if (t && !IS_ERR(t)) {
-   struct ipt_getinfo info;
-   struct xt_table_info *private = t->private;
-
-   info.valid_hooks = t->valid_hooks;
-   memcpy(info.hook_entry, private->hook_entry,
-          sizeof(info.hook_entry));
-   memcpy(info.underflow, private->underflow,
-          sizeof(info.underflow));
-   info.num_entries = private->number;
-   info.size = private->size;
-   memcpy(info.name, name, sizeof(info.name));
-
-   if (copy_to_user(user, &info, *len) != 0)
-    ret = -EFAULT;
-   else
-    ret = 0;
-   xt_table_unlock(t);
-   module_put(t->me);
-  } else
-   ret = t ? PTR_ERR(t) : -ENOENT;
- }
- break;
+#ifdef CONFIG_COMPAT
+ if (is_current_32bits())
+  return compat_do_ipt_get_ctl(sk, cmd, user, len);
+#endif

- case IPT_SO_GET_ENTRIES: {
-  struct ipt_get_entries get;
+ switch (cmd) {
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len);
+  break;

-  if (*len < sizeof(get)) {
-   duprintf("get_entries: %u < %u\n", *len, sizeof(get));
-   ret = -EINVAL;
-  } else if (copy_from_user(&get, user, sizeof(get)) != 0) {
-   ret = -EFAULT;
-  } else if (*len != sizeof(struct ipt_get_entries) + get.size) {
-   duprintf("get_entries: %u != %u\n", *len,
-     sizeof(struct ipt_get_entries) + get.size);
-   ret = -EINVAL;
-  } else
-   ret = get_entries(&get, user);
+ case IPT_SO_GET_ENTRIES:
+  ret = get_entries(user, len);
```

```
   break;
- }

  case IPT_SO_GET_REVISION_MATCH:
  case IPT_SO_GET_REVISION_TARGET: {
@@ -1327,6 +2285,9 @@ icmp_checkentry(const char *tablename,
 /* The built-in targets: standard (NULL) and error. */
 static struct ipt_target ipt_standard_target = {
  .name  = IPT_STANDARD_TARGET,
+#ifdef CONFIG_COMPAT
+ .compat  = &compat_ipt_standard_fn,
+#endif
 };

 static struct ipt_target ipt_error_target = {
@@ -1348,6 +2309,9 @@ static struct ipt_match icmp_matchstruct
  .name  = "icmp",
  .match  = &icmp_match,
  .checkentry = &icmp_checkentry,
+#ifdef CONFIG_COMPAT
+ .compat  = &icmp_compat,
+#endif
 };

 static int __init init(void)
@@ -1386,5 +2350,9 @@ static void __exit fini(void)
 EXPORT_SYMBOL(ipt_register_table);
 EXPORT_SYMBOL(ipt_unregister_table);
 EXPORT_SYMBOL(ipt_do_table);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(ipt_match_align_compat);
+EXPORT_SYMBOL(ipt_target_align_compat);
+#endif
 module_init(init);
 module_exit(fini);
--- ./net/netfilter/xt_tcpudp.c.iptcompat 2006-02-15 16:06:42.000000000 +0300
+++ ./net/netfilter/xt_tcpudp.c 2006-02-17 19:41:58.000000000 +0300
@@ -266,10 +266,35 @@ udp6_checkentry(const char *tablename,
  return 1;
 }

+#ifdef CONFIG_COMPAT
+static int tcp_compat(void *match,
+  void **dstptr, int *size, int convert)
+{
+ int off;
+
+ off = XT_ALIGN(sizeof(struct xt_tcp)) -
```

```
+  COMPAT_XT_ALIGN(sizeof(struct xt_tcp));
+ return ipt_match_align_compat(match, dstptr, size, off, convert);
+}
+
+static int udp_compat(void *match,
+  void **dstptr, int *size, int convert)
+{
+ int off;
+
+ off = XT_ALIGN(sizeof(struct xt_udp)) -
+  COMPAT_XT_ALIGN(sizeof(struct xt_udp));
+ return ipt_match_align_compat(match, dstptr, size, off, convert);
+}
+#endif
+
 static struct xt_match tcp_matchstruct = {
  .name  = "tcp",
  .match  = &tcp_match,
 .checkentry = &tcp_checkentry,
+#ifdef CONFIG_COMPAT
+ .compat  = &tcp_compat,
+#endif
 .me  = THIS_MODULE,
 };
 static struct xt_match tcp6_matchstruct = {
@@ -283,6 +308,9 @@ static struct xt_match udp_matchstruct =
 .name  = "udp",
 .match  = &udp_match,
 .checkentry = &udp_checkentry,
+#ifdef CONFIG_COMPAT
+ .compat  = &udp_compat,
+#endif
 .me  = THIS_MODULE,
 };
 static struct xt_match udp6_matchstruct = {
```

File Attachments

1) diff-ms-ipt-compat-20060217, downloaded 692 times

Subject: Re: [PATCH 1/2] iptables 32bit compat layer
Posted by davem on Mon, 20 Feb 2006 08:31:46 GMT
View Forum Message <> Reply to Message

From: Mishin Dmitry <dim@openvz.org>
Date: Mon, 20 Feb 2006 11:10:38 +0300

> This patch set extends current iptables compatibility layer in order

> to get 32bit iptables to work on 64bit kernel. Current layer is
> insufficient due to alignment checks both in kernel and user space
> tools.

Thanks a lot for doing this work Mishin.

Subject: Re: [PATCH 1/2] iptables 32bit compat layer
Posted by Arnd Bergmann on Mon, 20 Feb 2006 15:55:26 GMT
View Forum Message <> Reply to Message

On Monday 20 February 2006 09:10, Mishin Dmitry wrote:

> @@ -118,6 +125,10 @@ struct xt_match

> +#ifdef CONFIG_COMPAT

> +#endif

Is CONFIG_COMPAT the right conditional here? If the code is only used
for architectures that have different aligments, it should not need be
compiled in for the other architectures.

> @@ -154,6 +165,10 @@ struct xt_target

> +#ifdef CONFIG_COMPAT

> +#endif

> @@ -233,6 +248,34 @@ extern void xt_proto_fini(int af);

> +#ifdef CONFIG_COMPAT
> +#include <net/compat.h>
> +
> +/* FIXME: this works only on 32 bit tasks
> + * need to change whole approach in order to calculate align as function of
> + * current task alignment */
> +
> +struct compat_xt_counters
> +{

> +};

Hmm, maybe we should have something like

typedef u64 __attribute__((aligned(4))) compat_u64;

in order to get the right alignment on the architectures
where it makes a difference. Do all compiler versions
get that right?

+0300

> @@ -364,5 +365,62 @@ extern unsigned int ipt_do_table(struct

> +
> +#ifdef CONFIG_COMPAT
> +#include <net/compat.h>
> +
> +struct compat_ipt_getinfo
> +{

> +};

This structure looks like it does not need any
conversions. You should probably just use
struct ipt_getinfo then.

> +
> +struct compat_ipt_entry_match
> +{

> +};
> +
> +struct compat_ipt_entry_target
> +{

> +};

Dito

> +
> +extern int ipt_match_align_compat(void *match, void **dstptr,

> +extern int ipt_target_align_compat(void *target, void **dstptr,

> +
> +#endif /* CONFIG_COMPAT */

> @@ -23,6 +23,14 @@ struct compat_cmsghdr {

> +#if defined(CONFIG_X86_64)
> +#define is_current_32bits() (current_thread_info()->flags & _TIF_IA32)
> +#elif defined(CONFIG_IA64)
> +#define is_current_32bits() (IS_IA32_PROCESS(ia64_task_regs(current)))
> +#else

> +#endif
> +

This definition looks very wrong to me. For x86_64, the right thing to check
should be TS_COMPAT, no _TIF_IA32, since you can also call the 64 bit
syscall entry point from a i386 task running on x86_64. For most other
architectures, is_current_32bits returns something that is not reflected
in the name. I would e.g. expect the function to return '1' on i386 and
the correct task state on other compat platforms, instead of a bogus '0'.

There have been long discussions about the inclusions of the 'is_compat_task'
macro. Let's at least not define a second function that does almost the
same but gets it wrong.

I would much rather have either an extra 'compat' argument to to
sock_setsockopt and proto_ops->setsockopt than to spread the use
of is_compat_task further.

> @@ -308,107 +308,6 @@ void scm_detach_fds_compat(struct msghdr

> - * For now, we assume that the compatibility and native version

> - */
> -struct compat_ipt_replace {

> -};

Is the FIXME above the only reason that the code needs to be changed?
What is the reason that you did not just address this in the
compat_sys_setsockopt implementation?

Arnd <><

Subject: Re: [PATCH 1/2] iptables 32bit compat layer
Posted by Andi Kleen on Mon, 20 Feb 2006 21:23:25 GMT
View Forum Message <> Reply to Message

Mishin Dmitry <dim@openvz.org> writes:

> Hello,
>
> This patch set extends current iptables compatibility layer in order to get
> 32bit iptables to work on 64bit kernel. Current layer is insufficient
> due to alignment checks both in kernel and user space tools.
>
> This patch introduces base compatibility interface for other ip_tables modules

Nice. But some issues with the implementation

+#if defined(CONFIG_X86_64)
+#define is_current_32bits() (current_thread_info()->flags & _TIF_IA32)

This should be is_compat_task(). And we don't do such ifdefs
in generic code. And what you actually need here is a
is_compat_task_with_funny_u64_alignment() (better name sought)

So I would suggest you add macros for that to the ia64 and x86-64
asm/compat.hs and perhaps a ARCH_HAS_FUNNY_U64_ALIGNMENT #define in there.

+ ret = 0;
+ switch (convert) {
+ case COMPAT_TO_USER:
+ pt = (struct ipt_entry_target *)target;

etc. that looks ugly. why can't you just define different functions
for that? We don't really need in kernel ioctl

+#ifdef CONFIG_COMPAT
+ down(&compat_ipt_mutex);
+#endif

Why does it need an own lock?

Overall the implementation looks very complicated. Are you sure
it wasn't possible to do this simpler?

-Andi

Subject: Re: Re: [PATCH 1/2] iptables 32bit compat layer
Posted by dim on Tue, 21 Feb 2006 09:04:49 GMT
View Forum Message <> Reply to Message

On Monday 20 February 2006 18:55, Arnd Bergmann wrote:
> On Monday 20 February 2006 09:10, Mishin Dmitry wrote:

> > 16:16:02.000000000 +0300 +++

> > +0300 struct xt_match

> > @@ -118,6 +125,10 @@ struct xt_match

> > +#ifdef CONFIG_COMPAT

> > convert); +#endif

>
> Is CONFIG_COMPAT the right conditional here? If the code is only used
> for architectures that have different aligments, it should not need be
> compiled in for the other architectures.
So, I'll define ARCH_HAS_FUNNY_64_ALIGNMENT in x86_64 and ia64 code and will
check it, as Andi suggested.

>
> > @@ -154,6 +165,10 @@ struct xt_target

> > +#ifdef CONFIG_COMPAT

> > convert); +#endif

> > @@ -233,6 +248,34 @@ extern void xt_proto_fini(int af);

> > +#ifdef CONFIG_COMPAT
> > +#include <net/compat.h>
> > +

> > +/* FIXME: this works only on 32 bit tasks
> > + * need to change whole approach in order to calculate align as function
> > of + * current task alignment */
> > +
> > +struct compat_xt_counters
> > +{

> > +};
>
> Hmm, maybe we should have something like
>
> typedef u64 __attribute__((aligned(4))) compat_u64;
>
> in order to get the right alignment on the architectures
> where it makes a difference. Do all compiler versions
> get that right?
good point. I don't know this and that's why tried to avoid use of 'aligned'
attribute.

>
> > ---

> > 16:06:41.000000000 +0300 +++

> > +0300 @@ -364,5 +365,62 @@ extern unsigned int ipt_do_table(struct

> > +
> > +#ifdef CONFIG_COMPAT
> > +#include <net/compat.h>
> > +
> > +struct compat_ipt_getinfo
> > +{

> > +};
>
> This structure looks like it does not need any
> conversions. You should probably just use
> struct ipt_getinfo then.
I just saw compat_uint_t use in net/compat.c and thought, that it is a good
style to use it. Does anybody know arch, where sizeof(compat_uint_t) != 4?

>
> > +
> > +struct compat_ipt_entry_match
> > +{

> > +};
> > +
> > +struct compat_ipt_entry_target
> > +{

> > +};
>
> Dito
Disagree, ipt_entry_match and ipt_entry_target contain pointers which make
their alignment equal 8 byte on 64bits architectures.

>

> > +
> > +extern int ipt_match_align_compat(void *match, void **dstptr,

> > +extern int ipt_target_align_compat(void *target, void **dstptr,

> > +
> > +#endif /* CONFIG_COMPAT */

> > @@ -23,6 +23,14 @@ struct compat_cmsghdr {

> > +#if defined(CONFIG_X86_64)

> > +#define is_current_32bits() (current_thread_info()->flags & _TIF_IA32)
> > +#elif defined(CONFIG_IA64)
> > +#define is_current_32bits() (IS_IA32_PROCESS(ia64_task_regs(current)))
> > +#else

> > +#endif
> > +
>
> This definition looks very wrong to me. For x86_64, the right thing to
> check should be TS_COMPAT, no _TIF_IA32, since you can also call the 64 bit
> syscall entry point from a i386 task running on x86_64. For most other
> architectures, is_current_32bits returns something that is not reflected in
> the name. I would e.g. expect the function to return '1' on i386 and the
> correct task state on other compat platforms, instead of a bogus '0'.
>
> There have been long discussions about the inclusions of the
> 'is_compat_task' macro. Let's at least not define a second function that
> does almost the same but gets it wrong.
>
> I would much rather have either an extra 'compat' argument to to
> sock_setsockopt and proto_ops->setsockopt than to spread the use
> of is_compat_task further.
Another weak place in my code. is_compat_task() approach has one advantage -
it doesn't require a lot of current code modifications.
>

> > @@ -308,107 +308,6 @@ void scm_detach_fds_compat(struct msghdr

> > - * For now, we assume that the compatibility and native version

> > - */
> > -struct compat_ipt_replace {

> > -};

>
> Is the FIXME above the only reason that the code needs to be changed?
> What is the reason that you did not just address this in the
> compat_sys_setsockopt implementation?
Code above doesn't work. iptables with version >= 1.3 does alignment checks as
well as kernel does. So, we can't simply put entries with 8 bytes alignment
to userspace or with 4 bytes alignment to kernel - we need translate them
entry by entry. So, I tried to do this the most correct way - that userspace
will hide its alignment from kernel and vice versa, with not only
SET_REPLACE, but also GET_INFO, GET_ENTRIES and SET_COUNTERS translation.
First implementation was exactly in compat_sys_setsockopt, but David asked me
to do this in netfilter code itself.

>
> Arnd <><
>
--
Thanks,
Dmitry.

Subject: Re: Re: [PATCH 1/2] iptables 32bit compat layer
Posted by dim on Tue, 21 Feb 2006 09:24:27 GMT
View Forum Message <> Reply to Message

On Tuesday 21 February 2006 00:23, Andi Kleen wrote:
> Mishin Dmitry <dim@openvz.org> writes:
> > Hello,
> >
> > This patch set extends current iptables compatibility layer in order to
> > get 32bit iptables to work on 64bit kernel. Current layer is insufficient
> > due to alignment checks both in kernel and user space tools.
> >
> > This patch introduces base compatibility interface for other ip_tables
> > modules
>
> Nice. But some issues with the implementation
>
>
> +#if defined(CONFIG_X86_64)
> +#define is_current_32bits() (current_thread_info()->flags & _TIF_IA32)
>
> This should be is_compat_task(). And we don't do such ifdefs
> in generic code. And what you actually need here is a
> is_compat_task_with_funny_u64_alignment() (better name sought)
>
> So I would suggest you add macros for that to the ia64 and x86-64
> asm/compat.hs and perhaps a ARCH_HAS_FUNNY_U64_ALIGNMENT #define in there.

agree.

>
> + ret = 0;
> + switch (convert) {
> + case COMPAT_TO_USER:
> +   pt = (struct ipt_entry_target *)target;
>
> etc. that looks ugly. why can't you just define different functions
> for that? We don't really need in kernel ioctl
3 functions and the requirement that if defined one, than defined all of them?

>
> +#ifdef CONFIG_COMPAT
> + down(&compat_ipt_mutex);
> +#endif
>
> Why does it need an own lock?
Because it protects only compatibility translation. We spend a lot of time in
these cycles and I don't think that it is a good way to hold ipt_mutex for
this. The only reason of this lock is offset list - in the first iteration I
fill it, in the second - use it. If you know how to implement this better,
let me know.

>
> Overall the implementation looks very complicated. Are you sure
> it wasn't possible to do this simpler?
ughh...
I don't like this code as well. But seems that it is due to iptables code
itself, which was designed with no thoughts about compatibility in minds.

So, I see following approaches:
1) do translation before pass data to original do_replace or get_entries.
Disadvantage of such approach is additional 2 cycles through data.
2) do translation in compat_do_replace and compat_get_entries. Avoidance of
additional cycles, but some code duplication.
3) remove alignment checks in kernel - than we need only first time
translation from kernel to user. But such code will not work with both 32bit
and 64 bit iptables at the same time.

Any suggestions?

>
>
> -Andi
>
--
Thanks,

Dmitry.

Subject: Re: Re: [PATCH 1/2] iptables 32bit compat layer
Posted by Arnd Bergmann on Tue, 21 Feb 2006 11:56:00 GMT
View Forum Message <> Reply to Message

On Tuesday 21 February 2006 10:04, Dmitry Mishin wrote:
> On Monday 20 February 2006 18:55, Arnd Bergmann wrote:

> > Is CONFIG_COMPAT the right conditional here? If the code is only used
> > for architectures that have different aligments, it should not need be
> > compiled in for the other architectures.
> So, I'll define ARCH_HAS_FUNNY_64_ALIGNMENT in x86_64 and ia64 code and will
> check it, as Andi suggested.
>

I think nowadays, unconditionally setting CONFIG_FUNNY_64_ALIGNMENT from
arch/{ia64,x86_64}/Kconfig would be the preferred way to a #define in
include/asm.

> > > +
> > > +#ifdef CONFIG_COMPAT
> > > +#include <net/compat.h>
> > > +
> > > +struct compat_ipt_getinfo
> > > +{

> > > +};
> >
> > This structure looks like it does not need any
> > conversions. You should probably just use
> > struct ipt_getinfo then.
> I just saw compat_uint_t use in net/compat.c and thought, that it is a good
> style to use it. Does anybody know arch, where sizeof(compat_uint_t) != 4?

No, the compat layer already heavily depends on the fact that compat_uint_t
is always the same as unsigned int.

> >
> > Dito
> Disagree, ipt_entry_match and ipt_entry_target contain pointers which make

> their alignment equal 8 byte on 64bits architectures.

Ah, I see.

> > I would much rather have either an extra 'compat' argument to to
> > sock_setsockopt and proto_ops->setsockopt than to spread the use
> > of is_compat_task further.
> Another weak place in my code. is_compat_task() approach has one advantage -
> it doesn't require a lot of current code modifications.
> >
> > Is the FIXME above the only reason that the code needs to be changed?
> > What is the reason that you did not just address this in the
> > compat_sys_setsockopt implementation?
> Code above doesn't work. iptables with version >= 1.3 does alignment checks as
> well as kernel does. So, we can't simply put entries with 8 bytes alignment
> to userspace or with 4 bytes alignment to kernel - we need translate them
> entry by entry. So, I tried to do this the most correct way - that userspace
> will hide its alignment from kernel and vice versa, with not only
> SET_REPLACE, but also GET_INFO, GET_ENTRIES and SET_COUNTERS translation.
> First implementation was exactly in compat_sys_setsockopt, but David asked me
> to do this in netfilter code itself.

Ok, I see the point there. It's probably best to push down all the conversions
from compat_sys_setsockopt down to the protocol specific parts, similar to what
we do for the ioctl handlers.

I'm thinking of something like

```
int compat_sock_setsockopt(struct socket *sock, int level, int optname,
      char __user *optval, int optlen)
{
 switch (optname) {
 case SO_ATTACH_FILTER:
  return do_set_attach_filter(fd, level, optname,
         optval, optlen);
 case SO_SNDTIMEO:
  return do_set_sock_timeout(fd, level, optname,
        optval, optlen);
 default:
  break;
 }
 return sock_setsockopt(sock, level, optname, optval, optlen);
}

asmlinkage long compat_sys_setsockopt(int fd, int level, int optname,
    char __user *optval, int optlen)
{
 int err;
```

```
 struct socket *sock;

 if (optlen < 0)
  return -EINVAL;

 if ((sock = sockfd_lookup(fd, &err))!=NULL)
 {
  err = security_socket_setsockopt(sock,level,optname);
  if (err) {
   sockfd_put(sock);
   return err;
  }

  if (level == SOL_SOCKET)
   err = compat_sock_setsockopt(sock, level,
     optname, optval, optlen);
  else if (sock->ops->compat_setsockopt)
   err = sock->ops->compat_setsockopt(sock, level,
     optname, optval, optlen);
  else
   err = sock->ops->setsockopt(sock, level,
     optname, optval, optlen);
  sockfd_put(sock);
 }
 return err;
}

int tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen)
{
 int err = 0;

 err = ip_setsockopt(sk, level, optname, optval, optlen);

#ifdef CONFIG_NETFILTER
 if (err = -ENOPROTOOPT) {
  lock_sock(sk);
  err = nf_setsockopt(sk, PF_INET, optname, optval, optlen);
  release_sock(sk);
 }
#endif
 return err;
}

int compat_tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen)
{
 int err = 0;

 err = ip_setsockopt(sk, level, optname, optval, optlen);
```

```
#ifdef CONFIG_NETFILTER
 if (err = -ENOPROTOOPT) {
  lock_sock(sk);
  err = compat_nf_setsockopt(sk, PF_INET, optname, optval, optlen);
  release_sock(sk);
 }
#endif
 return err;
}
```

And the same for udp, raw, ipv6, decnet and each of those with getsockopt.
It is a bigger change, but it puts all the handlers where they belong
and it is more extensible to other sockopt handlers if we find more
fsckup in some of them.

Arnd <><

Subject: {get|set}sockopt compat layer
Posted by Mishin Dmitry on Tue, 07 Mar 2006 14:07:18 GMT
View Forum Message <> Reply to Message

Hello, Arnd!

Sorry for such delay, was on vacancy. Here is a patch, introducing
compat_(get|set)sockopt handlers, as you proposed.

On Tuesday 21 February 2006 14:56, Arnd Bergmann wrote:
> On Tuesday 21 February 2006 10:04, Dmitry Mishin wrote:
> > On Monday 20 February 2006 18:55, Arnd Bergmann wrote:
> > > Is CONFIG_COMPAT the right conditional here? If the code is only used
> > > for architectures that have different aligments, it should not need be
> > > compiled in for the other architectures.
> >
> > So, I'll define ARCH_HAS_FUNNY_64_ALIGNMENT in x86_64 and ia64 code and
> > will check it, as Andi suggested.
>
> I think nowadays, unconditionally setting CONFIG_FUNNY_64_ALIGNMENT from
> arch/{ia64,x86_64}/Kconfig would be the preferred way to a #define in
> include/asm.
>

> > > > +
> > > > +#ifdef CONFIG_COMPAT
> > > > +#include <net/compat.h>
> > > > +
> > > > +struct compat_ipt_getinfo

> > > > +{

> > > > +};
> > >
> > > This structure looks like it does not need any
> > > conversions. You should probably just use
> > > struct ipt_getinfo then.
> >
> > I just saw compat_uint_t use in net/compat.c and thought, that it is a
> > good style to use it. Does anybody know arch, where sizeof(compat_uint_t)
> > != 4?
>
> No, the compat layer already heavily depends on the fact that compat_uint_t
> is always the same as unsigned int.
>
> > > Dito
> >
> > Disagree, ipt_entry_match and ipt_entry_target contain pointers which
> > make their alignment equal 8 byte on 64bits architectures.
>
> Ah, I see.
>
> > > I would much rather have either an extra 'compat' argument to to
> > > sock_setsockopt and proto_ops->setsockopt than to spread the use
> > > of is_compat_task further.
> >
> > Another weak place in my code. is_compat_task() approach has one
> > advantage - it doesn't require a lot of current code modifications.
> >
> > > Is the FIXME above the only reason that the code needs to be changed?
> > > What is the reason that you did not just address this in the
> > > compat_sys_setsockopt implementation?
> >
> > Code above doesn't work. iptables with version >= 1.3 does alignment
> > checks as well as kernel does. So, we can't simply put entries with 8
> > bytes alignment to userspace or with 4 bytes alignment to kernel - we
> > need translate them entry by entry. So, I tried to do this the most
> > correct way - that userspace will hide its alignment from kernel and vice
> > versa, with not only SET_REPLACE, but also GET_INFO, GET_ENTRIES and
> > SET_COUNTERS translation. First implementation was exactly in
> > compat_sys_setsockopt, but David asked me to do this in netfilter code
> > itself.
>

> Ok, I see the point there. It's probably best to push down all the
> conversions from compat_sys_setsockopt down to the protocol specific parts,
> similar to what we do for the ioctl handlers.
>
> I'm thinking of something like
>
> int compat_sock_setsockopt(struct socket *sock, int level, int optname,
>   char __user *optval, int optlen)
> {
> switch (optname) {
> case SO_ATTACH_FILTER:
>   return do_set_attach_filter(fd, level, optname,
>   optval, optlen);
> case SO_SNDTIMEO:
>   return do_set_sock_timeout(fd, level, optname,
>   optval, optlen);
> default:
>   break;
> }
> return sock_setsockopt(sock, level, optname, optval, optlen);
> }
>
> asmlinkage long compat_sys_setsockopt(int fd, int level, int optname,
>   char __user *optval, int optlen)
> {
> int err;
> struct socket *sock;
>
> if (optlen < 0)
>   return -EINVAL;
>
> if ((sock = sockfd_lookup(fd, &err))!=NULL)
> {
>   err = security_socket_setsockopt(sock,level,optname);
>   if (err) {
>   sockfd_put(sock);
>   return err;
>   }
>
>   if (level == SOL_SOCKET)
>   err = compat_sock_setsockopt(sock, level,
>   optname, optval, optlen);
>   else if (sock->ops->compat_setsockopt)
>   err = sock->ops->compat_setsockopt(sock, level,
>   optname, optval, optlen);
>   else
>   err = sock->ops->setsockopt(sock, level,
>   optname, optval, optlen);

> sockfd_put(sock);
> }
> return err;
> }
>
> int tcp_setsockopt(struct sock *sk, int level, int optname, char __user
> *optval, int optlen) {
> int err = 0;
>
> err = ip_setsockopt(sk, level, optname, optval, optlen);
>
> #ifdef CONFIG_NETFILTER
> if (err = -ENOPROTOOPT) {
> lock_sock(sk);
> err = nf_setsockopt(sk, PF_INET, optname, optval, optlen);
> release_sock(sk);
> }
> #endif
> return err;
> }
>
> int compat_tcp_setsockopt(struct sock *sk, int level, int optname, char
> __user *optval, int optlen) {
> int err = 0;
>
> err = ip_setsockopt(sk, level, optname, optval, optlen);
>
> #ifdef CONFIG_NETFILTER
> if (err = -ENOPROTOOPT) {
> lock_sock(sk);
> err = compat_nf_setsockopt(sk, PF_INET, optname, optval, optlen);
> release_sock(sk);
> }
> #endif
> return err;
> }
>
> And the same for udp, raw, ipv6, decnet and each of those with getsockopt.
> It is a bigger change, but it puts all the handlers where they belong
> and it is more extensible to other sockopt handlers if we find more
> fsckup in some of them.
>
> Arnd <><

--
Thanks,
Dmitry.

```
--- ./include/linux/net.h.compat 2006-03-07 11:22:27.000000000 +0300
+++ ./include/linux/net.h 2006-03-07 11:20:07.000000000 +0300
@@ -149,6 +149,12 @@ struct proto_ops {
          int optname, char __user *optval, int optlen);
  int  (*getsockopt)(struct socket *sock, int level,
          int optname, char __user *optval, int __user *optlen);
+#ifdef CONFIG_COMPAT
+ int  (*compat_setsockopt)(struct socket *sock, int level,
+          int optname, char __user *optval, int optlen);
+ int  (*compat_getsockopt)(struct socket *sock, int level,
+          int optname, char __user *optval, int __user *optlen);
+#endif
  int  (*sendmsg)   (struct kiocb *iocb, struct socket *sock,
          struct msghdr *m, size_t total_len);
  int  (*recvmsg)   (struct kiocb *iocb, struct socket *sock,
--- ./include/linux/netfilter.h.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./include/linux/netfilter.h 2006-03-07 15:00:14.000000000 +0300
@@ -2,6 +2,7 @@
 #define __LINUX_NETFILTER_H

 #ifdef __KERNEL__
+#include <linux/config.h>
 #include <linux/init.h>
 #include <linux/types.h>
 #include <linux/skbuff.h>
@@ -80,10 +81,18 @@ struct nf_sockopt_ops
  int set_optmin;
  int set_optmax;
  int (*set)(struct sock *sk, int optval, void __user *user, unsigned int len);
+#ifdef CONFIG_COMPAT
+ int (*compat_set)(struct sock *sk, int optval,
+   void __user *user, unsigned int len);
+#endif

  int get_optmin;
  int get_optmax;
  int (*get)(struct sock *sk, int optval, void __user *user, int *len);
+#ifdef CONFIG_COMPAT
+ int (*compat_get)(struct sock *sk, int optval,
+   void __user *user, int *len);
+#endif

  /* Number of users inside set() or get(). */
  unsigned int use;
@@ -246,6 +255,13 @@ int nf_setsockopt(struct sock *sk, int p
 int nf_getsockopt(struct sock *sk, int pf, int optval, char __user *opt,
     int *len);
```

```
+#ifdef CONFIG_COMPAT
+int compat_nf_setsockopt(struct sock *sk, int pf, int optval,
+  char __user *opt, int len);
+int compat_nf_getsockopt(struct sock *sk, int pf, int optval,
+  char __user *opt, int *len);
+#endif
+
 /* Packet queuing */
 struct nf_queue_handler {
  int (*outfn)(struct sk_buff *skb, struct nf_info *info,
--- ./include/net/inet_connection_sock.h.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./include/net/inet_connection_sock.h 2006-03-07 15:46:20.000000000 +0300
@@ -15,6 +15,7 @@
 #ifndef _INET_CONNECTION_SOCK_H
 #define _INET_CONNECTION_SOCK_H

+#include <linux/config.h>
 #include <linux/compiler.h>
 #include <linux/string.h>
 #include <linux/timer.h>
@@ -50,6 +51,14 @@ struct inet_connection_sock_af_ops {
       char __user *optval, int optlen);
  int     (*getsockopt)(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen);
+#ifdef CONFIG_COMPAT
+ int     (*compat_setsockopt)(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int optlen);
+ int     (*compat_getsockopt)(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int __user *optlen);
+#endif
  void    (*addr2sockaddr)(struct sock *sk, struct sockaddr *);
  int sockaddr_len;
 };
--- ./include/net/ip.h.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./include/net/ip.h 2006-03-07 14:38:54.000000000 +0300
@@ -356,6 +356,12 @@ extern void ip_cmsg_recv(struct msghdr *
 extern int ip_cmsg_send(struct msghdr *msg, struct ipcm_cookie *ipc);
 extern int ip_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen);
 extern int ip_getsockopt(struct sock *sk, int level, int optname, char __user *optval, int __user
*optlen);
+#ifdef CONFIG_COMPAT
+extern int compat_ip_setsockopt(struct sock *sk, int level,
+   int optname, char __user *optval, int optlen);
+extern int compat_ip_getsockopt(struct sock *sk, int level,
+   int optname, char __user *optval, int __user *optlen);
+#endif
```

```
 extern int ip_ra_control(struct sock *sk, unsigned char on, void (*destructor)(struct sock *));

 extern int  ip_recv_error(struct sock *sk, struct msghdr *msg, int len);
--- ./include/net/sctp/structs.h.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./include/net/sctp/structs.h 2006-03-07 15:47:50.000000000 +0300
@@ -54,6 +54,7 @@
 #ifndef __sctp_structs_h__
 #define __sctp_structs_h__

+#include <linux/config.h>
 #include <linux/time.h>  /* We get struct timespec.    */
 #include <linux/socket.h> /* linux/in.h needs this!!    */
 #include <linux/in.h>  /* We get struct sockaddr_in. */
@@ -514,6 +515,18 @@ struct sctp_af {
        int optname,
        char __user *optval,
        int __user *optlen);
+#ifdef CONFIG_COMPAT
+ int  (*compat_setsockopt) (struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int optlen);
+ int  (*compat_getsockopt) (struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int __user *optlen);
+#endif
  struct dst_entry *(*get_dst) (struct sctp_association *asoc,
        union sctp_addr *daddr,
        union sctp_addr *saddr);
--- ./include/net/sock.h.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./include/net/sock.h 2006-03-07 15:56:13.000000000 +0300
@@ -520,6 +520,16 @@ struct proto {
  int   (*getsockopt)(struct sock *sk, int level,
      int optname, char __user *optval,
      int __user *option);
+#ifdef CONFIG_COMPAT
+ int   (*compat_setsockopt)(struct sock *sk,
+     int level,
+     int optname, char __user *optval,
+     int optlen);
+ int   (*compat_getsockopt)(struct sock *sk,
+     int level,
+     int optname, char __user *optval,
+     int __user *option);
+#endif
```

```
  int   (*sendmsg)(struct kiocb *iocb, struct sock *sk,
        struct msghdr *msg, size_t len);
  int   (*recvmsg)(struct kiocb *iocb, struct sock *sk,
@@ -816,6 +826,12 @@ extern int sock_common_recvmsg(struct ki
         struct msghdr *msg, size_t size, int flags);
 extern int sock_common_setsockopt(struct socket *sock, int level, int optname,
       char __user *optval, int optlen);
+#ifdef CONFIG_COMPAT
+extern int compat_sock_common_getsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int __user *optlen);
+extern int compat_sock_common_setsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int optlen);
+#endif

 extern void sk_common_release(struct sock *sk);

--- ./include/net/tcp.h.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./include/net/tcp.h 2006-03-07 15:43:43.000000000 +0300
@@ -347,6 +347,14 @@ extern int   tcp_getsockopt(struct sock
 extern int   tcp_setsockopt(struct sock *sk, int level,
            int optname, char __user *optval,
            int optlen);
+#ifdef CONFIG_COMPAT
+extern int   compat_tcp_getsockopt(struct sock *sk,
+     int level, int optname,
+     char __user *optval, int __user *optlen);
+extern int   compat_tcp_setsockopt(struct sock *sk,
+     int level, int optname,
+     char __user *optval, int optlen);
+#endif
 extern void   tcp_set_keepalive(struct sock *sk, int val);
 extern int   tcp_recvmsg(struct kiocb *iocb, struct sock *sk,
         struct msghdr *msg,
--- ./net/compat.c.compat 2006-03-07 11:21:00.000000000 +0300
+++ ./net/compat.c 2006-03-07 15:04:49.000000000 +0300
@@ -416,7 +416,7 @@ struct compat_sock_fprog {
  compat_uptr_t filter;  /* struct sock_filter * */
 };

