

[PATCH] usbmon: control the max amount of captured data for eachurb

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-10/msg02869.html>

- *From:* Paolo Abeni <paolo.abeni@xxxxxxxx>
 - *Date:* Sun, 08 Oct 2006 22:34:17 +0200
-

From: Paolo Abeni <paolo.abeni@xxxxxxxx>

An ioctl method is added to each usbmon text files. The ioctl implements two operation: retrieving the max size of data that usbmon is able to capture for each URB, and changing this value.

Captured data is stored in buffers allocated from a slab cache; when the max data value is changed, the old slab cache is destroyed, all captured data is discarded and a new slab is created, with appropriate buffer size. The size of the buffer used to format the data passed to user space is changed accordingly.

This change is useful to get full content of exchanged URB. A recently introduced libpcap patch make it possible to sniff from USB port via usbmon, but the full USB packet contents is currently unavailable.

Signed-off-by: Paolo Abeni <paolo.abeni@xxxxxxxx>

patch against linux 2.6.18

mon_text.c | 178

+++++-----

1 file changed, 161 insertions(+), 17 deletions(-)

--- linux-2.6.18-vanilla/drivers/usb/mon/mon_text.c

+++ linux-2.6.18-monioclt/drivers/usb/mon/mon_text.c

@@ -9,15 +9,17 @@

#include <linux/usb.h>

#include <linux/time.h>

#include <linux/mutex.h>

+#include <linux/ioctl.h>

#include <asm/uaccess.h>

#include "usb_mon.h"

/*

- * No, we do not want arbitrarily long data strings.

- * Use the binary interface if you want to capture bulk data!

+ * This limit can be changed using ioctl

*/

[PATCH] usbmon: control the max amount of captured data for eachurb

```

-#define DATA_MAX 32
+#define DATA_DFL 32
+#define DATA_MIN 16
+#define DATA_MAX 1024

/*
 * Defined by USB 2.0 clause 9.3, table 9.2.
 *@@ -33,6 +35,15 @@
#define EVENT_MAX (2*PAGE_SIZE / sizeof(struct mon_event_text))

#define PRINTF_DFL 160
+#define PRINTF_MIN 120
+
+/* ioctl macros */
+#define MON_IOC_MAGIC 0xff
+
+#define MON_IOCS_DATA_MAX_IOW(MON_IOC_MAGIC, 1, int)
+#define MON_IOC_G_DATA_MAX_IOR(MON_IOC_MAGIC, 2, int)
+
+#define MON_IOC_MAXNR 2

struct mon_event_text {
struct list_head e_link;
@@ -45,12 +56,13 @@ struct mon_event_text {
char setup_flag;
char data_flag;
unsigned char setup[SETUP_MAX];
- unsigned char data[DATA_MAX];
+ unsigned char* data;
};

#define SLAB_NAME_SZ 30
struct mon_reader_text {
kmem_cache_t *e_slab;
+ unsigned data_max;
int nevents;
struct list_head e_list;
struct mon_reader r; /* In C, parent class can be placed anywhere */
@@ -91,14 +103,14 @@ static inline char mon_text_get_setup(st
}

static inline char mon_text_get_data(struct mon_event_text *ep, struct urb *urb,
- int len, char ev_type)
+ int len, char ev_type, int data_max)
{
int pipe = urb->pipe;

if (len <= 0)
return 'L';
- if (len >= DATA_MAX)
- len = DATA_MAX;

