Tvheadend - Bug #5558

linuxdvb status / diseqc - global_lock problems with complex setup and/or slow drivers

2019-02-24 13:56 - Deep Thought

Status:	New	**Start date:**	2019-02-24
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	General	**Estimated time:**	0.00 hour
Target version:	4.6		
Found in version:	4.3 - 39db47829b65f140f337d4af3110a8906fed6ff8	**Affected Versions:**	

Description

With tvheadend 4.3 I noticed many problems with data corruption c,
which occur when background streams are running or epg scan is active.

This produces problems like
-Continuity counter errors in the log file
-mpegts: too much queued table input data (over 2MB)
-epg which populates slowly and which is incomplete
-minor block distortion in streams, often coinciding with epggrab tuning
to a different transponder, or a background stream starting.
-rarely: completely corrupted (unwatchable) streams when tuning while epg grab
is in progress (only on some transponders, so could be another problem)

Similar problems have been recorded by others in various tickets (some still open).

I know that these problems can be caused by faulty hardware or poor
signal, but I do not think this is the case here: strong transponders with
problems + problems also when transponders do not share any LNB or switch.

Based on a hunch that some threads cannot keep up with the incoming data,
I instrumented the code to find out how long threads lock the global_lock
and/or how long they are waiting to lock it.

My tests are not finished, I have not been able to instrument all occurences
of global_lock, and similar problems could occur for other locks.
However, I have already discovered several problems:

1. mpegts_input_table_thread has two phases: wait for data and process.
The process phase calls tvh_mutex_lock(&global_lock). This call often takes
very long: in my current log (covering 1 day)
I have 100 cases where it takes longer than 1 second.
In 3 cases, it takes 13 seconds! In 14 cases it takes over 4 seconds.

This is a time critical thread and these wait times are much too long. They could explain
why epg data is lost. More importantly, they show that some thread(s) keep global_lock
locked too long, and/or that the lock is too heavily contended. A good result should be
well below 200ms (preferably much less).

2. mtimer_thread runs delayed callbacks. I do not understand the code well enough, but I see
that often these callbacks are launched in bursts (e.g., at the end of scanning one epg
transponder)

The callbacks are started as follows:
```
tvh_mutex_lock(&global_lock);
dttime_init();
if (mtimer_running == mti) {
tprofile_start(&mtimer_profile, id);
cb(mti->mti_opaque);
tprofile_finish(&mtimer_profile);
}
```

dttime(500);
tvh_mutex_unlock(&global_lock);

In 50 cases, my logs show times longer than 500ms for some callbacks.
As these callbacks run with global_lock held, they delay other threads.
Even if a single run takes only a few 100 ms, remember that these calls occur
in bursts. So when an epg scan ends, many of these brief delays will add up to a
a very large one.

In fact, in 42 cases, the delay is longer than 1 second for a single callback.
In 6 cases, the delay for one callback is longer than 4 seconds.

These times are far too long. It could be what causes the delays
seen in mpegts_input_table_thread.

3. I see other delays (long locking of mutexes) in save_thread (idnode.c), but these
could be false alerts, as the locking/unlocking is bit complicated and my timing code
might measure the wrong thing. So I need to check those.

Currently my conclusion is that some callback in
mtimer_thread is taking too much time and causes some or all of the problems I have
noticed.

As the global_lock is used in many places, it is essential that it is never held
for a long period of time. As it is called many times, there is also a big risk for
livelock.

Note that in some parts of the code, lock/unlock pairs occur close too each other in the
code, which is good as it helps to identify problems and to clearly show that global_lock
is taken.

However, in other parts of the code, global_lock is acquired and released in non obvious places (e.g., in separate functions). This is quite risky, as it hides that the lock is taken.