-static int do_set_attach_filter(int fd, int level, int optname,
+static int do_set_attach_filter(struct socket *sock, int level, int optname,
     char __user *optval, int optlen)
 {
  struct compat_sock_fprog __user *fprog32 = (struct compat_sock_fprog __user *)optval;
@@ -432,11 +432,12 @@ static int do_set_attach_filter(int fd,
      __put_user(compat_ptr(ptr), &kfprog->filter))
   return -EFAULT;
```

```
- return sys_setsockopt(fd, level, optname, (char __user *)kfprog,
+ return sock_setsockopt(sock, level, optname, (char __user *)kfprog,
         sizeof(struct sock_fprog));
 }

-static int do_set_sock_timeout(int fd, int level, int optname, char __user *optval, int optlen)
+static int do_set_sock_timeout(struct socket *sock, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct compat_timeval __user *up = (struct compat_timeval __user *) optval;
  struct timeval ktime;
@@ -451,30 +452,61 @@ static int do_set_sock_timeout(int fd, i
   return -EFAULT;
  old_fs = get_fs();
  set_fs(KERNEL_DS);
- err = sys_setsockopt(fd, level, optname, (char *) &ktime, sizeof(ktime));
+ err = sock_setsockopt(sock, level, optname, (char *) &ktime, sizeof(ktime));
  set_fs(old_fs);

  return err;
 }

+static int compat_sock_setsockopt(struct socket *sock, int level, int optname,
+    char __user *optval, int optlen)
+{
+ if (optname == SO_ATTACH_FILTER)
+  return do_set_attach_filter(sock, level, optname,
+         optval, optlen);
+ if (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO)
+  return do_set_sock_timeout(sock, level, optname, optval, optlen);
+
+ return sock_setsockopt(sock, level, optname, optval, optlen);
+}
+
 asmlinkage long compat_sys_setsockopt(int fd, int level, int optname,
     char __user *optval, int optlen)
 {
+ int err;
+ struct socket *sock;
+
  /* SO_SET_REPLACE seems to be the same in all levels */
  if (optname == IPT_SO_SET_REPLACE)
   return do_netfilter_replace(fd, level, optname,
          optval, optlen);
- if (level == SOL_SOCKET && optname == SO_ATTACH_FILTER)
-  return do_set_attach_filter(fd, level, optname,
-         optval, optlen);
- if (level == SOL_SOCKET &&
```

```
-     (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO))
-  return do_set_sock_timeout(fd, level, optname, optval, optlen);

- return sys_setsockopt(fd, level, optname, optval, optlen);
+ if (optlen < 0)
+  return -EINVAL;
+
+ if ((sock = sockfd_lookup(fd, &err))!=NULL)
+ {
+  err = security_socket_setsockopt(sock,level,optname);
+  if (err) {
+   sockfd_put(sock);
+   return err;
+  }
+
+  if (level == SOL_SOCKET)
+   err = compat_sock_setsockopt(sock, level,
+     optname, optval, optlen);
+  else if (sock->ops->compat_setsockopt)
+   err = sock->ops->compat_setsockopt(sock, level,
+     optname, optval, optlen);
+  else
+   err = sock->ops->setsockopt(sock, level,
+     optname, optval, optlen);
+  sockfd_put(sock);
+ }
+ return err;
 }

-static int do_get_sock_timeout(int fd, int level, int optname,
+static int do_get_sock_timeout(struct socket *sock, int level, int optname,
    char __user *optval, int __user *optlen)
 {
  struct compat_timeval __user *up;
@@ -490,7 +522,7 @@ static int do_get_sock_timeout(int fd, i
  len = sizeof(ktime);
  old_fs = get_fs();
  set_fs(KERNEL_DS);
- err = sys_getsockopt(fd, level, optname, (char *) &ktime, &len);
+ err = sock_getsockopt(sock, level, optname, (char *) &ktime, &len);
  set_fs(old_fs);

  if (!err) {
@@ -503,15 +535,42 @@ static int do_get_sock_timeout(int fd, i
  return err;
 }

-asmlinkage long compat_sys_getsockopt(int fd, int level, int optname,
```

```
+static int compat_sock_getsockopt(struct socket *sock, int level, int optname,
     char __user *optval, int __user *optlen)
 {
- if (level == SOL_SOCKET &&
-     (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO))
-  return do_get_sock_timeout(fd, level, optname, optval, optlen);
- return sys_getsockopt(fd, level, optname, optval, optlen);
+ if (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO)
+  return do_get_sock_timeout(sock, level, optname, optval, optlen);
+ return sock_getsockopt(sock, level, optname, optval, optlen);
 }

+asmlinkage long compat_sys_getsockopt(int fd, int level, int optname,
+    char __user *optval, int __user *optlen)
+{
+ int err;
+ struct socket *sock;
+
+ if ((sock = sockfd_lookup(fd, &err))!=NULL)
+ {
+  err = security_socket_getsockopt(sock, level,
+          optname);
+  if (err) {
+   sockfd_put(sock);
+   return err;
+  }
+
+  if (level == SOL_SOCKET)
+   err = compat_sock_getsockopt(sock, level,
+     optname, optval, optlen);
+  else if (sock->ops->compat_getsockopt)
+   err = sock->ops->compat_getsockopt(sock, level,
+     optname, optval, optlen);
+  else
+   err = sock->ops->getsockopt(sock, level,
+     optname, optval, optlen);
+  sockfd_put(sock);
+ }
+ return err;
+}
 /* Argument list sizes for compat_sys_socketcall */
 #define AL(x) ((x) * sizeof(u32))
 static unsigned char nas[18]={AL(0),AL(3),AL(3),AL(3),AL(2),AL(3),
--- ./net/core/sock.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/core/sock.c 2006-03-07 16:24:57.000000000 +0300
@@ -1385,6 +1385,20 @@ int sock_common_getsockopt(struct socket

 EXPORT_SYMBOL(sock_common_getsockopt);
```

```
+#ifdef CONFIG_COMPAT
+int compat_sock_common_getsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_prot->compat_setsockopt)
+  return sk->sk_prot->compat_getsockopt(sk, level,
+   optname, optval, optlen);
+ return sk->sk_prot->getsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL(compat_sock_common_getsockopt);
+#endif
+
 int sock_common_recvmsg(struct kiocb *iocb, struct socket *sock,
    struct msghdr *msg, size_t size, int flags)
 {
@@ -1414,6 +1428,20 @@ int sock_common_setsockopt(struct socket

 EXPORT_SYMBOL(sock_common_setsockopt);

+#ifdef CONFIG_COMPAT
+int compat_sock_common_setsockopt(struct socket *sock,
+  int level, int optname, char __user *optval, int optlen)
+{
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_prot->compat_setsockopt)
+  return sk->sk_prot->compat_setsockopt(sk, level,
+   optname, optval, optlen);
+ return sk->sk_prot->setsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL(compat_sock_common_setsockopt);
+#endif
+
 void sk_common_release(struct sock *sk)
 {
  if (sk->sk_prot->destroy)
--- ./net/dccp/dccp.h.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/dccp/dccp.h 2006-03-07 15:24:36.000000000 +0300
@@ -246,6 +246,14 @@ extern int    dccp_getsockopt(struct soc
        char __user *optval, int __user *optlen);
 extern int    dccp_setsockopt(struct sock *sk, int level, int optname,
        char __user *optval, int optlen);
+#ifdef CONFIG_COMPAT
+extern int    compat_dccp_getsockopt(struct sock *sk,
+    int level, int optname,
```

```
+    char __user *optval, int __user *optlen);
+extern int    compat_dccp_setsockopt(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int optlen);
+#endif
 extern int    dccp_ioctl(struct sock *sk, int cmd, unsigned long arg);
 extern int    dccp_sendmsg(struct kiocb *iocb, struct sock *sk,
     struct msghdr *msg, size_t size);
--- ./net/dccp/ipv4.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/dccp/ipv4.c 2006-03-07 15:24:42.000000000 +0300
@@ -1028,6 +1028,10 @@ struct inet_connection_sock_af_ops dccp_
  .net_header_len = sizeof(struct iphdr),
  .setsockopt = ip_setsockopt,
  .getsockopt = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ip_setsockopt,
+ .compat_getsockopt = compat_ip_getsockopt,
+#endif
  .addr2sockaddr = inet_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in),
 };
@@ -1152,6 +1156,10 @@ struct proto dccp_prot = {
  .init   = dccp_v4_init_sock,
  .setsockopt  = dccp_setsockopt,
  .getsockopt  = dccp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_dccp_setsockopt,
+ .compat_getsockopt = compat_dccp_getsockopt,
+#endif
  .sendmsg  = dccp_sendmsg,
  .recvmsg  = dccp_recvmsg,
  .backlog_rcv  = dccp_v4_do_rcv,
--- ./net/dccp/ipv6.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/dccp/ipv6.c 2006-03-07 15:57:23.000000000 +0300
@@ -1170,6 +1170,10 @@ static struct proto dccp_v6_prot = {
  .init   = dccp_v6_init_sock,
  .setsockopt  = dccp_setsockopt,
  .getsockopt  = dccp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_dccp_setsockopt,
+ .compat_getsockopt = compat_dccp_getsockopt,
+#endif
  .sendmsg  = dccp_sendmsg,
  .recvmsg  = dccp_recvmsg,
  .backlog_rcv  = dccp_v6_do_rcv,
@@ -1207,6 +1211,10 @@ static struct proto_ops inet6_dccp_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
```

```
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap  = sock_no_mmap,
--- ./net/dccp/proto.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/dccp/proto.c 2006-03-07 16:49:11.000000000 +0300
@@ -255,18 +255,13 @@ static int dccp_setsockopt_service(struc
  return 0;
 }

-int dccp_setsockopt(struct sock *sk, int level, int optname,
-      char __user *optval, int optlen)
+static int do_dccp_setsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int optlen)
 {
  struct dccp_sock *dp;
  int err;
  int val;

- if (level != SOL_DCCP)
-  return inet_csk(sk)->icsk_af_ops->setsockopt(sk, level,
-            optname, optval,
-            optlen);
-
  if (optlen < sizeof(int))
   return -EINVAL;

@@ -293,8 +288,34 @@ int dccp_setsockopt(struct sock *sk, int
  return err;
 }

+int dccp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ if (level != SOL_DCCP)
+  return inet_csk(sk)->icsk_af_ops->setsockopt(sk, level,
+            optname, optval,
+            optlen);
+ return do_dccp_setsockopt(sk, level, optname, optval, optlen);
+}
 EXPORT_SYMBOL_GPL(dccp_setsockopt);

+#ifdef CONFIG_COMPAT
+int compat_dccp_setsockopt(struct sock *sk, int level, int optname,
```

```
+      char __user *optval, int optlen)
+{
+ if (level != SOL_DCCP) {
+  if (inet_csk(sk)->icsk_af_ops->compat_setsockopt)
+   return inet_csk(sk)->icsk_af_ops->compat_setsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return inet_csk(sk)->icsk_af_ops->setsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_dccp_setsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL_GPL(compat_dccp_setsockopt);
+#endif
+
 static int dccp_getsockopt_service(struct sock *sk, int len,
        u32 __user *optval,
        int __user *optlen)
@@ -326,16 +347,12 @@ out:
  return err;
 }

-int dccp_getsockopt(struct sock *sk, int level, int optname,
+static int do_dccp_getsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen)
 {
  struct dccp_sock *dp;
  int val, len;

- if (level != SOL_DCCP)
-  return inet_csk(sk)->icsk_af_ops->getsockopt(sk, level,
-            optname, optval,
-            optlen);
  if (get_user(len, optlen))
   return -EFAULT;

@@ -368,8 +385,34 @@ int dccp_getsockopt(struct sock *sk, int
  return 0;
 }

+int dccp_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ if (level != SOL_DCCP)
+  return inet_csk(sk)->icsk_af_ops->getsockopt(sk, level,
+            optname, optval,
+            optlen);
+ return do_dccp_getsockopt(sk, level, optname, optval, optlen);
```

```
+}
 EXPORT_SYMBOL_GPL(dccp_getsockopt);

+#ifdef CONFIG_COMPAT
+int compat_dccp_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ if (level != SOL_DCCP) {
+  if (inet_csk(sk)->icsk_af_ops->compat_setsockopt)
+   return inet_csk(sk)->icsk_af_ops->compat_getsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return inet_csk(sk)->icsk_af_ops->getsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_dccp_getsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL_GPL(compat_dccp_getsockopt);
+#endif
+
 int dccp_sendmsg(struct kiocb *iocb, struct sock *sk, struct msghdr *msg,
    size_t len)
 {
@@ -696,6 +739,10 @@ static const struct proto_ops inet_dccp_
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap  = sock_no_mmap,
--- ./net/ipv4/af_inet.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv4/af_inet.c 2006-03-07 16:25:58.000000000 +0300
@@ -802,6 +802,10 @@ const struct proto_ops inet_stream_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
@@ -823,6 +827,10 @@ const struct proto_ops inet_dgram_ops =
```

```
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
@@ -848,6 +856,10 @@ static const struct proto_ops inet_sockr
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
--- ./net/ipv4/ip_sockglue.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv4/ip_sockglue.c 2006-03-07 14:41:47.000000000 +0300
@@ -380,14 +380,12 @@ out:
  * an IP socket.
  */

-int ip_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen)
+static int do_ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct inet_sock *inet = inet_sk(sk);
  int val=0,err;

- if (level != SOL_IP)
-  return -ENOPROTOOPT;
-
  if (((1<<optname) & ((1<<IP_PKTINFO) | (1<<IP_RECVTTL) |
        (1<<IP_RECVOPTS) | (1<<IP_RECVTOS) |
        (1<<IP_RETOPTS) | (1<<IP_TOS) |
@@ -849,12 +847,7 @@ mc_msf_out:
    break;

   default:
-#ifdef CONFIG_NETFILTER
-   err = nf_setsockopt(sk, PF_INET, optname, optval,
-         optlen);
-#else
```

```
    err = -ENOPROTOOPT;
-#endif
    break;
  }
  release_sock(sk);
@@ -865,12 +858,66 @@ e_inval:
  return -EINVAL;
 }

+int ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ int err;
+
+ if (level != SOL_IP)
+  return -ENOPROTOOPT;
+
+ err = do_ip_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_HDRINCL &&
+  optname != IP_IPSEC_POLICY && optname != IP_XFRM_POLICY
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > (MRT_BASE + 10))
+#endif
+    ) {
+  lock_sock(sk);
+  err = nf_setsockopt(sk, PF_INET, optname, optval, optlen);
+  release_sock(sk);
+ }
+#endif
+ return err;
+}
+
+#ifdef CONFIG_COMPAT
+int compat_ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ int err;
+
+ if (level != SOL_IP)
+  return -ENOPROTOOPT;
+
+ err = do_ip_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_HDRINCL &&
+  optname != IP_IPSEC_POLICY && optname != IP_XFRM_POLICY
```

```
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > (MRT_BASE + 10))
+#endif
+    ) {
+  lock_sock(sk);
+  err = compat_nf_setsockopt(sk, PF_INET,
+    optname, optval, optlen);
+  release_sock(sk);
+ }
+#endif
+ return err;
+}
+#endif
+
 /*
  * Get the options. Note for future reference. The GET of IP options gets the
  * _received_ ones. The set sets the _sent_ ones.
  */

-int ip_getsockopt(struct sock *sk, int level, int optname, char __user *optval, int __user *optlen)
+static int do_ip_getsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int __user *optlen)
 {
  struct inet_sock *inet = inet_sk(sk);
  int val;
@@ -1051,17 +1098,8 @@ int ip_getsockopt(struct sock *sk, int l
    val = inet->freebind;
    break;
   default:
-#ifdef CONFIG_NETFILTER
-   val = nf_getsockopt(sk, PF_INET, optname, optval,
-         &len);
-   release_sock(sk);
-   if (val >= 0)
-    val = put_user(len, optlen);
-   return val;
-#else
    release_sock(sk);
    return -ENOPROTOOPT;
-#endif
  }
  release_sock(sk);

@@ -1082,7 +1120,73 @@ int ip_getsockopt(struct sock *sk, int l
  return 0;
 }

+int ip_getsockopt(struct sock *sk, int level,
```

```
+  int optname, char __user *optval, int __user *optlen)
+{
+ int err;
+
+ err = do_ip_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_PKTOPTIONS
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > MRT_BASE+10)
+#endif
+    ) {
+     int len;
+
+  if(get_user(len,optlen))
+   return -EFAULT;
+
+  lock_sock(sk);
+  err = nf_getsockopt(sk, PF_INET, optname, optval,
+    &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+  return err;
+ }
+#endif
+ return err;
+}
+
+#ifdef CONFIG_COMPAT
+int compat_ip_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ int err;
+
+ err = do_ip_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_PKTOPTIONS
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > MRT_BASE+10)
+#endif
+    ) {
+     int len;
+
+  if(get_user(len,optlen))
+   return -EFAULT;
+
```

```
+  lock_sock(sk);
+  err = compat_nf_getsockopt(sk, PF_INET,
+    optname, optval, &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+  return err;
+ }
+#endif
+ return err;
+}
+#endif
+
 EXPORT_SYMBOL(ip_cmsg_recv);

 EXPORT_SYMBOL(ip_getsockopt);
 EXPORT_SYMBOL(ip_setsockopt);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(compat_ip_getsockopt);
+EXPORT_SYMBOL(compat_ip_setsockopt);
+#endif
--- ./net/ipv4/raw.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv4/raw.c 2006-03-07 15:34:44.000000000 +0300
@@ -660,12 +660,9 @@ static int raw_geticmpfilter(struct sock
 out: return ret;
 }

-static int raw_setsockopt(struct sock *sk, int level, int optname,
+static int do_raw_setsockopt(struct sock *sk, int level, int optname,
      char __user *optval, int optlen)
 {
- if (level != SOL_RAW)
-  return ip_setsockopt(sk, level, optname, optval, optlen);
-
  if (optname == ICMP_FILTER) {
   if (inet_sk(sk)->num != IPPROTO_ICMP)
    return -EOPNOTSUPP;
@@ -675,12 +672,28 @@ static int raw_setsockopt(struct sock *s
  return -ENOPROTOOPT;
 }

-static int raw_getsockopt(struct sock *sk, int level, int optname,
-     char __user *optval, int __user *optlen)
+static int raw_setsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int optlen)
 {
  if (level != SOL_RAW)
-  return ip_getsockopt(sk, level, optname, optval, optlen);
```

```
+  return ip_setsockopt(sk, level, optname, optval, optlen);
+ return do_raw_setsockopt(sk, level, optname, optval, optlen);
+}

+#ifdef CONFIG_COMPAT
+static int compat_raw_setsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int optlen)
+{
+ if (level != SOL_RAW)
+  return compat_ip_setsockopt(sk, level,
+    optname, optval, optlen);
+ return do_raw_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
+static int do_raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
  if (optname == ICMP_FILTER) {
   if (inet_sk(sk)->num != IPPROTO_ICMP)
    return -EOPNOTSUPP;
@@ -690,6 +703,25 @@ static int raw_getsockopt(struct sock *s
  return -ENOPROTOOPT;
 }

+static int raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_RAW)
+  return ip_getsockopt(sk, level, optname, optval, optlen);
+ return do_raw_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_RAW)
+  return compat_ip_getsockopt(sk, level,
+    optname, optval, optlen);
+ return do_raw_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 static int raw_ioctl(struct sock *sk, int cmd, unsigned long arg)
 {
  switch (cmd) {
@@ -728,6 +760,10 @@ struct proto raw_prot = {
```

```
  .init =  raw_init,
  .setsockopt = raw_setsockopt,
  .getsockopt = raw_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_raw_setsockopt,
+ .compat_getsockopt = compat_raw_getsockopt,
+#endif
  .sendmsg = raw_sendmsg,
  .recvmsg = raw_recvmsg,
  .bind =  raw_bind,
--- ./net/ipv4/tcp.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv4/tcp.c 2006-03-07 16:36:21.000000000 +0300
@@ -1687,18 +1687,14 @@ int tcp_disconnect(struct sock *sk, int
 /*
  * Socket option code for TCP.
  */
-int tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval,
-     int optlen)
+static int do_tcp_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct tcp_sock *tp = tcp_sk(sk);
  struct inet_connection_sock *icsk = inet_csk(sk);
  int val;
  int err = 0;

- if (level != SOL_TCP)
-  return icsk->icsk_af_ops->setsockopt(sk, level, optname,
-           optval, optlen);
-
  /* This is a string value all the others are int's */
  if (optname == TCP_CONGESTION) {
   char name[TCP_CA_NAME_MAX];
@@ -1871,6 +1867,35 @@ int tcp_setsockopt(struct sock *sk, int
  return err;
 }

+int tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval,
+     int optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP)
+  return icsk->icsk_af_ops->setsockopt(sk, level, optname,
+           optval, optlen);
+ return do_tcp_setsockopt(sk, level, optname, optval, optlen);
+}
+
```

```
+#ifdef CONFIG_COMPAT
+int compat_tcp_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP) {
+  if (icsk->icsk_af_ops->compat_setsockopt)
+   return icsk->icsk_af_ops->compat_setsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return icsk->icsk_af_ops->setsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_tcp_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 /* Return information about state of tcp endpoint in API format. */
 void tcp_get_info(struct sock *sk, struct tcp_info *info)
 {
@@ -1931,17 +1956,13 @@ void tcp_get_info(struct sock *sk, struc

 EXPORT_SYMBOL_GPL(tcp_get_info);

-int tcp_getsockopt(struct sock *sk, int level, int optname, char __user *optval,
-     int __user *optlen)
+static int do_tcp_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
 {
  struct inet_connection_sock *icsk = inet_csk(sk);
  struct tcp_sock *tp = tcp_sk(sk);
  int val, len;

- if (level != SOL_TCP)
-  return icsk->icsk_af_ops->getsockopt(sk, level, optname,
-           optval, optlen);
-
  if (get_user(len, optlen))
   return -EFAULT;

@@ -2025,6 +2046,34 @@ int tcp_getsockopt(struct sock *sk, int
  return 0;
 }

+int tcp_getsockopt(struct sock *sk, int level, int optname, char __user *optval,
+     int __user *optlen)
+{
```

```
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP)
+  return icsk->icsk_af_ops->getsockopt(sk, level, optname,
+           optval, optlen);
+ return do_tcp_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+int compat_tcp_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP) {
+  if (icsk->icsk_af_ops->compat_getsockopt)
+   return icsk->icsk_af_ops->compat_getsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return icsk->icsk_af_ops->getsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_tcp_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif

 extern void __skb_cb_too_small_for_tcp(int, int);
 extern struct tcp_congestion_ops tcp_reno;
@@ -2142,3 +2191,7 @@ EXPORT_SYMBOL(tcp_sendpage);
 EXPORT_SYMBOL(tcp_setsockopt);
 EXPORT_SYMBOL(tcp_shutdown);
 EXPORT_SYMBOL(tcp_statistics);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(compat_tcp_setsockopt);
+EXPORT_SYMBOL(compat_tcp_getsockopt);
+#endif
--- ./net/ipv4/tcp_ipv4.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv4/tcp_ipv4.c 2006-03-07 15:46:22.000000000 +0300
@@ -1225,6 +1225,10 @@ struct inet_connection_sock_af_ops ipv4_
  .net_header_len = sizeof(struct iphdr),
  .setsockopt = ip_setsockopt,
  .getsockopt = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ip_setsockopt,
+ .compat_getsockopt = compat_ip_getsockopt,
+#endif
  .addr2sockaddr = inet_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in),
```

```
 };
@@ -1807,6 +1811,10 @@ struct proto tcp_prot = {
  .shutdown  = tcp_shutdown,
  .setsockopt  = tcp_setsockopt,
  .getsockopt  = tcp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_tcp_setsockopt,
+ .compat_getsockopt = compat_tcp_getsockopt,
+#endif
  .sendmsg  = tcp_sendmsg,
  .recvmsg  = tcp_recvmsg,
  .backlog_rcv  = tcp_v4_do_rcv,
--- ./net/ipv4/udp.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv4/udp.c 2006-03-07 12:38:44.000000000 +0300
@@ -1207,16 +1207,13 @@ static int udp_destroy_sock(struct sock
 /*
  * Socket option code for UDP
  */
-static int udp_setsockopt(struct sock *sk, int level, int optname,
+static int do_udp_setsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val;
  int err = 0;

- if (level != SOL_UDP)
-  return ip_setsockopt(sk, level, optname, optval, optlen);
-
  if(optlen<sizeof(int))
   return -EINVAL;

@@ -1256,15 +1253,31 @@ static int udp_setsockopt(struct sock *s
  return err;
 }

-static int udp_getsockopt(struct sock *sk, int level, int optname,
+static int udp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ if (level != SOL_UDP)
+  return ip_setsockopt(sk, level, optname, optval, optlen);
+ return do_udp_setsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_udp_setsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int optlen)
```

```
+{
+ if (level != SOL_UDP)
+  return compat_ip_setsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udp_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
+static int do_udp_getsockopt(struct sock *sk, int level, int optname,
      char __user *optval, int __user *optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val, len;

- if (level != SOL_UDP)
-  return ip_getsockopt(sk, level, optname, optval, optlen);
-
  if(get_user(len,optlen))
   return -EFAULT;

@@ -1293,6 +1306,24 @@ static int udp_getsockopt(struct sock *s
    return 0;
 }

+static int udp_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return ip_getsockopt(sk, level, optname, optval, optlen);
+ return do_udp_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_udp_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return compat_ip_getsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udp_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
 /**
  *  udp_poll - wait for a UDP event.
  * @file - file struct
@@ -1350,6 +1381,10 @@ struct proto udp_prot = {
  .destroy = udp_destroy_sock,
  .setsockopt = udp_setsockopt,
```

```
  .getsockopt = udp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_udp_setsockopt,
+ .compat_getsockopt = compat_udp_getsockopt,
+#endif
  .sendmsg = udp_sendmsg,
  .recvmsg = udp_recvmsg,
  .sendpage = udp_sendpage,
--- ./net/ipv6/af_inet6.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv6/af_inet6.c 2006-03-07 16:26:21.000000000 +0300
@@ -470,6 +470,10 @@ const struct proto_ops inet6_stream_ops
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
@@ -491,6 +495,10 @@ const struct proto_ops inet6_dgram_ops =
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
@@ -519,6 +527,10 @@ static const struct proto_ops inet6_sock
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
--- ./net/ipv6/tcp_ipv6.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/ipv6/tcp_ipv6.c 2006-03-07 15:42:11.000000000 +0300
@@ -1565,6 +1565,10 @@ struct proto tcpv6_prot = {
  .shutdown  = tcp_shutdown,
  .setsockopt  = tcp_setsockopt,
```

```
  .getsockopt  = tcp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_tcp_setsockopt,
+ .compat_getsockopt = compat_tcp_getsockopt,
+#endif
  .sendmsg  = tcp_sendmsg,
  .recvmsg  = tcp_recvmsg,
  .backlog_rcv  = tcp_v6_do_rcv,
--- ./net/netfilter/nf_sockopt.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/netfilter/nf_sockopt.c 2006-03-07 14:58:57.000000000 +0300
@@ -130,3 +130,72 @@ int nf_getsockopt(struct sock *sk, int p
 }
 EXPORT_SYMBOL(nf_getsockopt);

+#ifdef CONFIG_COMPAT
+static int compat_nf_sockopt(struct sock *sk, int pf, int val,
+        char __user *opt, int *len, int get)
+{
+ struct list_head *i;
+ struct nf_sockopt_ops *ops;
+ int ret;
+
+ if (down_interruptible(&nf_sockopt_mutex) != 0)
+  return -EINTR;
+
+ list_for_each(i, &nf_sockopts) {
+  ops = (struct nf_sockopt_ops *)i;
+  if (ops->pf == pf) {
+   if (get) {
+    if (val >= ops->get_optmin
+        && val < ops->get_optmax) {
+     ops->use++;
+     up(&nf_sockopt_mutex);
+     if (ops->compat_get)
+      ret = ops->compat_get(sk,
+       val, opt, len);
+     else
+      ret = ops->get(sk,
+       val, opt, len);
+     goto out;
+    }
+   } else {
+    if (val >= ops->set_optmin
+        && val < ops->set_optmax) {
+     ops->use++;
+     up(&nf_sockopt_mutex);
+     if (ops->compat_set)
+      ret = ops->compat_set(sk,
```

```
+       val, opt, *len);
+     else
+      ret = ops->set(sk,
+       val, opt, *len);
+     goto out;
+    }
+   }
+  }
+ }
+ up(&nf_sockopt_mutex);
+ return -ENOPROTOOPT;
+
+ out:
+ down(&nf_sockopt_mutex);
+ ops->use--;
+ if (ops->cleanup_task)
+  wake_up_process(ops->cleanup_task);
+ up(&nf_sockopt_mutex);
+ return ret;
+}
+
+int compat_nf_setsockopt(struct sock *sk, int pf,
+  int val, char __user *opt, int len)
+{
+ return compat_nf_sockopt(sk, pf, val, opt, &len, 0);
+}
+EXPORT_SYMBOL(compat_nf_setsockopt);
+
+int compat_nf_getsockopt(struct sock *sk, int pf,
+  int val, char __user *opt, int *len)
+{
+ return compat_nf_sockopt(sk, pf, val, opt, len, 1);
+}
+EXPORT_SYMBOL(compat_nf_getsockopt);
+#endif
--- ./net/sctp/ipv6.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/sctp/ipv6.c 2006-03-07 16:26:56.000000000 +0300
@@ -875,6 +875,10 @@ static const struct proto_ops inet6_seqp
  .shutdown   = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg    = inet_sendmsg,
  .recvmsg    = sock_common_recvmsg,
  .mmap       = sock_no_mmap,
```

```
--- ./net/sctp/protocol.c.compat 2006-03-06 12:06:34.000000000 +0300
+++ ./net/sctp/protocol.c 2006-03-07 15:51:18.000000000 +0300
@@ -845,6 +845,10 @@ static const struct proto_ops inet_seqpa
  .shutdown    = inet_shutdown,     /* Looks harmless.  */
  .setsockopt  = sock_common_setsockopt,   /* IP_SOL IP_OPTION is a problem. */
  .getsockopt  = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt  = compat_sock_common_setsockopt,
+ .compat_getsockopt  = compat_sock_common_getsockopt,
+#endif
  .sendmsg     = inet_sendmsg,
  .recvmsg     = sock_common_recvmsg,
  .mmap        = sock_no_mmap,
@@ -883,6 +887,10 @@ static struct sctp_af sctp_ipv4_specific
  .sctp_xmit      = sctp_v4_xmit,
  .setsockopt     = ip_setsockopt,
  .getsockopt     = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt     = compat_ip_setsockopt,
+ .compat_getsockopt     = compat_ip_getsockopt,
+#endif
  .get_dst = sctp_v4_get_dst,
  .get_saddr = sctp_v4_get_saddr,
  .copy_addrlist  = sctp_v4_copy_addrlist,
```

File Attachments

1) diff-ms-sockopts-compat-20060307, downloaded 702 times

Subject: Re: {get|set}sockopt compat layer
Posted by Arnd Bergmann on Tue, 07 Mar 2006 15:05:38 GMT
View Forum Message <> Reply to Message

On Tuesday 07 March 2006 15:07, Dmitry Mishin wrote:
> Sorry for such delay, was on vacancy. Here is a patch, introducing
> compat_(get|set)sockopt handlers, as you proposed.

Looks pretty good to me, just a few nits I like to pick:

> @@ -149,6 +149,12 @@ struct proto_ops {

> +#ifdef CONFIG_COMPAT

> +#endif

For the compat_ioctl stuff, we don't have the function pointer inside an #ifdef, the overhead is relatively small since there is only one of these structures per module implementing a protocol, but it avoids having to rebuild everything when changing CONFIG_COMPAT.

It's probably not a big issue either way, maybe davem has a stronger opinion on it either way.

> @@ -2,6 +2,7 @@

> +#include <linux/config.h>

You don't need to add new <linux/config.h> includes any more, these are automatic now.

> @@ -80,10 +81,18 @@ struct nf_sockopt_ops

> +#ifdef CONFIG_COMPAT

> +#endif

> +#ifdef CONFIG_COMPAT

> +#endif

see above, same for some more of these.

> @@ -816,6 +826,12 @@ extern int sock_common_recvmsg(struct ki

> +#ifdef CONFIG_COMPAT
> +extern int compat_sock_common_getsockopt(struct socket *sock, int level,

> +extern int compat_sock_common_setsockopt(struct socket *sock, int level,

> +#endif

Declarations don't belong inside #ifdef.

Arnd <><

Subject: Re: {get|set}sockopt compat layer
Posted by Mishin Dmitry on Thu, 09 Mar 2006 10:23:59 GMT
View Forum Message <> Reply to Message

Hello, Arnd!

> For the compat_ioctl stuff, we don't have the function pointer inside an
> #ifdef, the overhead is relatively small since there is only one of these
> structures per module implementing a protocol, but it avoids having to
> rebuild everything when changing CONFIG_COMPAT.
>
> It's probably not a big issue either way, maybe davem has a stronger
> opinion on it either way.
>
Done.

--
Thanks,
Dmitry.

