

---

Subject: [PATCH -mm 1/2] i386: semi-rewrite of PTRACE\_PEEKUSR, PTRACE\_POKEUSR

Posted by [Alexey Dobriyan](#) on Wed, 20 Jun 2007 15:07:34 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Refactor PTRACE\_PEEKUSR, PTRACE\_POKEUSR implementation on i386.  
Ideas and concepts borrowed from utrace patchset by Roland McGrath.

Patch adds only two new concepts: struct ptrace\_usr\_area and struct regset.  
The former describes registers accessible through PTRACE\_PEEKUSR, PTRACE\_POKEUSR in generic way. Where it starts, where it ends, if it's a hole (needed to know when to return 0 and -EIO).

The latter is for abstracting various registers and operations on them (general-purpose regs, FPU regs, etc).

Unlike utrace patchset, usr\_area has direct pointer to corresponding register set instead of magic number in specific array.

Signed-off-by: Alexey Dobriyan <[adobriyan@sw.ru](mailto:adobriyan@sw.ru)>

---

```
arch/i386/kernel/ptrace.c | 192 ++++++-----
include/linux/ptrace.h   | 15 +++
kernel/ptrace.c          | 42 ++++++
3 files changed, 160 insertions(+), 89 deletions(-)
```

--- a/arch/i386/kernel/ptrace.c

+++ b/arch/i386/kernel/ptrace.c

```
@@ -141,6 +141,11 @@ static unsigned long getreg(struct task_struct *child,
    return retval;
}
```

```
+static struct regset i386_gp_regset = {
+ .getreg = getreg,
+ .setreg = putreg,
+};
+
+#define LDT_SEGMENT 4
```

```
static unsigned long convert_eip_to_linear(struct task_struct *child, struct pt_regs *regs)
@@ -349,9 +354,103 @@ ptrace_set_thread_area(struct task_struct *child,
    return 0;
}
```

```
+static unsigned long i386_db_getreg(struct task_struct *tsk, unsigned long addr)
+{
+ addr -= offsetof(struct user, u_debugreg);
```

```

+ addr >>= 2;
+ BUG_ON(addr > 7);
+ return tsk->thread.debugreg;
+}
+
+static int i386_db_setreg(struct task_struct *tsk, unsigned long addr, unsigned long val)
+{
+ int i;
+
+ /*
+  * We need to be very careful here. We implicitly want to modify a
+  * portion of the task_struct, and we have to be selective about what
+  * portions we allow someone to modify.
+  */
+ switch (addr) {
+ case offsetof(struct user, u_debugreg):
+ case offsetof(struct user, u_debugreg):
+ return -EIO;
+ case offsetof(struct user, u_debugreg):
+ case offsetof(struct user, u_debugreg):
+ case offsetof(struct user, u_debugreg):
+ case offsetof(struct user, u_debugreg):
+ if (val > TASK_SIZE - 3)
+ return -EIO;
+ break;
+ case offsetof(struct user, u_debugreg):
+ /*
+  * Sanity-check data. Take one half-byte at once with
+  * check = (val >> (16 + 4*i)) & 0xf. It contains the R/Wi and
+  * LENi bits; bits 0 and 1 are R/Wi, and bits 2 and 3 are LENi.
+  * Given a list of invalid values, we do
+  * mask |= 1 << invalid_value, so that (mask >> check) & 1 is a
+  * correct test for invalid values.
+  *
+  * R/Wi contains the type of the breakpoint / watchpoint, LENi
+  * contains the length of the watched data in the watchpoint
+  * case.
+  *
+  * The invalid values are:
+  * - LENi == 0x10 (undefined), so mask |= 0x0f00.
+  * - R/Wi == 0x10 (break on I/O reads or writes), so
+  *   mask |= 0x4444.
+  * - R/Wi == 0x00 && LENi != 0x00, so we have mask |= 0x1110.
+  *
+  * Finally, mask = 0x0f00 | 0x4444 | 0x1110 == 0x5f54.
+  *
+  * See the Intel Manual "System Programming Guide", 15.2.4
+  */