History

#1 - 2019-02-24 14:01 - Deep Thought

Here is the code I use for timing (recycled from old software):

```
#include <sys/timeb.h>
inline int _dttime(struct timeb *dt_timer,int timeout,const char*func,int line)
{
    struct timeb now;
    ftime(&now);
    int ret=(now.time-dt_timer->time)*1000+now.millitm-dt_timer->millitm;
    memcpy((void*)dt_timer,(void*)&now,sizeof(now));
    if(timeout>=0&&ret>=timeout) {
        tvherror(LS_MPEGTS,"%s_%d TIME: %d\n",func,line,ret);
    }
    return ret;
}

#define dttime_init()\
   struct timeb dt_timer; \
   ftime(&dt_timer);\

#define dttime(timeout) \
    _dttime(&dt_timer,timeout,__FUNCTION__,__LINE__)
```

How to use it:

1. call dttime_init() within the function you want to test. It declares
a time variable and initialises it to curent time

2. calling dttime(timeout) will compare the current time to the last time
and report an error if it is longer than timeout milliseconds. It will also
update the "current" time.

Calling dttime(-1) simply updates the timer without reporting anything.

Typical usage:

```
dttime_init();
...
tvh_mutex_lock(&global_lock);
dttime(-1); //record current time
... code to time...
dttime(500); //report if code took longer than 500 ms
```

#2 - 2019-02-24 14:28 - Deep Thought

- File 20180224.log added

Log file attached.

near line 24: continuity errors at startup. mtimer_thread called
long running callback (4 seconds)

near line 89: huge number of continuity counter errors when scanning freesat
transponder (high bandwith epg). mtimer_thread has called long running callback
(1.4 second). mpegts_input_table_thread has trouble acquiring lock (three times
2 seconds)

near line 173: continuity counter errors on channel film4
(only recording in progress)
mtimer_thread has just called slow callback (1026ms). mpegts_input_table_thread
experiences delays acauiring global lock (1.8second and 2.0 second)

near line 332: possibly unrelated errors: invalid checksums in epg tables,
changes in ts_id. This is a multistream transponder. It is possible that
tvheadend gets confusion between streams, resulting in the the ts_id change
problem. The invalid checksum still remains to be explained.

#3 - 2019-02-24 14:39 - Deep Thought

Possibly related tickets:

#5459 Something is slow in 4.3
"Continuity errors errors are around"

#5446 Continuity errors while recording

#5353 Locked up with "too much queued input data" and "too much queued table input data"

#5326 tvheadend stuck at DTS and PCR diff is very big and after this error pvr.hts says - Command hello failed: No response received.
"mpegts: too much queued input data (over 50MB)"

#5142 TV headend stops working and recording frequently but the service itself is running
"mpegts: too much queued table input data (over 2MB), discarding new"

#4 - 2019-02-24 18:17 - Deep Thought

After several hours of debugging, I tracked down the slow callbacks, which
are called with global_lock locked.

There are two:

- linuxdvb_frontend_monitor: takes about 300ms. Probably does not cause
  serious problems, but 300ms is still a long time to keep other threads waiting
  and it is not clear why the global_lock has to be held in this function
- epggrab_ota_kick_cb: this one is really bad as it often takes 1000 to 3000
  milliseconds to complete. It seems that the whole epg tuning process is
  part of this callback. As tuning is quite slow and involves many steps,
  it is not surprising to see these long call times.

The details are below.

A good solution probably requires significant changes
to the code. Specifically, global locks should be avoided. Assuming that
the global lock is mostly needed to access the idnode database, it should be
used only for that. That means that for tuning, first the tuning data should be looked up while locking global_lock and copied. Then tuning should use the
values, without locking global_lock, or perhaps just locking it if further
idnode access is needed.

=> Is this indeed correct as an approach? Or is it more complicated?

A workaround may be to release global_lock in some of the slow calls and reacquire
it when needed. I do not know if this allowed. Also remember that my experiment
only checked for problems with global_lock. There could be similar problems
with other locks...

==========
Here are the details:
epggrab_ota_kick_cb oftren taken 800 - 3000 ms to complete
it calls mpegts_mux_subscribe, which calls subscription_create_from_channel_or_service

subscription_create_from_channel_or_service often takes 1000 - 2700 ms
The slow part is
if (flags & SUBSCRIPTION_ONESHOT) {
if ((si = subscription_start_instance(s, error)) == NULL) {
subscription_unsubscribe(s, UNSUBSCRIBE_QUIET | UNSUBSCRIBE_FINAL);
return NULL;
}
subscription_link_service(s, si->si_s);
subscription_show_info(s);

This can be traced to subscription_start_instance which often takes 1 to 2.6 seconds
subscription_start_instance calls service_find_instance.