```
--- ./include/linux/net.h.compat 2006-03-09 12:57:53.000000000 +0300
+++ ./include/linux/net.h 2006-03-09 12:58:53.000000000 +0300
@@ -149,6 +149,10 @@ struct proto_ops {
         int optname, char __user *optval, int optlen);
  int  (*getsockopt)(struct socket *sock, int level,
         int optname, char __user *optval, int __user *optlen);
```

```
+ int  (*compat_setsockopt)(struct socket *sock, int level,
+          int optname, char __user *optval, int optlen);
+ int  (*compat_getsockopt)(struct socket *sock, int level,
+          int optname, char __user *optval, int __user *optlen);
  int  (*sendmsg)   (struct kiocb *iocb, struct socket *sock,
          struct msghdr *m, size_t total_len);
  int  (*recvmsg)   (struct kiocb *iocb, struct socket *sock,
--- ./include/linux/netfilter.h.compat 2006-03-09 12:57:53.000000000 +0300
+++ ./include/linux/netfilter.h 2006-03-09 12:59:44.000000000 +0300
@@ -80,10 +80,14 @@ struct nf_sockopt_ops
  int set_optmin;
  int set_optmax;
  int (*set)(struct sock *sk, int optval, void __user *user, unsigned int len);
+ int (*compat_set)(struct sock *sk, int optval,
+   void __user *user, unsigned int len);

  int get_optmin;
  int get_optmax;
  int (*get)(struct sock *sk, int optval, void __user *user, int *len);
+ int (*compat_get)(struct sock *sk, int optval,
+   void __user *user, int *len);

  /* Number of users inside set() or get(). */
  unsigned int use;
@@ -246,6 +250,11 @@ int nf_setsockopt(struct sock *sk, int p
 int nf_getsockopt(struct sock *sk, int pf, int optval, char __user *opt,
     int *len);

+int compat_nf_setsockopt(struct sock *sk, int pf, int optval,
+  char __user *opt, int len);
+int compat_nf_getsockopt(struct sock *sk, int pf, int optval,
+  char __user *opt, int *len);
+
 /* Packet queuing */
 struct nf_queue_handler {
  int (*outfn)(struct sk_buff *skb, struct nf_info *info,
--- ./include/net/inet_connection_sock.h.compat 2006-03-09 12:57:53.000000000 +0300
+++ ./include/net/inet_connection_sock.h 2006-03-09 12:59:58.000000000 +0300
@@ -50,6 +50,12 @@ struct inet_connection_sock_af_ops {
       char __user *optval, int optlen);
  int     (*getsockopt)(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen);
+ int     (*compat_setsockopt)(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int optlen);
+ int     (*compat_getsockopt)(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int __user *optlen);
```

```
  void     (*addr2sockaddr)(struct sock *sk, struct sockaddr *);
  int sockaddr_len;
 };
--- ./include/net/ip.h.compat 2006-03-09 12:57:53.000000000 +0300
+++ ./include/net/ip.h 2006-03-09 13:00:15.000000000 +0300
@@ -356,6 +356,10 @@ extern void ip_cmsg_recv(struct msghdr *
 extern int ip_cmsg_send(struct msghdr *msg, struct ipcm_cookie *ipc);
 extern int ip_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen);
 extern int ip_getsockopt(struct sock *sk, int level, int optname, char __user *optval, int __user
*optlen);
+extern int compat_ip_setsockopt(struct sock *sk, int level,
+   int optname, char __user *optval, int optlen);
+extern int compat_ip_getsockopt(struct sock *sk, int level,
+   int optname, char __user *optval, int __user *optlen);
 extern int ip_ra_control(struct sock *sk, unsigned char on, void (*destructor)(struct sock *));

 extern int  ip_recv_error(struct sock *sk, struct msghdr *msg, int len);
--- ./include/net/sctp/structs.h.compat 2006-03-09 12:57:53.000000000 +0300
+++ ./include/net/sctp/structs.h 2006-03-09 13:00:36.000000000 +0300
@@ -514,6 +514,16 @@ struct sctp_af {
       int optname,
       char __user *optval,
       int __user *optlen);
+ int  (*compat_setsockopt) (struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int optlen);
+ int  (*compat_getsockopt) (struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int __user *optlen);
  struct dst_entry *(*get_dst) (struct sctp_association *asoc,
       union sctp_addr *daddr,
       union sctp_addr *saddr);
--- ./include/net/sock.h.compat 2006-03-09 12:57:53.000000000 +0300
+++ ./include/net/sock.h 2006-03-09 13:01:10.000000000 +0300
@@ -520,6 +520,14 @@ struct proto {
  int   (*getsockopt)(struct sock *sk, int level,
      int optname, char __user *optval,
      int __user *option);
+ int   (*compat_setsockopt)(struct sock *sk,
+     int level,
+     int optname, char __user *optval,
+     int optlen);
+ int   (*compat_getsockopt)(struct sock *sk,
+     int level,
```

```
+     int optname, char __user *optval,
+     int __user *option);
  int   (*sendmsg)(struct kiocb *iocb, struct sock *sk,
         struct msghdr *msg, size_t len);
  int   (*recvmsg)(struct kiocb *iocb, struct sock *sk,
@@ -816,6 +824,10 @@ extern int sock_common_recvmsg(struct ki
           struct msghdr *msg, size_t size, int flags);
 extern int sock_common_setsockopt(struct socket *sock, int level, int optname,
       char __user *optval, int optlen);
+extern int compat_sock_common_getsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int __user *optlen);
+extern int compat_sock_common_setsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int optlen);

 extern void sk_common_release(struct sock *sk);

--- ./include/net/tcp.h.compat 2006-03-09 12:57:53.000000000 +0300
+++ ./include/net/tcp.h 2006-03-09 13:02:50.000000000 +0300
@@ -347,6 +347,12 @@ extern int   tcp_getsockopt(struct sock
 extern int   tcp_setsockopt(struct sock *sk, int level,
            int optname, char __user *optval,
            int optlen);
+extern int   compat_tcp_getsockopt(struct sock *sk,
+     int level, int optname,
+     char __user *optval, int __user *optlen);
+extern int   compat_tcp_setsockopt(struct sock *sk,
+     int level, int optname,
+     char __user *optval, int optlen);
 extern void   tcp_set_keepalive(struct sock *sk, int val);
 extern int   tcp_recvmsg(struct kiocb *iocb, struct sock *sk,
         struct msghdr *msg,
--- ./net/compat.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/compat.c 2006-03-09 12:58:23.000000000 +0300
@@ -416,7 +416,7 @@ struct compat_sock_fprog {
  compat_uptr_t filter;  /* struct sock_filter * */
 };

-static int do_set_attach_filter(int fd, int level, int optname,
+static int do_set_attach_filter(struct socket *sock, int level, int optname,
     char __user *optval, int optlen)
 {
  struct compat_sock_fprog __user *fprog32 = (struct compat_sock_fprog __user *)optval;
@@ -432,11 +432,12 @@ static int do_set_attach_filter(int fd,
      __put_user(compat_ptr(ptr), &kfprog->filter))
   return -EFAULT;

- return sys_setsockopt(fd, level, optname, (char __user *)kfprog,
+ return sock_setsockopt(sock, level, optname, (char __user *)kfprog,
```

```
         sizeof(struct sock_fprog));
 }

-static int do_set_sock_timeout(int fd, int level, int optname, char __user *optval, int optlen)
+static int do_set_sock_timeout(struct socket *sock, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct compat_timeval __user *up = (struct compat_timeval __user *) optval;
  struct timeval ktime;
@@ -451,30 +452,61 @@ static int do_set_sock_timeout(int fd, i
   return -EFAULT;
  old_fs = get_fs();
  set_fs(KERNEL_DS);
- err = sys_setsockopt(fd, level, optname, (char *) &ktime, sizeof(ktime));
+ err = sock_setsockopt(sock, level, optname, (char *) &ktime, sizeof(ktime));
  set_fs(old_fs);

  return err;
 }

+static int compat_sock_setsockopt(struct socket *sock, int level, int optname,
+    char __user *optval, int optlen)
+{
+ if (optname == SO_ATTACH_FILTER)
+  return do_set_attach_filter(sock, level, optname,
+         optval, optlen);
+ if (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO)
+  return do_set_sock_timeout(sock, level, optname, optval, optlen);
+
+ return sock_setsockopt(sock, level, optname, optval, optlen);
+}
+
 asmlinkage long compat_sys_setsockopt(int fd, int level, int optname,
     char __user *optval, int optlen)
 {
+ int err;
+ struct socket *sock;
+
  /* SO_SET_REPLACE seems to be the same in all levels */
  if (optname == IPT_SO_SET_REPLACE)
   return do_netfilter_replace(fd, level, optname,
          optval, optlen);
- if (level == SOL_SOCKET && optname == SO_ATTACH_FILTER)
-  return do_set_attach_filter(fd, level, optname,
-         optval, optlen);
- if (level == SOL_SOCKET &&
-     (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO))
-  return do_set_sock_timeout(fd, level, optname, optval, optlen);
```

```
- return sys_setsockopt(fd, level, optname, optval, optlen);
+ if (optlen < 0)
+  return -EINVAL;
+
+ if ((sock = sockfd_lookup(fd, &err))!=NULL)
+ {
+  err = security_socket_setsockopt(sock,level,optname);
+  if (err) {
+   sockfd_put(sock);
+   return err;
+  }
+
+  if (level == SOL_SOCKET)
+   err = compat_sock_setsockopt(sock, level,
+     optname, optval, optlen);
+  else if (sock->ops->compat_setsockopt)
+   err = sock->ops->compat_setsockopt(sock, level,
+     optname, optval, optlen);
+  else
+   err = sock->ops->setsockopt(sock, level,
+     optname, optval, optlen);
+  sockfd_put(sock);
+ }
+ return err;
 }

-static int do_get_sock_timeout(int fd, int level, int optname,
+static int do_get_sock_timeout(struct socket *sock, int level, int optname,
   char __user *optval, int __user *optlen)
 {
  struct compat_timeval __user *up;
@@ -490,7 +522,7 @@ static int do_get_sock_timeout(int fd, i
  len = sizeof(ktime);
  old_fs = get_fs();
  set_fs(KERNEL_DS);
- err = sys_getsockopt(fd, level, optname, (char *) &ktime, &len);
+ err = sock_getsockopt(sock, level, optname, (char *) &ktime, &len);
  set_fs(old_fs);

  if (!err) {
@@ -503,15 +535,42 @@ static int do_get_sock_timeout(int fd, i
  return err;
 }

-asmlinkage long compat_sys_getsockopt(int fd, int level, int optname,
+static int compat_sock_getsockopt(struct socket *sock, int level, int optname,
     char __user *optval, int __user *optlen)
```

```
 {
- if (level == SOL_SOCKET &&
-     (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO))
-  return do_get_sock_timeout(fd, level, optname, optval, optlen);
- return sys_getsockopt(fd, level, optname, optval, optlen);
+ if (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO)
+  return do_get_sock_timeout(sock, level, optname, optval, optlen);
+ return sock_getsockopt(sock, level, optname, optval, optlen);
 }

+asmlinkage long compat_sys_getsockopt(int fd, int level, int optname,
+    char __user *optval, int __user *optlen)
+{
+ int err;
+ struct socket *sock;
+
+ if ((sock = sockfd_lookup(fd, &err))!=NULL)
+ {
+  err = security_socket_getsockopt(sock, level,
+          optname);
+  if (err) {
+   sockfd_put(sock);
+   return err;
+  }
+
+  if (level == SOL_SOCKET)
+   err = compat_sock_getsockopt(sock, level,
+     optname, optval, optlen);
+  else if (sock->ops->compat_getsockopt)
+   err = sock->ops->compat_getsockopt(sock, level,
+     optname, optval, optlen);
+  else
+   err = sock->ops->getsockopt(sock, level,
+     optname, optval, optlen);
+  sockfd_put(sock);
+ }
+ return err;
+}
 /* Argument list sizes for compat_sys_socketcall */
 #define AL(x) ((x) * sizeof(u32))
 static unsigned char nas[18]={AL(0),AL(3),AL(3),AL(3),AL(2),AL(3),
--- ./net/core/sock.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/core/sock.c 2006-03-09 12:58:23.000000000 +0300
@@ -1385,6 +1385,20 @@ int sock_common_getsockopt(struct socket

 EXPORT_SYMBOL(sock_common_getsockopt);

+#ifdef CONFIG_COMPAT
```

```
+int compat_sock_common_getsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_prot->compat_setsockopt)
+  return sk->sk_prot->compat_getsockopt(sk, level,
+   optname, optval, optlen);
+ return sk->sk_prot->getsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL(compat_sock_common_getsockopt);
+#endif
+
 int sock_common_recvmsg(struct kiocb *iocb, struct socket *sock,
    struct msghdr *msg, size_t size, int flags)
 {
@@ -1414,6 +1428,20 @@ int sock_common_setsockopt(struct socket

 EXPORT_SYMBOL(sock_common_setsockopt);

+#ifdef CONFIG_COMPAT
+int compat_sock_common_setsockopt(struct socket *sock,
+  int level, int optname, char __user *optval, int optlen)
+{
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_prot->compat_setsockopt)
+  return sk->sk_prot->compat_setsockopt(sk, level,
+   optname, optval, optlen);
+ return sk->sk_prot->setsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL(compat_sock_common_setsockopt);
+#endif
+
 void sk_common_release(struct sock *sk)
 {
  if (sk->sk_prot->destroy)
--- ./net/dccp/dccp.h.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/dccp/dccp.h 2006-03-09 12:58:23.000000000 +0300
@@ -246,6 +246,14 @@ extern int    dccp_getsockopt(struct soc
        char __user *optval, int __user *optlen);
 extern int    dccp_setsockopt(struct sock *sk, int level, int optname,
        char __user *optval, int optlen);
+#ifdef CONFIG_COMPAT
+extern int    compat_dccp_getsockopt(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int __user *optlen);
+extern int    compat_dccp_setsockopt(struct sock *sk,
```

```
+    int level, int optname,
+    char __user *optval, int optlen);
+#endif
 extern int    dccp_ioctl(struct sock *sk, int cmd, unsigned long arg);
 extern int    dccp_sendmsg(struct kiocb *iocb, struct sock *sk,
     struct msghdr *msg, size_t size);
--- ./net/dccp/ipv4.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/dccp/ipv4.c 2006-03-09 12:58:23.000000000 +0300
@@ -1028,6 +1028,10 @@ struct inet_connection_sock_af_ops dccp_
  .net_header_len = sizeof(struct iphdr),
  .setsockopt = ip_setsockopt,
  .getsockopt = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ip_setsockopt,
+ .compat_getsockopt = compat_ip_getsockopt,
+#endif
  .addr2sockaddr = inet_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in),
 };
@@ -1152,6 +1156,10 @@ struct proto dccp_prot = {
  .init   = dccp_v4_init_sock,
  .setsockopt  = dccp_setsockopt,
  .getsockopt  = dccp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_dccp_setsockopt,
+ .compat_getsockopt = compat_dccp_getsockopt,
+#endif
  .sendmsg  = dccp_sendmsg,
  .recvmsg  = dccp_recvmsg,
  .backlog_rcv  = dccp_v4_do_rcv,
--- ./net/dccp/ipv6.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/dccp/ipv6.c 2006-03-09 12:58:23.000000000 +0300
@@ -1170,6 +1170,10 @@ static struct proto dccp_v6_prot = {
  .init   = dccp_v6_init_sock,
  .setsockopt  = dccp_setsockopt,
  .getsockopt  = dccp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_dccp_setsockopt,
+ .compat_getsockopt = compat_dccp_getsockopt,
+#endif
  .sendmsg  = dccp_sendmsg,
  .recvmsg  = dccp_recvmsg,
  .backlog_rcv  = dccp_v6_do_rcv,
@@ -1207,6 +1211,10 @@ static struct proto_ops inet6_dccp_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
```

```
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap  = sock_no_mmap,
--- ./net/dccp/proto.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/dccp/proto.c 2006-03-09 12:58:23.000000000 +0300
@@ -255,18 +255,13 @@ static int dccp_setsockopt_service(struc
  return 0;
 }

-int dccp_setsockopt(struct sock *sk, int level, int optname,
-      char __user *optval, int optlen)
+static int do_dccp_setsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int optlen)
 {
  struct dccp_sock *dp;
  int err;
  int val;

- if (level != SOL_DCCP)
-  return inet_csk(sk)->icsk_af_ops->setsockopt(sk, level,
-            optname, optval,
-            optlen);
-
  if (optlen < sizeof(int))
   return -EINVAL;

@@ -293,8 +288,34 @@ int dccp_setsockopt(struct sock *sk, int
  return err;
 }

+int dccp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ if (level != SOL_DCCP)
+  return inet_csk(sk)->icsk_af_ops->setsockopt(sk, level,
+            optname, optval,
+            optlen);
+ return do_dccp_setsockopt(sk, level, optname, optval, optlen);
+}
 EXPORT_SYMBOL_GPL(dccp_setsockopt);

+#ifdef CONFIG_COMPAT
+int compat_dccp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
```

```
+ if (level != SOL_DCCP) {
+  if (inet_csk(sk)->icsk_af_ops->compat_setsockopt)
+   return inet_csk(sk)->icsk_af_ops->compat_setsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return inet_csk(sk)->icsk_af_ops->setsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_dccp_setsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL_GPL(compat_dccp_setsockopt);
+#endif
+
 static int dccp_getsockopt_service(struct sock *sk, int len,
        u32 __user *optval,
        int __user *optlen)
@@ -326,16 +347,12 @@ out:
  return err;
 }

-int dccp_getsockopt(struct sock *sk, int level, int optname,
+static int do_dccp_getsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen)
 {
  struct dccp_sock *dp;
  int val, len;

- if (level != SOL_DCCP)
-  return inet_csk(sk)->icsk_af_ops->getsockopt(sk, level,
-            optname, optval,
-            optlen);
  if (get_user(len, optlen))
   return -EFAULT;

@@ -368,8 +385,34 @@ int dccp_getsockopt(struct sock *sk, int
  return 0;
 }

+int dccp_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ if (level != SOL_DCCP)
+  return inet_csk(sk)->icsk_af_ops->getsockopt(sk, level,
+            optname, optval,
+            optlen);
+ return do_dccp_getsockopt(sk, level, optname, optval, optlen);
+}
 EXPORT_SYMBOL_GPL(dccp_getsockopt);
```

```
+#ifdef CONFIG_COMPAT
+int compat_dccp_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ if (level != SOL_DCCP) {
+  if (inet_csk(sk)->icsk_af_ops->compat_setsockopt)
+   return inet_csk(sk)->icsk_af_ops->compat_getsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return inet_csk(sk)->icsk_af_ops->getsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_dccp_getsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL_GPL(compat_dccp_getsockopt);
+#endif
+
 int dccp_sendmsg(struct kiocb *iocb, struct sock *sk, struct msghdr *msg,
    size_t len)
 {
@@ -696,6 +739,10 @@ static const struct proto_ops inet_dccp_
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap  = sock_no_mmap,
--- ./net/ipv4/af_inet.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv4/af_inet.c 2006-03-09 12:58:23.000000000 +0300
@@ -802,6 +802,10 @@ const struct proto_ops inet_stream_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
@@ -823,6 +827,10 @@ const struct proto_ops inet_dgram_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
```

```
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
@@ -848,6 +856,10 @@ static const struct proto_ops inet_sockr
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
--- ./net/ipv4/ip_sockglue.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv4/ip_sockglue.c 2006-03-09 12:58:23.000000000 +0300
@@ -380,14 +380,12 @@ out:
  * an IP socket.
  */

-int ip_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen)
+static int do_ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct inet_sock *inet = inet_sk(sk);
  int val=0,err;

- if (level != SOL_IP)
-  return -ENOPROTOOPT;
-
  if (((1<<optname) & ((1<<IP_PKTINFO) | (1<<IP_RECVTTL) |
        (1<<IP_RECVOPTS) | (1<<IP_RECVTOS) |
        (1<<IP_RETOPTS) | (1<<IP_TOS) |
@@ -849,12 +847,7 @@ mc_msf_out:
    break;

   default:
-#ifdef CONFIG_NETFILTER
-   err = nf_setsockopt(sk, PF_INET, optname, optval,
-         optlen);
-#else
    err = -ENOPROTOOPT;
-#endif
```

```
    break;
  }
  release_sock(sk);
@@ -865,12 +858,66 @@ e_inval:
  return -EINVAL;
 }

+int ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ int err;
+
+ if (level != SOL_IP)
+  return -ENOPROTOOPT;
+
+ err = do_ip_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_HDRINCL &&
+  optname != IP_IPSEC_POLICY && optname != IP_XFRM_POLICY
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > (MRT_BASE + 10))
+#endif
+    ) {
+  lock_sock(sk);
+  err = nf_setsockopt(sk, PF_INET, optname, optval, optlen);
+  release_sock(sk);
+ }
+#endif
+ return err;
+}
+
+#ifdef CONFIG_COMPAT
+int compat_ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ int err;
+
+ if (level != SOL_IP)
+  return -ENOPROTOOPT;
+
+ err = do_ip_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_HDRINCL &&
+  optname != IP_IPSEC_POLICY && optname != IP_XFRM_POLICY
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > (MRT_BASE + 10))
```

```
+#endif
+    ) {
+  lock_sock(sk);
+  err = compat_nf_setsockopt(sk, PF_INET,
+    optname, optval, optlen);
+  release_sock(sk);
+ }
+#endif
+ return err;
+}
+#endif
+
 /*
  * Get the options. Note for future reference. The GET of IP options gets the
  * _received_ ones. The set sets the _sent_ ones.
  */

-int ip_getsockopt(struct sock *sk, int level, int optname, char __user *optval, int __user *optlen)
+static int do_ip_getsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int __user *optlen)
 {
  struct inet_sock *inet = inet_sk(sk);
  int val;
@@ -1051,17 +1098,8 @@ int ip_getsockopt(struct sock *sk, int l
    val = inet->freebind;
    break;
   default:
-#ifdef CONFIG_NETFILTER
-   val = nf_getsockopt(sk, PF_INET, optname, optval,
-         &len);
-   release_sock(sk);
-   if (val >= 0)
-    val = put_user(len, optlen);
-   return val;
-#else
    release_sock(sk);
    return -ENOPROTOOPT;
-#endif
  }
  release_sock(sk);

@@ -1082,7 +1120,73 @@ int ip_getsockopt(struct sock *sk, int l
  return 0;
 }

+int ip_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
```

```
+ int err;
+
+ err = do_ip_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_PKTOPTIONS
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > MRT_BASE+10)
+#endif
+    ) {
+     int len;
+
+  if(get_user(len,optlen))
+   return -EFAULT;
+
+  lock_sock(sk);
+  err = nf_getsockopt(sk, PF_INET, optname, optval,
+    &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+  return err;
+ }
+#endif
+ return err;
+}
+
+#ifdef CONFIG_COMPAT
+int compat_ip_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ int err;
+
+ err = do_ip_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_PKTOPTIONS
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > MRT_BASE+10)
+#endif
+    ) {
+     int len;
+
+  if(get_user(len,optlen))
+   return -EFAULT;
+
+  lock_sock(sk);
+  err = compat_nf_getsockopt(sk, PF_INET,
```

```
+    optname, optval, &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+  return err;
+ }
+#endif
+ return err;
+}
+#endif
+
 EXPORT_SYMBOL(ip_cmsg_recv);

 EXPORT_SYMBOL(ip_getsockopt);
 EXPORT_SYMBOL(ip_setsockopt);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(compat_ip_getsockopt);
+EXPORT_SYMBOL(compat_ip_setsockopt);
+#endif
--- ./net/ipv4/raw.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv4/raw.c 2006-03-09 12:58:23.000000000 +0300
@@ -660,12 +660,9 @@ static int raw_geticmpfilter(struct sock
 out: return ret;
 }

-static int raw_setsockopt(struct sock *sk, int level, int optname,
+static int do_raw_setsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int optlen)
 {
- if (level != SOL_RAW)
-  return ip_setsockopt(sk, level, optname, optval, optlen);
-
   if (optname == ICMP_FILTER) {
    if (inet_sk(sk)->num != IPPROTO_ICMP)
     return -EOPNOTSUPP;
@@ -675,12 +672,28 @@ static int raw_setsockopt(struct sock *s
  return -ENOPROTOOPT;
 }

-static int raw_getsockopt(struct sock *sk, int level, int optname,
-      char __user *optval, int __user *optlen)
+static int raw_setsockopt(struct sock *sk, int level, int optname,
+       char __user *optval, int optlen)
 {
  if (level != SOL_RAW)
-  return ip_getsockopt(sk, level, optname, optval, optlen);
+  return ip_setsockopt(sk, level, optname, optval, optlen);
+ return do_raw_setsockopt(sk, level, optname, optval, optlen);
```

```
+}

+#ifdef CONFIG_COMPAT
+static int compat_raw_setsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int optlen)
+{
+ if (level != SOL_RAW)
+  return compat_ip_setsockopt(sk, level,
+    optname, optval, optlen);
+ return do_raw_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
+static int do_raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
  if (optname == ICMP_FILTER) {
   if (inet_sk(sk)->num != IPPROTO_ICMP)
    return -EOPNOTSUPP;
@@ -690,6 +703,25 @@ static int raw_getsockopt(struct sock *s
  return -ENOPROTOOPT;
 }

+static int raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_RAW)
+  return ip_getsockopt(sk, level, optname, optval, optlen);
+ return do_raw_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_RAW)
+  return compat_ip_getsockopt(sk, level,
+    optname, optval, optlen);
+ return do_raw_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 static int raw_ioctl(struct sock *sk, int cmd, unsigned long arg)
 {
  switch (cmd) {
@@ -728,6 +760,10 @@ struct proto raw_prot = {
  .init =  raw_init,
  .setsockopt = raw_setsockopt,
```

```
  .getsockopt = raw_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_raw_setsockopt,
+ .compat_getsockopt = compat_raw_getsockopt,
+#endif
  .sendmsg = raw_sendmsg,
  .recvmsg = raw_recvmsg,
  .bind =  raw_bind,
--- ./net/ipv4/tcp.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv4/tcp.c 2006-03-09 12:58:23.000000000 +0300
@@ -1687,18 +1687,14 @@ int tcp_disconnect(struct sock *sk, int
 /*
  * Socket option code for TCP.
  */
-int tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval,
-     int optlen)
+static int do_tcp_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct tcp_sock *tp = tcp_sk(sk);
  struct inet_connection_sock *icsk = inet_csk(sk);
  int val;
  int err = 0;

- if (level != SOL_TCP)
-  return icsk->icsk_af_ops->setsockopt(sk, level, optname,
-           optval, optlen);
-
  /* This is a string value all the others are int's */
  if (optname == TCP_CONGESTION) {
   char name[TCP_CA_NAME_MAX];
@@ -1871,6 +1867,35 @@ int tcp_setsockopt(struct sock *sk, int
  return err;
 }

+int tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval,
+     int optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP)
+  return icsk->icsk_af_ops->setsockopt(sk, level, optname,
+           optval, optlen);
+ return do_tcp_setsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+int compat_tcp_setsockopt(struct sock *sk, int level,
```

```
+  int optname, char __user *optval, int optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP) {
+  if (icsk->icsk_af_ops->compat_setsockopt)
+   return icsk->icsk_af_ops->compat_setsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return icsk->icsk_af_ops->setsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_tcp_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 /* Return information about state of tcp endpoint in API format. */
 void tcp_get_info(struct sock *sk, struct tcp_info *info)
 {
@@ -1931,17 +1956,13 @@ void tcp_get_info(struct sock *sk, struc

 EXPORT_SYMBOL_GPL(tcp_get_info);

-int tcp_getsockopt(struct sock *sk, int level, int optname, char __user *optval,
-     int __user *optlen)
+static int do_tcp_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
 {
  struct inet_connection_sock *icsk = inet_csk(sk);
  struct tcp_sock *tp = tcp_sk(sk);
  int val, len;

- if (level != SOL_TCP)
-  return icsk->icsk_af_ops->getsockopt(sk, level, optname,
-           optval, optlen);
-
  if (get_user(len, optlen))
   return -EFAULT;

@@ -2025,6 +2046,34 @@ int tcp_getsockopt(struct sock *sk, int
  return 0;
 }

+int tcp_getsockopt(struct sock *sk, int level, int optname, char __user *optval,
+     int __user *optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
```

```
+ if (level != SOL_TCP)
+  return icsk->icsk_af_ops->getsockopt(sk, level, optname,
+           optval, optlen);
+ return do_tcp_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+int compat_tcp_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP) {
+  if (icsk->icsk_af_ops->compat_getsockopt)
+   return icsk->icsk_af_ops->compat_getsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return icsk->icsk_af_ops->getsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_tcp_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif

 extern void __skb_cb_too_small_for_tcp(int, int);
 extern struct tcp_congestion_ops tcp_reno;
@@ -2142,3 +2191,7 @@ EXPORT_SYMBOL(tcp_sendpage);
 EXPORT_SYMBOL(tcp_setsockopt);
 EXPORT_SYMBOL(tcp_shutdown);
 EXPORT_SYMBOL(tcp_statistics);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(compat_tcp_setsockopt);
+EXPORT_SYMBOL(compat_tcp_getsockopt);
+#endif
--- ./net/ipv4/tcp_ipv4.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv4/tcp_ipv4.c 2006-03-09 12:58:23.000000000 +0300
@@ -1225,6 +1225,10 @@ struct inet_connection_sock_af_ops ipv4_
  .net_header_len = sizeof(struct iphdr),
  .setsockopt = ip_setsockopt,
  .getsockopt = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ip_setsockopt,
+ .compat_getsockopt = compat_ip_getsockopt,
+#endif
  .addr2sockaddr = inet_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in),
 };
@@ -1807,6 +1811,10 @@ struct proto tcp_prot = {
```

```
  .shutdown  = tcp_shutdown,
  .setsockopt  = tcp_setsockopt,
  .getsockopt  = tcp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_tcp_setsockopt,
+ .compat_getsockopt = compat_tcp_getsockopt,
+#endif
  .sendmsg  = tcp_sendmsg,
  .recvmsg  = tcp_recvmsg,
  .backlog_rcv  = tcp_v4_do_rcv,
--- ./net/ipv4/udp.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv4/udp.c 2006-03-09 12:58:23.000000000 +0300
@@ -1207,16 +1207,13 @@ static int udp_destroy_sock(struct sock
 /*
  * Socket option code for UDP
  */
-static int udp_setsockopt(struct sock *sk, int level, int optname,
+static int do_udp_setsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val;
  int err = 0;

- if (level != SOL_UDP)
-  return ip_setsockopt(sk, level, optname, optval, optlen);
-
  if(optlen<sizeof(int))
   return -EINVAL;

@@ -1256,15 +1253,31 @@ static int udp_setsockopt(struct sock *s
  return err;
 }

-static int udp_getsockopt(struct sock *sk, int level, int optname,
+static int udp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ if (level != SOL_UDP)
+  return ip_setsockopt(sk, level, optname, optval, optlen);
+ return do_udp_setsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_udp_setsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int optlen)
+{
+ if (level != SOL_UDP)
```

```
+  return compat_ip_setsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udp_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
+static int do_udp_getsockopt(struct sock *sk, int level, int optname,
      char __user *optval, int __user *optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val, len;

- if (level != SOL_UDP)
-  return ip_getsockopt(sk, level, optname, optval, optlen);
-
  if(get_user(len,optlen))
   return -EFAULT;

@@ -1293,6 +1306,24 @@ static int udp_getsockopt(struct sock *s
    return 0;
 }

+static int udp_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return ip_getsockopt(sk, level, optname, optval, optlen);
+ return do_udp_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_udp_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return compat_ip_getsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udp_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
 /**
  *  udp_poll - wait for a UDP event.
  * @file - file struct
@@ -1350,6 +1381,10 @@ struct proto udp_prot = {
  .destroy = udp_destroy_sock,
  .setsockopt = udp_setsockopt,
  .getsockopt = udp_getsockopt,
+#ifdef CONFIG_COMPAT
```

```
+ .compat_setsockopt = compat_udp_setsockopt,
+ .compat_getsockopt = compat_udp_getsockopt,
+#endif
  .sendmsg = udp_sendmsg,
  .recvmsg = udp_recvmsg,
  .sendpage = udp_sendpage,
--- ./net/ipv6/af_inet6.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv6/af_inet6.c 2006-03-09 12:58:23.000000000 +0300
@@ -470,6 +470,10 @@ const struct proto_ops inet6_stream_ops
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
@@ -491,6 +495,10 @@ const struct proto_ops inet6_dgram_ops =
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
@@ -519,6 +527,10 @@ static const struct proto_ops inet6_sock
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
--- ./net/ipv6/tcp_ipv6.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/ipv6/tcp_ipv6.c 2006-03-09 12:58:23.000000000 +0300
@@ -1565,6 +1565,10 @@ struct proto tcpv6_prot = {
  .shutdown  = tcp_shutdown,
  .setsockopt  = tcp_setsockopt,
  .getsockopt  = tcp_getsockopt,
+#ifdef CONFIG_COMPAT
```

```
+ .compat_setsockopt = compat_tcp_setsockopt,
+ .compat_getsockopt = compat_tcp_getsockopt,
+#endif
  .sendmsg  = tcp_sendmsg,
  .recvmsg  = tcp_recvmsg,
  .backlog_rcv  = tcp_v6_do_rcv,
--- ./net/netfilter/nf_sockopt.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/netfilter/nf_sockopt.c 2006-03-09 12:58:23.000000000 +0300
@@ -130,3 +130,72 @@ int nf_getsockopt(struct sock *sk, int p
 }
 EXPORT_SYMBOL(nf_getsockopt);

+#ifdef CONFIG_COMPAT
+static int compat_nf_sockopt(struct sock *sk, int pf, int val,
+        char __user *opt, int *len, int get)
+{
+ struct list_head *i;
+ struct nf_sockopt_ops *ops;
+ int ret;
+
+ if (down_interruptible(&nf_sockopt_mutex) != 0)
+  return -EINTR;
+
+ list_for_each(i, &nf_sockopts) {
+  ops = (struct nf_sockopt_ops *)i;
+  if (ops->pf == pf) {
+   if (get) {
+    if (val >= ops->get_optmin
+        && val < ops->get_optmax) {
+     ops->use++;
+     up(&nf_sockopt_mutex);
+     if (ops->compat_get)
+      ret = ops->compat_get(sk,
+       val, opt, len);
+     else
+      ret = ops->get(sk,
+       val, opt, len);
+     goto out;
+    }
+   } else {
+    if (val >= ops->set_optmin
+        && val < ops->set_optmax) {
+     ops->use++;
+     up(&nf_sockopt_mutex);
+     if (ops->compat_set)
+      ret = ops->compat_set(sk,
+       val, opt, *len);
+     else
```

```
+      ret = ops->set(sk,
+       val, opt, *len);
+     goto out;
+    }
+   }
+  }
+ }
+ up(&nf_sockopt_mutex);
+ return -ENOPROTOOPT;
+
+ out:
+ down(&nf_sockopt_mutex);
+ ops->use--;
+ if (ops->cleanup_task)
+  wake_up_process(ops->cleanup_task);
+ up(&nf_sockopt_mutex);
+ return ret;
+}
+
+int compat_nf_setsockopt(struct sock *sk, int pf,
+  int val, char __user *opt, int len)
+{
+ return compat_nf_sockopt(sk, pf, val, opt, &len, 0);
+}
+EXPORT_SYMBOL(compat_nf_setsockopt);
+
+int compat_nf_getsockopt(struct sock *sk, int pf,
+  int val, char __user *opt, int *len)
+{
+ return compat_nf_sockopt(sk, pf, val, opt, len, 1);
+}
+EXPORT_SYMBOL(compat_nf_getsockopt);
+#endif
--- ./net/sctp/ipv6.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/sctp/ipv6.c 2006-03-09 12:58:23.000000000 +0300
@@ -875,6 +875,10 @@ static const struct proto_ops inet6_seqp
  .shutdown   = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg    = inet_sendmsg,
  .recvmsg    = sock_common_recvmsg,
  .mmap       = sock_no_mmap,
--- ./net/sctp/protocol.c.compat 2006-03-09 12:57:54.000000000 +0300
+++ ./net/sctp/protocol.c 2006-03-09 12:58:23.000000000 +0300
```

```
@@ -845,6 +845,10 @@ static const struct proto_ops inet_seqpa
  .shutdown    = inet_shutdown,     /* Looks harmless.  */
  .setsockopt  = sock_common_setsockopt,   /* IP_SOL IP_OPTION is a problem. */
  .getsockopt  = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt  = compat_sock_common_setsockopt,
+ .compat_getsockopt  = compat_sock_common_getsockopt,
+#endif
  .sendmsg     = inet_sendmsg,
  .recvmsg     = sock_common_recvmsg,
  .mmap        = sock_no_mmap,
@@ -883,6 +887,10 @@ static struct sctp_af sctp_ipv4_specific
  .sctp_xmit      = sctp_v4_xmit,
  .setsockopt     = ip_setsockopt,
  .getsockopt     = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt     = compat_ip_setsockopt,
+ .compat_getsockopt     = compat_ip_getsockopt,
+#endif
  .get_dst = sctp_v4_get_dst,
  .get_saddr = sctp_v4_get_saddr,
  .copy_addrlist  = sctp_v4_copy_addrlist,
```

File Attachments

1) diff-ms-sockopts-compat-20060309, downloaded 683 times

Subject: Re: {get|set}sockopt compat layer
Posted by davem on Thu, 09 Mar 2006 23:29:34 GMT
View Forum Message <> Reply to Message

From: Dmitry Mishin <dim@openvz.org>
Date: Thu, 9 Mar 2006 13:23:59 +0300

> Hello, Arnd!
>
> > For the compat_ioctl stuff, we don't have the function pointer inside an
> > #ifdef, the overhead is relatively small since there is only one of these
> > structures per module implementing a protocol, but it avoids having to
> > rebuild everything when changing CONFIG_COMPAT.
> >
> > It's probably not a big issue either way, maybe davem has a stronger
> > opinion on it either way.
> >
> Done.

I think this looks fine but it doesn't apply cleanly to the
current net-2.6.17 tree.

Could you cook up a fresh patch, and send it with a complete
changelog entry and appropriate Signed-off-by: lines?

Thanks a lot Dmitry.

Subject: [PATCH] {get|set}sockopt compatibility layer
Posted by Mishin Dmitry on Fri, 10 Mar 2006 11:21:10 GMT
View Forum Message <> Reply to Message

This patch extends {get|set}sockopt compatibility layer in order to move
protocol specific parts to their place and avoid huge universal net/compat.c
file in the future.

Signed-off-by: Dmitry Mishin <dim@openvz.org>

--
Thanks,
Dmitry.