```

```

+ * Note that LENi == 0x10 is defined on x86_64 in long mode
+ * (i.e. even for 32-bit userspace software, but 64-bit
+ * kernel), so the x86_64 mask value is 0x5454. See the AMD
+ * manual no. 24593 (AMD64 System Programming)
+ */
+ val &= ~DR_CONTROL_RESERVED;
+ for (i = 0; i < 4; i++)
+ if ((0x5f54 >> ((val >> (16 + 4 * i)) & 0xf)) & 1)
+ return -EIO;
+ if (val)
+ set_tsk_thread_flag(tsk, TIF_DEBUG);
+ else
+ clear_tsk_thread_flag(tsk, TIF_DEBUG);
+ break;
+ }
+ addr -= offsetof(struct user, u_debugreg);
+ addr >>= 2;
+ BUG_ON(addr > 7);
+ tsk->thread.debugreg = val;
+ return 0;
+}
+
+static struct regset i386_db_regset = {
+ .getreg = i386_db_getreg,
+ .setreg = i386_db_setreg,
+};
+
+static struct ptrace_usr_area i386_usr_area[] = {
+ {
+ .start = 0,
+ .end = FRAME_SIZE * sizeof(unsigned long),
+ .regset = &i386_gp_regset,
+ }, {
+ .start = FRAME_SIZE * sizeof(unsigned long),
+ .end = offsetof(struct user, u_debugreg),
+ .hole = 1,
+ }, {
+ .start = offsetof(struct user, u_debugreg),
+ .end = offsetof(struct user, u_debugreg),
+ .regset = &i386_db_regset,
+ },
+ {}
+};
+
+long arch_ptrace(struct task_struct *child, long request, long addr, long data)
+{
+ struct user * dummy = NULL;
+ int i, ret;

```

```

unsigned long __user *datap = (unsigned long __user *)data;

@@ -363,26 +462,9 @@ long arch_ptrace(struct task_struct *child, long request, long addr, long
data)
    break;

    /* read the word at location addr in the USER area. */
- case PTRACE_PEEKUSR: {
-     unsigned long tmp;
-
-     ret = -EIO;
-     if ((addr & 3) || addr < 0 ||
-         addr > sizeof(struct user) - 3)
-         break;
-
-     tmp = 0; /* Default return condition */
-     if(addr < FRAME_SIZE*sizeof(long))
-     tmp = getreg(child, addr);
-     if(addr >= (long) &dummy->u_debugreg &&
-         addr <= (long) &dummy->u_debugreg){
-         addr -= (long) &dummy->u_debugreg;
-         addr = addr >> 2;
-         tmp = child->thread.debugreg;
-     }
-     ret = put_user(tmp, datap);
+ case PTRACE_PEEKUSR:
+     ret = ptrace_peekusr(child, i386_usr_area, addr, data);
    break;
- }

    /* when I and D space are separate, this will have to be fixed. */
    case PTRACE_POKETEXT: /* write the word at location addr. */
@@ -391,74 +473,7 @@ long arch_ptrace(struct task_struct *child, long request, long addr, long
data)
    break;

    case PTRACE_POKEUSR: /* write the word at location addr in the USER area */
-     ret = -EIO;
-     if ((addr & 3) || addr < 0 ||
-         addr > sizeof(struct user) - 3)
-         break;
-
-     if (addr < FRAME_SIZE*sizeof(long)) {
-         ret = putreg(child, addr, data);
-         break;
-     }
-     /* We need to be very careful here. We implicitly
-        want to modify a portion of the task_struct, and we

```

```

- have to be selective about what portions we allow someone
- to modify. */
-
- ret = -EIO;
- if(addr >= (long) &dummy->u_debugreg &&
-   addr <= (long) &dummy->u_debugreg){
-
-   if(addr == (long) &dummy->u_debugreg) break;
-   if(addr == (long) &dummy->u_debugreg) break;
-   if(addr < (long) &dummy->u_debugreg &&
-     ((unsigned long) data) >= TASK_SIZE-3) break;
-
-   /* Sanity-check data. Take one half-byte at once with
-    * check = (val >> (16 + 4*i)) & 0xf. It contains the
-    * R/Wi and LENi bits; bits 0 and 1 are R/Wi, and bits
-    * 2 and 3 are LENi. Given a list of invalid values,
-    * we do mask |= 1 << invalid_value, so that
-    * (mask >> check) & 1 is a correct test for invalid
-    * values.
-    *
-    * R/Wi contains the type of the breakpoint /
-    * watchpoint, LENi contains the length of the watched
-    * data in the watchpoint case.
-    *
-    * The invalid values are:
-    * - LENi == 0x10 (undefined), so mask |= 0x0f00.
-    * - R/Wi == 0x10 (break on I/O reads or writes), so
-    *   mask |= 0x4444.
-    * - R/Wi == 0x00 && LENi != 0x00, so we have mask |=
-    *   0x1110.
-    *
-    * Finally, mask = 0x0f00 | 0x4444 | 0x1110 == 0x5f54.
-    *
-    * See the Intel Manual "System Programming Guide",
-    * 15.2.4
-    *
-    * Note that LENi == 0x10 is defined on x86_64 in long
-    * mode (i.e. even for 32-bit userspace software, but
-    * 64-bit kernel), so the x86_64 mask value is 0x5454.
-    * See the AMD manual no. 24593 (AMD64 System
-    * Programming)*/
-
-   if(addr == (long) &dummy->u_debugreg) {
-     data &= ~DR_CONTROL_RESERVED;
-     for(i=0; i<4; i++)
-       if ((0x5f54 >> ((data >> (16 + 4*i)) & 0xf)) & 1)
-         goto out_tsk;
-     if (data)