```

[PATCH] usbmon: control the max amount of captured data for eachurb

[PATCH] usbmon: control the max amount of captured data for eachurb

```
+ if (len >= data_max)
+ len = data_max;

if (usb_pipein(pipe)) {
if (ev_type == 'S')
@@ -138,6 +150,84 @@ static inline unsigned int mon_get_times
return stamp;
}

+/*
+ * [Re]allocate printf buffer and slab cache accoring to specified sizes.
+ * mon_lock must be hold due to mon_reader_add/remove
+ */
+static int mon_text_resize(struct mon_reader_text* rp, int printf_size,
+ int data_max)
+{
+ unsigned long flags;
+ int ret=0, pos;
+ kmem_cache_t * new_e_slab;
+ char* new_printf_buf, id;
+ char new_name[SLAB_NAME_SZ];
+
+ /* remove this reader from list to avoid urb_submit accessing slab
+ * cache*/
+ mutex_lock(&mon_lock);
+ mon_reader_del(rp->r.m_bus, &rp->r);
+ mutex_unlock(&mon_lock);
+
+ /* try to allocate early all required buffer and preserve old values
+ * in case of failure */
+ if (!(new_printf_buf = kmalloc(printf_size, GFP_KERNEL)))
+ {
+ ret = -1;
+ goto out;
+ }
+
+ /* avoid trying to create two slab with same name: it will fail
+ * swap the last char from '0' to '1' and vice versa*/
+ strcpy(new_name, rp->slab_name);
+ pos = strlen(new_name) - 1;
+ id = new_name[pos];
+ new_name[pos] = (1 - (id - '0')) + '0';
+
+ if (!(new_e_slab = kmem_cache_create(new_name,
+ sizeof(struct mon_event_text)+data_max, sizeof(long), 0,
+ mon_text_ctor, NULL))) {
+ kfree(new_printf_buf);
+ ret = -1;
+ goto out;
+ }
+
+ }
```

[PATCH] usbmon: control the max amount of captured data for eachurb

```
+ /* hold printf lock to avoid read syscall accessing printf_buf/slab*/
+ mutex_lock(&rp->printf_lock);
+ if (rp->e_slab)
+ kmem_cache_destroy(rp->e_slab);
+ rp->e_slab = new_e_slab;
+
+ /* event list must be flush because old entries have differnd size */
+ spin_lock_irqsave(&rp->r.m_bus->lock, flags);
+ INIT_LIST_HEAD(&rp->e_list);
+ rp->nevents = 0;
+ spin_unlock_irqrestore(&rp->r.m_bus->lock, flags);
+ if (rp->printf_buf)
+ kfree(rp->printf_buf);
+ rp->printf_buf = new_printf_buf;
+ rp->printf_size = printf_size;
+ rp->data_max = data_max;
+ strcpy(rp->slab_name, new_name);
+ mutex_unlock(&rp->printf_lock);
+
+out:
+ mutex_lock(&mon_lock);
+ mon_reader_add(rp->r.m_bus, &rp->r);
+ mutex_unlock(&mon_lock);
+ return ret;
+}
+
+static inline struct mon_event_text * mon_event_alloc(
+ struct mon_reader_text *rp)
+{
+ struct mon_event_text *ep = kmem_cache_alloc(rp->e_slab, SLAB_ATOMIC);
+ if (!ep)
+ return 0;
+ ep->data = ((unsigned char*)ep) + sizeof(struct mon_event_text);
+ return ep;
+}
+
+static void mon_text_event(struct mon_reader_text *rp, struct urb *urb,
+ char ev_type)
+{
@@ -147,7 +237,7 @@ static void mon_text_event(struct mon_re
stamp = mon_get_timestamp();

if (rp->nevents >= EVENT_MAX ||
- (ep = kmem_cache_alloc(rp->e_slab, SLAB_ATOMIC)) == NULL) {
+ (ep = mon_event_alloc(rp)) == NULL) {
rp->r.m_bus->cnt_text_lost++;
return;
}
@@ -162,7 +252,8 @@ static void mon_text_event(struct mon_re
ep->status = urb->status;
```