```
--- ./include/linux/net.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/linux/net.h 2006-03-10 12:24:11.000000000 +0300
@@ -149,6 +149,10 @@ struct proto_ops {
          int optname, char __user *optval, int optlen);
  int  (*getsockopt)(struct socket *sock, int level,
          int optname, char __user *optval, int __user *optlen);
+ int  (*compat_setsockopt)(struct socket *sock, int level,
+          int optname, char __user *optval, int optlen);
+ int  (*compat_getsockopt)(struct socket *sock, int level,
+          int optname, char __user *optval, int __user *optlen);
  int  (*sendmsg)   (struct kiocb *iocb, struct socket *sock,
          struct msghdr *m, size_t total_len);
  int  (*recvmsg)   (struct kiocb *iocb, struct socket *sock,
--- ./include/linux/netfilter.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/linux/netfilter.h 2006-03-10 12:24:11.000000000 +0300
@@ -80,10 +80,14 @@ struct nf_sockopt_ops
  int set_optmin;
  int set_optmax;
  int (*set)(struct sock *sk, int optval, void __user *user, unsigned int len);
+ int (*compat_set)(struct sock *sk, int optval,
+   void __user *user, unsigned int len);

  int get_optmin;
  int get_optmax;
  int (*get)(struct sock *sk, int optval, void __user *user, int *len);
+ int (*compat_get)(struct sock *sk, int optval,
+   void __user *user, int *len);
```

```
  /* Number of users inside set() or get(). */
  unsigned int use;
@@ -246,6 +250,11 @@ int nf_setsockopt(struct sock *sk, int p
 int nf_getsockopt(struct sock *sk, int pf, int optval, char __user *opt,
     int *len);

+int compat_nf_setsockopt(struct sock *sk, int pf, int optval,
+  char __user *opt, int len);
+int compat_nf_getsockopt(struct sock *sk, int pf, int optval,
+  char __user *opt, int *len);
+
 /* Packet queuing */
 struct nf_queue_handler {
  int (*outfn)(struct sk_buff *skb, struct nf_info *info,
--- ./include/net/inet_connection_sock.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/net/inet_connection_sock.h 2006-03-10 12:24:11.000000000 +0300
@@ -50,6 +50,12 @@ struct inet_connection_sock_af_ops {
       char __user *optval, int optlen);
  int     (*getsockopt)(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen);
+ int     (*compat_setsockopt)(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int optlen);
+ int     (*compat_getsockopt)(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int __user *optlen);
  void    (*addr2sockaddr)(struct sock *sk, struct sockaddr *);
  int sockaddr_len;
 };
--- ./include/net/ip.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/net/ip.h 2006-03-10 12:24:11.000000000 +0300
@@ -356,6 +356,10 @@ extern void ip_cmsg_recv(struct msghdr *
 extern int ip_cmsg_send(struct msghdr *msg, struct ipcm_cookie *ipc);
 extern int ip_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen);
 extern int ip_getsockopt(struct sock *sk, int level, int optname, char __user *optval, int __user
*optlen);
+extern int compat_ip_setsockopt(struct sock *sk, int level,
+   int optname, char __user *optval, int optlen);
+extern int compat_ip_getsockopt(struct sock *sk, int level,
+   int optname, char __user *optval, int __user *optlen);
 extern int ip_ra_control(struct sock *sk, unsigned char on, void (*destructor)(struct sock *));

 extern int  ip_recv_error(struct sock *sk, struct msghdr *msg, int len);
--- ./include/net/ipv6.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/net/ipv6.h 2006-03-10 13:16:18.000000000 +0300
@@ -520,6 +520,16 @@ extern int   ipv6_getsockopt(struct sock
       int optname,
```

```
       char __user *optval,
       int __user *optlen);
+extern int   compat_ipv6_setsockopt(struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int optlen);
+extern int   compat_ipv6_getsockopt(struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int __user *optlen);

 extern void   ipv6_packet_init(void);

--- ./include/net/sctp/structs.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/net/sctp/structs.h 2006-03-10 12:24:11.000000000 +0300
@@ -514,6 +514,16 @@ struct sctp_af {
       int optname,
       char __user *optval,
       int __user *optlen);
+ int  (*compat_setsockopt) (struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int optlen);
+ int  (*compat_getsockopt) (struct sock *sk,
+      int level,
+      int optname,
+      char __user *optval,
+      int __user *optlen);
  struct dst_entry *(*get_dst) (struct sctp_association *asoc,
       union sctp_addr *daddr,
       union sctp_addr *saddr);
--- ./include/net/sock.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/net/sock.h 2006-03-10 12:24:11.000000000 +0300
@@ -520,6 +520,14 @@ struct proto {
  int   (*getsockopt)(struct sock *sk, int level,
      int optname, char __user *optval,
      int __user *option);
+ int   (*compat_setsockopt)(struct sock *sk,
+     int level,
+     int optname, char __user *optval,
+     int optlen);
+ int   (*compat_getsockopt)(struct sock *sk,
+     int level,
+     int optname, char __user *optval,
+     int __user *option);
```

```
  int   (*sendmsg)(struct kiocb *iocb, struct sock *sk,
        struct msghdr *msg, size_t len);
  int   (*recvmsg)(struct kiocb *iocb, struct sock *sk,
@@ -816,6 +824,10 @@ extern int sock_common_recvmsg(struct ki
         struct msghdr *msg, size_t size, int flags);
 extern int sock_common_setsockopt(struct socket *sock, int level, int optname,
       char __user *optval, int optlen);
+extern int compat_sock_common_getsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int __user *optlen);
+extern int compat_sock_common_setsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int optlen);

 extern void sk_common_release(struct sock *sk);

--- ./include/net/tcp.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./include/net/tcp.h 2006-03-10 12:24:11.000000000 +0300
@@ -352,6 +352,12 @@ extern int   tcp_getsockopt(struct sock
 extern int   tcp_setsockopt(struct sock *sk, int level,
            int optname, char __user *optval,
            int optlen);
+extern int   compat_tcp_getsockopt(struct sock *sk,
+     int level, int optname,
+     char __user *optval, int __user *optlen);
+extern int   compat_tcp_setsockopt(struct sock *sk,
+     int level, int optname,
+     char __user *optval, int optlen);
 extern void   tcp_set_keepalive(struct sock *sk, int val);
 extern int   tcp_recvmsg(struct kiocb *iocb, struct sock *sk,
         struct msghdr *msg,
--- ./net/compat.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/compat.c 2006-03-10 12:24:11.000000000 +0300
@@ -416,7 +416,7 @@ struct compat_sock_fprog {
  compat_uptr_t filter;  /* struct sock_filter * */
 };

-static int do_set_attach_filter(int fd, int level, int optname,
+static int do_set_attach_filter(struct socket *sock, int level, int optname,
     char __user *optval, int optlen)
 {
  struct compat_sock_fprog __user *fprog32 = (struct compat_sock_fprog __user *)optval;
@@ -432,11 +432,12 @@ static int do_set_attach_filter(int fd,
      __put_user(compat_ptr(ptr), &kfprog->filter))
   return -EFAULT;

- return sys_setsockopt(fd, level, optname, (char __user *)kfprog,
+ return sock_setsockopt(sock, level, optname, (char __user *)kfprog,
         sizeof(struct sock_fprog));
 }
```

```
-static int do_set_sock_timeout(int fd, int level, int optname, char __user *optval, int optlen)
+static int do_set_sock_timeout(struct socket *sock, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct compat_timeval __user *up = (struct compat_timeval __user *) optval;
  struct timeval ktime;
@@ -451,30 +452,61 @@ static int do_set_sock_timeout(int fd, i
   return -EFAULT;
  old_fs = get_fs();
  set_fs(KERNEL_DS);
- err = sys_setsockopt(fd, level, optname, (char *) &ktime, sizeof(ktime));
+ err = sock_setsockopt(sock, level, optname, (char *) &ktime, sizeof(ktime));
  set_fs(old_fs);

  return err;
 }

+static int compat_sock_setsockopt(struct socket *sock, int level, int optname,
+    char __user *optval, int optlen)
+{
+ if (optname == SO_ATTACH_FILTER)
+  return do_set_attach_filter(sock, level, optname,
+         optval, optlen);
+ if (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO)
+  return do_set_sock_timeout(sock, level, optname, optval, optlen);
+
+ return sock_setsockopt(sock, level, optname, optval, optlen);
+}
+
 asmlinkage long compat_sys_setsockopt(int fd, int level, int optname,
     char __user *optval, int optlen)
 {
+ int err;
+ struct socket *sock;
+
  /* SO_SET_REPLACE seems to be the same in all levels */
  if (optname == IPT_SO_SET_REPLACE)
   return do_netfilter_replace(fd, level, optname,
          optval, optlen);
- if (level == SOL_SOCKET && optname == SO_ATTACH_FILTER)
-  return do_set_attach_filter(fd, level, optname,
-         optval, optlen);
- if (level == SOL_SOCKET &&
-     (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO))
-  return do_set_sock_timeout(fd, level, optname, optval, optlen);

- return sys_setsockopt(fd, level, optname, optval, optlen);
```

```
+ if (optlen < 0)
+  return -EINVAL;
+
+ if ((sock = sockfd_lookup(fd, &err))!=NULL)
+ {
+  err = security_socket_setsockopt(sock,level,optname);
+  if (err) {
+   sockfd_put(sock);
+   return err;
+  }
+
+  if (level == SOL_SOCKET)
+   err = compat_sock_setsockopt(sock, level,
+     optname, optval, optlen);
+  else if (sock->ops->compat_setsockopt)
+   err = sock->ops->compat_setsockopt(sock, level,
+     optname, optval, optlen);
+  else
+   err = sock->ops->setsockopt(sock, level,
+     optname, optval, optlen);
+  sockfd_put(sock);
+ }
+ return err;
 }

-static int do_get_sock_timeout(int fd, int level, int optname,
+static int do_get_sock_timeout(struct socket *sock, int level, int optname,
   char __user *optval, int __user *optlen)
 {
  struct compat_timeval __user *up;
@@ -490,7 +522,7 @@ static int do_get_sock_timeout(int fd, i
  len = sizeof(ktime);
  old_fs = get_fs();
  set_fs(KERNEL_DS);
- err = sys_getsockopt(fd, level, optname, (char *) &ktime, &len);
+ err = sock_getsockopt(sock, level, optname, (char *) &ktime, &len);
  set_fs(old_fs);

  if (!err) {
@@ -503,15 +535,42 @@ static int do_get_sock_timeout(int fd, i
  return err;
 }

-asmlinkage long compat_sys_getsockopt(int fd, int level, int optname,
+static int compat_sock_getsockopt(struct socket *sock, int level, int optname,
     char __user *optval, int __user *optlen)
 {
- if (level == SOL_SOCKET &&
```

```
-     (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO))
-  return do_get_sock_timeout(fd, level, optname, optval, optlen);
- return sys_getsockopt(fd, level, optname, optval, optlen);
+ if (optname == SO_RCVTIMEO || optname == SO_SNDTIMEO)
+  return do_get_sock_timeout(sock, level, optname, optval, optlen);
+ return sock_getsockopt(sock, level, optname, optval, optlen);
 }

+asmlinkage long compat_sys_getsockopt(int fd, int level, int optname,
+    char __user *optval, int __user *optlen)
+{
+ int err;
+ struct socket *sock;
+
+ if ((sock = sockfd_lookup(fd, &err))!=NULL)
+ {
+  err = security_socket_getsockopt(sock, level,
+          optname);
+  if (err) {
+   sockfd_put(sock);
+   return err;
+  }
+
+  if (level == SOL_SOCKET)
+   err = compat_sock_getsockopt(sock, level,
+     optname, optval, optlen);
+  else if (sock->ops->compat_getsockopt)
+   err = sock->ops->compat_getsockopt(sock, level,
+     optname, optval, optlen);
+  else
+   err = sock->ops->getsockopt(sock, level,
+     optname, optval, optlen);
+  sockfd_put(sock);
+ }
+ return err;
+}
 /* Argument list sizes for compat_sys_socketcall */
 #define AL(x) ((x) * sizeof(u32))
 static unsigned char nas[18]={AL(0),AL(3),AL(3),AL(3),AL(2),AL(3),
--- ./net/core/sock.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/core/sock.c 2006-03-10 12:24:11.000000000 +0300
@@ -1385,6 +1385,20 @@ int sock_common_getsockopt(struct socket

 EXPORT_SYMBOL(sock_common_getsockopt);

+#ifdef CONFIG_COMPAT
+int compat_sock_common_getsockopt(struct socket *sock, int level,
+  int optname, char __user *optval, int __user *optlen)
```

```
+{
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_prot->compat_setsockopt)
+  return sk->sk_prot->compat_getsockopt(sk, level,
+   optname, optval, optlen);
+ return sk->sk_prot->getsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL(compat_sock_common_getsockopt);
+#endif
+
 int sock_common_recvmsg(struct kiocb *iocb, struct socket *sock,
    struct msghdr *msg, size_t size, int flags)
 {
@@ -1414,6 +1428,20 @@ int sock_common_setsockopt(struct socket

 EXPORT_SYMBOL(sock_common_setsockopt);

+#ifdef CONFIG_COMPAT
+int compat_sock_common_setsockopt(struct socket *sock,
+  int level, int optname, char __user *optval, int optlen)
+{
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_prot->compat_setsockopt)
+  return sk->sk_prot->compat_setsockopt(sk, level,
+   optname, optval, optlen);
+ return sk->sk_prot->setsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL(compat_sock_common_setsockopt);
+#endif
+
 void sk_common_release(struct sock *sk)
 {
  if (sk->sk_prot->destroy)
--- ./net/dccp/dccp.h.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/dccp/dccp.h 2006-03-10 12:24:11.000000000 +0300
@@ -192,6 +192,14 @@ extern int    dccp_getsockopt(struct soc
        char __user *optval, int __user *optlen);
 extern int    dccp_setsockopt(struct sock *sk, int level, int optname,
        char __user *optval, int optlen);
+#ifdef CONFIG_COMPAT
+extern int    compat_dccp_getsockopt(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int __user *optlen);
+extern int    compat_dccp_setsockopt(struct sock *sk,
+    int level, int optname,
+    char __user *optval, int optlen);
```

```
+#endif
 extern int    dccp_ioctl(struct sock *sk, int cmd, unsigned long arg);
 extern int    dccp_sendmsg(struct kiocb *iocb, struct sock *sk,
     struct msghdr *msg, size_t size);
--- ./net/dccp/ipv4.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/dccp/ipv4.c 2006-03-10 12:30:33.000000000 +0300
@@ -994,6 +994,10 @@ static struct inet_connection_sock_af_op
  .net_header_len = sizeof(struct iphdr),
  .setsockopt = ip_setsockopt,
  .getsockopt = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ip_setsockopt,
+ .compat_getsockopt = compat_ip_getsockopt,
+#endif
  .addr2sockaddr = inet_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in),
 };
@@ -1040,6 +1044,10 @@ static struct proto dccp_v4_prot = {
  .init   = dccp_v4_init_sock,
  .setsockopt  = dccp_setsockopt,
  .getsockopt  = dccp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_dccp_setsockopt,
+ .compat_getsockopt = compat_dccp_getsockopt,
+#endif
  .sendmsg  = dccp_sendmsg,
  .recvmsg  = dccp_recvmsg,
  .backlog_rcv  = dccp_v4_do_rcv,
@@ -1079,6 +1087,10 @@ static const struct proto_ops inet_dccp_
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap  = sock_no_mmap,
--- ./net/dccp/ipv6.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/dccp/ipv6.c 2006-03-10 13:18:04.000000000 +0300
@@ -1114,6 +1114,10 @@ static struct inet_connection_sock_af_op
  .net_header_len = sizeof(struct ipv6hdr),
  .setsockopt = ipv6_setsockopt,
  .getsockopt = ipv6_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ipv6_setsockopt,
+ .compat_getsockopt = compat_ipv6_getsockopt,
```

```
+#endif
  .addr2sockaddr = inet6_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in6)
 };
@@ -1130,6 +1134,10 @@ static struct inet_connection_sock_af_op
  .net_header_len = sizeof(struct iphdr),
  .setsockopt = ipv6_setsockopt,
  .getsockopt = ipv6_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ipv6_setsockopt,
+ .compat_getsockopt = compat_ipv6_getsockopt,
+#endif
  .addr2sockaddr = inet6_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in6)
 };
@@ -1167,6 +1175,10 @@ static struct proto dccp_v6_prot = {
  .init  = dccp_v6_init_sock,
  .setsockopt = dccp_setsockopt,
  .getsockopt = dccp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_dccp_setsockopt,
+ .compat_getsockopt = compat_dccp_getsockopt,
+#endif
  .sendmsg = dccp_sendmsg,
  .recvmsg = dccp_recvmsg,
  .backlog_rcv = dccp_v6_do_rcv,
@@ -1204,6 +1216,10 @@ static struct proto_ops inet6_dccp_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap  = sock_no_mmap,
--- ./net/dccp/proto.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/dccp/proto.c 2006-03-10 12:24:11.000000000 +0300
@@ -455,18 +455,13 @@ out_free_val:
  goto out;
 }

-int dccp_setsockopt(struct sock *sk, int level, int optname,
-      char __user *optval, int optlen)
+static int do_dccp_setsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int optlen)
 {
```

```
  struct dccp_sock *dp;
  int err;
  int val;

- if (level != SOL_DCCP)
-  return inet_csk(sk)->icsk_af_ops->setsockopt(sk, level,
-            optname, optval,
-            optlen);
-
  if (optlen < sizeof(int))
   return -EINVAL;

@@ -512,8 +507,34 @@ int dccp_setsockopt(struct sock *sk, int
  return err;
 }

+int dccp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ if (level != SOL_DCCP)
+  return inet_csk(sk)->icsk_af_ops->setsockopt(sk, level,
+            optname, optval,
+            optlen);
+ return do_dccp_setsockopt(sk, level, optname, optval, optlen);
+}
 EXPORT_SYMBOL_GPL(dccp_setsockopt);

+#ifdef CONFIG_COMPAT
+int compat_dccp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ if (level != SOL_DCCP) {
+  if (inet_csk(sk)->icsk_af_ops->compat_setsockopt)
+   return inet_csk(sk)->icsk_af_ops->compat_setsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return inet_csk(sk)->icsk_af_ops->setsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_dccp_setsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL_GPL(compat_dccp_setsockopt);
+#endif
+
 static int dccp_getsockopt_service(struct sock *sk, int len,
        __be32 __user *optval,
        int __user *optlen)
@@ -545,16 +566,12 @@ out:
```

```
  return err;
 }

-int dccp_getsockopt(struct sock *sk, int level, int optname,
+static int do_dccp_getsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen)
 {
  struct dccp_sock *dp;
  int val, len;

- if (level != SOL_DCCP)
-  return inet_csk(sk)->icsk_af_ops->getsockopt(sk, level,
-            optname, optval,
-            optlen);
  if (get_user(len, optlen))
   return -EFAULT;

@@ -587,8 +604,34 @@ int dccp_getsockopt(struct sock *sk, int
  return 0;
 }

+int dccp_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ if (level != SOL_DCCP)
+  return inet_csk(sk)->icsk_af_ops->getsockopt(sk, level,
+            optname, optval,
+            optlen);
+ return do_dccp_getsockopt(sk, level, optname, optval, optlen);
+}
 EXPORT_SYMBOL_GPL(dccp_getsockopt);

+#ifdef CONFIG_COMPAT
+int compat_dccp_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ if (level != SOL_DCCP) {
+  if (inet_csk(sk)->icsk_af_ops->compat_setsockopt)
+   return inet_csk(sk)->icsk_af_ops->compat_getsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return inet_csk(sk)->icsk_af_ops->getsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_dccp_getsockopt(sk, level, optname, optval, optlen);
+}
+EXPORT_SYMBOL_GPL(compat_dccp_getsockopt);
+#endif
```

```
+
 int dccp_sendmsg(struct kiocb *iocb, struct sock *sk, struct msghdr *msg,
    size_t len)
 {
--- ./net/ipv4/af_inet.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv4/af_inet.c 2006-03-10 12:24:11.000000000 +0300
@@ -802,6 +802,10 @@ const struct proto_ops inet_stream_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
@@ -823,6 +827,10 @@ const struct proto_ops inet_dgram_ops =
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
@@ -848,6 +856,10 @@ static const struct proto_ops inet_sockr
  .shutdown = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,
  .recvmsg = sock_common_recvmsg,
  .mmap =  sock_no_mmap,
--- ./net/ipv4/ip_sockglue.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv4/ip_sockglue.c 2006-03-10 12:24:11.000000000 +0300
@@ -380,14 +380,12 @@ out:
  * an IP socket.
  */

-int ip_setsockopt(struct sock *sk, int level, int optname, char __user *optval, int optlen)
+static int do_ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
```

```
 {
  struct inet_sock *inet = inet_sk(sk);
  int val=0,err;

- if (level != SOL_IP)
-  return -ENOPROTOOPT;
-
  if (((1<<optname) & ((1<<IP_PKTINFO) | (1<<IP_RECVTTL) |
        (1<<IP_RECVOPTS) | (1<<IP_RECVTOS) |
        (1<<IP_RETOPTS) | (1<<IP_TOS) |
@@ -849,12 +847,7 @@ mc_msf_out:
    break;

   default:
-#ifdef CONFIG_NETFILTER
-   err = nf_setsockopt(sk, PF_INET, optname, optval,
-         optlen);
-#else
    err = -ENOPROTOOPT;
-#endif
    break;
  }
  release_sock(sk);
@@ -865,12 +858,66 @@ e_inval:
  return -EINVAL;
 }

+int ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ int err;
+
+ if (level != SOL_IP)
+  return -ENOPROTOOPT;
+
+ err = do_ip_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_HDRINCL &&
+  optname != IP_IPSEC_POLICY && optname != IP_XFRM_POLICY
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > (MRT_BASE + 10))
+#endif
+    ) {
+  lock_sock(sk);
+  err = nf_setsockopt(sk, PF_INET, optname, optval, optlen);
+  release_sock(sk);
+ }
```

```
+#endif
+ return err;
+}
+
+#ifdef CONFIG_COMPAT
+int compat_ip_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ int err;
+
+ if (level != SOL_IP)
+  return -ENOPROTOOPT;
+
+ err = do_ip_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_HDRINCL &&
+  optname != IP_IPSEC_POLICY && optname != IP_XFRM_POLICY
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > (MRT_BASE + 10))
+#endif
+    ) {
+  lock_sock(sk);
+  err = compat_nf_setsockopt(sk, PF_INET,
+    optname, optval, optlen);
+  release_sock(sk);
+ }
+#endif
+ return err;
+}
+#endif
+
 /*
  * Get the options. Note for future reference. The GET of IP options gets the
  * _received_ ones. The set sets the _sent_ ones.
  */

-int ip_getsockopt(struct sock *sk, int level, int optname, char __user *optval, int __user *optlen)
+static int do_ip_getsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int __user *optlen)
 {
  struct inet_sock *inet = inet_sk(sk);
  int val;
@@ -1051,17 +1098,8 @@ int ip_getsockopt(struct sock *sk, int l
    val = inet->freebind;
    break;
   default:
-#ifdef CONFIG_NETFILTER
```

```
-   val = nf_getsockopt(sk, PF_INET, optname, optval,
-         &len);
-   release_sock(sk);
-   if (val >= 0)
-    val = put_user(len, optlen);
-   return val;
-#else
    release_sock(sk);
    return -ENOPROTOOPT;
-#endif
  }
  release_sock(sk);

@@ -1082,7 +1120,73 @@ int ip_getsockopt(struct sock *sk, int l
  return 0;
 }

+int ip_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ int err;
+
+ err = do_ip_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_PKTOPTIONS
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > MRT_BASE+10)
+#endif
+    ) {
+     int len;
+
+  if(get_user(len,optlen))
+   return -EFAULT;
+
+  lock_sock(sk);
+  err = nf_getsockopt(sk, PF_INET, optname, optval,
+    &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+  return err;
+ }
+#endif
+ return err;
+}
+
+#ifdef CONFIG_COMPAT
```

```
+int compat_ip_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ int err;
+
+ err = do_ip_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IP_PKTOPTIONS
+#ifdef CONFIG_IP_MROUTE
+  && (optname < MRT_BASE || optname > MRT_BASE+10)
+#endif
+    ) {
+     int len;
+
+  if(get_user(len,optlen))
+   return -EFAULT;
+
+  lock_sock(sk);
+  err = compat_nf_getsockopt(sk, PF_INET,
+    optname, optval, &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+  return err;
+ }
+#endif
+ return err;
+}
+#endif
+
 EXPORT_SYMBOL(ip_cmsg_recv);

 EXPORT_SYMBOL(ip_getsockopt);
 EXPORT_SYMBOL(ip_setsockopt);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(compat_ip_getsockopt);
+EXPORT_SYMBOL(compat_ip_setsockopt);
+#endif
--- ./net/ipv4/raw.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv4/raw.c 2006-03-10 12:24:11.000000000 +0300
@@ -660,12 +660,9 @@ static int raw_geticmpfilter(struct sock
 out: return ret;
 }

-static int raw_setsockopt(struct sock *sk, int level, int optname,
+static int do_raw_setsockopt(struct sock *sk, int level, int optname,
      char __user *optval, int optlen)
```

```
 {
- if (level != SOL_RAW)
-  return ip_setsockopt(sk, level, optname, optval, optlen);
-
  if (optname == ICMP_FILTER) {
   if (inet_sk(sk)->num != IPPROTO_ICMP)
    return -EOPNOTSUPP;
@@ -675,12 +672,28 @@ static int raw_setsockopt(struct sock *s
  return -ENOPROTOOPT;
 }

-static int raw_getsockopt(struct sock *sk, int level, int optname,
-     char __user *optval, int __user *optlen)
+static int raw_setsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int optlen)
 {
  if (level != SOL_RAW)
-  return ip_getsockopt(sk, level, optname, optval, optlen);
+  return ip_setsockopt(sk, level, optname, optval, optlen);
+ return do_raw_setsockopt(sk, level, optname, optval, optlen);
+}

+#ifdef CONFIG_COMPAT
+static int compat_raw_setsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int optlen)
+{
+ if (level != SOL_RAW)
+  return compat_ip_setsockopt(sk, level,
+    optname, optval, optlen);
+ return do_raw_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
+static int do_raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
  if (optname == ICMP_FILTER) {
   if (inet_sk(sk)->num != IPPROTO_ICMP)
    return -EOPNOTSUPP;
@@ -690,6 +703,25 @@ static int raw_getsockopt(struct sock *s
  return -ENOPROTOOPT;
 }

+static int raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_RAW)
+  return ip_getsockopt(sk, level, optname, optval, optlen);
```

```
+ return do_raw_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_raw_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_RAW)
+  return compat_ip_getsockopt(sk, level,
+    optname, optval, optlen);
+ return do_raw_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 static int raw_ioctl(struct sock *sk, int cmd, unsigned long arg)
 {
  switch (cmd) {
@@ -728,6 +760,10 @@ struct proto raw_prot = {
  .init =  raw_init,
  .setsockopt = raw_setsockopt,
  .getsockopt = raw_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_raw_setsockopt,
+ .compat_getsockopt = compat_raw_getsockopt,
+#endif
  .sendmsg = raw_sendmsg,
  .recvmsg = raw_recvmsg,
  .bind =  raw_bind,
--- ./net/ipv4/tcp.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv4/tcp.c 2006-03-10 12:24:11.000000000 +0300
@@ -1687,18 +1687,14 @@ int tcp_disconnect(struct sock *sk, int
 /*
  * Socket option code for TCP.
  */
-int tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval,
-     int optlen)
+static int do_tcp_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
 {
  struct tcp_sock *tp = tcp_sk(sk);
  struct inet_connection_sock *icsk = inet_csk(sk);
  int val;
  int err = 0;

- if (level != SOL_TCP)
-  return icsk->icsk_af_ops->setsockopt(sk, level, optname,
-           optval, optlen);
-
```

```
  /* This is a string value all the others are int's */
  if (optname == TCP_CONGESTION) {
   char name[TCP_CA_NAME_MAX];
@@ -1871,6 +1867,35 @@ int tcp_setsockopt(struct sock *sk, int
  return err;
 }

+int tcp_setsockopt(struct sock *sk, int level, int optname, char __user *optval,
+     int optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP)
+  return icsk->icsk_af_ops->setsockopt(sk, level, optname,
+           optval, optlen);
+ return do_tcp_setsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+int compat_tcp_setsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP) {
+  if (icsk->icsk_af_ops->compat_setsockopt)
+   return icsk->icsk_af_ops->compat_setsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return icsk->icsk_af_ops->setsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_tcp_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 /* Return information about state of tcp endpoint in API format. */
 void tcp_get_info(struct sock *sk, struct tcp_info *info)
 {
@@ -1931,17 +1956,13 @@ void tcp_get_info(struct sock *sk, struc

 EXPORT_SYMBOL_GPL(tcp_get_info);

-int tcp_getsockopt(struct sock *sk, int level, int optname, char __user *optval,
-     int __user *optlen)
+static int do_tcp_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
 {
```

```
  struct inet_connection_sock *icsk = inet_csk(sk);
  struct tcp_sock *tp = tcp_sk(sk);
  int val, len;

- if (level != SOL_TCP)
-  return icsk->icsk_af_ops->getsockopt(sk, level, optname,
-           optval, optlen);
-
  if (get_user(len, optlen))
   return -EFAULT;

@@ -2025,6 +2046,34 @@ int tcp_getsockopt(struct sock *sk, int
  return 0;
 }

+int tcp_getsockopt(struct sock *sk, int level, int optname, char __user *optval,
+     int __user *optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP)
+  return icsk->icsk_af_ops->getsockopt(sk, level, optname,
+           optval, optlen);
+ return do_tcp_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+int compat_tcp_getsockopt(struct sock *sk, int level,
+  int optname, char __user *optval, int __user *optlen)
+{
+ struct inet_connection_sock *icsk = inet_csk(sk);
+
+ if (level != SOL_TCP) {
+  if (icsk->icsk_af_ops->compat_getsockopt)
+   return icsk->icsk_af_ops->compat_getsockopt(sk,
+    level, optname, optval, optlen);
+  else
+   return icsk->icsk_af_ops->getsockopt(sk,
+    level, optname, optval, optlen);
+ }
+ return do_tcp_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif

 extern void __skb_cb_too_small_for_tcp(int, int);
 extern struct tcp_congestion_ops tcp_reno;
@@ -2142,3 +2191,7 @@ EXPORT_SYMBOL(tcp_sendpage);
 EXPORT_SYMBOL(tcp_setsockopt);
```

```
 EXPORT_SYMBOL(tcp_shutdown);
 EXPORT_SYMBOL(tcp_statistics);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(compat_tcp_setsockopt);
+EXPORT_SYMBOL(compat_tcp_getsockopt);
+#endif
--- ./net/ipv4/tcp_ipv4.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv4/tcp_ipv4.c 2006-03-10 12:24:11.000000000 +0300
@@ -1226,6 +1226,10 @@ struct inet_connection_sock_af_ops ipv4_
  .net_header_len = sizeof(struct iphdr),
  .setsockopt = ip_setsockopt,
  .getsockopt = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ip_setsockopt,
+ .compat_getsockopt = compat_ip_getsockopt,
+#endif
  .addr2sockaddr = inet_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in),
 };
@@ -1808,6 +1812,10 @@ struct proto tcp_prot = {
  .shutdown  = tcp_shutdown,
  .setsockopt  = tcp_setsockopt,
  .getsockopt  = tcp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_tcp_setsockopt,
+ .compat_getsockopt = compat_tcp_getsockopt,
+#endif
  .sendmsg  = tcp_sendmsg,
  .recvmsg  = tcp_recvmsg,
  .backlog_rcv  = tcp_v4_do_rcv,
--- ./net/ipv4/udp.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv4/udp.c 2006-03-10 12:24:11.000000000 +0300
@@ -1207,16 +1207,13 @@ static int udp_destroy_sock(struct sock
 /*
  * Socket option code for UDP
  */
-static int udp_setsockopt(struct sock *sk, int level, int optname,
+static int do_udp_setsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val;
  int err = 0;

- if (level != SOL_UDP)
-  return ip_setsockopt(sk, level, optname, optval, optlen);
-
  if(optlen<sizeof(int))
```

```
   return -EINVAL;

@@ -1256,15 +1253,31 @@ static int udp_setsockopt(struct sock *s
  return err;
 }

-static int udp_getsockopt(struct sock *sk, int level, int optname,
+static int udp_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ if (level != SOL_UDP)
+  return ip_setsockopt(sk, level, optname, optval, optlen);
+ return do_udp_setsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_udp_setsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int optlen)
+{
+ if (level != SOL_UDP)
+  return compat_ip_setsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udp_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
+static int do_udp_getsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val, len;

- if (level != SOL_UDP)
-  return ip_getsockopt(sk, level, optname, optval, optlen);
-
  if(get_user(len,optlen))
   return -EFAULT;

@@ -1293,6 +1306,24 @@ static int udp_getsockopt(struct sock *s
    return 0;
 }

+static int udp_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return ip_getsockopt(sk, level, optname, optval, optlen);
+ return do_udp_getsockopt(sk, level, optname, optval, optlen);
```

```
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_udp_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return compat_ip_getsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udp_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
 /**
  *  udp_poll - wait for a UDP event.
  * @file - file struct
@@ -1350,6 +1381,10 @@ struct proto udp_prot = {
  .destroy = udp_destroy_sock,
  .setsockopt = udp_setsockopt,
  .getsockopt = udp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_udp_setsockopt,
+ .compat_getsockopt = compat_udp_getsockopt,
+#endif
  .sendmsg = udp_sendmsg,
  .recvmsg = udp_recvmsg,
  .sendpage = udp_sendpage,
--- ./net/ipv6/af_inet6.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv6/af_inet6.c 2006-03-10 12:24:11.000000000 +0300
@@ -470,6 +470,10 @@ const struct proto_ops inet6_stream_ops
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
@@ -491,6 +495,10 @@ const struct proto_ops inet6_dgram_ops =
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
```