The slow call in service_find_instance
is service_start which taks 1200- 2600 ms

service_start calls t->s_start_feed, which is mpegts_service_start

mpegts_service_start often takes 900 - 2500 ms

mpegts_service_start calls mpegts_mux_instance_start
which often takes 900 - 2800 ms.

mpegts_mux_instance_start calls mi->mi_start_mux which is linuxdvb_frontend_start_mux
which often takes 900 -2500 ms.

linuxdvb_frontend_start_mux calls
linuxdvb_satconf_start_mux which often takes 1000 - 2000ms

linuxdvb_satconf_start_mux calls
linuxdvb_satconf_ele_tune which ofen takes more than 1000ms

The slow parts of linuxdvb_satconf_ele_tune are:
lds[i]->ld_tune often takes more than 1000ms
linuxdvb_frontend_tune1: takes 900ms

#5 - 2019-02-24 18:58 - Flole Systems

Just to be sure I understand this right: You are refering to continuity errors when epg-grabbing or background scanning os doing it's job? Based on your analysis it is related to linuxdvb, so SAT-IP should not be affected? I am seeing continuity errors with SAT-IP aswell, I have tracked down the SAT-IP Server as issue here, I am wondering is this might make the issues I am having worse though.

#6 - 2019-02-24 19:23 - Deep Thought

Based on my analysis it is based on locking problems in the code.
The dvb code is fine but is called while holding the global lock.
Probably it needs to be like that because of the new database (protect data stuctures)
but it causes problems.

It could very well affect other parts of the code as well, including sat-ip,
but I have not tested that.

#7 - 2019-02-24 19:46 - saen acro

Read also #5554 wink.png

#8 - 2019-02-24 20:13 - Jaroslav Kysela

I guess that the linuxdvb code is the culprit (diseqc waiting - not sure why the wait times are too long for your config). The satip client has tuning in completely another thread, so the tuning should not block other things. Ideally, the goal should be to create a framework which makes the satip client code more abstract and reuse it in other backends. Another way (until the code is rewritten - absolutely no ETA from my side) is to install minisatip or any other SAT>IP server and try to pass the data through the network loopback to tvh.

#9 - 2019-02-24 20:15 - Jaroslav Kysela

BTW: Could you track where the diseqc code blocks? It seems that you really configured some big delays there. Or... the driver is just broken (slow ioctl).

#10 - 2019-02-25 20:49 - Deep Thought

I have been checking.

First of all: I did notice that the number of diseqc repeats on some tuners
is quite high (2 or 3). This is a reminant from an old configuration. The reason for it
is that tvheadend had (or has) problems tuning to some satellites with my configuration
(10 port uncommitted switch connected to tuner, 4 port commited switch connected to one of
the outputs of the first switch; this requires a diseqc repeat because tvheadend expects the
switches to be connected in a different order).

So this explains that some of the tune times are a bit larger. However this type of tuning,
or in fact any tuning, should not affect live tv (which uses a completely different dish
in my experiment, with a single committed switch).

I have now reduced diseqc repeats to 1, but the problems still occur. I still see continuity
errors.

The times for calls are now sometimes but not always shorter:
e.g, now I see linuxdvb_satconf_ele_tune_1079 TIME: 1168
Still very large. In six cases I see values larger than 500ms.

timer_thread still locks for 1.5 seconds sometimes, which is a sign of things going wrong.

When further instrumenting the code, I see no problems with any of the ioctls.
None of the diseqc commands take longer than 200ms.

#11 - 2019-02-25 21:17 - Deep Thought

I must correct one thing: I have nw found one ioctl which takes a long time
and thus also can create problems. However, this seems unrelated to the
problems I report above (as the error occurs at different times):

The following code in linuxdvb_frontend.c:

```
 if (ioctl(lfe->lfe_fe_fd, FE_READ_STATUS, &fe_status) == -1) {
    tvhwarn(LS_LINUXDVB, "%s - FE_READ_STATUS error %s", buf, strerror(errno));
    /* TODO: check error value */
    return;

  } else if (fe_status & FE_HAS_LOCK)
    status = SIGNAL_GOOD;
  else if (fe_status & (FE_HAS_SYNC | FE_HAS_VITERBI | FE_HAS_CARRIER))
    status = SIGNAL_BAD;
  else if (fe_status & FE_HAS_SIGNAL)
    status = SIGNAL_FAINT;
  else
    status = SIGNAL_NONE;
```

takes more than 1.6 - 3.1 seconds. 16 cases sofar.
This happens primarily when tuning to multistream
transponders on 5.0W. In rare cases, it also happens on 28.2E, but not very often.