```

```

-   set_tsk_thread_flag(child, TIF_DEBUG);
-   else
-   clear_tsk_thread_flag(child, TIF_DEBUG);
-   }
-   addr -= (long) &dummy->u_debugreg;
-   addr = addr >> 2;
-   child->thread.debugreg[addr] = data;
-   ret = 0;
-   }
+ ret = ptrace_pokeusr(child, i386_usr_area, addr, data);
  break;

```

```

  case PTRACE_SYSEMU: /* continue and stop at next syscall, which will not be executed */
@@ -613,7 +628,6 @@ long arch_ptrace(struct task_struct *child, long request, long addr, long
data)
  ret = ptrace_request(child, request, addr, data);
  break;
}
- out_tsk:
  return ret;
}

```

--- a/include/linux/ptrace.h

+++ b/include/linux/ptrace.h

```

@@ -113,6 +113,21 @@ static inline void ptrace_unlink(struct task_struct *child)
int generic_ptrace_peekdata(struct task_struct *tsk, long addr, long data);
int generic_ptrace_pokedata(struct task_struct *tsk, long addr, long data);

```

```

+struct regset {
+ unsigned long (*getreg)(struct task_struct *tsk, unsigned long addr);
+ int (*setreg)(struct task_struct *tsk, unsigned long addr, unsigned long val);
+};
+
+struct ptrace_usr_area {
+ long start, end; /* [start, end) */
+ struct regset *regset;
+ /* if set, reads from [start, end) return 0, writes return -EIO */
+ unsigned int hole:1;
+};
+
+int ptrace_peekusr(struct task_struct *tsk, struct ptrace_usr_area *usr_area, long addr, long
data);
+int ptrace_pokeusr(struct task_struct *tsk, struct ptrace_usr_area *usr_area, long addr, long
data);
+
+
#ifdef force_successful_syscall_return
/*
 * System call handlers that, upon successful completion, need to return a

```

```

--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -510,3 +510,45 @@ int generic_ptrace_pokedata(struct task_struct *tsk, long addr, long
data)
    copied = access_process_vm(tsk, addr, &data, sizeof(data), 1);
    return (copied == sizeof(data)) ? 0 : -EIO;
}
+
+int ptrace_peekusr(struct task_struct *tsk, struct ptrace_usr_area *usr_area,
+    long addr, long data)
+{
+    if (addr & (sizeof(unsigned long) - 1))
+        return -EIO;
+
+    while (usr_area->end != 0) {
+        if (usr_area->start <= addr && addr < usr_area->end) {
+            unsigned long val;
+
+            if (usr_area->hole)
+                val = 0;
+            else
+                val = usr_area->regset->getreg(tsk, addr);
+            return put_user(val, (unsigned long __user *)data);
+        }
+        usr_area++;
+    }
+    return -EIO;
+}
+
+int ptrace_pokeusr(struct task_struct *tsk, struct ptrace_usr_area *usr_area,
+    long addr, long data)
+{
+    if (addr & (sizeof(unsigned long) - 1))
+        return -EIO;
+
+    while (usr_area->end != 0) {
+        if (usr_area->start <= addr && addr < usr_area->end) {
+            int rv;
+
+            if (usr_area->hole)
+                rv = -EIO;
+            else
+                rv = usr_area->regset->setreg(tsk, addr, data);
+            return rv;
+        }
+        usr_area++;
+    }
+    return -EIO;

```

+}

---