```
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
@@ -519,6 +527,10 @@ static const struct proto_ops inet6_sock
  .shutdown = inet_shutdown,   /* ok  */
  .setsockopt = sock_common_setsockopt,  /* ok  */
  .getsockopt = sock_common_getsockopt,  /* ok  */
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
  .sendmsg = inet_sendmsg,   /* ok  */
  .recvmsg = sock_common_recvmsg,  /* ok  */
  .mmap =  sock_no_mmap,
--- ./net/ipv6/ipv6_sockglue.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv6/ipv6_sockglue.c 2006-03-10 14:02:07.000000000 +0300
@@ -109,19 +109,13 @@ int ip6_ra_control(struct sock *sk, int
  return 0;
 }

-int ipv6_setsockopt(struct sock *sk, int level, int optname,
+static int do_ipv6_setsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int optlen)
 {
  struct ipv6_pinfo *np = inet6_sk(sk);
  int val, valbool;
  int retv = -ENOPROTOOPT;

- if (level == SOL_IP && sk->sk_type != SOCK_RAW)
-  return udp_prot.setsockopt(sk, level, optname, optval, optlen);
-
- if(level!=SOL_IPV6)
-  goto out;
-
  if (optval == NULL)
   val=0;
  else if (get_user(val, (int __user *) optval))
@@ -613,17 +607,9 @@ done:
   retv = xfrm_user_policy(sk, optname, optval, optlen);
   break;

-#ifdef CONFIG_NETFILTER
- default:
-  retv = nf_setsockopt(sk, PF_INET6, optname, optval,
-         optlen);
-  break;
-#endif
-
  }
```

```
  release_sock(sk);

-out:
  return retv;

 e_inval:
@@ -631,6 +617,65 @@ e_inval:
  return -EINVAL;
 }

+int ipv6_setsockopt(struct sock *sk, int level, int optname,
+       char __user *optval, int optlen)
+{
+ int err;
+
+ if (level == SOL_IP && sk->sk_type != SOCK_RAW)
+  return udp_prot.setsockopt(sk, level, optname, optval, optlen);
+
+ if (level != SOL_IPV6)
+  return -ENOPROTOOPT;
+
+ err = do_ipv6_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IPV6_IPSEC_POLICY &&
+    optname != IPV6_XFRM_POLICY) {
+  lock_sock(sk);
+  err = nf_setsockopt(sk, PF_INET6, optname, optval,
+     optlen);
+  release_sock(sk);
+ }
+#endif
+ return err;
+}
+
+
+#ifdef CONFIG_COMPAT
+int compat_ipv6_setsockopt(struct sock *sk, int level, int optname,
+       char __user *optval, int optlen)
+{
+ int err;
+
+ if (level == SOL_IP && sk->sk_type != SOCK_RAW) {
+  if (udp_prot.compat_setsockopt)
+   return udp_prot.compat_setsockopt(sk, level,
+      optname, optval, optlen);
+  else
+   return udp_prot.setsockopt(sk, level,
```

```
+     optname, optval, optlen);
+ }
+
+ if (level != SOL_IPV6)
+  return -ENOPROTOOPT;
+
+ err = do_ipv6_setsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible ENOPROTOOPTs except default case */
+ if (err == -ENOPROTOOPT && optname != IPV6_IPSEC_POLICY &&
+    optname != IPV6_XFRM_POLICY) {
+  lock_sock(sk);
+  err = compat_nf_setsockopt(sk, PF_INET6, optname, optval,
+     optlen);
+  release_sock(sk);
+ }
+#endif
+ return err;
+}
+#endif
+
 static int ipv6_getsockopt_sticky(struct sock *sk, struct ipv6_opt_hdr *hdr,
       char __user *optval, int len)
 {
@@ -642,17 +687,13 @@ static int ipv6_getsockopt_sticky(struct
  return len;
 }

-int ipv6_getsockopt(struct sock *sk, int level, int optname,
+static int do_ipv6_getsockopt(struct sock *sk, int level, int optname,
       char __user *optval, int __user *optlen)
 {
  struct ipv6_pinfo *np = inet6_sk(sk);
  int len;
  int val;

- if (level == SOL_IP && sk->sk_type != SOCK_RAW)
-  return udp_prot.getsockopt(sk, level, optname, optval, optlen);
- if(level!=SOL_IPV6)
-  return -ENOPROTOOPT;
  if (get_user(len, optlen))
   return -EFAULT;
  switch (optname) {
@@ -842,17 +883,7 @@ int ipv6_getsockopt(struct sock *sk, int
   break;

  default:
-#ifdef CONFIG_NETFILTER
```

```
-  lock_sock(sk);
-  val = nf_getsockopt(sk, PF_INET6, optname, optval,
-        &len);
-  release_sock(sk);
-  if (val >= 0)
-   val = put_user(len, optlen);
-  return val;
-#else
   return -EINVAL;
-#endif
  }
  len = min_t(unsigned int, sizeof(int), len);
  if(put_user(len, optlen))
@@ -862,6 +893,78 @@ int ipv6_getsockopt(struct sock *sk, int
  return 0;
 }

+int ipv6_getsockopt(struct sock *sk, int level, int optname,
+       char __user *optval, int __user *optlen)
+{
+ int err;
+
+ if (level == SOL_IP && sk->sk_type != SOCK_RAW)
+  return udp_prot.getsockopt(sk, level, optname, optval, optlen);
+
+ if(level != SOL_IPV6)
+  return -ENOPROTOOPT;
+
+ err = do_ipv6_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible EINVALs except default case */
+ if (err == -ENOPROTOOPT && optname != IPV6_ADDRFORM &&
+    optname != MCAST_MSFILTER) {
+  int len;
+
+  if (get_user(len, optlen))
+   return -EFAULT;
+
+  lock_sock(sk);
+  err = nf_getsockopt(sk, PF_INET6, optname, optval,
+     &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+ }
+#endif
+ return err;
+}
```

```
+
+#ifdef CONFIG_COMPAT
+int compat_ipv6_getsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int __user *optlen)
+{
+ int err;
+
+ if (level == SOL_IP && sk->sk_type != SOCK_RAW) {
+  if (udp_prot.compat_getsockopt)
+   return udp_prot.compat_getsockopt(sk, level,
+     optname, optval, optlen);
+  else
+   return udp_prot.getsockopt(sk, level,
+     optname, optval, optlen);
+ }
+
+ if(level != SOL_IPV6)
+  return -ENOPROTOOPT;
+
+ err = do_ipv6_getsockopt(sk, level, optname, optval, optlen);
+#ifdef CONFIG_NETFILTER
+ /* we need to exclude all possible EINVALs except default case */
+ if (err == -ENOPROTOOPT && optname != IPV6_ADDRFORM &&
+   optname != MCAST_MSFILTER) {
+  int len;
+
+  if (get_user(len, optlen))
+   return -EFAULT;
+
+  lock_sock(sk);
+  err = compat_nf_getsockopt(sk, PF_INET6, optname, optval,
+    &len);
+  release_sock(sk);
+  if (err >= 0)
+   err = put_user(len, optlen);
+ }
+#endif
+ return err;
+}
+#endif
+
 void __init ipv6_packet_init(void)
 {
  dev_add_pack(&ipv6_packet_type);
--- ./net/ipv6/ipv6_syms.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv6/ipv6_syms.c 2006-03-10 13:16:56.000000000 +0300
@@ -18,6 +18,10 @@ EXPORT_SYMBOL(ip6_route_output);
 EXPORT_SYMBOL(addrconf_lock);
```

```
 EXPORT_SYMBOL(ipv6_setsockopt);
 EXPORT_SYMBOL(ipv6_getsockopt);
+#ifdef CONFIG_COMPAT
+EXPORT_SYMBOL(compat_ipv6_setsockopt);
+EXPORT_SYMBOL(compat_ipv6_getsockopt);
+#endif
 EXPORT_SYMBOL(inet6_register_protosw);
 EXPORT_SYMBOL(inet6_unregister_protosw);
 EXPORT_SYMBOL(inet6_add_protocol);
--- ./net/ipv6/raw.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv6/raw.c 2006-03-10 13:34:28.000000000 +0300
@@ -859,29 +859,12 @@ static int rawv6_geticmpfilter(struct so
 }


-static int rawv6_setsockopt(struct sock *sk, int level, int optname,
+static int do_rawv6_setsockopt(struct sock *sk, int level, int optname,
        char __user *optval, int optlen)
 {
  struct raw6_sock *rp = raw6_sk(sk);
  int val;

- switch(level) {
-  case SOL_RAW:
-   break;
-
-  case SOL_ICMPV6:
-   if (inet_sk(sk)->num != IPPROTO_ICMPV6)
-    return -EOPNOTSUPP;
-   return rawv6_seticmpfilter(sk, level, optname, optval,
-         optlen);
-  case SOL_IPV6:
-   if (optname == IPV6_CHECKSUM)
-    break;
-  default:
-   return ipv6_setsockopt(sk, level, optname, optval,
-            optlen);
- };
-
    if (get_user(val, (int __user *)optval))
   return -EFAULT;

@@ -906,12 +889,9 @@ static int rawv6_setsockopt(struct sock
  }
 }

-static int rawv6_getsockopt(struct sock *sk, int level, int optname,
-       char __user *optval, int __user *optlen)
```

```
+static int rawv6_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
 {
- struct raw6_sock *rp = raw6_sk(sk);
- int val, len;
-
  switch(level) {
   case SOL_RAW:
    break;
@@ -919,15 +899,47 @@ static int rawv6_getsockopt(struct sock
   case SOL_ICMPV6:
    if (inet_sk(sk)->num != IPPROTO_ICMPV6)
     return -EOPNOTSUPP;
-   return rawv6_geticmpfilter(sk, level, optname, optval,
+   return rawv6_seticmpfilter(sk, level, optname, optval,
          optlen);
   case SOL_IPV6:
    if (optname == IPV6_CHECKSUM)
     break;
   default:
-   return ipv6_getsockopt(sk, level, optname, optval,
+   return ipv6_setsockopt(sk, level, optname, optval,
            optlen);
  };
+ return do_rawv6_setsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_rawv6_setsockopt(struct sock *sk, int level, int optname,
+      char __user *optval, int optlen)
+{
+ switch(level) {
+  case SOL_RAW:
+   break;
+
+  case SOL_ICMPV6:
+   if (inet_sk(sk)->num != IPPROTO_ICMPV6)
+    return -EOPNOTSUPP;
+   return rawv6_seticmpfilter(sk, level, optname, optval,
+         optlen);
+  case SOL_IPV6:
+   if (optname == IPV6_CHECKSUM)
+    break;
+  default:
+   return compat_ipv6_setsockopt(sk, level,
+      optname, optval, optlen);
+ };
+ return do_rawv6_setsockopt(sk, level, optname, optval, optlen);
```

```
+}
+#endif
+
+static int do_rawv6_getsockopt(struct sock *sk, int level, int optname,
+       char __user *optval, int __user *optlen)
+{
+ struct raw6_sock *rp = raw6_sk(sk);
+ int val, len;

  if (get_user(len,optlen))
   return -EFAULT;
@@ -953,6 +965,52 @@ static int rawv6_getsockopt(struct sock
  return 0;
 }

+static int rawv6_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ switch(level) {
+  case SOL_RAW:
+   break;
+
+  case SOL_ICMPV6:
+   if (inet_sk(sk)->num != IPPROTO_ICMPV6)
+    return -EOPNOTSUPP;
+   return rawv6_geticmpfilter(sk, level, optname, optval,
+         optlen);
+  case SOL_IPV6:
+   if (optname == IPV6_CHECKSUM)
+    break;
+  default:
+   return ipv6_getsockopt(sk, level, optname, optval,
+            optlen);
+ };
+ return do_rawv6_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_rawv6_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ switch(level) {
+  case SOL_RAW:
+   break;
+
+  case SOL_ICMPV6:
+   if (inet_sk(sk)->num != IPPROTO_ICMPV6)
+    return -EOPNOTSUPP;
```

```
+   return rawv6_geticmpfilter(sk, level, optname, optval,
+         optlen);
+  case SOL_IPV6:
+   if (optname == IPV6_CHECKSUM)
+    break;
+  default:
+   return compat_ipv6_getsockopt(sk, level,
+     optname, optval, optlen);
+ };
+ return do_rawv6_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 static int rawv6_ioctl(struct sock *sk, int cmd, unsigned long arg)
 {
  switch(cmd) {
@@ -1008,6 +1066,10 @@ struct proto rawv6_prot = {
  .destroy = inet6_destroy_sock,
  .setsockopt = rawv6_setsockopt,
  .getsockopt = rawv6_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_rawv6_setsockopt,
+ .compat_getsockopt = compat_rawv6_getsockopt,
+#endif
  .sendmsg = rawv6_sendmsg,
  .recvmsg = rawv6_recvmsg,
  .bind =  rawv6_bind,
--- ./net/ipv6/tcp_ipv6.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv6/tcp_ipv6.c 2006-03-10 13:19:49.000000000 +0300
@@ -1308,6 +1308,10 @@ static struct inet_connection_sock_af_op

  .setsockopt = ipv6_setsockopt,
  .getsockopt = ipv6_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ipv6_setsockopt,
+ .compat_getsockopt = compat_ipv6_getsockopt,
+#endif
  .addr2sockaddr = inet6_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in6)
 };
@@ -1327,6 +1331,10 @@ static struct inet_connection_sock_af_op

  .setsockopt = ipv6_setsockopt,
  .getsockopt = ipv6_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ipv6_setsockopt,
+ .compat_getsockopt = compat_ipv6_getsockopt,
+#endif
```

```
  .addr2sockaddr = inet6_csk_addr2sockaddr,
  .sockaddr_len = sizeof(struct sockaddr_in6)
 };
@@ -1566,6 +1574,10 @@ struct proto tcpv6_prot = {
  .shutdown  = tcp_shutdown,
  .setsockopt  = tcp_setsockopt,
  .getsockopt  = tcp_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_tcp_setsockopt,
+ .compat_getsockopt = compat_tcp_getsockopt,
+#endif
  .sendmsg  = tcp_sendmsg,
  .recvmsg  = tcp_recvmsg,
  .backlog_rcv  = tcp_v6_do_rcv,
--- ./net/ipv6/udp.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/ipv6/udp.c 2006-03-10 13:26:47.000000000 +0300
@@ -880,16 +880,13 @@ static int udpv6_destroy_sock(struct soc
 /*
  * Socket option code for UDP
  */
-static int udpv6_setsockopt(struct sock *sk, int level, int optname,
+static int do_udpv6_setsockopt(struct sock *sk, int level, int optname,
      char __user *optval, int optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val;
  int err = 0;

- if (level != SOL_UDP)
-  return ipv6_setsockopt(sk, level, optname, optval, optlen);
-
  if(optlen<sizeof(int))
   return -EINVAL;

@@ -927,15 +924,31 @@ static int udpv6_setsockopt(struct sock
  return err;
 }

-static int udpv6_getsockopt(struct sock *sk, int level, int optname,
+static int udpv6_setsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int optlen)
+{
+ if (level != SOL_UDP)
+  return ipv6_setsockopt(sk, level, optname, optval, optlen);
+ return do_udpv6_setsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
```

```
+static int compat_udpv6_setsockopt(struct sock *sk, int level, int optname,
+  char __user *optval, int optlen)
+{
+ if (level != SOL_UDP)
+  return compat_ipv6_setsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udpv6_setsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
+static int do_udpv6_getsockopt(struct sock *sk, int level, int optname,
      char __user *optval, int __user *optlen)
 {
  struct udp_sock *up = udp_sk(sk);
  int val, len;

- if (level != SOL_UDP)
-  return ipv6_getsockopt(sk, level, optname, optval, optlen);
-
  if(get_user(len,optlen))
   return -EFAULT;

@@ -964,6 +977,25 @@ static int udpv6_getsockopt(struct sock
    return 0;
 }

+static int udpv6_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return ipv6_getsockopt(sk, level, optname, optval, optlen);
+ return do_udpv6_getsockopt(sk, level, optname, optval, optlen);
+}
+
+#ifdef CONFIG_COMPAT
+static int compat_udpv6_getsockopt(struct sock *sk, int level, int optname,
+     char __user *optval, int __user *optlen)
+{
+ if (level != SOL_UDP)
+  return compat_ipv6_getsockopt(sk, level,
+    optname, optval, optlen);
+ return do_udpv6_getsockopt(sk, level, optname, optval, optlen);
+}
+#endif
+
 static struct inet6_protocol udpv6_protocol = {
  .handler = udpv6_rcv,
  .err_handler = udpv6_err,
```

```
@@ -1046,6 +1078,10 @@ struct proto udpv6_prot = {
  .destroy = udpv6_destroy_sock,
  .setsockopt = udpv6_setsockopt,
  .getsockopt = udpv6_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_udpv6_setsockopt,
+ .compat_getsockopt = compat_udpv6_getsockopt,
+#endif
  .sendmsg = udpv6_sendmsg,
  .recvmsg = udpv6_recvmsg,
  .backlog_rcv = udpv6_queue_rcv_skb,
--- ./net/netfilter/nf_sockopt.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/netfilter/nf_sockopt.c 2006-03-10 13:46:12.000000000 +0300
@@ -131,3 +131,72 @@ int nf_getsockopt(struct sock *sk, int p
 }
 EXPORT_SYMBOL(nf_getsockopt);

+#ifdef CONFIG_COMPAT
+static int compat_nf_sockopt(struct sock *sk, int pf, int val,
+        char __user *opt, int *len, int get)
+{
+ struct list_head *i;
+ struct nf_sockopt_ops *ops;
+ int ret;
+
+ if (mutex_lock_interruptible(&nf_sockopt_mutex) != 0)
+  return -EINTR;
+
+ list_for_each(i, &nf_sockopts) {
+  ops = (struct nf_sockopt_ops *)i;
+  if (ops->pf == pf) {
+   if (get) {
+    if (val >= ops->get_optmin
+        && val < ops->get_optmax) {
+     ops->use++;
+     mutex_unlock(&nf_sockopt_mutex);
+     if (ops->compat_get)
+      ret = ops->compat_get(sk,
+       val, opt, len);
+     else
+      ret = ops->get(sk,
+       val, opt, len);
+     goto out;
+    }
+   } else {
+    if (val >= ops->set_optmin
+        && val < ops->set_optmax) {
+     ops->use++;
```

```
+     mutex_unlock(&nf_sockopt_mutex);
+     if (ops->compat_set)
+      ret = ops->compat_set(sk,
+       val, opt, *len);
+     else
+      ret = ops->set(sk,
+       val, opt, *len);
+     goto out;
+    }
+   }
+  }
+ }
+ mutex_unlock(&nf_sockopt_mutex);
+ return -ENOPROTOOPT;
+
+ out:
+ mutex_lock(&nf_sockopt_mutex);
+ ops->use--;
+ if (ops->cleanup_task)
+  wake_up_process(ops->cleanup_task);
+ mutex_unlock(&nf_sockopt_mutex);
+ return ret;
+}
+
+int compat_nf_setsockopt(struct sock *sk, int pf,
+  int val, char __user *opt, int len)
+{
+ return compat_nf_sockopt(sk, pf, val, opt, &len, 0);
+}
+EXPORT_SYMBOL(compat_nf_setsockopt);
+
+int compat_nf_getsockopt(struct sock *sk, int pf,
+  int val, char __user *opt, int *len)
+{
+ return compat_nf_sockopt(sk, pf, val, opt, len, 1);
+}
+EXPORT_SYMBOL(compat_nf_getsockopt);
+#endif
--- ./net/sctp/ipv6.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/sctp/ipv6.c 2006-03-10 13:21:06.000000000 +0300
@@ -875,6 +875,10 @@ static const struct proto_ops inet6_seqp
  .shutdown   = inet_shutdown,
  .setsockopt = sock_common_setsockopt,
  .getsockopt = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_sock_common_setsockopt,
+ .compat_getsockopt = compat_sock_common_getsockopt,
+#endif
```

```
  .sendmsg    = inet_sendmsg,
  .recvmsg    = sock_common_recvmsg,
  .mmap       = sock_no_mmap,
@@ -914,6 +918,10 @@ static struct sctp_af sctp_ipv6_specific
  .sctp_xmit       = sctp_v6_xmit,
  .setsockopt      = ipv6_setsockopt,
  .getsockopt      = ipv6_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt = compat_ipv6_setsockopt,
+ .compat_getsockopt = compat_ipv6_getsockopt,
+#endif
  .get_dst  = sctp_v6_get_dst,
  .get_saddr  = sctp_v6_get_saddr,
  .copy_addrlist   = sctp_v6_copy_addrlist,
--- ./net/sctp/protocol.c.compat 2006-03-10 11:58:11.000000000 +0300
+++ ./net/sctp/protocol.c 2006-03-10 12:24:11.000000000 +0300
@@ -845,6 +845,10 @@ static const struct proto_ops inet_seqpa
  .shutdown    = inet_shutdown,     /* Looks harmless.  */
  .setsockopt  = sock_common_setsockopt,   /* IP_SOL IP_OPTION is a problem. */
  .getsockopt  = sock_common_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt  = compat_sock_common_setsockopt,
+ .compat_getsockopt  = compat_sock_common_getsockopt,
+#endif
  .sendmsg     = inet_sendmsg,
  .recvmsg     = sock_common_recvmsg,
  .mmap        = sock_no_mmap,
@@ -883,6 +887,10 @@ static struct sctp_af sctp_ipv4_specific
  .sctp_xmit      = sctp_v4_xmit,
  .setsockopt     = ip_setsockopt,
  .getsockopt     = ip_getsockopt,
+#ifdef CONFIG_COMPAT
+ .compat_setsockopt     = compat_ip_setsockopt,
+ .compat_getsockopt     = compat_ip_getsockopt,
+#endif
  .get_dst = sctp_v4_get_dst,
  .get_saddr = sctp_v4_get_saddr,
  .copy_addrlist  = sctp_v4_copy_addrlist,
```

File Attachments

1) diff-ms-sockopts-compat-20060310, downloaded 674 times

Subject: Re: [PATCH] {get|set}sockopt compatibility layer
Posted by davem on Fri, 10 Mar 2006 11:34:46 GMT
View Forum Message <> Reply to Message

From: Dmitry Mishin <dim@openvz.org>

Date: Fri, 10 Mar 2006 14:21:10 +0300

> This patch extends {get|set}sockopt compatibility layer in order to move
> protocol specific parts to their place and avoid huge universal net/compat.c
> file in the future.
>
> Signed-off-by: Dmitry Mishin <dim@openvz.org>

Applied, thanks Dmitry.

Please give "-p1" format patches in the future, I fixed your's
up by hand so could feed it to git.

Thanks again.

Subject: [PATCH] iptables 32bit compat layer
Posted by Mishin Dmitry on Thu, 23 Mar 2006 10:24:33 GMT
View Forum Message <> Reply to Message

This patch extends current iptables compatibility layer in order to get
32bit iptables to work on 64bit kernel. Current layer is insufficient due to
alignment checks both in kernel and user space tools.

Patch is for current net-2.6.17 with addition of move of ipt_entry_{match|
target} definitions to xt_entry_{match|target}.

Signed-off-by: Dmitry Mishin <dim@openvz.org>
Acked-off-by: Kirill Korotaev <dev@openvz.org>

--
Thanks,
Dmitry.

diff --git a/include/linux/netfilter/x_tables.h b/include/linux/netfilter/x_tables.h
index ad72a4f..9d4fa6d 100644
--- a/include/linux/netfilter/x_tables.h
+++ b/include/linux/netfilter/x_tables.h
@@ -142,6 +142,12 @@ struct xt_counters_info
 #define ASSERT_WRITE_LOCK(x)
 #include <linux/netfilter_ipv4/listhelp.h>

+#ifdef CONFIG_COMPAT
+#define COMPAT_TO_USER 1
+#define COMPAT_FROM_USER -1
+#define COMPAT_CALC_SIZE 0
+#endif
+

```
 struct xt_match
 {
  struct list_head list;
@@ -175,6 +181,9 @@ struct xt_match
  void (*destroy)(const struct xt_match *match, void *matchinfo,
    unsigned int matchinfosize);

+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *match, void **dstptr, int *size, int convert);
+
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;

@@ -220,6 +229,9 @@ struct xt_target
  void (*destroy)(const struct xt_target *target, void *targinfo,
    unsigned int targinfosize);

+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *target, void **dstptr, int *size, int convert);
+
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;

@@ -314,6 +326,61 @@ extern void xt_proto_fini(int af);
 extern struct xt_table_info *xt_alloc_table_info(unsigned int size);
 extern void xt_free_table_info(struct xt_table_info *info);

+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
+
+struct compat_xt_entry_match
+{
+ union {
+  struct {
+   u_int16_t match_size;
+   char name[XT_FUNCTION_MAXNAMELEN - 1];
+   u_int8_t revision;
+  } user;
+  u_int16_t match_size;
+ } u;
+ unsigned char data[0];
+};
+
+struct compat_xt_entry_target
+{
+ union {
+  struct {
+   u_int16_t target_size;
```

```
+   char name[XT_FUNCTION_MAXNAMELEN - 1];
+   u_int8_t revision;
+  } user;
+  u_int16_t target_size;
+ } u;
+ unsigned char data[0];
+};
+
+/* FIXME: this works only on 32 bit tasks
+ * need to change whole approach in order to calculate align as function of
+ * current task alignment */
+
+struct compat_xt_counters
+{
+ u_int32_t cnt[4];
+};
+
+struct compat_xt_counters_info
+{
+ char name[XT_TABLE_MAXNAMELEN];
+ compat_uint_t num_counters;
+ struct compat_xt_counters counters[0];
+};
+
+#define COMPAT_XT_ALIGN(s) (((s) + (__alignof__(struct compat_xt_counters)-1)) \
+  & ~(__alignof__(struct compat_xt_counters)-1))
+
+extern void xt_compat_lock(int af);
+extern void xt_compat_unlock(int af);
+extern int xt_compat_match(void *match, void **dstptr, int *size, int convert);
+extern int xt_compat_target(void *target, void **dstptr, int *size,
+  int convert);
+
+#endif /* CONFIG_COMPAT */
 #endif /* __KERNEL__ */

 #endif /* _X_TABLES_H */
diff --git a/include/linux/netfilter_ipv4/ip_tables.h b/include/linux/netfilter_ipv4/ip_tables.h
index 56eebc6..9e11e32 100644
--- a/include/linux/netfilter_ipv4/ip_tables.h
+++ b/include/linux/netfilter_ipv4/ip_tables.h
@@ -312,5 +312,23 @@ extern unsigned int ipt_do_table(struct
      void *userdata);

 #define IPT_ALIGN(s) XT_ALIGN(s)
+
+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
```

```
+
+struct compat_ipt_entry
+{
+ struct ipt_ip ip;
+ compat_uint_t nfcache;
+ u_int16_t target_offset;
+ u_int16_t next_offset;
+ compat_uint_t comefrom;
+ struct compat_xt_counters counters;
+ unsigned char elems[0];
+};
+
+#define COMPAT_IPT_ALIGN(s)  COMPAT_XT_ALIGN(s)
+
+#endif /* CONFIG_COMPAT */
 #endif /*__KERNEL__*/
 #endif /* _IPTABLES_H */
diff --git a/net/compat.c b/net/compat.c
index 13177a1..6a7028e 100644
--- a/net/compat.c
+++ b/net/compat.c
@@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
  int err;
  struct socket *sock;

- /* SO_SET_REPLACE seems to be the same in all levels */
- if (optname == IPT_SO_SET_REPLACE)
+ if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
   return do_netfilter_replace(fd, level, optname,
          optval, optlen);

diff --git a/net/ipv4/netfilter/ip_tables.c b/net/ipv4/netfilter/ip_tables.c
index 39705f9..9149b24 100644
--- a/net/ipv4/netfilter/ip_tables.c
+++ b/net/ipv4/netfilter/ip_tables.c
@@ -24,6 +24,7 @@
 #include <linux/module.h>
 #include <linux/icmp.h>
 #include <net/ip.h>
+#include <net/compat.h>
 #include <asm/uaccess.h>
 #include <linux/mutex.h>
 #include <linux/proc_fs.h>
@@ -799,17 +800,11 @@ get_counters(const struct xt_table_info
  }
 }

-static int
```

```
-copy_entries_to_user(unsigned int total_size,
-       struct ipt_table *table,
-       void __user *userptr)
+static inline struct xt_counters * alloc_counters(struct ipt_table *table)
 {
- unsigned int off, num, countersize;
- struct ipt_entry *e;
+ unsigned int countersize;
  struct xt_counters *counters;
  struct xt_table_info *private = table->private;
- int ret = 0;
- void *loc_cpu_entry;

  /* We need atomic snapshot of counters: rest doesn't change
     (other than comefrom, which userspace doesn't care
@@ -818,13 +813,32 @@ copy_entries_to_user(unsigned int total_
  counters = vmalloc_node(countersize, numa_node_id());

  if (counters == NULL)
-  return -ENOMEM;
+  return ERR_PTR(-ENOMEM);

  /* First, sum counters... */
  write_lock_bh(&table->lock);
  get_counters(private, counters);
  write_unlock_bh(&table->lock);

+ return counters;
+}
+
+static int
+copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table,
+       void __user *userptr)
+{
+ unsigned int off, num;
+ struct ipt_entry *e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ int ret = 0;
+ void *loc_cpu_entry;
+
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
  /* choose the copy that is on our node/cpu, ...
   * This choice is lazy (because current thread is
```

```
   * allowed to migrate to another cpu)
@@ -884,44 +898,899 @@ copy_entries_to_user(unsigned int total_
  return ret;
 }

+#ifdef CONFIG_COMPAT
+struct compat_delta {
+ struct compat_delta *next;
+ u_int16_t offset;
+ short delta;
+};
+
+static struct compat_delta *compat_offsets = NULL;
+
+static int compat_add_offset(u_int16_t offset, short delta)
+{
+ struct compat_delta *tmp;
+
+ tmp = kmalloc(sizeof(struct compat_delta), GFP_KERNEL);
+ if (!tmp)
+  return -ENOMEM;
+ tmp->offset = offset;
+ tmp->delta = delta;
+ if (compat_offsets) {
+  tmp->next = compat_offsets->next;
+  compat_offsets->next = tmp;
+ } else {
+  compat_offsets = tmp;
+  tmp->next = NULL;
+ }
+ return 0;
+}
+
+static void compat_flush_offsets(void)
+{
+ struct compat_delta *tmp, *next;
+
+ if (compat_offsets) {
+  for(tmp = compat_offsets; tmp; tmp = next) {
+   next = tmp->next;
+   kfree(tmp);
+  }
+  compat_offsets = NULL;
+ }
+}
+
+static short compat_calc_jump(u_int16_t offset)
+{
```

```
+ struct compat_delta *tmp;
+ short delta;
+
+ for(tmp = compat_offsets, delta = 0; tmp; tmp = tmp->next)
+  if (tmp->offset < offset)
+   delta += tmp->delta;
+ return delta;
+}
+
+struct compat_ipt_standard_target
+{
+ struct compat_xt_entry_target target;
+ compat_int_t verdict;
+};
+
+#define IPT_ST_OFFSET (sizeof(struct ipt_standard_target) - \
+    sizeof(struct compat_ipt_standard_target))
+
+struct compat_ipt_standard
+{
+ struct compat_ipt_entry entry;
+ struct compat_ipt_standard_target target;
+};
+
+static int compat_ipt_standard_fn(void *target,
+  void **dstptr, int *size, int convert)
+{
+ struct compat_ipt_standard_target compat_st, *pcompat_st;
+ struct ipt_standard_target st, *pst;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pst = (struct ipt_standard_target *)target;
+   memcpy(&compat_st.target, &pst->target,
+     sizeof(struct ipt_entry_target));
+   compat_st.verdict = pst->verdict;
+   if (compat_st.verdict > 0)
+    compat_st.verdict -=
+     compat_calc_jump(compat_st.verdict);
+   compat_st.target.u.user.target_size =
+   sizeof(struct compat_ipt_standard_target);
+   if (__copy_to_user(*dstptr, &compat_st,
+    sizeof(struct compat_ipt_standard_target)))
+    ret = -EFAULT;
+   *size -= IPT_ST_OFFSET;
+   *dstptr += sizeof(struct compat_ipt_standard_target);
```

```
+   break;
+  case COMPAT_FROM_USER:
+   pcompat_st =
+    (struct compat_ipt_standard_target *)target;
+   memcpy(&st.target, &pcompat_st->target,
+     sizeof(struct ipt_entry_target));
+   st.verdict = pcompat_st->verdict;
+   if (st.verdict > 0)
+    st.verdict += compat_calc_jump(st.verdict);
+   st.target.u.user.target_size =
+   sizeof(struct ipt_standard_target);
+   memcpy(*dstptr, &st,
+     sizeof(struct ipt_standard_target));
+   *size += IPT_ST_OFFSET;
+   *dstptr += sizeof(struct ipt_standard_target);
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += IPT_ST_OFFSET;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+
+static inline int
+compat_calc_match(struct ipt_entry_match *m, int * size)
+{
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+ else
+  xt_compat_match(m, NULL, size, COMPAT_CALC_SIZE);
+ return 0;
+}
+
+static int compat_calc_entry(struct ipt_entry *e, struct xt_table_info *info,
+  void *base, struct xt_table_info *newinfo)
+{
+ struct ipt_entry_target *t;
+ u_int16_t entry_offset;
+ int off, i, ret;
+
+ off = 0;
+ entry_offset = (void *)e - base;
+ IPT_MATCH_ITERATE(e, compat_calc_match, &off);
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
```

```
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ else
+  xt_compat_target(t, NULL, &off, COMPAT_CALC_SIZE);
+ newinfo->size -= off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  return ret;
+
+ for (i = 0; i< NF_IP_NUMHOOKS; i++) {
+  if (info->hook_entry[i] && (e < (struct ipt_entry *)
+    (base + info->hook_entry[i])))
+   newinfo->hook_entry[i] -= off;
+  if (info->underflow[i] && (e < (struct ipt_entry *)
+    (base + info->underflow[i])))
+   newinfo->underflow[i] -= off;
+ }
+ return 0;
+}
+
+static int compat_table_info(struct xt_table_info *info,
+  struct xt_table_info *newinfo)
+{
+ void *loc_cpu_entry;
+ int i;
+
+ if (!newinfo || !info)
+  return -EINVAL;
+
+ memset(newinfo, 0, sizeof(struct xt_table_info));
+ newinfo->size = info->size;
+ newinfo->number = info->number;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ loc_cpu_entry = info->entries[raw_smp_processor_id()];
+ return IPT_ENTRY_ITERATE(loc_cpu_entry, info->size,
+   compat_calc_entry, info, loc_cpu_entry, newinfo);
+}
+#endif
+
+static int get_info(void __user *user, int *len, int compat)
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ struct ipt_table *t;
+ int ret;
+
+ if (*len != sizeof(struct ipt_getinfo)) {
```