In the results I reported above, this problem did not occur.
I will report this specific problem to the driver maintainers.

I repeat that it does NOT explain the results in this ticket!

#12 - 2019-02-26 08:06 - Jaroslav Kysela

Deep Thought wrote:

> So this explains that some of the tune times are a bit larger. However this type of tuning,
> or in fact any tuning, should not affect live tv (which uses a completely different dish
> in my experiment, with a single committed switch).

The really simple diseqc should be send within 100ms with no repeats.

15ms power on

diseqc
25ms wait
tune...

src/input/mpegts/linuxdvb/linuxdvb_switch.c - linuxdvb_switch_tune()

> When further instrumenting the code, I see no problems with any of the ioctls.
> None of the diseqc commands take longer than 200ms.

200ms is really big time. Note that there are also delays in the code to satisfy the communication requirements - look for tvh_safe_usleep() in src/input/mpegts/linuxdvb/linuxdvb_switch.c and src/input/mpegts/linuxdvb/linuxdvb_satconf.c .

#13 - 2019-02-26 18:59 - Deep Thought

Yes 200ms is big, but I only said that the calls take less than that, and therefore not
worth investigating further. I did not measure how long they take.

#14 - 2019-02-26 19:23 - Deep Thought

According to crazycat69 on https://github.com/tbsdtv/media_build/issues/14
it is indeed possible that FE_READ_STATUS ioctl blocks for up to 2 seconds.
This is expected behaviour.

This is therfore another case where the global_lock can be locked for up to 2 seconds,
blocking any other thread that needs it (e.g., the SI table processing).

This type of problem would be worse when tuning to weak transponders.

#15 - 2019-02-27 14:40 - Jaroslav Kysela

Deep Thought wrote:

> Yes 200ms is big, but I only said that the calls take less than that, and therefore not
> worth investigating further. I did not measure how long they take.

You should measure the whole linuxdvb_satconf_start_mux() execution.

> it is indeed possible that FE_READ_STATUS ioctl blocks for up to 2 seconds.

TVH opens the DVB FE file descriptor in the O_NONBLOCK mode, so the kernel code should not sleep. It's against the basic unix rule. The ioctl() must return ASAP.

#16 - 2019-02-27 20:04 - Deep Thought

About "the whole call". I measured it (see start of this report).

Here are some recent results for this call (with only 1 diseqc repeat on tuners
which need it)
linuxdvb_frontend_start_mux_790 is the place where the time taken by
linuxdvb_satconf_start_mux is reported. So this is the time you want to see
and it is the last number on each line.

At 14:04 an epg scan starts.
Feb 27 14:04:01 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 454
Feb 27 14:04:02 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 920
Feb 27 14:04:04 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1997
Feb 27 14:04:05 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1383
Feb 27 14:06:13 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 276
Feb 27 14:06:14 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 838
Feb 27 14:06:18 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1738
Feb 27 14:06:19 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 733
Feb 27 14:08:25 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 374
Feb 27 14:08:26 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 755
Feb 27 14:08:31 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1398
Feb 27 14:08:32 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 750
Feb 27 14:10:37 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 292
Feb 27 14:10:38 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1131
Feb 27 14:10:45 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1728
Feb 27 14:10:45 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 830
Feb 27 14:12:49 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 276
Feb 27 14:12:50 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 772
Feb 27 14:12:58 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1602