[PATCH] usbmon: control the max amount of captured data for eachurb

```
ep->setup_flag = mon_text_get_setup(ep, urb, ev_type);
- ep->data_flag = mon_text_get_data(ep, urb, ep->length, ev_type);
+ ep->data_flag = mon_text_get_data(ep, urb, ep->length, ev_type,
+ rp->data_max);

rp->nevents++;
list_add_tail(&ep->e_link, &rp->e_list);
@@ -187,7 +278,7 @@ static void mon_text_error(void *data, s
struct mon_event_text *ep;

if (rp->nevents >= EVENT_MAX ||
- (ep = kmem_cache_alloc(rp->e_slab, SLAB_ATOMIC)) == NULL) {
+ (ep = mon_event_alloc(rp)) == NULL) {
rp->r.m_bus->cnt_text_lost++;
return;
}
@@ -249,7 +340,7 @@ static int mon_text_open(struct inode *i
INIT_LIST_HEAD(&rp->e_list);
init_waitqueue_head(&rp->wait);
mutex_init(&rp->printf_lock);
-
+
rp->printf_size = PRINTF_DFL;
rp->printf_buf = kmalloc(rp->printf_size, GFP_KERNEL);
if (rp->printf_buf == NULL) {
@@ -265,8 +356,9 @@ static int mon_text_open(struct inode *i

snprintf(rp->slab_name, SLAB_NAME_SZ, "mon%dt_%lx", ubus->busnum,
(long)rp);
+ rp->data_max = DATA_DFL;
rp->e_slab = kmem_cache_create(rp->slab_name,
- sizeof(struct mon_event_text), sizeof(long), 0,
+ sizeof(struct mon_event_text)+rp->data_max, sizeof(long), 0,
mon_text_ctor, NULL);
if (rp->e_slab == NULL) {
rc = -ENOMEM;
@@ -366,8 +458,8 @@ static ssize_t mon_text_read(struct file
if ((data_len = ep->length) > 0) {
if (ep->data_flag == 0) {
cnt += snprintf(pbuf + cnt, limit - cnt, "=");
- if (data_len >= DATA_MAX)
- data_len = DATA_MAX;
+ if (data_len >= rp->data_max)
+ data_len = rp->data_max;
for (i = 0; i < data_len; i++) {
if (i % 4 == 0) {
cnt += snprintf(pbuf + cnt, limit - cnt,
@@ -435,6 +527,59 @@ static int mon_text_release(struct inode
return 0;
}
}
```

[PATCH] usbmon: control the max amount of captured data for eachurb

```
+static int mon_text_ioctl(struct inode *inode, struct file *file,
+ unsigned int cmd, unsigned long arg)
+{
+ int ret=0, new_data_max;
+ u32 __user *argp = (u32 __user *)arg;
+ struct mon_reader_text *rp;
+
+ /* basic sanity check */
+ if (!(rp = file->private_data))
+ return -ENODEV;
+ if (_IOC_TYPE(cmd) != MON_IOC_MAGIC)
+ return -ENOTTY;
+ if (!capable(CAP_SYS_ADMIN))
+ return -EPERM;
+
+ switch (cmd) {
+ case MON_IOCTL_DATA_MAX:
+ if (put_user(rp->data_max, argp))
+ ret = -EFAULT;
+ break;
+
+ case MON_IOCS_DATA_MAX:
+ if (get_user(new_data_max, argp))
+ {
+ ret = -EFAULT;
+ break;
+ }
+
+ /* buffer should be big enough to handle the "header" and
+ * we want to avoid large memory consumption */
+ if ((new_data_max < DATA_MIN)|| (new_data_max > DATA_MAX))
+ {
+ ret = -EINVAL;
+ break;
+ }
+ if (new_data_max == rp->data_max)
+ break;
+
+ /* try to allocate new buffer before relasing old one
+ * to be safe*/
+ if (mon_text_resize(rp, new_data_max*2 + PRINTF_MIN,
+ new_data_max))
+ ret = -ENOMEM;
+ break;
+
+ default:
+ ret = -ENOTTY;
+ break;
+ }
+
+ return ret;
```

[PATCH] usbmon: control the max amount of captured data for eachurb

```
+}
+
struct file_operations mon_fops_text = {
.owner = THIS_MODULE,
.open = mon_text_open,
@@ -442,7 +587,7 @@ struct file_operations mon_fops_text = {
.read = mon_text_read,
/* .write = mon_text_write, */
/* .poll = mon_text_poll, */
- /* .ioctl = mon_text_ioctl, */
+ .ioctl = mon_text_ioctl,
.release = mon_text_release,
};

@@ -455,6 +600,5 @@ static void mon_text_ctor(void *mem, kme
* Nothing to initialize. No, really!
* So, we fill it with garbage to emulate a reused object.
*/
- memset(mem, 0xe5, sizeof(struct mon_event_text));
+ memset(mem, 0xe5, kmem_cache_size(slab));
}
-
```

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>