```
+  duprintf("length %u != %u\n", *len,
+   (unsigned int)sizeof(struct ipt_getinfo));
+  return -EINVAL;
+ }
+
+ if (copy_from_user(name, user, sizeof(name)) != 0)
+  return -EFAULT;
+
+ name[IPT_TABLE_MAXNAMELEN-1] = '\0';
+#ifdef CONFIG_COMPAT
+ if (compat)
+  xt_compat_lock(AF_INET);
+#endif
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+   "iptable_%s", name);
+ if (t && !IS_ERR(t)) {
+  struct ipt_getinfo info;
+  struct xt_table_info *private = t->private;
+
+#ifdef CONFIG_COMPAT
+  if (compat) {
+   struct xt_table_info tmp;
+   ret = compat_table_info(private, &tmp);
+   compat_flush_offsets();
+   private =  &tmp;
+  }
+#endif
+  info.valid_hooks = t->valid_hooks;
+  memcpy(info.hook_entry, private->hook_entry,
+    sizeof(info.hook_entry));
+  memcpy(info.underflow, private->underflow,
+    sizeof(info.underflow));
+  info.num_entries = private->number;
+  info.size = private->size;
+  strcpy(info.name, name);
+
+  if (copy_to_user(user, &info, *len) != 0)
+   ret = -EFAULT;
+  else
+   ret = 0;
+
+  xt_table_unlock(t);
+  module_put(t->me);
+ } else
+  ret = t ? PTR_ERR(t) : -ENOENT;
+#ifdef CONFIG_COMPAT
+ if (compat)
+  xt_compat_unlock(AF_INET);
```

```
+#endif
+ return ret;
+}
+
+static int
+get_entries(struct ipt_get_entries __user *uptr, int *len)
+{
+ int ret;
+ struct ipt_get_entries get;
+ struct ipt_table *t;
+
+ if (*len < sizeof(get)) {
+  duprintf("get_entries: %u < %d\n", *len,
+    (unsigned int)sizeof(get));
+  return -EINVAL;
+ }
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;
+ if (*len != sizeof(struct ipt_get_entries) + get.size) {
+  duprintf("get_entries: %u != %u\n", *len,
+    (unsigned int)(sizeof(struct ipt_get_entries) +
+    get.size));
+  return -EINVAL;
+ }
+
+ t = xt_find_table_lock(AF_INET, get.name);
+ if (t && !IS_ERR(t)) {
+  struct xt_table_info *private = t->private;
+  duprintf("t->private->number = %u\n",
+    private->number);
+  if (get.size == private->size)
+   ret = copy_entries_to_user(private->size,
+         t, uptr->entrytable);
+  else {
+   duprintf("get_entries: I've got %u not %u!\n",
+     private->size,
+     get.size);
+   ret = -EINVAL;
+  }
+  module_put(t->me);
+  xt_table_unlock(t);
+ } else
+  ret = t ? PTR_ERR(t) : -ENOENT;
+
+ return ret;
+}
+
+static int
```

```
+__do_replace(const char *name, unsigned int valid_hooks,
+  struct xt_table_info *newinfo, unsigned int num_counters,
+  void __user *counters_ptr)
+{
+ int ret;
+ struct ipt_table *t;
+ struct xt_table_info *oldinfo;
+ struct xt_counters *counters;
+ void *loc_cpu_old_entry;
+
+ ret = 0;
+ counters = vmalloc(num_counters * sizeof(struct xt_counters));
+ if (!counters) {
+  ret = -ENOMEM;
+  goto out;
+ }
+
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+        "iptable_%s", name);
+ if (!t || IS_ERR(t)) {
+  ret = t ? PTR_ERR(t) : -ENOENT;
+  goto free_newinfo_counters_untrans;
+ }
+
+ /* You lied! */
+ if (valid_hooks != t->valid_hooks) {
+  duprintf("Valid hook crap: %08X vs %08X\n",
+    valid_hooks, t->valid_hooks);
+  ret = -EINVAL;
+  goto put_module;
+ }
+
+ oldinfo = xt_replace_table(t, num_counters, newinfo, &ret);
+ if (!oldinfo)
+  goto put_module;
+
+ /* Update module usage count based on number of rules */
+ duprintf("do_replace: oldnum=%u, initnum=%u, newnum=%u\n",
+  oldinfo->number, oldinfo->initial_entries, newinfo->number);
+ if ((oldinfo->number > oldinfo->initial_entries) ||
+     (newinfo->number <= oldinfo->initial_entries))
+  module_put(t->me);
+ if ((oldinfo->number > oldinfo->initial_entries) &&
+     (newinfo->number <= oldinfo->initial_entries))
+  module_put(t->me);
+
+ /* Get the old counters. */
+ get_counters(oldinfo, counters);
```

```
+ /* Decrease module usage counts and free resource */
+ loc_cpu_old_entry = oldinfo->entries[raw_smp_processor_id()];
+ IPT_ENTRY_ITERATE(loc_cpu_old_entry, oldinfo->size, cleanup_entry,NULL);
+ xt_free_table_info(oldinfo);
+ if (copy_to_user(counters_ptr, counters,
+    sizeof(struct xt_counters) * num_counters) != 0)
+  ret = -EFAULT;
+ vfree(counters);
+ xt_table_unlock(t);
+ return ret;
+
+ put_module:
+ module_put(t->me);
+ xt_table_unlock(t);
+ free_newinfo_counters_untrans:
+ vfree(counters);
+ out:
+ return ret;
+}
+
+static int
+do_replace(void __user *user, unsigned int len)
+{
+ int ret;
+ struct ipt_replace tmp;
+ struct xt_table_info *newinfo;
+ void *loc_cpu_entry;
+
+ if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
+  return -EFAULT;
+
+ /* Hack: Causes ipchains to give correct error msg --RR */
+ if (len != sizeof(tmp) + tmp.size)
+  return -ENOPROTOOPT;
+
+ /* overflow check */
+ if (tmp.size >= (INT_MAX - sizeof(struct xt_table_info)) / NR_CPUS -
+   SMP_CACHE_BYTES)
+  return -ENOMEM;
+ if (tmp.num_counters >= INT_MAX / sizeof(struct xt_counters))
+  return -ENOMEM;
+
+ newinfo = xt_alloc_table_info(tmp.size);
+ if (!newinfo)
+  return -ENOMEM;
+
+ /* choose the copy that is our node/cpu */
+ loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
```

```
+ if (copy_from_user(loc_cpu_entry, user + sizeof(tmp),
+      tmp.size) != 0) {
+  ret = -EFAULT;
+  goto free_newinfo;
+ }
+
+ ret = translate_table(tmp.name, tmp.valid_hooks,
+         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
+         tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
+  goto free_newinfo;
+
+ duprintf("ip_tables: Translated table\n");
+
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         tmp.counters);
+ if (ret)
+  goto free_newinfo_untrans;
+ return 0;
+
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}
+
+/* We're lazy, and add to the first CPU; overflow works its fey magic
+ * and everything is OK. */
+static inline int
+add_counter_to_entry(struct ipt_entry *e,
+       const struct xt_counters addme[],
+       unsigned int *i)
+{
+#if 0
+ duprintf("add_counter: Entry %u %lu/%lu + %lu/%lu\n",
+    *i,
+    (long unsigned int)e->counters.pcnt,
+    (long unsigned int)e->counters.bcnt,
+    (long unsigned int)addme[*i].pcnt,
+    (long unsigned int)addme[*i].bcnt);
+#endif
+
+ ADD_COUNTER(e->counters, addme[*i].bcnt, addme[*i].pcnt);
+
+ (*i)++;
+ return 0;
```

```
+}
+
+static int
+do_add_counters(void __user *user, unsigned int len, int compat)
+{
+ unsigned int i;
+ struct xt_counters_info tmp;
+ struct xt_counters *paddc;
+ unsigned int num_counters;
+ char *name;
+ int size;
+ void *ptmp;
+ struct ipt_table *t;
+ struct xt_table_info *private;
+ int ret = 0;
+ void *loc_cpu_entry;
+#ifdef CONFIG_COMPAT
+ struct compat_xt_counters_info compat_tmp;
+
+ if (compat) {
+  ptmp = &compat_tmp;
+  size = sizeof(struct compat_xt_counters_info);
+ } else
+#endif
+ {
+  ptmp = &tmp;
+  size = sizeof(struct xt_counters_info);
+ }
+
+ if (copy_from_user(ptmp, user, size) != 0)
+  return -EFAULT;
+
+#ifdef CONFIG_COMPAT
+ if (compat) {
+  num_counters = compat_tmp.num_counters;
+  name = compat_tmp.name;
+ } else
+#endif
+ {
+  num_counters = tmp.num_counters;
+  name = tmp.name;
+ }
+
+ if (len != size + num_counters * sizeof(struct xt_counters))
+  return -EINVAL;
+
+ paddc = vmalloc_node(len - size, numa_node_id());
+ if (!paddc)
```

```
+  return -ENOMEM;
+
+ if (copy_from_user(paddc, user + size, len - size) != 0) {
+  ret = -EFAULT;
+  goto free;
+ }
+
+ t = xt_find_table_lock(AF_INET, name);
+ if (!t || IS_ERR(t)) {
+  ret = t ? PTR_ERR(t) : -ENOENT;
+  goto free;
+ }
+
+ write_lock_bh(&t->lock);
+ private = t->private;
+ if (private->number != num_counters) {
+  ret = -EINVAL;
+  goto unlock_up_free;
+ }
+
+ i = 0;
+ /* Choose the copy that is on our node */
+ loc_cpu_entry = private->entries[raw_smp_processor_id()];
+ IPT_ENTRY_ITERATE(loc_cpu_entry,
+     private->size,
+     add_counter_to_entry,
+     paddc,
+     &i);
+ unlock_up_free:
+ write_unlock_bh(&t->lock);
+ xt_table_unlock(t);
+ module_put(t->me);
+ free:
+ vfree(paddc);
+
+ return ret;
+}
+
+#ifdef CONFIG_COMPAT
+struct compat_ipt_replace {
+ char   name[IPT_TABLE_MAXNAMELEN];
+ u32   valid_hooks;
+ u32   num_entries;
+ u32   size;
+ u32   hook_entry[NF_IP_NUMHOOKS];
+ u32   underflow[NF_IP_NUMHOOKS];
+ u32   num_counters;
+ compat_uptr_t  counters; /* struct ipt_counters * */
```

```
+ struct compat_ipt_entry entries[0];
+};
+
+static inline int compat_copy_match_to_user(struct ipt_entry_match *m,
+  void __user **dstptr, compat_uint_t *size)
+{
+ if (m->u.kernel.match->compat)
+  return m->u.kernel.match->compat(m, dstptr, size,
+    COMPAT_TO_USER);
+ else
+  return xt_compat_match(m, dstptr, size, COMPAT_TO_USER);
+}
+
+static int compat_copy_entry_to_user(struct ipt_entry *e,
+  void __user **dstptr, compat_uint_t *size)
+{
+ struct ipt_entry_target __user *t;
+ struct compat_ipt_entry __user *ce;
+ u_int16_t target_offset, next_offset;
+ compat_uint_t origsize;
+ int ret;
+
+ ret = -EFAULT;
+ origsize = *size;
+ ce = (struct compat_ipt_entry __user *)*dstptr;
+ if (__copy_to_user(ce, e, sizeof(struct ipt_entry)))
+  goto out;
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_to_user, dstptr, size);
+ target_offset = e->target_offset - (origsize - *size);
+ if (ret)
+  goto out;
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
+  ret = t->u.kernel.target->compat(t, dstptr, size,
+    COMPAT_TO_USER);
+ else
+  ret = xt_compat_target(t, dstptr, size, COMPAT_TO_USER);
+ if (ret)
+  goto out;
+ ret = -EFAULT;
+ next_offset = e->next_offset - (origsize - *size);
+ if (__put_user(target_offset, &ce->target_offset))
+  goto out;
+ if (__put_user(next_offset, &ce->next_offset))
+  goto out;
+ return 0;
```

```
+out:
+ return ret;
+}
+
+static inline int
+compat_check_calc_match(struct ipt_entry_match *m,
+     const char *name,
+     const struct ipt_ip *ip,
+     unsigned int hookmask,
+     int *size, int *i)
+{
+ struct ipt_match *match;
+
+ match = try_then_request_module(xt_find_match(AF_INET, m->u.user.name,
+         m->u.user.revision),
+     "ipt_%s", m->u.user.name);
+ if (IS_ERR(match) || !match) {
+  duprintf("compat_check_calc_match: `%s' not found\n",
+    m->u.user.name);
+  return match ? PTR_ERR(match) : -ENOENT;
+ }
+ m->u.kernel.match = match;
+
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+ else
+  xt_compat_match(m, NULL, size, COMPAT_CALC_SIZE);
+
+ (*i)++;
+ return 0;
+}
+
+static inline int
+check_compat_entry_size_and_hooks(struct ipt_entry *e,
+      struct xt_table_info *newinfo,
+      unsigned int *size,
+      unsigned char *base,
+      unsigned char *limit,
+      unsigned int *hook_entries,
+      unsigned int *underflows,
+      unsigned int *i,
+      const char *name)
+{
+ struct ipt_entry_target *t;
+ struct ipt_target *target;
+ u_int16_t entry_offset;
+ int ret, off, h, j;
+
```

```
+ duprintf("check_compat_entry_size_and_hooks %p\n", e);
+ if ((unsigned long)e % __alignof__(struct compat_ipt_entry) != 0
+     || (unsigned char *)e + sizeof(struct compat_ipt_entry) >= limit) {
+  duprintf("Bad offset %p, limit = %p\n", e, limit);
+  return -EINVAL;
+ }
+
+ if (e->next_offset < sizeof(struct compat_ipt_entry) +
+   sizeof(struct compat_xt_entry_target)) {
+  duprintf("checking: element %p size %u\n",
+    e, e->next_offset);
+  return -EINVAL;
+ }
+
+ if (!ip_checkentry(&e->ip)) {
+  duprintf("ip_tables: ip check failed %p %s.\n", e, name);
+  return -EINVAL;
+ }
+
+ off = 0;
+ entry_offset = (void *)e - (void *)base;
+ j = 0;
+ ret = IPT_MATCH_ITERATE(e, compat_check_calc_match, name, &e->ip,
+   e->comefrom, &off, &j);
+ if (ret != 0)
+  goto out;
+
+ t = ipt_get_target(e);
+ target = try_then_request_module(xt_find_target(AF_INET,
+          t->u.user.name,
+          t->u.user.revision),
+      "ipt_%s", t->u.user.name);
+ if (IS_ERR(target) || !target) {
+  duprintf("check_entry: `%s' not found\n", t->u.user.name);
+  ret = target ? PTR_ERR(target) : -ENOENT;
+  goto out;
+ }
+ t->u.kernel.target = target;
+
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ else
+  xt_compat_target(t, NULL, &off, COMPAT_CALC_SIZE);
+ *size += off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  goto out;
+
```

```
+ /* Check hooks & underflows */
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)e - base == hook_entries[h])
+   newinfo->hook_entry[h] = hook_entries[h];
+  if ((unsigned char *)e - base == underflows[h])
+   newinfo->underflow[h] = underflows[h];
+ }
+
+ /* Clear counters and comefrom */
+ e->counters = ((struct ipt_counters) { 0, 0 });
+ e->comefrom = 0;
+
+ (*i)++;
+ return 0;
+out:
+ IPT_MATCH_ITERATE(e, cleanup_match, &j);
+ return ret;
+}
+
+static inline int compat_copy_match_from_user(struct ipt_entry_match *m,
+ void **dstptr, compat_uint_t *size, const char *name,
+ const struct ipt_ip *ip, unsigned int hookmask)
+{
+ struct ipt_entry_match *dm;
+ struct ipt_match *match;
+ int ret;
+
+ dm = (struct ipt_entry_match *)*dstptr;
+ match = m->u.kernel.match;
+ if (match->compat)
+  match->compat(m, dstptr, size, COMPAT_FROM_USER);
+ else
+  xt_compat_match(m, dstptr, size, COMPAT_FROM_USER);
+
+ ret = xt_check_match(match, AF_INET, dm->u.match_size - sizeof(*dm),
+       name, hookmask, ip->proto,
+       ip->invflags & IPT_INV_PROTO);
+ if (ret)
+  return ret;
+
+ if (m->u.kernel.match->checkentry
+     && !m->u.kernel.match->checkentry(name, ip, match, dm->data,
+          dm->u.match_size - sizeof(*dm),
+          hookmask)) {
+  duprintf("ip_tables: check failed for `%s'.\n",
+    m->u.kernel.match->name);
+  return -EINVAL;
+ }
```

```
+ return 0;
+}
+
+static int compat_copy_entry_from_user(struct ipt_entry *e, void **dstptr,
+ unsigned int *size, const char *name,
+ struct xt_table_info *newinfo, unsigned char *base)
+{
+ struct ipt_entry_target *t;
+ struct ipt_target *target;
+ struct ipt_entry *de;
+ unsigned int origsize;
+ int ret, h;
+
+ ret = 0;
+ origsize = *size;
+ de = (struct ipt_entry *)*dstptr;
+ memcpy(de, e, sizeof(struct ipt_entry));
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_from_user, dstptr, size,
+   name, &de->ip, de->comefrom);
+ if (ret)
+  goto out;
+ de->target_offset = e->target_offset - (origsize - *size);
+ t = ipt_get_target(e);
+ target = t->u.kernel.target;
+ if (target->compat)
+  target->compat(t, dstptr, size, COMPAT_FROM_USER);
+ else
+  xt_compat_target(t, dstptr, size, COMPAT_FROM_USER);
+
+ de->next_offset = e->next_offset - (origsize - *size);
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)de - base < newinfo->hook_entry[h])
+   newinfo->hook_entry[h] -= origsize - *size;
+  if ((unsigned char *)de - base < newinfo->underflow[h])
+   newinfo->underflow[h] -= origsize - *size;
+ }
+
+ t = ipt_get_target(de);
+ target = t->u.kernel.target;
+ ret = xt_check_target(target, AF_INET, t->u.target_size - sizeof(*t),
+         name, e->comefrom, e->ip.proto,
+         e->ip.invflags & IPT_INV_PROTO);
+ if (ret)
+  goto out;
+
+ ret = -EINVAL;
```

```
+ if (t->u.kernel.target == &ipt_standard_target) {
+  if (!standard_check(t, *size))
+   goto out;
+ } else if (t->u.kernel.target->checkentry
+     && !t->u.kernel.target->checkentry(name, de, target,
+    t->data, t->u.target_size - sizeof(*t),
+    de->comefrom)) {
+  duprintf("ip_tables: compat: check failed for `%s'.\n",
+    t->u.kernel.target->name);
+  goto out;
+ }
+ ret = 0;
+out:
+ return ret;
+}
+
 static int
-get_entries(const struct ipt_get_entries *entries,
-     struct ipt_get_entries __user *uptr)
+translate_compat_table(const char *name,
+  unsigned int valid_hooks,
+  struct xt_table_info **pinfo,
+  void **pentry0,
+  unsigned int total_size,
+  unsigned int number,
+  unsigned int *hook_entries,
+  unsigned int *underflows)
 {
+ unsigned int i;
+ struct xt_table_info *newinfo, *info;
+ void *pos, *entry0, *entry1;
+ unsigned int size;
  int ret;
- struct ipt_table *t;

- t = xt_find_table_lock(AF_INET, entries->name);
- if (t && !IS_ERR(t)) {
-  struct xt_table_info *private = t->private;
-  duprintf("t->private->number = %u\n",
-    private->number);
-  if (entries->size == private->size)
-   ret = copy_entries_to_user(private->size,
-         t, uptr->entrytable);
-  else {
-   duprintf("get_entries: I've got %u not %u!\n",
-     private->size,
-     entries->size);
-   ret = -EINVAL;
```

```
+ info = *pinfo;
+ entry0 = *pentry0;
+ size = total_size;
+ info->number = number;
+
+ /* Init all hooks to impossible value. */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  info->hook_entry[i] = 0xFFFFFFFF;
+  info->underflow[i] = 0xFFFFFFFF;
+ }
+
+ duprintf("translate_compat_table: size %u\n", info->size);
+ i = 0;
+ xt_compat_lock(AF_INET);
+ /* Walk through entries, checking offsets. */
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+    check_compat_entry_size_and_hooks,
+    info, &size, entry0,
+    entry0 + total_size,
+    hook_entries, underflows, &i, name);
+ if (ret != 0)
+  goto out_unlock;
+
+ ret = -EINVAL;
+ if (i != number) {
+  duprintf("translate_compat_table: %u not %u entries\n",
+    i, number);
+  goto out_unlock;
+ }
+
+ /* Check hooks all assigned */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  /* Only hooks which are valid */
+  if (!(valid_hooks & (1 << i)))
+   continue;
+  if (info->hook_entry[i] == 0xFFFFFFFF) {
+   duprintf("Invalid hook entry %u %u\n",
+     i, hook_entries[i]);
+   goto out_unlock;
   }
-  module_put(t->me);
-  xt_table_unlock(t);
- } else
-  ret = t ? PTR_ERR(t) : -ENOENT;
+  if (info->underflow[i] == 0xFFFFFFFF) {
+   duprintf("Invalid underflow %u %u\n",
+     i, underflows[i]);
+   goto out_unlock;
```

```
+  }
+ }
+
+ ret = -ENOMEM;
+ newinfo = xt_alloc_table_info(size);
+ if (!newinfo)
+  goto out_unlock;
+
+ newinfo->number = number;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ entry1 = newinfo->entries[raw_smp_processor_id()];
+ pos = entry1;
+ size =  total_size;
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+   compat_copy_entry_from_user, &pos, &size,
+   name, newinfo, entry1);
+ compat_flush_offsets();
+ xt_compat_unlock(AF_INET);
+ if (ret)
+  goto free_newinfo;
+
+ ret = -ELOOP;
+ if (!mark_source_chains(newinfo, valid_hooks, entry1))
+  goto free_newinfo;
+
+ /* And one copy for every other CPU */
+ for_each_cpu(i)
+  if (newinfo->entries[i] && newinfo->entries[i] != entry1)
+   memcpy(newinfo->entries[i], entry1, newinfo->size);
+
+ *pinfo = newinfo;
+ *pentry0 = entry1;
+ xt_free_table_info(info);
+ return 0;

+free_newinfo:
+ xt_free_table_info(newinfo);
+out:
  return ret;
+out_unlock:
+ xt_compat_unlock(AF_INET);
+ goto out;
 }

 static int
```

```
-do_replace(void __user *user, unsigned int len)
+compat_do_replace(void __user *user, unsigned int len)
 {
  int ret;
- struct ipt_replace tmp;
- struct ipt_table *t;
- struct xt_table_info *newinfo, *oldinfo;
- struct xt_counters *counters;
- void *loc_cpu_entry, *loc_cpu_old_entry;
+ struct compat_ipt_replace tmp;
+ struct xt_table_info *newinfo;
+ void *loc_cpu_entry;

  if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
   return -EFAULT;
@@ -949,151 +1818,201 @@ do_replace(void __user *user, unsigned i
   goto free_newinfo;
  }

- counters = vmalloc(tmp.num_counters * sizeof(struct xt_counters));
- if (!counters) {
-  ret = -ENOMEM;
+ ret = translate_compat_table(tmp.name, tmp.valid_hooks,
+         &newinfo, &loc_cpu_entry, tmp.size,
+         tmp.num_entries, tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
   goto free_newinfo;
- }

- ret = translate_table(tmp.name, tmp.valid_hooks,
-         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
-         tmp.hook_entry, tmp.underflow);
- if (ret != 0)
-  goto free_newinfo_counters;
+ duprintf("compat_do_replace: Translated table\n");

- duprintf("ip_tables: Translated table\n");
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         compat_ptr(tmp.counters));
+ if (ret)
+  goto free_newinfo_untrans;
+ return 0;

- t = try_then_request_module(xt_find_table_lock(AF_INET, tmp.name),
-        "iptable_%s", tmp.name);
- if (!t || IS_ERR(t)) {
-  ret = t ? PTR_ERR(t) : -ENOENT;
```

```
-  goto free_newinfo_counters_untrans;
- }
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}

- /* You lied! */
- if (tmp.valid_hooks != t->valid_hooks) {
-  duprintf("Valid hook crap: %08X vs %08X\n",
-    tmp.valid_hooks, t->valid_hooks);
-  ret = -EINVAL;
-  goto put_module;
- }
+static int
+compat_do_ipt_set_ctl(struct sock *sk, int cmd, void __user *user,
+  unsigned int len)
+{
+ int ret;

- oldinfo = xt_replace_table(t, tmp.num_counters, newinfo, &ret);
- if (!oldinfo)
-  goto put_module;
+ if (!capable(CAP_NET_ADMIN))
+  return -EPERM;

- /* Update module usage count based on number of rules */
- duprintf("do_replace: oldnum=%u, initnum=%u, newnum=%u\n",
-  oldinfo->number, oldinfo->initial_entries, newinfo->number);
- if ((oldinfo->number > oldinfo->initial_entries) ||
-     (newinfo->number <= oldinfo->initial_entries))
-  module_put(t->me);
- if ((oldinfo->number > oldinfo->initial_entries) &&
-     (newinfo->number <= oldinfo->initial_entries))
-  module_put(t->me);
+ switch (cmd) {
+ case IPT_SO_SET_REPLACE:
+  ret = compat_do_replace(user, len);
+  break;

- /* Get the old counters. */
- get_counters(oldinfo, counters);
- /* Decrease module usage counts and free resource */
- loc_cpu_old_entry = oldinfo->entries[raw_smp_processor_id()];
- IPT_ENTRY_ITERATE(loc_cpu_old_entry, oldinfo->size, cleanup_entry,NULL);
- xt_free_table_info(oldinfo);
```

```
- if (copy_to_user(tmp.counters, counters,
-    sizeof(struct xt_counters) * tmp.num_counters) != 0)
-  ret = -EFAULT;
- vfree(counters);
- xt_table_unlock(t);
- return ret;
+ case IPT_SO_SET_ADD_COUNTERS:
+  ret = do_add_counters(user, len, 1);
+  break;
+
+ default:
+  duprintf("do_ipt_set_ctl:  unknown request %i\n", cmd);
+  ret = -EINVAL;
+ }

- put_module:
- module_put(t->me);
- xt_table_unlock(t);
- free_newinfo_counters_untrans:
- IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
- free_newinfo_counters:
- vfree(counters);
- free_newinfo:
- xt_free_table_info(newinfo);
  return ret;
 }

-/* We're lazy, and add to the first CPU; overflow works its fey magic
- * and everything is OK. */
-static inline int
-add_counter_to_entry(struct ipt_entry *e,
-       const struct xt_counters addme[],
-       unsigned int *i)
+struct compat_ipt_get_entries
 {
-#if 0
- duprintf("add_counter: Entry %u %lu/%lu + %lu/%lu\n",
-   *i,
-   (long unsigned int)e->counters.pcnt,
-   (long unsigned int)e->counters.bcnt,
-   (long unsigned int)addme[*i].pcnt,
-   (long unsigned int)addme[*i].bcnt);
-#endif
+ char name[IPT_TABLE_MAXNAMELEN];
+ compat_uint_t size;
+ struct compat_ipt_entry entrytable[0];
+};
```

```
- ADD_COUNTER(e->counters, addme[*i].bcnt, addme[*i].pcnt);
+static int compat_copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table, void __user *userptr)
+{
+ unsigned int off, num;
+ struct compat_ipt_entry e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ void __user *pos;
+ unsigned int size;
+ int ret = 0;
+ void *loc_cpu_entry;

- (*i)++;
- return 0;
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
+ /* choose the copy that is on our node/cpu, ...
+  * This choice is lazy (because current thread is
+  * allowed to migrate to another cpu)
+  */
+ loc_cpu_entry = private->entries[raw_smp_processor_id()];
+ pos = userptr;
+ size = total_size;
+ ret = IPT_ENTRY_ITERATE(loc_cpu_entry, total_size,
+   compat_copy_entry_to_user, &pos, &size);
+ if (ret)
+  goto free_counters;
+
+ /* ... then go back and fix counters and names */
+ for (off = 0, num = 0; off < size; off += e.next_offset, num++) {
+  unsigned int i;
+  struct ipt_entry_match m;
+  struct ipt_entry_target t;
+
+  ret = -EFAULT;
+  if (copy_from_user(&e, userptr + off,
+     sizeof(struct compat_ipt_entry)))
+   goto free_counters;
+  if (copy_to_user(userptr + off +
+   offsetof(struct compat_ipt_entry, counters),
+    &counters[num], sizeof(counters[num])))
+   goto free_counters;
+
+  for (i = sizeof(struct compat_ipt_entry);
+    i < e.target_offset; i += m.u.match_size) {
```

```
+   if (copy_from_user(&m, userptr + off + i,
+     sizeof(struct ipt_entry_match)))
+    goto free_counters;
+   if (copy_to_user(userptr + off + i +
+    offsetof(struct ipt_entry_match, u.user.name),
+    m.u.kernel.match->name,
+    strlen(m.u.kernel.match->name) + 1))
+    goto free_counters;
+  }
+
+  if (copy_from_user(&t, userptr + off + e.target_offset,
+      sizeof(struct ipt_entry_target)))
+   goto free_counters;
+  if (copy_to_user(userptr + off + e.target_offset +
+   offsetof(struct ipt_entry_target, u.user.name),
+   t.u.kernel.target->name,
+   strlen(t.u.kernel.target->name) + 1))
+   goto free_counters;
+ }
+ ret = 0;
+free_counters:
+ vfree(counters);
+ return ret;
 }

 static int
-do_add_counters(void __user *user, unsigned int len)
+compat_get_entries(struct compat_ipt_get_entries __user *uptr, int *len)
 {
- unsigned int i;
- struct xt_counters_info tmp, *paddc;
+ int ret;
+ struct compat_ipt_get_entries get;
  struct ipt_table *t;
- struct xt_table_info *private;
- int ret = 0;
- void *loc_cpu_entry;

- if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
-  return -EFAULT;

- if (len != sizeof(tmp) + tmp.num_counters*sizeof(struct xt_counters))
+ if (*len < sizeof(get)) {
+  duprintf("compat_get_entries: %u < %u\n",
+    *len, (unsigned int)sizeof(get));
   return -EINVAL;
+ }
```

```
- paddc = vmalloc_node(len, numa_node_id());
- if (!paddc)
-  return -ENOMEM;
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;

- if (copy_from_user(paddc, user, len) != 0) {
-  ret = -EFAULT;
-  goto free;
+ if (*len != sizeof(struct compat_ipt_get_entries) + get.size) {
+  duprintf("compat_get_entries: %u != %u\n", *len,
+   (unsigned int)(sizeof(struct compat_ipt_get_entries) +
+   get.size));
+  return -EINVAL;
  }

- t = xt_find_table_lock(AF_INET, tmp.name);
- if (!t || IS_ERR(t)) {
+ xt_compat_lock(AF_INET);
+ t = xt_find_table_lock(AF_INET, get.name);
+ if (t && !IS_ERR(t)) {
+  struct xt_table_info *private = t->private;
+  struct xt_table_info info;
+  duprintf("t->private->number = %u\n",
+    private->number);
+  ret = compat_table_info(private, &info);
+  if (!ret && get.size == info.size) {
+   ret = compat_copy_entries_to_user(private->size,
+         t, uptr->entrytable);
+  } else if (!ret) {
+   duprintf("compat_get_entries: I've got %u not %u!\n",
+     private->size,
+     get.size);
+   ret = -EINVAL;
+  }
+  compat_flush_offsets();
+  module_put(t->me);
+  xt_table_unlock(t);
+ } else
   ret = t ? PTR_ERR(t) : -ENOENT;
-  goto free;
- }

- write_lock_bh(&t->lock);
- private = t->private;
- if (private->number != paddc->num_counters) {
-  ret = -EINVAL;
-  goto unlock_up_free;
```

```
- }
+ xt_compat_unlock(AF_INET);
+ return ret;
+}

- i = 0;
- /* Choose the copy that is on our node */
- loc_cpu_entry = private->entries[raw_smp_processor_id()];
- IPT_ENTRY_ITERATE(loc_cpu_entry,
-     private->size,
-     add_counter_to_entry,
-     paddc->counters,
-     &i);
- unlock_up_free:
- write_unlock_bh(&t->lock);
- xt_table_unlock(t);
- module_put(t->me);
- free:
- vfree(paddc);
+static int
+compat_do_ipt_get_ctl(struct sock *sk, int cmd, void __user *user, int *len)
+{
+ int ret;

+ switch (cmd) {
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len, 1);
+  break;
+ case IPT_SO_GET_ENTRIES:
+  ret = compat_get_entries(user, len);
+  break;
+ default:
+  duprintf("compat_do_ipt_get_ctl: unknown request %i\n", cmd);
+  ret = -EINVAL;
+ }
  return ret;
 }
+#endif

 static int
 do_ipt_set_ctl(struct sock *sk, int cmd, void __user *user, unsigned int len)
@@ -1109,7 +2028,7 @@ do_ipt_set_ctl(struct sock *sk, int cmd,
   break;

  case IPT_SO_SET_ADD_COUNTERS:
-  ret = do_add_counters(user, len);
+  ret = do_add_counters(user, len, 0);
   break;
```

```
  default:
@@ -1129,65 +2048,13 @@ do_ipt_get_ctl(struct sock *sk, int cmd,
   return -EPERM;

  switch (cmd) {
- case IPT_SO_GET_INFO: {
-  char name[IPT_TABLE_MAXNAMELEN];
-  struct ipt_table *t;
-
-  if (*len != sizeof(struct ipt_getinfo)) {
-   duprintf("length %u != %u\n", *len,
-     sizeof(struct ipt_getinfo));
-   ret = -EINVAL;
-   break;
-  }
-
-  if (copy_from_user(name, user, sizeof(name)) != 0) {
-   ret = -EFAULT;
-   break;
-  }
-  name[IPT_TABLE_MAXNAMELEN-1] = '\0';
-
-  t = try_then_request_module(xt_find_table_lock(AF_INET, name),
-         "iptable_%s", name);
-  if (t && !IS_ERR(t)) {
-   struct ipt_getinfo info;
-   struct xt_table_info *private = t->private;
-
-   info.valid_hooks = t->valid_hooks;
-   memcpy(info.hook_entry, private->hook_entry,
-          sizeof(info.hook_entry));
-   memcpy(info.underflow, private->underflow,
-          sizeof(info.underflow));
-   info.num_entries = private->number;
-   info.size = private->size;
-   memcpy(info.name, name, sizeof(info.name));
-
-   if (copy_to_user(user, &info, *len) != 0)
-    ret = -EFAULT;
-   else
-    ret = 0;
-   xt_table_unlock(t);
-   module_put(t->me);
-  } else
-   ret = t ? PTR_ERR(t) : -ENOENT;
- }
- break;
```

```
-
- case IPT_SO_GET_ENTRIES: {
-  struct ipt_get_entries get;
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len, 0);
+  break;

-  if (*len < sizeof(get)) {
-   duprintf("get_entries: %u < %u\n", *len, sizeof(get));
-   ret = -EINVAL;
-  } else if (copy_from_user(&get, user, sizeof(get)) != 0) {
-   ret = -EFAULT;
-  } else if (*len != sizeof(struct ipt_get_entries) + get.size) {
-   duprintf("get_entries: %u != %u\n", *len,
-     sizeof(struct ipt_get_entries) + get.size);
-   ret = -EINVAL;
-  } else
-   ret = get_entries(&get, user);
+ case IPT_SO_GET_ENTRIES:
+  ret = get_entries(user, len);
   break;
- }

  case IPT_SO_GET_REVISION_MATCH:
  case IPT_SO_GET_REVISION_TARGET: {
@@ -1335,6 +2202,9 @@ icmp_checkentry(const char *tablename,
 static struct ipt_target ipt_standard_target = {
  .name  = IPT_STANDARD_TARGET,
  .targetsize = sizeof(int),
+#ifdef CONFIG_COMPAT
+ .compat  = &compat_ipt_standard_fn,
+#endif
 };

 static struct ipt_target ipt_error_target = {
@@ -1348,9 +2218,11 @@ static struct nf_sockopt_ops ipt_sockopt
  .set_optmin = IPT_BASE_CTL,
  .set_optmax = IPT_SO_SET_MAX+1,
  .set  = do_ipt_set_ctl,
+ .compat_set = compat_do_ipt_set_ctl,
  .get_optmin = IPT_BASE_CTL,
  .get_optmax = IPT_SO_GET_MAX+1,
  .get  = do_ipt_get_ctl,
+ .compat_get = compat_do_ipt_get_ctl,
 };

 static struct ipt_match icmp_matchstruct = {
diff --git a/net/netfilter/x_tables.c b/net/netfilter/x_tables.c
```