Feb 27 14:12:59 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 974
Feb 27 14:13:32 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 633
Feb 27 14:14:00 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 670
Feb 27 14:14:33 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 831
Feb 27 14:15:00 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 401
Feb 27 14:15:02 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 602
Feb 27 14:15:10 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1289
Feb 27 14:15:48 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 716
Feb 27 14:17:12 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 311
Feb 27 14:17:14 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 731
Feb 27 14:17:23 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1667
Feb 27 14:18:00 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 840
Feb 27 14:18:04 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 308
Feb 27 14:18:07 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 821
Feb 27 14:19:35 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1236
Feb 27 14:19:50 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 926
Feb 27 14:20:12 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 795
Feb 27 14:20:25 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 688
Feb 27 14:20:30 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 2908
Feb 27 14:21:27 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 602
Feb 27 14:22:02 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 743
Feb 27 14:22:42 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1299
Feb 27 14:23:39 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 696
Feb 27 14:24:13 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 664
Feb 27 14:24:54 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1281
Feb 27 14:25:51 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 717
Feb 27 14:26:25 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 622
Feb 27 14:27:07 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1531
Feb 27 14:28:02 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 659
Feb 27 14:28:15 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 605
Feb 27 14:28:24 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 599
Feb 27 14:29:23 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 5654

At 20:20 live viewing starts with a few channel changes:

Feb 27 20:22:59 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 473
Feb 27 20:23:38 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 297

At 20:31 I trigger an OTA epg scan via de webinterface:

Feb 27 20:31:30 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 764
Feb 27 20:31:33 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 2652
Feb 27 20:31:34 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1433
Feb 27 20:33:42 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 710
Feb 27 20:33:45 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1504
Feb 27 20:33:47 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 590
Feb 27 20:35:53 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 588
Feb 27 20:35:58 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1618
Feb 27 20:36:00 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 818
Feb 27 20:38:05 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 711
Feb 27 20:38:11 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1929
Feb 27 20:38:13 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 600
Feb 27 20:40:17 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 803
Feb 27 20:40:23 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1313
Feb 27 20:40:26 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 722
Feb 27 20:42:29 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 759
Feb 27 20:42:36 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 1788
Feb 27 20:42:39 streacom tvheadend[1270]: mpegts: linuxdvb_frontend_start_mux_790 TIME: 917

The problem really is that global_lock is taken for much too long.

Note that in these lasts results, I have already modified the code (as an experiment, not as a fix) to release the global_lock around the FE_READ_STATUS ioctl. This could affect the results.

According to the ioctl man page, there are no guarantees:

```
CONFORMING TO
       No single standard.  Arguments, returns, and semantics of ioctl() vary according to the device driver i
n ques-
       tion  (the  call is used as a catch-all for operations that don't cleanly fit the UNIX stream I/O model
).  See
       ioctl_list(2) for a list of many of the known ioctl() calls.  The ioctl() system call appeared  in  Ver
sion  7
       AT&T UNIX.
```

Also:
The fact that EAGAIN or EWOULDBLOCK is not listed as a possible error for ioctl
tends to confirm this. If ioctl would be able to function in non-blocking mode, then
the driver would need to be able to report EAGAIN or EWOULDBLOCK, but it cannot

I therefore believe that blocking/nonblocking does not apply to ioctl.
This is also agrees with my experience (years ago) with another receiver.
From my experience with working with older receivers, tuning always
takes several hundreds of milliseconds, especially with lnb switches or rotors involved.
ioctl calls are not instantaneous, but tuning involves many of them and also code to wait for
tuner locking ...
So as it is a slow process, it should be considered as such.

#17 - 2019-02-28 07:57 - Jaroslav Kysela

- Subject changed from global_lock problems to linuxdvb status / diseqc - global_lock problems with complex setup and/or slow drivers

- Target version set to 4.4

Okay, I don't want to follow further this, all was written, so the conclusion:

1) the used driver is a crap with the problematic timing of the status / diseqc ioctls
2) tvh helds the global_lock for the diseqc / status I/O - it should be moved outside global_lock (no ETA from my side)
3) the satip client might be temporary used to avoid such problems (use minisatip server with the threading config or other SAT>IP server locally)
4) buy another hardware which has better drivers

#18 - 2019-03-18 22:20 - Deep Thought

- File inuxdvb_frontend_monitor.txt added

The attached path releases the global_lock when linux_dvb_monitor is called
and reacquires it when needed only.

I have tested this patch for a few days and the result is that fewer cases of "slow channel changing" occur. It is ONLY a PARTIAL fix for this issue, but it helps.

#19 - 2021-02-20 22:32 - Flole Systems

- Target version changed from 4.4 to 4.6

Files

20180224.log	38.2 KB	2019-02-24	Deep Thought
inuxdvb_frontend_monitor.txt	1.75 KB	2019-03-18	Deep Thought