

hci_usb: races, races, races

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-03/msg12130.html>

- *From:* Pavel Machek <pavel@xxxxxx>
 - *Date:* Mon, 31 Mar 2008 22:18:33 +0200
-

Hi!

hci_usb interacts badly with system suspend... it corrupts memory after it was freed. To test, just run ppp-over-bluetooth, do some data transfers, and s2ram. Unplugging hci_usb would probably kill it as well.

```
@@ -447,11 +449,20 @@ static int hci_usb_close(struct hci_dev
```

```
BT_DBG("%s", hdev->name);
```

```
/* Synchronize with completion handlers */  
write_lock_irqsave(&husb->completion_lock, flags);  
write_unlock_irqrestore(&husb->completion_lock, flags);
```

```
- hci_usb_unlink_urbs(husb);  
+// hci_usb_unlink_urbs(husb);  
hci_usb_flush(hdev);  
return 0;  
}
```

Now... hci_usb_unlink_urbs is running in paralel with completions, freeing memory.

Unfortunately, hci_usb_tx_complete touches that memory:

```
@@ -735,14 +748,15 @@ static void hci_usb_tx_complete(struct u  
BT_DBG("%s urb %p status %d flags %x", hdev->name, urb,  
urb->status, urb->transfer_flags);
```

```
+ /* _urb may have been already cleared by hci_usb_unlink_urbs  
*/  
atomic_dec(__pending_tx(husb, _urb->type));
```

```
urb->transfer_buffer = NULL;  
kfree_skb((struct sk_buff *) _urb->priv);
```

```
if (!test_bit(HCI_RUNNING, &hdev->flags))  
return;
```

```

if (!urb->status)
hdev->stat.byte_tx += urb->transfer_buffer_length;
else

```

HCI_RUNNING test is too late... and anyway, USB stack was touching that memory while we were freeing it. I don't know about simple fix.

Workaround is:

```

@@ -447,11 +449,20 @@ static int hci_usb_close(struct hci_dev

```

```

BT_DBG("%s", hdev->name);

```

```

/* Synchronize with completion handlers */
write_lock_irqsave(&husb->completion_lock, flags);
write_unlock_irqrestore(&husb->completion_lock, flags);

```

```

- hci_usb_unlink_urbs(husb);
+// hci_usb_unlink_urbs(husb);
hci_usb_flush(hdev);
return 0;
}

```

hci_usb.h is racy, too:

_urb->queue = q is used for locking, without barriers etc... this could fix it, but it is still ugly. If BUG() never triggers (it really should not) we should remove the test, and we'll get non-ugly solution.

hci_usb.h

```

@@ -75,7 +70,7 @@ static inline void _urb_queue_head(struc
{
unsigned long flags;
spin_lock_irqsave(&q->lock, flags);
- list_add(&_urb->list, &q->head); _urb->queue = q;
+ _urb->queue = q; mb(); list_add(&_urb->list, &q->head);
spin_unlock_irqrestore(&q->lock, flags);
}

```

```

@@ -83,19 +78,32 @@ static inline void _urb_queue_tail(struc
{
unsigned long flags;
spin_lock_irqsave(&q->lock, flags);
- list_add_tail(&_urb->list, &q->head); _urb->queue = q;
+ /* Hmm... we only update _urb->queue _after_ putting it on
+ * list. Someone may see the 0 value... and urb_unlink just
+ * silently does nothing when it sees NULL. */
+ _urb->queue = q; mb(); list_add_tail(&_urb->list, &q->head);
spin_unlock_irqrestore(&q->lock, flags);
}

```

```
}  
  
static inline void _urb_unlink(struct _urb *_urb)  
{  
- struct _urb_queue *q = _urb->queue;  
+ struct _urb_queue *q;  
unsigned long flags;  
+ /* This is at least strange. If we rely on _urb->queue for  
locking  
+ we should test it _inside_ some lock.  
+  
+ In combination with no locking in urb_queue_tail, that  
+ means that we may return with urb still on the list.  
+ */  
+  
+ mb();  
+ q = _urb->queue;  
+  
if (q) {  
spin_lock_irqsave(&q->lock, flags);  
list_del(&_urb->list); _urb->queue = NULL;  
spin_unlock_irqrestore(&q->lock, flags);  
- }  
+ } else BUG();  
}
```

Pavel

--

(english) <http://www.livejournal.com/~pavelmachek>

(cesky, pictures) <http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html>

Attachment: [bt.1.mesg.gz](#)

Description: Binary data

Attachment: [bt.2.mesg.gz](#)

Description: Binary data

Attachment: [bt.3.mesg.gz](#)

Description: Binary data