```
index 750b928..72b8928 100644
--- a/net/netfilter/x_tables.c
+++ b/net/netfilter/x_tables.c
@@ -36,6 +36,7 @@ struct xt_af {
  struct list_head match;
  struct list_head target;
  struct list_head tables;
+ struct semaphore compat_mutex;
 };

 static struct xt_af *xt;
@@ -266,6 +267,54 @@ int xt_check_match(const struct xt_match
 }
 EXPORT_SYMBOL_GPL(xt_check_match);

+#ifdef CONFIG_COMPAT
+int xt_compat_match(void *match, void **dstptr, int *size, int convert)
+{
+ struct xt_match *m;
+ struct compat_xt_entry_match *pcompat_m;
+ struct xt_entry_match *pm;
+ u_int16_t msize;
+ int off, ret;
+
+ ret = 0;
+ m = ((struct xt_entry_match *)match)->u.kernel.match;
+ off = XT_ALIGN(m->matchsize) - COMPAT_XT_ALIGN(m->matchsize);
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pm = (struct xt_entry_match *)match;
+   msize = pm->u.user.match_size;
+   if (__copy_to_user(*dstptr, pm, msize)) {
+    ret = -EFAULT;
+    break;
+   }
+   msize -= off;
+   if (put_user(msize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += msize;
+   break;
+  case COMPAT_FROM_USER:
+   pcompat_m = (struct compat_xt_entry_match *)match;
+   pm = (struct xt_entry_match *)*dstptr;
+   msize = pcompat_m->u.user.match_size;
+   memcpy(pm, pcompat_m, msize);
+   msize += off;
+   pm->u.user.match_size = msize;
```

```
+   *size += off;
+   *dstptr += msize;
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+EXPORT_SYMBOL_GPL(xt_compat_match);
+#endif
+
 int xt_check_target(const struct xt_target *target, unsigned short family,
       unsigned int size, const char *table, unsigned int hook_mask,
       unsigned short proto, int inv_proto)
@@ -295,6 +344,54 @@ int xt_check_target(const struct xt_targ
 }
 EXPORT_SYMBOL_GPL(xt_check_target);

+#ifdef CONFIG_COMPAT
+int xt_compat_target(void *target, void **dstptr, int *size, int convert)
+{
+ struct xt_target *t;
+ struct compat_xt_entry_target *pcompat;
+ struct xt_entry_target *pt;
+ u_int16_t tsize;
+ int off, ret;
+
+ ret = 0;
+ t = ((struct xt_entry_target *)target)->u.kernel.target;
+ off = XT_ALIGN(t->targetsize) - COMPAT_XT_ALIGN(t->targetsize);
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pt = (struct xt_entry_target *)target;
+   tsize = pt->u.user.target_size;
+   if (__copy_to_user(*dstptr, pt, tsize)) {
+    ret = -EFAULT;
+    break;
+   }
+   tsize -= off;
+   if (put_user(tsize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += tsize;
+   break;
```

```
+  case COMPAT_FROM_USER:
+   pcompat = (struct compat_xt_entry_target *)target;
+   pt = (struct xt_entry_target *)*dstptr;
+   tsize = pcompat->u.user.target_size;
+   memcpy(pt, pcompat, tsize);
+   tsize += off;
+   pt->u.user.target_size = tsize;
+   *size += off;
+   *dstptr += tsize;
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+EXPORT_SYMBOL_GPL(xt_compat_target);
+#endif
+
 struct xt_table_info *xt_alloc_table_info(unsigned int size)
 {
  struct xt_table_info *newinfo;
@@ -365,6 +462,19 @@ void xt_table_unlock(struct xt_table *ta
 }
 EXPORT_SYMBOL_GPL(xt_table_unlock);

+#ifdef CONFIG_COMPAT
+void xt_compat_lock(int af)
+{
+ down(&xt[af].compat_mutex);
+}
+EXPORT_SYMBOL_GPL(xt_compat_lock);
+
+void xt_compat_unlock(int af)
+{
+ up(&xt[af].compat_mutex);
+}
+EXPORT_SYMBOL_GPL(xt_compat_unlock);
+#endif

 struct xt_table_info *
 xt_replace_table(struct xt_table *table,
@@ -665,6 +775,9 @@ static int __init xt_init(void)

  for (i = 0; i < NPROTO; i++) {
```

```
   init_MUTEX(&xt[i].mutex);
+#ifdef CONFIG_COMPAT
+  init_MUTEX(&xt[i].compat_mutex);
+#endif
   INIT_LIST_HEAD(&xt[i].target);
   INIT_LIST_HEAD(&xt[i].match);
   INIT_LIST_HEAD(&xt[i].tables);
```

File Attachments

1) diff-ms-netfilter-ipt-compat-20060322, downloaded 689 times

---

Subject: Re: [PATCH] iptables 32bit compat layer
Posted by Patrick McHardy on Wed, 29 Mar 2006 09:28:39 GMT
View Forum Message <> Reply to Message

Dmitry Mishin wrote:
> This patch extends current iptables compatibility layer in order to get
> 32bit iptables to work on 64bit kernel. Current layer is insufficient due to
> alignment checks both in kernel and user space tools.
>
> Patch is for current net-2.6.17 with addition of move of ipt_entry_{match|
> target} definitions to xt_entry_{match|target}.

Thanks, this looks good. Two small issues so far:

```
> diff --git a/net/compat.c b/net/compat.c
> index 13177a1..6a7028e 100644
> --- a/net/compat.c
> +++ b/net/compat.c
> @@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
>   int err;
>   struct socket *sock;
>
> - /* SO_SET_REPLACE seems to be the same in all levels */
> - if (optname == IPT_SO_SET_REPLACE)
> + if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
>    return do_netfilter_replace(fd, level, optname,
>           optval, optlen);
```

I don't understand the reason for this change. If its not a mistake,
it would make more sense to check for IP6T_SO_SET_REPLACE I guess ..

```
> +#ifdef CONFIG_COMPAT
> +void xt_compat_lock(int af)
> +{
```

> + down(&xt[af].compat_mutex);
> +}
> +EXPORT_SYMBOL_GPL(xt_compat_lock);
> +
> +void xt_compat_unlock(int af)
> +{
> + up(&xt[af].compat_mutex);
> +}
> +EXPORT_SYMBOL_GPL(xt_compat_unlock);
> +#endif

Won't a seperate compat-mutex introduce races between compat- and non-compat users? BTW, the up/down calls have been replaced by the new mutex API in Linus' tree, please resend the patch against the current tree.

Subject: Re: [PATCH] iptables 32bit compat layer
Posted by Mishin Dmitry on Wed, 29 Mar 2006 11:36:16 GMT
View Forum Message <> Reply to Message

On Wednesday 29 March 2006 13:28, Patrick McHardy wrote:
> Dmitry Mishin wrote:
> > This patch extends current iptables compatibility layer in order to get
> > 32bit iptables to work on 64bit kernel. Current layer is insufficient due
> > to alignment checks both in kernel and user space tools.
> >
> > Patch is for current net-2.6.17 with addition of move of
> > ipt_entry_{match| target} definitions to xt_entry_{match|target}.
>
> Thanks, this looks good. Two small issues so far:
> > diff --git a/net/compat.c b/net/compat.c
> > index 13177a1..6a7028e 100644
> > --- a/net/compat.c
> > +++ b/net/compat.c
> > @@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
> > int err;
> > struct socket *sock;
> >
> > - /* SO_SET_REPLACE seems to be the same in all levels */
> > - if (optname == IPT_SO_SET_REPLACE)
> > + if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
> > return do_netfilter_replace(fd, level, optname,
> > optval, optlen);
>
> I don't understand the reason for this change. If its not a mistake,
> it would make more sense to check for IP6T_SO_SET_REPLACE I guess ..
IP6T_SO_SET_REPLACE == IPT_SO_SET_REPLACE == XT_SO_SET_REPLACE.

Rename will require respective #include directive rename, so, I just leave
this as it is. BTW, I'll make respective patch for IPV6 in the near future
and this hunk will be removed at all.

>
> > +#ifdef CONFIG_COMPAT
> > +void xt_compat_lock(int af)
> > +{
> > + down(&xt[af].compat_mutex);
> > +}
> > +EXPORT_SYMBOL_GPL(xt_compat_lock);
> > +
> > +void xt_compat_unlock(int af)
> > +{
> > + up(&xt[af].compat_mutex);
> > +}
> > +EXPORT_SYMBOL_GPL(xt_compat_unlock);
> > +#endif
>
> Won't a seperate compat-mutex introduce races between compat- and
> non-compat users? BTW, the up/down calls have been replaced by the
> new mutex API in Linus' tree, please resend the patch against the
> current tree.
compat_mutex is always over xt[af].mutex and can't be taken under the last
one, so, there should be no races.
New patch is attached.

--
Thanks,
Dmitry.

diff --git a/include/linux/netfilter/x_tables.h b/include/linux/netfilter/x_tables.h
index 1350e47..f6bdef8 100644
--- a/include/linux/netfilter/x_tables.h
+++ b/include/linux/netfilter/x_tables.h
@@ -142,6 +142,12 @@ struct xt_counters_info
 #define ASSERT_WRITE_LOCK(x)
 #include <linux/netfilter_ipv4/listhelp.h>

+#ifdef CONFIG_COMPAT
+#define COMPAT_TO_USER 1
+#define COMPAT_FROM_USER -1
+#define COMPAT_CALC_SIZE 0
+#endif
+
 struct xt_match
 {
  struct list_head list;

```
@@ -175,6 +181,9 @@ struct xt_match
  void (*destroy)(const struct xt_match *match, void *matchinfo,
    unsigned int matchinfosize);

+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *match, void **dstptr, int *size, int convert);
+
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;

@@ -220,6 +229,9 @@ struct xt_target
  void (*destroy)(const struct xt_target *target, void *targinfo,
    unsigned int targinfosize);

+ /* Called when userspace align differs from kernel space one */
+ int (*compat)(void *target, void **dstptr, int *size, int convert);
+
  /* Set this to THIS_MODULE if you are a module, otherwise NULL */
  struct module *me;

@@ -314,6 +326,61 @@ extern void xt_proto_fini(int af);
 extern struct xt_table_info *xt_alloc_table_info(unsigned int size);
 extern void xt_free_table_info(struct xt_table_info *info);

+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
+
+struct compat_xt_entry_match
+{
+ union {
+  struct {
+   u_int16_t match_size;
+   char name[XT_FUNCTION_MAXNAMELEN - 1];
+   u_int8_t revision;
+  } user;
+  u_int16_t match_size;
+ } u;
+ unsigned char data[0];
+};
+
+struct compat_xt_entry_target
+{
+ union {
+  struct {
+   u_int16_t target_size;
+   char name[XT_FUNCTION_MAXNAMELEN - 1];
+   u_int8_t revision;
+  } user;
```

```
+  u_int16_t target_size;
+ } u;
+ unsigned char data[0];
+};
+
+/* FIXME: this works only on 32 bit tasks
+ * need to change whole approach in order to calculate align as function of
+ * current task alignment */
+
+struct compat_xt_counters
+{
+ u_int32_t cnt[4];
+};
+
+struct compat_xt_counters_info
+{
+ char name[XT_TABLE_MAXNAMELEN];
+ compat_uint_t num_counters;
+ struct compat_xt_counters counters[0];
+};
+
+#define COMPAT_XT_ALIGN(s) (((s) + (__alignof__(struct compat_xt_counters)-1)) \
+  & ~(__alignof__(struct compat_xt_counters)-1))
+
+extern void xt_compat_lock(int af);
+extern void xt_compat_unlock(int af);
+extern int xt_compat_match(void *match, void **dstptr, int *size, int convert);
+extern int xt_compat_target(void *target, void **dstptr, int *size,
+  int convert);
+
+#endif /* CONFIG_COMPAT */
 #endif /* __KERNEL__ */

 #endif /* _X_TABLES_H */
diff --git a/include/linux/netfilter_ipv4/ip_tables.h b/include/linux/netfilter_ipv4/ip_tables.h
index d5b8c0d..c0dac16 100644
--- a/include/linux/netfilter_ipv4/ip_tables.h
+++ b/include/linux/netfilter_ipv4/ip_tables.h
@@ -316,5 +316,23 @@ extern unsigned int ipt_do_table(struct
      void *userdata);

 #define IPT_ALIGN(s) XT_ALIGN(s)
+
+#ifdef CONFIG_COMPAT
+#include <net/compat.h>
+
+struct compat_ipt_entry
+{
```

```
+ struct ipt_ip ip;
+ compat_uint_t nfcache;
+ u_int16_t target_offset;
+ u_int16_t next_offset;
+ compat_uint_t comefrom;
+ struct compat_xt_counters counters;
+ unsigned char elems[0];
+};
+
+#define COMPAT_IPT_ALIGN(s)  COMPAT_XT_ALIGN(s)
+
+#endif /* CONFIG_COMPAT */
 #endif /*__KERNEL__*/
 #endif /* _IPTABLES_H */
diff --git a/net/compat.c b/net/compat.c
index 8fd37cd..d5d69fa 100644
--- a/net/compat.c
+++ b/net/compat.c
@@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
  int err;
  struct socket *sock;

- /* SO_SET_REPLACE seems to be the same in all levels */
- if (optname == IPT_SO_SET_REPLACE)
+ if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
   return do_netfilter_replace(fd, level, optname,
          optval, optlen);

diff --git a/net/ipv4/netfilter/ip_tables.c b/net/ipv4/netfilter/ip_tables.c
index a7b194c..34df287 100644
--- a/net/ipv4/netfilter/ip_tables.c
+++ b/net/ipv4/netfilter/ip_tables.c
@@ -24,6 +24,7 @@
 #include <linux/module.h>
 #include <linux/icmp.h>
 #include <net/ip.h>
+#include <net/compat.h>
 #include <asm/uaccess.h>
 #include <linux/mutex.h>
 #include <linux/proc_fs.h>
@@ -799,17 +800,11 @@ get_counters(const struct xt_table_info
  }
 }

-static int
-copy_entries_to_user(unsigned int total_size,
-       struct ipt_table *table,
-       void __user *userptr)
```

```
+static inline struct xt_counters * alloc_counters(struct ipt_table *table)
 {
- unsigned int off, num, countersize;
- struct ipt_entry *e;
+ unsigned int countersize;
  struct xt_counters *counters;
  struct xt_table_info *private = table->private;
- int ret = 0;
- void *loc_cpu_entry;

  /* We need atomic snapshot of counters: rest doesn't change
     (other than comefrom, which userspace doesn't care
@@ -818,13 +813,32 @@ copy_entries_to_user(unsigned int total_
  counters = vmalloc_node(countersize, numa_node_id());

  if (counters == NULL)
-  return -ENOMEM;
+  return ERR_PTR(-ENOMEM);

  /* First, sum counters... */
  write_lock_bh(&table->lock);
  get_counters(private, counters);
  write_unlock_bh(&table->lock);

+ return counters;
+}
+
+static int
+copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table,
+       void __user *userptr)
+{
+ unsigned int off, num;
+ struct ipt_entry *e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ int ret = 0;
+ void *loc_cpu_entry;
+
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
  /* choose the copy that is on our node/cpu, ...
   * This choice is lazy (because current thread is
   * allowed to migrate to another cpu)
@@ -878,50 +892,905 @@ copy_entries_to_user(unsigned int total_
    goto free_counters;
```

```
   }
  }
-
- free_counters:
- vfree(counters);
+
+ free_counters:
+ vfree(counters);
+ return ret;
+}
+
+#ifdef CONFIG_COMPAT
+struct compat_delta {
+ struct compat_delta *next;
+ u_int16_t offset;
+ short delta;
+};
+
+static struct compat_delta *compat_offsets = NULL;
+
+static int compat_add_offset(u_int16_t offset, short delta)
+{
+ struct compat_delta *tmp;
+
+ tmp = kmalloc(sizeof(struct compat_delta), GFP_KERNEL);
+ if (!tmp)
+  return -ENOMEM;
+ tmp->offset = offset;
+ tmp->delta = delta;
+ if (compat_offsets) {
+  tmp->next = compat_offsets->next;
+  compat_offsets->next = tmp;
+ } else {
+  compat_offsets = tmp;
+  tmp->next = NULL;
+ }
+ return 0;
+}
+
+static void compat_flush_offsets(void)
+{
+ struct compat_delta *tmp, *next;
+
+ if (compat_offsets) {
+  for(tmp = compat_offsets; tmp; tmp = next) {
+   next = tmp->next;
+   kfree(tmp);
+  }
```

```
+  compat_offsets = NULL;
+ }
+}
+
+static short compat_calc_jump(u_int16_t offset)
+{
+ struct compat_delta *tmp;
+ short delta;
+
+ for(tmp = compat_offsets, delta = 0; tmp; tmp = tmp->next)
+  if (tmp->offset < offset)
+   delta += tmp->delta;
+ return delta;
+}
+
+struct compat_ipt_standard_target
+{
+ struct compat_xt_entry_target target;
+ compat_int_t verdict;
+};
+
+#define IPT_ST_OFFSET (sizeof(struct ipt_standard_target) - \
+    sizeof(struct compat_ipt_standard_target))
+
+struct compat_ipt_standard
+{
+ struct compat_ipt_entry entry;
+ struct compat_ipt_standard_target target;
+};
+
+static int compat_ipt_standard_fn(void *target,
+  void **dstptr, int *size, int convert)
+{
+ struct compat_ipt_standard_target compat_st, *pcompat_st;
+ struct ipt_standard_target st, *pst;
+ int ret;
+
+ ret = 0;
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pst = (struct ipt_standard_target *)target;
+   memcpy(&compat_st.target, &pst->target,
+     sizeof(struct ipt_entry_target));
+   compat_st.verdict = pst->verdict;
+   if (compat_st.verdict > 0)
+    compat_st.verdict -=
+     compat_calc_jump(compat_st.verdict);
+   compat_st.target.u.user.target_size =
```

```
+   sizeof(struct compat_ipt_standard_target);
+   if (__copy_to_user(*dstptr, &compat_st,
+    sizeof(struct compat_ipt_standard_target)))
+    ret = -EFAULT;
+   *size -= IPT_ST_OFFSET;
+   *dstptr += sizeof(struct compat_ipt_standard_target);
+   break;
+  case COMPAT_FROM_USER:
+   pcompat_st =
+    (struct compat_ipt_standard_target *)target;
+   memcpy(&st.target, &pcompat_st->target,
+     sizeof(struct ipt_entry_target));
+   st.verdict = pcompat_st->verdict;
+   if (st.verdict > 0)
+    st.verdict += compat_calc_jump(st.verdict);
+   st.target.u.user.target_size =
+   sizeof(struct ipt_standard_target);
+   memcpy(*dstptr, &st,
+     sizeof(struct ipt_standard_target));
+   *size += IPT_ST_OFFSET;
+   *dstptr += sizeof(struct ipt_standard_target);
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += IPT_ST_OFFSET;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+
+static inline int
+compat_calc_match(struct ipt_entry_match *m, int * size)
+{
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+ else
+  xt_compat_match(m, NULL, size, COMPAT_CALC_SIZE);
+ return 0;
+}
+
+static int compat_calc_entry(struct ipt_entry *e, struct xt_table_info *info,
+  void *base, struct xt_table_info *newinfo)
+{
+ struct ipt_entry_target *t;
+ u_int16_t entry_offset;
+ int off, i, ret;
```

```
+
+ off = 0;
+ entry_offset = (void *)e - base;
+ IPT_MATCH_ITERATE(e, compat_calc_match, &off);
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ else
+  xt_compat_target(t, NULL, &off, COMPAT_CALC_SIZE);
+ newinfo->size -= off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  return ret;
+
+ for (i = 0; i< NF_IP_NUMHOOKS; i++) {
+  if (info->hook_entry[i] && (e < (struct ipt_entry *)
+    (base + info->hook_entry[i])))
+   newinfo->hook_entry[i] -= off;
+  if (info->underflow[i] && (e < (struct ipt_entry *)
+    (base + info->underflow[i])))
+   newinfo->underflow[i] -= off;
+ }
+ return 0;
+}
+
+static int compat_table_info(struct xt_table_info *info,
+  struct xt_table_info *newinfo)
+{
+ void *loc_cpu_entry;
+ int i;
+
+ if (!newinfo || !info)
+  return -EINVAL;
+
+ memset(newinfo, 0, sizeof(struct xt_table_info));
+ newinfo->size = info->size;
+ newinfo->number = info->number;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ loc_cpu_entry = info->entries[raw_smp_processor_id()];
+ return IPT_ENTRY_ITERATE(loc_cpu_entry, info->size,
+   compat_calc_entry, info, loc_cpu_entry, newinfo);
+}
+#endif
+
+static int get_info(void __user *user, int *len, int compat)
```

```
+{
+ char name[IPT_TABLE_MAXNAMELEN];
+ struct ipt_table *t;
+ int ret;
+
+ if (*len != sizeof(struct ipt_getinfo)) {
+  duprintf("length %u != %u\n", *len,
+   (unsigned int)sizeof(struct ipt_getinfo));
+  return -EINVAL;
+ }
+
+ if (copy_from_user(name, user, sizeof(name)) != 0)
+  return -EFAULT;
+
+ name[IPT_TABLE_MAXNAMELEN-1] = '\0';
+#ifdef CONFIG_COMPAT
+ if (compat)
+  xt_compat_lock(AF_INET);
+#endif
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+   "iptable_%s", name);
+ if (t && !IS_ERR(t)) {
+  struct ipt_getinfo info;
+  struct xt_table_info *private = t->private;
+
+#ifdef CONFIG_COMPAT
+  if (compat) {
+   struct xt_table_info tmp;
+   ret = compat_table_info(private, &tmp);
+   compat_flush_offsets();
+   private =  &tmp;
+  }
+#endif
+  info.valid_hooks = t->valid_hooks;
+  memcpy(info.hook_entry, private->hook_entry,
+    sizeof(info.hook_entry));
+  memcpy(info.underflow, private->underflow,
+    sizeof(info.underflow));
+  info.num_entries = private->number;
+  info.size = private->size;
+  strcpy(info.name, name);
+
+  if (copy_to_user(user, &info, *len) != 0)
+   ret = -EFAULT;
+  else
+   ret = 0;
+
+  xt_table_unlock(t);
```

```
+  module_put(t->me);
+ } else
+  ret = t ? PTR_ERR(t) : -ENOENT;
+#ifdef CONFIG_COMPAT
+ if (compat)
+  xt_compat_unlock(AF_INET);
+#endif
+ return ret;
+}
+
+static int
+get_entries(struct ipt_get_entries __user *uptr, int *len)
+{
+ int ret;
+ struct ipt_get_entries get;
+ struct ipt_table *t;
+
+ if (*len < sizeof(get)) {
+  duprintf("get_entries: %u < %d\n", *len,
+    (unsigned int)sizeof(get));
+  return -EINVAL;
+ }
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;
+ if (*len != sizeof(struct ipt_get_entries) + get.size) {
+  duprintf("get_entries: %u != %u\n", *len,
+    (unsigned int)(sizeof(struct ipt_get_entries) +
+    get.size));
+  return -EINVAL;
+ }
+
+ t = xt_find_table_lock(AF_INET, get.name);
+ if (t && !IS_ERR(t)) {
+  struct xt_table_info *private = t->private;
+  duprintf("t->private->number = %u\n",
+    private->number);
+  if (get.size == private->size)
+   ret = copy_entries_to_user(private->size,
+         t, uptr->entrytable);
+  else {
+   duprintf("get_entries: I've got %u not %u!\n",
+     private->size,
+     get.size);
+   ret = -EINVAL;
+  }
+  module_put(t->me);
+  xt_table_unlock(t);
+ } else
```

```
+  ret = t ? PTR_ERR(t) : -ENOENT;
+
+ return ret;
+}
+
+static int
+__do_replace(const char *name, unsigned int valid_hooks,
+  struct xt_table_info *newinfo, unsigned int num_counters,
+  void __user *counters_ptr)
+{
+ int ret;
+ struct ipt_table *t;
+ struct xt_table_info *oldinfo;
+ struct xt_counters *counters;
+ void *loc_cpu_old_entry;
+
+ ret = 0;
+ counters = vmalloc(num_counters * sizeof(struct xt_counters));
+ if (!counters) {
+  ret = -ENOMEM;
+  goto out;
+ }
+
+ t = try_then_request_module(xt_find_table_lock(AF_INET, name),
+        "iptable_%s", name);
+ if (!t || IS_ERR(t)) {
+  ret = t ? PTR_ERR(t) : -ENOENT;
+  goto free_newinfo_counters_untrans;
+ }
+
+ /* You lied! */
+ if (valid_hooks != t->valid_hooks) {
+  duprintf("Valid hook crap: %08X vs %08X\n",
+    valid_hooks, t->valid_hooks);
+  ret = -EINVAL;
+  goto put_module;
+ }
+
+ oldinfo = xt_replace_table(t, num_counters, newinfo, &ret);
+ if (!oldinfo)
+  goto put_module;
+
+ /* Update module usage count based on number of rules */
+ duprintf("do_replace: oldnum=%u, initnum=%u, newnum=%u\n",
+  oldinfo->number, oldinfo->initial_entries, newinfo->number);
+ if ((oldinfo->number > oldinfo->initial_entries) ||
+     (newinfo->number <= oldinfo->initial_entries))
+  module_put(t->me);
```

```
+ if ((oldinfo->number > oldinfo->initial_entries) &&
+     (newinfo->number <= oldinfo->initial_entries))
+  module_put(t->me);
+
+ /* Get the old counters. */
+ get_counters(oldinfo, counters);
+ /* Decrease module usage counts and free resource */
+ loc_cpu_old_entry = oldinfo->entries[raw_smp_processor_id()];
+ IPT_ENTRY_ITERATE(loc_cpu_old_entry, oldinfo->size, cleanup_entry,NULL);
+ xt_free_table_info(oldinfo);
+ if (copy_to_user(counters_ptr, counters,
+    sizeof(struct xt_counters) * num_counters) != 0)
+  ret = -EFAULT;
+ vfree(counters);
+ xt_table_unlock(t);
+ return ret;
+
+ put_module:
+ module_put(t->me);
+ xt_table_unlock(t);
+ free_newinfo_counters_untrans:
+ vfree(counters);
+ out:
+ return ret;
+}
+
+static int
+do_replace(void __user *user, unsigned int len)
+{
+ int ret;
+ struct ipt_replace tmp;
+ struct xt_table_info *newinfo;
+ void *loc_cpu_entry;
+
+ if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
+  return -EFAULT;
+
+ /* Hack: Causes ipchains to give correct error msg --RR */
+ if (len != sizeof(tmp) + tmp.size)
+  return -ENOPROTOOPT;
+
+ /* overflow check */
+ if (tmp.size >= (INT_MAX - sizeof(struct xt_table_info)) / NR_CPUS -
+   SMP_CACHE_BYTES)
+  return -ENOMEM;
+ if (tmp.num_counters >= INT_MAX / sizeof(struct xt_counters))
+  return -ENOMEM;
+
```

```
+ newinfo = xt_alloc_table_info(tmp.size);
+ if (!newinfo)
+  return -ENOMEM;
+
+ /* choose the copy that is our node/cpu */
+ loc_cpu_entry = newinfo->entries[raw_smp_processor_id()];
+ if (copy_from_user(loc_cpu_entry, user + sizeof(tmp),
+      tmp.size) != 0) {
+  ret = -EFAULT;
+  goto free_newinfo;
+ }
+
+ ret = translate_table(tmp.name, tmp.valid_hooks,
+         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
+         tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
+  goto free_newinfo;
+
+ duprintf("ip_tables: Translated table\n");
+
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         tmp.counters);
+ if (ret)
+  goto free_newinfo_untrans;
+ return 0;
+
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}
+
+/* We're lazy, and add to the first CPU; overflow works its fey magic
+ * and everything is OK. */
+static inline int
+add_counter_to_entry(struct ipt_entry *e,
+       const struct xt_counters addme[],
+       unsigned int *i)
+{
+#if 0
+ duprintf("add_counter: Entry %u %lu/%lu + %lu/%lu\n",
+    *i,
+    (long unsigned int)e->counters.pcnt,
+    (long unsigned int)e->counters.bcnt,
+    (long unsigned int)addme[*i].pcnt,
+    (long unsigned int)addme[*i].bcnt);
```

```
+#endif
+
+ ADD_COUNTER(e->counters, addme[*i].bcnt, addme[*i].pcnt);
+
+ (*i)++;
+ return 0;
+}
+
+static int
+do_add_counters(void __user *user, unsigned int len, int compat)
+{
+ unsigned int i;
+ struct xt_counters_info tmp;
+ struct xt_counters *paddc;
+ unsigned int num_counters;
+ char *name;
+ int size;
+ void *ptmp;
+ struct ipt_table *t;
+ struct xt_table_info *private;
+ int ret = 0;
+ void *loc_cpu_entry;
+#ifdef CONFIG_COMPAT
+ struct compat_xt_counters_info compat_tmp;
+
+ if (compat) {
+  ptmp = &compat_tmp;
+  size = sizeof(struct compat_xt_counters_info);
+ } else
+#endif
+ {
+  ptmp = &tmp;
+  size = sizeof(struct xt_counters_info);
+ }
+
+ if (copy_from_user(ptmp, user, size) != 0)
+  return -EFAULT;
+
+#ifdef CONFIG_COMPAT
+ if (compat) {
+  num_counters = compat_tmp.num_counters;
+  name = compat_tmp.name;
+ } else
+#endif
+ {
+  num_counters = tmp.num_counters;
+  name = tmp.name;
+ }
```

```
+
+ if (len != size + num_counters * sizeof(struct xt_counters))
+  return -EINVAL;
+
+ paddc = vmalloc_node(len - size, numa_node_id());
+ if (!paddc)
+  return -ENOMEM;
+
+ if (copy_from_user(paddc, user + size, len - size) != 0) {
+  ret = -EFAULT;
+  goto free;
+ }
+
+ t = xt_find_table_lock(AF_INET, name);
+ if (!t || IS_ERR(t)) {
+  ret = t ? PTR_ERR(t) : -ENOENT;
+  goto free;
+ }
+
+ write_lock_bh(&t->lock);
+ private = t->private;
+ if (private->number != num_counters) {
+  ret = -EINVAL;
+  goto unlock_up_free;
+ }
+
+ i = 0;
+ /* Choose the copy that is on our node */
+ loc_cpu_entry = private->entries[raw_smp_processor_id()];
+ IPT_ENTRY_ITERATE(loc_cpu_entry,
+     private->size,
+     add_counter_to_entry,
+     paddc,
+     &i);
+ unlock_up_free:
+ write_unlock_bh(&t->lock);
+ xt_table_unlock(t);
+ module_put(t->me);
+ free:
+ vfree(paddc);
+
+ return ret;
+}
+
+#ifdef CONFIG_COMPAT
+struct compat_ipt_replace {
+ char   name[IPT_TABLE_MAXNAMELEN];
+ u32   valid_hooks;
```

```
+ u32   num_entries;
+ u32   size;
+ u32   hook_entry[NF_IP_NUMHOOKS];
+ u32   underflow[NF_IP_NUMHOOKS];
+ u32   num_counters;
+ compat_uptr_t  counters; /* struct ipt_counters * */
+ struct compat_ipt_entry entries[0];
+};
+
+static inline int compat_copy_match_to_user(struct ipt_entry_match *m,
+  void __user **dstptr, compat_uint_t *size)
+{
+ if (m->u.kernel.match->compat)
+  return m->u.kernel.match->compat(m, dstptr, size,
+    COMPAT_TO_USER);
+ else
+  return xt_compat_match(m, dstptr, size, COMPAT_TO_USER);
+}
+
+static int compat_copy_entry_to_user(struct ipt_entry *e,
+  void __user **dstptr, compat_uint_t *size)
+{
+ struct ipt_entry_target __user *t;
+ struct compat_ipt_entry __user *ce;
+ u_int16_t target_offset, next_offset;
+ compat_uint_t origsize;
+ int ret;
+
+ ret = -EFAULT;
+ origsize = *size;
+ ce = (struct compat_ipt_entry __user *)*dstptr;
+ if (__copy_to_user(ce, e, sizeof(struct ipt_entry)))
+  goto out;
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_to_user, dstptr, size);
+ target_offset = e->target_offset - (origsize - *size);
+ if (ret)
+  goto out;
+ t = ipt_get_target(e);
+ if (t->u.kernel.target->compat)
+  ret = t->u.kernel.target->compat(t, dstptr, size,
+    COMPAT_TO_USER);
+ else
+  ret = xt_compat_target(t, dstptr, size, COMPAT_TO_USER);
+ if (ret)
+  goto out;
+ ret = -EFAULT;
```

```
+ next_offset = e->next_offset - (origsize - *size);
+ if (__put_user(target_offset, &ce->target_offset))
+  goto out;
+ if (__put_user(next_offset, &ce->next_offset))
+  goto out;
+ return 0;
+out:
+ return ret;
+}
+
+static inline int
+compat_check_calc_match(struct ipt_entry_match *m,
+     const char *name,
+     const struct ipt_ip *ip,
+     unsigned int hookmask,
+     int *size, int *i)
+{
+ struct ipt_match *match;
+
+ match = try_then_request_module(xt_find_match(AF_INET, m->u.user.name,
+         m->u.user.revision),
+     "ipt_%s", m->u.user.name);
+ if (IS_ERR(match) || !match) {
+  duprintf("compat_check_calc_match: `%s' not found\n",
+    m->u.user.name);
+  return match ? PTR_ERR(match) : -ENOENT;
+ }
+ m->u.kernel.match = match;
+
+ if (m->u.kernel.match->compat)
+  m->u.kernel.match->compat(m, NULL, size, COMPAT_CALC_SIZE);
+ else
+  xt_compat_match(m, NULL, size, COMPAT_CALC_SIZE);
+
+ (*i)++;
+ return 0;
+}
+
+static inline int
+check_compat_entry_size_and_hooks(struct ipt_entry *e,
+      struct xt_table_info *newinfo,
+      unsigned int *size,
+      unsigned char *base,
+      unsigned char *limit,
+      unsigned int *hook_entries,
+      unsigned int *underflows,
+      unsigned int *i,
+      const char *name)
```

