Simple cpu load test

This is a simple run of a cpu load test using two CAN channels on one CPC-PCI two channel passive CAN board.

One channel is used as a receiver, using the 'horch' software to record the CAN messages. It's not a very big application, but at least receives the messages pulls it from the driver and writes to the disk.

The other channel uses 'can_send' to produce some bus-load. We once wrote a little extension to create a defined bus-load, but this still does work perfectly. But after some approaches

```
$ ./can_send -l 50 -b 1000 -T 100000
send 100000 mesages, 83 messages  every 20 ms cycle
$
```

-l 50	produces not 50% (-l 50) but nearly 30% bus load.
-b 1000	at 1 Mbit/s.
-T 100000	generate this number of messages.

On a standard desktop linux we have a minimal time slice of 20 ms. That's why the program calculates how many messages it should write into the write buffer, before going to sleep until it is waken up again. The message is always with the same ID and 8 data bytes, approximately 113µs at 1MBit/s.

Both CAN channels are directly connected on the same PC. The PC is

```
vendor_id       : AuthenticAMD
cpu family      : 6
model           : 8
model name      : AMD Athlon(tm) XP 2400+
stepping        : 1
cpu MHz         : 1994.362
cache size      : 256 KB
bogomips        : 3971.48
```

Results

After finishing **can_send**, it should have sent 100000 Messages, which took about 36s, the logfile of the receiving process contains exactly 100000 CAN messages and 13 status messages containing CAN controller status information and the bus load, like:

```
:: sja1000 1000 12 96 0 0 170 30.4
:: sja1000 1000 28 96 0 0 170 30.5
:: sja1000 1000 12 96 0 0 170 30.5
:: sja1000 1000 12 96 0 0 170 30.1
:: sja1000 1000 28 96 0 0 170 30.4
:: sja1000 1000 28 96 0 0 170 30.4
:: sja1000 1000 12 96 0 0 170 30.4
:: sja1000 1000 28 96 0 0 170 30.4
:: sja1000 1000 28 96 0 0 170 30.4
:: sja1000 1000 12 96 0 0 170 30.4
:: sja1000 1000 28 96 0 0 170 30.4
:: sja1000 1000 28 96 0 0 170 30.4
:: sja1000 1000 12 96 0 0 170 25.8
            |   |          |    |
            |   +status    |    |
            |              |    + load
            + bit rate     + error code 170=0xaa
```

error_code=0x55 means something like a stuff error occured somewhere in the data field while receiving. Not serious.

Looking at the CPU load statistic without running the CAN test:

```
top - 19:47:30 up 13 days,  2:21, 19 users,  load average: 0.00, 0.03, 0.06
Tasks: 135 total,   3 running, 131 sleeping,   0 stopped,   1 zombie
Cpu(s):   0.3% user,   1.0% system,   0.0% nice,  98.7% idle
```

Because on the desktop are a lot of processes we see some noise and a basic load. running the test:

```
Cpu(s):   4.6% user,  15.8% system,   0.0% nice,  79.5% idle
Cpu(s):   5.6% user,  12.5% system,   0.0% nice,  81.9% idle
Cpu(s):   7.6% user,  11.2% system,   0.0% nice,  81.2% idle
Cpu(s):   5.0% user,   8.6% system,   0.0% nice,  86.5% idle
Cpu(s):   5.0% user,  10.6% system,   0.0% nice,  84.5% idle
Cpu(s):   2.6% user,  11.6% system,   0.0% nice,  85.8% idle
Cpu(s):   2.6% user,  11.6% system,   0.0% nice,  85.8% idle
Cpu(s):   3.3% user,  12.2% system,   0.0% nice,  84.5% idle
Cpu(s):   5.3% user,   9.2% system,   0.0% nice,  85.5% idle
Cpu(s):   5.0% user,  17.8% system,   0.0% nice,  77.2% idle
```

This gives slightly incorrect values, because the values were catched with the mouse, which increases both, the user and system load as well. Looking at the running process I can read (approximating with eyes and head):

3% user
6% system

Where **user** is the time spend in user processes, and **system** the time spend within system calls, that is in this case the CAN driver and also disk I/O. The result without disk I/0 is approximately 1% better. that means the idle process is 92% instead of 91%.

Subtracting the base load of 1% **we can estimate that the can driver needs 5% of the systems processing power for sending and receiving 2650 Messages/s .**

On Linux we can count the interrupts processed with:

```
$ cat /proc/interrupts
 ....
 12:    1668480          XT-PIC  usb-uhci, VIA8233, Can
 ....

$ time ./can_send -l 50 -b 1000 -T 100000

$ cat /proc/interrupts
 ....
 12:    1768480          XT-PIC  usb-uhci, VIA8233, Can
 ....
```

The difference between the two counts are exactly 100000. That means we have one interrupt for each CAN message. The driver, handling both channels, gets the **Transmit ready** and the **Message received** interrupt nearly at the same time and handles both interrupts with one call to the ISR. We also see that the system is fast enough, otherwise it would have handled more than one interrupt with one call to the ISR. Otherwise the interrupt counter should show a number smaller than 100000.

Busload calculation

To verify the 30% we got from horch, we can:

30 % from 1s is
 300.000 µs
divided by 113µs we need for one message
 300000/113 = 2654
results in 2654 messages / s
To transfer 100000 messages we need
 100000/2654 = 37 s