```
+{
+ struct ipt_entry_target *t;
+ struct ipt_target *target;
+ u_int16_t entry_offset;
+ int ret, off, h, j;
+
+ duprintf("check_compat_entry_size_and_hooks %p\n", e);
+ if ((unsigned long)e % __alignof__(struct compat_ipt_entry) != 0
+     || (unsigned char *)e + sizeof(struct compat_ipt_entry) >= limit) {
+  duprintf("Bad offset %p, limit = %p\n", e, limit);
+  return -EINVAL;
+ }
+
+ if (e->next_offset < sizeof(struct compat_ipt_entry) +
+    sizeof(struct compat_xt_entry_target)) {
+  duprintf("checking: element %p size %u\n",
+     e, e->next_offset);
+  return -EINVAL;
+ }
+
+ if (!ip_checkentry(&e->ip)) {
+  duprintf("ip_tables: ip check failed %p %s.\n", e, name);
+  return -EINVAL;
+ }
+
+ off = 0;
+ entry_offset = (void *)e - (void *)base;
+ j = 0;
+ ret = IPT_MATCH_ITERATE(e, compat_check_calc_match, name, &e->ip,
+    e->comefrom, &off, &j);
+ if (ret != 0)
+  goto out;
+
+ t = ipt_get_target(e);
+ target = try_then_request_module(xt_find_target(AF_INET,
+           t->u.user.name,
+           t->u.user.revision),
+       "ipt_%s", t->u.user.name);
+ if (IS_ERR(target) || !target) {
+  duprintf("check_entry: `%s' not found\n", t->u.user.name);
+  ret = target ? PTR_ERR(target) : -ENOENT;
+  goto out;
+ }
+ t->u.kernel.target = target;
+
+ if (t->u.kernel.target->compat)
+  t->u.kernel.target->compat(t, NULL, &off, COMPAT_CALC_SIZE);
+ else
```

```
+  xt_compat_target(t, NULL, &off, COMPAT_CALC_SIZE);
+ *size += off;
+ ret = compat_add_offset(entry_offset, off);
+ if (ret)
+  goto out;
+
+ /* Check hooks & underflows */
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)e - base == hook_entries[h])
+   newinfo->hook_entry[h] = hook_entries[h];
+  if ((unsigned char *)e - base == underflows[h])
+   newinfo->underflow[h] = underflows[h];
+ }
+
+ /* Clear counters and comefrom */
+ e->counters = ((struct ipt_counters) { 0, 0 });
+ e->comefrom = 0;
+
+ (*i)++;
+ return 0;
+out:
+ IPT_MATCH_ITERATE(e, cleanup_match, &j);
+ return ret;
+}
+
+static inline int compat_copy_match_from_user(struct ipt_entry_match *m,
+ void **dstptr, compat_uint_t *size, const char *name,
+ const struct ipt_ip *ip, unsigned int hookmask)
+{
+ struct ipt_entry_match *dm;
+ struct ipt_match *match;
+ int ret;
+
+ dm = (struct ipt_entry_match *)*dstptr;
+ match = m->u.kernel.match;
+ if (match->compat)
+  match->compat(m, dstptr, size, COMPAT_FROM_USER);
+ else
+  xt_compat_match(m, dstptr, size, COMPAT_FROM_USER);
+
+ ret = xt_check_match(match, AF_INET, dm->u.match_size - sizeof(*dm),
+       name, hookmask, ip->proto,
+       ip->invflags & IPT_INV_PROTO);
+ if (ret)
+  return ret;
+
+ if (m->u.kernel.match->checkentry
+     && !m->u.kernel.match->checkentry(name, ip, match, dm->data,
```

```
+           dm->u.match_size - sizeof(*dm),
+           hookmask)) {
+  duprintf("ip_tables: check failed for `%s'.\n",
+    m->u.kernel.match->name);
+  return -EINVAL;
+ }
+ return 0;
+}
+
+static int compat_copy_entry_from_user(struct ipt_entry *e, void **dstptr,
+ unsigned int *size, const char *name,
+ struct xt_table_info *newinfo, unsigned char *base)
+{
+ struct ipt_entry_target *t;
+ struct ipt_target *target;
+ struct ipt_entry *de;
+ unsigned int origsize;
+ int ret, h;
+
+ ret = 0;
+ origsize = *size;
+ de = (struct ipt_entry *)*dstptr;
+ memcpy(de, e, sizeof(struct ipt_entry));
+
+ *dstptr += sizeof(struct compat_ipt_entry);
+ ret = IPT_MATCH_ITERATE(e, compat_copy_match_from_user, dstptr, size,
+   name, &de->ip, de->comefrom);
+ if (ret)
+  goto out;
+ de->target_offset = e->target_offset - (origsize - *size);
+ t = ipt_get_target(e);
+ target = t->u.kernel.target;
+ if (target->compat)
+  target->compat(t, dstptr, size, COMPAT_FROM_USER);
+ else
+  xt_compat_target(t, dstptr, size, COMPAT_FROM_USER);
+
+ de->next_offset = e->next_offset - (origsize - *size);
+ for (h = 0; h < NF_IP_NUMHOOKS; h++) {
+  if ((unsigned char *)de - base < newinfo->hook_entry[h])
+   newinfo->hook_entry[h] -= origsize - *size;
+  if ((unsigned char *)de - base < newinfo->underflow[h])
+   newinfo->underflow[h] -= origsize - *size;
+ }
+
+ t = ipt_get_target(de);
+ target = t->u.kernel.target;
+ ret = xt_check_target(target, AF_INET, t->u.target_size - sizeof(*t),
```

```
+         name, e->comefrom, e->ip.proto,
+         e->ip.invflags & IPT_INV_PROTO);
+ if (ret)
+  goto out;
+
+ ret = -EINVAL;
+ if (t->u.kernel.target == &ipt_standard_target) {
+  if (!standard_check(t, *size))
+   goto out;
+ } else if (t->u.kernel.target->checkentry
+     && !t->u.kernel.target->checkentry(name, de, target,
+    t->data, t->u.target_size - sizeof(*t),
+    de->comefrom)) {
+  duprintf("ip_tables: compat: check failed for `%s'.\n",
+    t->u.kernel.target->name);
+  goto out;
+ }
+ ret = 0;
+out:
  return ret;
 }

 static int
-get_entries(const struct ipt_get_entries *entries,
-     struct ipt_get_entries __user *uptr)
+translate_compat_table(const char *name,
+  unsigned int valid_hooks,
+  struct xt_table_info **pinfo,
+  void **pentry0,
+  unsigned int total_size,
+  unsigned int number,
+  unsigned int *hook_entries,
+  unsigned int *underflows)
 {
+ unsigned int i;
+ struct xt_table_info *newinfo, *info;
+ void *pos, *entry0, *entry1;
+ unsigned int size;
  int ret;
- struct ipt_table *t;

- t = xt_find_table_lock(AF_INET, entries->name);
- if (t && !IS_ERR(t)) {
-  struct xt_table_info *private = t->private;
-  duprintf("t->private->number = %u\n",
-    private->number);
-  if (entries->size == private->size)
-   ret = copy_entries_to_user(private->size,
```

```
-         t, uptr->entrytable);
-  else {
-   duprintf("get_entries: I've got %u not %u!\n",
-     private->size,
-     entries->size);
-   ret = -EINVAL;
+ info = *pinfo;
+ entry0 = *pentry0;
+ size = total_size;
+ info->number = number;
+
+ /* Init all hooks to impossible value. */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  info->hook_entry[i] = 0xFFFFFFFF;
+  info->underflow[i] = 0xFFFFFFFF;
+ }
+
+ duprintf("translate_compat_table: size %u\n", info->size);
+ i = 0;
+ xt_compat_lock(AF_INET);
+ /* Walk through entries, checking offsets. */
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+     check_compat_entry_size_and_hooks,
+     info, &size, entry0,
+     entry0 + total_size,
+     hook_entries, underflows, &i, name);
+ if (ret != 0)
+  goto out_unlock;
+
+ ret = -EINVAL;
+ if (i != number) {
+  duprintf("translate_compat_table: %u not %u entries\n",
+     i, number);
+  goto out_unlock;
+ }
+
+ /* Check hooks all assigned */
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  /* Only hooks which are valid */
+  if (!(valid_hooks & (1 << i)))
+   continue;
+  if (info->hook_entry[i] == 0xFFFFFFFF) {
+   duprintf("Invalid hook entry %u %u\n",
+     i, hook_entries[i]);
+   goto out_unlock;
   }
-  module_put(t->me);
-  xt_table_unlock(t);
```

```
- } else
-  ret = t ? PTR_ERR(t) : -ENOENT;
+  if (info->underflow[i] == 0xFFFFFFFF) {
+   duprintf("Invalid underflow %u %u\n",
+     i, underflows[i]);
+   goto out_unlock;
+  }
+ }
+
+ ret = -ENOMEM;
+ newinfo = xt_alloc_table_info(size);
+ if (!newinfo)
+  goto out_unlock;
+
+ newinfo->number = number;
+ for (i = 0; i < NF_IP_NUMHOOKS; i++) {
+  newinfo->hook_entry[i] = info->hook_entry[i];
+  newinfo->underflow[i] = info->underflow[i];
+ }
+ entry1 = newinfo->entries[raw_smp_processor_id()];
+ pos = entry1;
+ size =  total_size;
+ ret = IPT_ENTRY_ITERATE(entry0, total_size,
+   compat_copy_entry_from_user, &pos, &size,
+   name, newinfo, entry1);
+ compat_flush_offsets();
+ xt_compat_unlock(AF_INET);
+ if (ret)
+  goto free_newinfo;
+
+ ret = -ELOOP;
+ if (!mark_source_chains(newinfo, valid_hooks, entry1))
+  goto free_newinfo;
+
+ /* And one copy for every other CPU */
+ for_each_cpu(i)
+  if (newinfo->entries[i] && newinfo->entries[i] != entry1)
+   memcpy(newinfo->entries[i], entry1, newinfo->size);
+
+ *pinfo = newinfo;
+ *pentry0 = entry1;
+ xt_free_table_info(info);
+ return 0;

+free_newinfo:
+ xt_free_table_info(newinfo);
+out:
  return ret;
```

```
+out_unlock:
+ xt_compat_unlock(AF_INET);
+ goto out;
 }

 static int
-do_replace(void __user *user, unsigned int len)
+compat_do_replace(void __user *user, unsigned int len)
 {
  int ret;
- struct ipt_replace tmp;
- struct ipt_table *t;
- struct xt_table_info *newinfo, *oldinfo;
- struct xt_counters *counters;
- void *loc_cpu_entry, *loc_cpu_old_entry;
+ struct compat_ipt_replace tmp;
+ struct xt_table_info *newinfo;
+ void *loc_cpu_entry;

  if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
   return -EFAULT;
@@ -949,151 +1818,201 @@ do_replace(void __user *user, unsigned i
   goto free_newinfo;
  }

- counters = vmalloc(tmp.num_counters * sizeof(struct xt_counters));
- if (!counters) {
-  ret = -ENOMEM;
+ ret = translate_compat_table(tmp.name, tmp.valid_hooks,
+         &newinfo, &loc_cpu_entry, tmp.size,
+         tmp.num_entries, tmp.hook_entry, tmp.underflow);
+ if (ret != 0)
   goto free_newinfo;
- }

- ret = translate_table(tmp.name, tmp.valid_hooks,
-         newinfo, loc_cpu_entry, tmp.size, tmp.num_entries,
-         tmp.hook_entry, tmp.underflow);
- if (ret != 0)
-  goto free_newinfo_counters;
+ duprintf("compat_do_replace: Translated table\n");

- duprintf("ip_tables: Translated table\n");
+ ret = __do_replace(tmp.name, tmp.valid_hooks,
+         newinfo, tmp.num_counters,
+         compat_ptr(tmp.counters));
+ if (ret)
+  goto free_newinfo_untrans;
```

```
+ return 0;

- t = try_then_request_module(xt_find_table_lock(AF_INET, tmp.name),
-        "iptable_%s", tmp.name);
- if (!t || IS_ERR(t)) {
-  ret = t ? PTR_ERR(t) : -ENOENT;
-  goto free_newinfo_counters_untrans;
- }
+ free_newinfo_untrans:
+ IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
+ free_newinfo:
+ xt_free_table_info(newinfo);
+ return ret;
+}

- /* You lied! */
- if (tmp.valid_hooks != t->valid_hooks) {
-  duprintf("Valid hook crap: %08X vs %08X\n",
-    tmp.valid_hooks, t->valid_hooks);
-  ret = -EINVAL;
-  goto put_module;
- }
+static int
+compat_do_ipt_set_ctl(struct sock *sk, int cmd, void __user *user,
+  unsigned int len)
+{
+ int ret;

- oldinfo = xt_replace_table(t, tmp.num_counters, newinfo, &ret);
- if (!oldinfo)
-  goto put_module;
+ if (!capable(CAP_NET_ADMIN))
+  return -EPERM;

- /* Update module usage count based on number of rules */
- duprintf("do_replace: oldnum=%u, initnum=%u, newnum=%u\n",
-  oldinfo->number, oldinfo->initial_entries, newinfo->number);
- if ((oldinfo->number > oldinfo->initial_entries) ||
-     (newinfo->number <= oldinfo->initial_entries))
-  module_put(t->me);
- if ((oldinfo->number > oldinfo->initial_entries) &&
-     (newinfo->number <= oldinfo->initial_entries))
-  module_put(t->me);
+ switch (cmd) {
+ case IPT_SO_SET_REPLACE:
+  ret = compat_do_replace(user, len);
+  break;
```

```
- /* Get the old counters. */
- get_counters(oldinfo, counters);
- /* Decrease module usage counts and free resource */
- loc_cpu_old_entry = oldinfo->entries[raw_smp_processor_id()];
- IPT_ENTRY_ITERATE(loc_cpu_old_entry, oldinfo->size, cleanup_entry,NULL);
- xt_free_table_info(oldinfo);
- if (copy_to_user(tmp.counters, counters,
-    sizeof(struct xt_counters) * tmp.num_counters) != 0)
-  ret = -EFAULT;
- vfree(counters);
- xt_table_unlock(t);
- return ret;
+ case IPT_SO_SET_ADD_COUNTERS:
+  ret = do_add_counters(user, len, 1);
+  break;
+
+ default:
+  duprintf("do_ipt_set_ctl:  unknown request %i\n", cmd);
+  ret = -EINVAL;
+ }

- put_module:
- module_put(t->me);
- xt_table_unlock(t);
- free_newinfo_counters_untrans:
- IPT_ENTRY_ITERATE(loc_cpu_entry, newinfo->size, cleanup_entry,NULL);
- free_newinfo_counters:
- vfree(counters);
- free_newinfo:
- xt_free_table_info(newinfo);
  return ret;
 }

-/* We're lazy, and add to the first CPU; overflow works its fey magic
- * and everything is OK. */
-static inline int
-add_counter_to_entry(struct ipt_entry *e,
-       const struct xt_counters addme[],
-       unsigned int *i)
+struct compat_ipt_get_entries
 {
-#if 0
- duprintf("add_counter: Entry %u %lu/%lu + %lu/%lu\n",
-   *i,
-   (long unsigned int)e->counters.pcnt,
-   (long unsigned int)e->counters.bcnt,
-   (long unsigned int)addme[*i].pcnt,
-   (long unsigned int)addme[*i].bcnt);
```

```
-#endif
+ char name[IPT_TABLE_MAXNAMELEN];
+ compat_uint_t size;
+ struct compat_ipt_entry entrytable[0];
+};

- ADD_COUNTER(e->counters, addme[*i].bcnt, addme[*i].pcnt);
+static int compat_copy_entries_to_user(unsigned int total_size,
+       struct ipt_table *table, void __user *userptr)
+{
+ unsigned int off, num;
+ struct compat_ipt_entry e;
+ struct xt_counters *counters;
+ struct xt_table_info *private = table->private;
+ void __user *pos;
+ unsigned int size;
+ int ret = 0;
+ void *loc_cpu_entry;

- (*i)++;
- return 0;
+ counters = alloc_counters(table);
+ if (IS_ERR(counters))
+  return PTR_ERR(counters);
+
+ /* choose the copy that is on our node/cpu, ...
+  * This choice is lazy (because current thread is
+  * allowed to migrate to another cpu)
+  */
+ loc_cpu_entry = private->entries[raw_smp_processor_id()];
+ pos = userptr;
+ size = total_size;
+ ret = IPT_ENTRY_ITERATE(loc_cpu_entry, total_size,
+   compat_copy_entry_to_user, &pos, &size);
+ if (ret)
+  goto free_counters;
+
+ /* ... then go back and fix counters and names */
+ for (off = 0, num = 0; off < size; off += e.next_offset, num++) {
+  unsigned int i;
+  struct ipt_entry_match m;
+  struct ipt_entry_target t;
+
+  ret = -EFAULT;
+  if (copy_from_user(&e, userptr + off,
+     sizeof(struct compat_ipt_entry)))
+   goto free_counters;
+  if (copy_to_user(userptr + off +
```

```
+   offsetof(struct compat_ipt_entry, counters),
+    &counters[num], sizeof(counters[num])))
+   goto free_counters;
+
+  for (i = sizeof(struct compat_ipt_entry);
+    i < e.target_offset; i += m.u.match_size) {
+   if (copy_from_user(&m, userptr + off + i,
+     sizeof(struct ipt_entry_match)))
+    goto free_counters;
+   if (copy_to_user(userptr + off + i +
+    offsetof(struct ipt_entry_match, u.user.name),
+    m.u.kernel.match->name,
+    strlen(m.u.kernel.match->name) + 1))
+    goto free_counters;
+  }
+
+  if (copy_from_user(&t, userptr + off + e.target_offset,
+     sizeof(struct ipt_entry_target)))
+   goto free_counters;
+  if (copy_to_user(userptr + off + e.target_offset +
+   offsetof(struct ipt_entry_target, u.user.name),
+   t.u.kernel.target->name,
+   strlen(t.u.kernel.target->name) + 1))
+   goto free_counters;
+ }
+ ret = 0;
+free_counters:
+ vfree(counters);
+ return ret;
 }

 static int
-do_add_counters(void __user *user, unsigned int len)
+compat_get_entries(struct compat_ipt_get_entries __user *uptr, int *len)
 {
- unsigned int i;
- struct xt_counters_info tmp, *paddc;
+ int ret;
+ struct compat_ipt_get_entries get;
  struct ipt_table *t;
- struct xt_table_info *private;
- int ret = 0;
- void *loc_cpu_entry;

- if (copy_from_user(&tmp, user, sizeof(tmp)) != 0)
-  return -EFAULT;

- if (len != sizeof(tmp) + tmp.num_counters*sizeof(struct xt_counters))
```

```
+ if (*len < sizeof(get)) {
+  duprintf("compat_get_entries: %u < %u\n",
+    *len, (unsigned int)sizeof(get));
   return -EINVAL;
+ }

- paddc = vmalloc_node(len, numa_node_id());
- if (!paddc)
-  return -ENOMEM;
+ if (copy_from_user(&get, uptr, sizeof(get)) != 0)
+  return -EFAULT;

- if (copy_from_user(paddc, user, len) != 0) {
-  ret = -EFAULT;
-  goto free;
+ if (*len != sizeof(struct compat_ipt_get_entries) + get.size) {
+  duprintf("compat_get_entries: %u != %u\n", *len,
+   (unsigned int)(sizeof(struct compat_ipt_get_entries) +
+   get.size));
+  return -EINVAL;
  }

- t = xt_find_table_lock(AF_INET, tmp.name);
- if (!t || IS_ERR(t)) {
+ xt_compat_lock(AF_INET);
+ t = xt_find_table_lock(AF_INET, get.name);
+ if (t && !IS_ERR(t)) {
+  struct xt_table_info *private = t->private;
+  struct xt_table_info info;
+  duprintf("t->private->number = %u\n",
+    private->number);
+  ret = compat_table_info(private, &info);
+  if (!ret && get.size == info.size) {
+   ret = compat_copy_entries_to_user(private->size,
+         t, uptr->entrytable);
+  } else if (!ret) {
+   duprintf("compat_get_entries: I've got %u not %u!\n",
+     private->size,
+     get.size);
+   ret = -EINVAL;
+  }
+  compat_flush_offsets();
+  module_put(t->me);
+  xt_table_unlock(t);
+ } else
   ret = t ? PTR_ERR(t) : -ENOENT;
-  goto free;
- }
```

```
- write_lock_bh(&t->lock);
- private = t->private;
- if (private->number != paddc->num_counters) {
-  ret = -EINVAL;
-  goto unlock_up_free;
- }
+ xt_compat_unlock(AF_INET);
+ return ret;
+}

- i = 0;
- /* Choose the copy that is on our node */
- loc_cpu_entry = private->entries[raw_smp_processor_id()];
- IPT_ENTRY_ITERATE(loc_cpu_entry,
-     private->size,
-     add_counter_to_entry,
-     paddc->counters,
-     &i);
- unlock_up_free:
- write_unlock_bh(&t->lock);
- xt_table_unlock(t);
- module_put(t->me);
- free:
- vfree(paddc);
+static int
+compat_do_ipt_get_ctl(struct sock *sk, int cmd, void __user *user, int *len)
+{
+ int ret;

+ switch (cmd) {
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len, 1);
+  break;
+ case IPT_SO_GET_ENTRIES:
+  ret = compat_get_entries(user, len);
+  break;
+ default:
+  duprintf("compat_do_ipt_get_ctl: unknown request %i\n", cmd);
+  ret = -EINVAL;
+ }
  return ret;
 }
+#endif

 static int
 do_ipt_set_ctl(struct sock *sk, int cmd, void __user *user, unsigned int len)
@@ -1109,7 +2028,7 @@ do_ipt_set_ctl(struct sock *sk, int cmd,
```

```
   break;

  case IPT_SO_SET_ADD_COUNTERS:
-  ret = do_add_counters(user, len);
+  ret = do_add_counters(user, len, 0);
   break;

  default:
@@ -1129,65 +2048,13 @@ do_ipt_get_ctl(struct sock *sk, int cmd,
   return -EPERM;

  switch (cmd) {
- case IPT_SO_GET_INFO: {
-  char name[IPT_TABLE_MAXNAMELEN];
-  struct ipt_table *t;
-
-  if (*len != sizeof(struct ipt_getinfo)) {
-   duprintf("length %u != %u\n", *len,
-     sizeof(struct ipt_getinfo));
-   ret = -EINVAL;
-   break;
-  }
-
-  if (copy_from_user(name, user, sizeof(name)) != 0) {
-   ret = -EFAULT;
-   break;
-  }
-  name[IPT_TABLE_MAXNAMELEN-1] = '\0';
-
-  t = try_then_request_module(xt_find_table_lock(AF_INET, name),
-         "iptable_%s", name);
-  if (t && !IS_ERR(t)) {
-   struct ipt_getinfo info;
-   struct xt_table_info *private = t->private;
-
-   info.valid_hooks = t->valid_hooks;
-   memcpy(info.hook_entry, private->hook_entry,
-          sizeof(info.hook_entry));
-   memcpy(info.underflow, private->underflow,
-          sizeof(info.underflow));
-   info.num_entries = private->number;
-   info.size = private->size;
-   memcpy(info.name, name, sizeof(info.name));
-
-   if (copy_to_user(user, &info, *len) != 0)
-    ret = -EFAULT;
-   else
-    ret = 0;
```

```
-   xt_table_unlock(t);
-   module_put(t->me);
-  } else
-   ret = t ? PTR_ERR(t) : -ENOENT;
- }
- break;
-
- case IPT_SO_GET_ENTRIES: {
-  struct ipt_get_entries get;
+ case IPT_SO_GET_INFO:
+  ret = get_info(user, len, 0);
+  break;

-  if (*len < sizeof(get)) {
-   duprintf("get_entries: %u < %u\n", *len, sizeof(get));
-   ret = -EINVAL;
-  } else if (copy_from_user(&get, user, sizeof(get)) != 0) {
-   ret = -EFAULT;
-  } else if (*len != sizeof(struct ipt_get_entries) + get.size) {
-   duprintf("get_entries: %u != %u\n", *len,
-     sizeof(struct ipt_get_entries) + get.size);
-   ret = -EINVAL;
-  } else
-   ret = get_entries(&get, user);
+ case IPT_SO_GET_ENTRIES:
+  ret = get_entries(user, len);
   break;
- }

  case IPT_SO_GET_REVISION_MATCH:
  case IPT_SO_GET_REVISION_TARGET: {
@@ -1336,6 +2203,9 @@ static struct ipt_target ipt_standard_ta
  .name  = IPT_STANDARD_TARGET,
  .targetsize = sizeof(int),
  .family  = AF_INET,
+#ifdef CONFIG_COMPAT
+ .compat  = &compat_ipt_standard_fn,
+#endif
 };

 static struct ipt_target ipt_error_target = {
@@ -1350,9 +2220,11 @@ static struct nf_sockopt_ops ipt_sockopt
  .set_optmin = IPT_BASE_CTL,
  .set_optmax = IPT_SO_SET_MAX+1,
  .set  = do_ipt_set_ctl,
+ .compat_set = compat_do_ipt_set_ctl,
  .get_optmin = IPT_BASE_CTL,
  .get_optmax = IPT_SO_GET_MAX+1,
```

```
  .get  = do_ipt_get_ctl,
+ .compat_get = compat_do_ipt_get_ctl,
 };

 static struct ipt_match icmp_matchstruct = {
diff --git a/net/netfilter/x_tables.c b/net/netfilter/x_tables.c
index a657ab5..feb8a9e 100644
--- a/net/netfilter/x_tables.c
+++ b/net/netfilter/x_tables.c
@@ -38,6 +38,7 @@ struct xt_af {
  struct list_head match;
  struct list_head target;
  struct list_head tables;
+ struct mutex compat_mutex;
 };

 static struct xt_af *xt;
@@ -272,6 +273,54 @@ int xt_check_match(const struct xt_match
 }
 EXPORT_SYMBOL_GPL(xt_check_match);

+#ifdef CONFIG_COMPAT
+int xt_compat_match(void *match, void **dstptr, int *size, int convert)
+{
+ struct xt_match *m;
+ struct compat_xt_entry_match *pcompat_m;
+ struct xt_entry_match *pm;
+ u_int16_t msize;
+ int off, ret;
+
+ ret = 0;
+ m = ((struct xt_entry_match *)match)->u.kernel.match;
+ off = XT_ALIGN(m->matchsize) - COMPAT_XT_ALIGN(m->matchsize);
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pm = (struct xt_entry_match *)match;
+   msize = pm->u.user.match_size;
+   if (__copy_to_user(*dstptr, pm, msize)) {
+    ret = -EFAULT;
+    break;
+   }
+   msize -= off;
+   if (put_user(msize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += msize;
+   break;
+  case COMPAT_FROM_USER:
```

```
+   pcompat_m = (struct compat_xt_entry_match *)match;
+   pm = (struct xt_entry_match *)*dstptr;
+   msize = pcompat_m->u.user.match_size;
+   memcpy(pm, pcompat_m, msize);
+   msize += off;
+   pm->u.user.match_size = msize;
+   *size += off;
+   *dstptr += msize;
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+EXPORT_SYMBOL_GPL(xt_compat_match);
+#endif
+
 int xt_check_target(const struct xt_target *target, unsigned short family,
       unsigned int size, const char *table, unsigned int hook_mask,
       unsigned short proto, int inv_proto)
@@ -301,6 +350,54 @@ int xt_check_target(const struct xt_targ
 }
 EXPORT_SYMBOL_GPL(xt_check_target);

+#ifdef CONFIG_COMPAT
+int xt_compat_target(void *target, void **dstptr, int *size, int convert)
+{
+ struct xt_target *t;
+ struct compat_xt_entry_target *pcompat;
+ struct xt_entry_target *pt;
+ u_int16_t tsize;
+ int off, ret;
+
+ ret = 0;
+ t = ((struct xt_entry_target *)target)->u.kernel.target;
+ off = XT_ALIGN(t->targetsize) - COMPAT_XT_ALIGN(t->targetsize);
+ switch (convert) {
+  case COMPAT_TO_USER:
+   pt = (struct xt_entry_target *)target;
+   tsize = pt->u.user.target_size;
+   if (__copy_to_user(*dstptr, pt, tsize)) {
+    ret = -EFAULT;
+    break;
+   }
```

```
+   tsize -= off;
+   if (put_user(tsize, (u_int16_t *)*dstptr))
+    ret = -EFAULT;
+   *size -= off;
+   *dstptr += tsize;
+   break;
+  case COMPAT_FROM_USER:
+   pcompat = (struct compat_xt_entry_target *)target;
+   pt = (struct xt_entry_target *)*dstptr;
+   tsize = pcompat->u.user.target_size;
+   memcpy(pt, pcompat, tsize);
+   tsize += off;
+   pt->u.user.target_size = tsize;
+   *size += off;
+   *dstptr += tsize;
+   break;
+  case COMPAT_CALC_SIZE:
+   *size += off;
+   break;
+  default:
+   ret = -ENOPROTOOPT;
+   break;
+ }
+ return ret;
+}
+EXPORT_SYMBOL_GPL(xt_compat_target);
+#endif
+
 struct xt_table_info *xt_alloc_table_info(unsigned int size)
 {
  struct xt_table_info *newinfo;
@@ -371,6 +468,19 @@ void xt_table_unlock(struct xt_table *ta
 }
 EXPORT_SYMBOL_GPL(xt_table_unlock);

+#ifdef CONFIG_COMPAT
+void xt_compat_lock(int af)
+{
+ mutex_lock(&xt[af].compat_mutex);
+}
+EXPORT_SYMBOL_GPL(xt_compat_lock);
+
+void xt_compat_unlock(int af)
+{
+ mutex_unlock(&xt[af].compat_mutex);
+}
+EXPORT_SYMBOL_GPL(xt_compat_unlock);
+#endif
```

```
 struct xt_table_info *
 xt_replace_table(struct xt_table *table,
@@ -671,6 +781,9 @@ static int __init xt_init(void)

  for (i = 0; i < NPROTO; i++) {
   mutex_init(&xt[i].mutex);
+#ifdef CONFIG_COMPAT
+  mutex_init(&xt[i].compat_mutex);
+#endif
   INIT_LIST_HEAD(&xt[i].target);
   INIT_LIST_HEAD(&xt[i].match);
   INIT_LIST_HEAD(&xt[i].tables);
```

File Attachments

1) diff-ms-netfilter-ipt-compat-20060329, downloaded 690 times

---

Subject: Re: [PATCH] iptables 32bit compat layer
Posted by Patrick McHardy on Wed, 29 Mar 2006 12:32:50 GMT
View Forum Message <> Reply to Message

Dmitry Mishin wrote:
> On Wednesday 29 March 2006 13:28, Patrick McHardy wrote:
>
>>>diff --git a/net/compat.c b/net/compat.c
>>>index 13177a1..6a7028e 100644
>>>--- a/net/compat.c
>>>+++ b/net/compat.c
>>>@@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
>>> int err;
>>> struct socket *sock;
>>>
>>>- /* SO_SET_REPLACE seems to be the same in all levels */
>>>- if (optname == IPT_SO_SET_REPLACE)
>>>+ if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
>>>   return do_netfilter_replace(fd, level, optname,
>>>   optval, optlen);
>>
>>I don't understand the reason for this change. If its not a mistake,
>>it would make more sense to check for IP6T_SO_SET_REPLACE I guess ..
>
> IP6T_SO_SET_REPLACE == IPT_SO_SET_REPLACE == XT_SO_SET_REPLACE.
> Rename will require respective #include directive rename, so, I just leave
> this as it is. BTW, I'll make respective patch for IPV6 in the near future
> and this hunk will be removed at all.

I know, but SOL_IPV6 implies IP6T_* - but please don't bother sending

a new patch for this :) So the point of the change is to exclude IPv6
from the compat layer because its not implemented yet?

Subject: Re: [PATCH] iptables 32bit compat layer
Posted by Mishin Dmitry on Wed, 29 Mar 2006 12:38:01 GMT
View Forum Message <> Reply to Message

On Wednesday 29 March 2006 16:32, Patrick McHardy wrote:
> Dmitry Mishin wrote:
> > On Wednesday 29 March 2006 13:28, Patrick McHardy wrote:
> >>>diff --git a/net/compat.c b/net/compat.c
> >>>index 13177a1..6a7028e 100644
> >>>--- a/net/compat.c
> >>>+++ b/net/compat.c
> >>>@@ -476,8 +476,7 @@ asmlinkage long compat_sys_setsockopt(in
> >>> int err;
> >>> struct socket *sock;
> >>>
> >>>- /* SO_SET_REPLACE seems to be the same in all levels */
> >>>- if (optname == IPT_SO_SET_REPLACE)
> >>>+ if (level == SOL_IPV6 && optname == IPT_SO_SET_REPLACE)
> >>>   return do_netfilter_replace(fd, level, optname,
> >>>   optval, optlen);
> >>
> >>I don't understand the reason for this change. If its not a mistake,
> >>it would make more sense to check for IP6T_SO_SET_REPLACE I guess ..
> >
> > IP6T_SO_SET_REPLACE == IPT_SO_SET_REPLACE == XT_SO_SET_REPLACE.
> > Rename will require respective #include directive rename, so, I just
> > leave this as it is. BTW, I'll make respective patch for IPV6 in the near
> > future and this hunk will be removed at all.
>
> I know, but SOL_IPV6 implies IP6T_* - but please don't bother sending
> a new patch for this :) So the point of the change is to exclude IPv6
> from the compat layer because its not implemented yet?
Exactly. Because do_netfilter_replace still works for some cases, but newer
replacement isn't ready yet.

--
Thanks,
Dmitry.

Subject: Re: [PATCH] iptables 32bit compat layer
Posted by Patrick McHardy on Wed, 29 Mar 2006 12:47:23 GMT
View Forum Message <> Reply to Message

Dmitry Mishin wrote:
> On Wednesday 29 March 2006 16:32, Patrick McHardy wrote:
>>
>>I know, but SOL_IPV6 implies IP6T_* - but please don't bother sending
>>a new patch for this :) So the point of the change is to exclude IPv6
>>from the compat layer because its not implemented yet?
>
> Exactly. Because do_netfilter_replace still works for some cases, but newer
> replacement isn't ready yet.

Ah OK. One last thing before I'll pass it on. There was one 64 bit
architecture where iptables already worked with 32 bit userspace for
some reason. I would like to make sure that it still works before it
goes in because I can't debug it if it fails afterwards. I don't
recall which arch it was, but I think Martin had one of these machines.
Martin, could you give the compat patch a try?

Subject: Re: [PATCH] iptables 32bit compat layer
Posted by davem on Wed, 29 Mar 2006 21:53:51 GMT
View Forum Message <> Reply to Message

From: Martin Josefsson <gandalf@wlug.westbo.se>
Date: Wed, 29 Mar 2006 21:04:45 +0200

> That machine (an old ultra1) hasn't seen electricity in a long time,
> I'll see if I can dig it out this weekend unless someone else (dave?)
> beats me to testing the patch.

I think such userland hacks should not be worried about
and we should put the new correct kernel compat netfilter
stuff in.

If anything explodes on sparc64 for whatever reason, I will take care
of it. :-)

Subject: Re: [PATCH] iptables 32bit compat layer
Posted by Patrick McHardy on Wed, 29 Mar 2006 23:01:15 GMT
View Forum Message <> Reply to Message

David S. Miller wrote:
> From: Martin Josefsson <gandalf@wlug.westbo.se>
> Date: Wed, 29 Mar 2006 21:04:45 +0200
>
>
>>That machine (an old ultra1) hasn't seen electricity in a long time,

>>I'll see if I can dig it out this weekend unless someone else (dave?)
>>beats me to testing the patch.
>
>
> I think such userland hacks should not be worried about
> and we should put the new correct kernel compat netfilter
> stuff in.
>
> If anything explodes on sparc64 for whatever reason, I will take care
> of it. :-)

Fine with me :) I've added it to my tree, I'll pass it on a couple of
hours.
