Virtueller CAN-Bus

Diese Übung dient als Einführung zur Kommunikation mit einem CAN-Bus-Interface. Lassen Sie sich dabei von dem Umfang der Aufgaben und den technischen Details nicht abschrecken, es wird vieles vorgegeben und erklärt.

Vorbereitungen: CAN-Interface

In diesem Teil der Aufgabe werden wir ein CAN-Interface starten, ein Beispiel kompilieren, das eine einzelne CAN-Nachricht an das Interface sendet und ein kleines Programm starten, welches an das Interface gesendete Nachrichten anzeigt.

Hinweis: Ein echtes CAN-Interface benötigt spezielle Hardware, die auf einem normalen Computer in der Regel nicht vorhanden ist. Damit Sie die Aufgaben auch von zu Hause aus bearbeiten können, verwenden wir ein virtuelles CAN-Interface, kurz **VCAN**.

VCAN initialisieren und überprüfen

Im Folgenden möchten wir einen VCAN initialisieren. Im Terminal (Konsole) führen wir folgenden Befehl aus, um zu überprüfen ob bereits ein VCAN existiert.

```
ifconfig vcan0
```

Hier sollte zunächst **ERROR** - „No Device found“ auf dem Terminal angezeigt werden.
Mit den untenstehenden Befehlen können wir nun den VCAN initialisieren.

```
sudo modprobe vcan
sudo ip link add dev vcan0 type vcan
sudo ip link set up vcan0
```

Die Befehle erzeugen im Normalfall keine Ausgaben. Führen wir nun

```
ifconfig vcan0
```

erneut aus, so sollte folgende Ausgabe angezeigt werden.

```
vcan0
Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-[..]
UP RUNNING NOARP MTU:16 Metric:1
RX packets:1083 errors:0 dropped:0 overruns:0 frame:0 TX packets:1083 errors:0 dropped:0
overruns:0 carrier:0 collisions:0 txqueuelen:0
RX bytes:8152 (8.1 KB) TX bytes:8152 (8.1 KB)
```

Wichtig ist, auf das “**UP RUNNING**” in der zweiten Zeile der Ausgabe zu achten. Der vcan ist nun korrekt initialisiert.

CanLogger

Importieren Sie das mitgelieferte Projekt **canlogger** als “Existing Projects into Workspace“. Tragen Sie in der **Run-Config** unter Arguments **-f=vcan0** ein und starten den CanLogger.

Alternativ rufen Sie im **Vorlagen-Workspace** im **canlogger** Projektverzeichnis das vorkompilierte ausführbare Programm direkt im Terminal aus. Navigieren Sie in das richtige Verzeichnis. Die Ausführbare Datei befindet sich i.d.r. im **Debug-** oder **bin-Ordner**. Starten Sie in diesem Verzeichnis mit Rechtsklick ein Terminal. Hier folgenden Befehl eintragen:

```
./canlogger -f=vcan0
```

Anschließend das Terminal geöffnet lassen.

Hinweis: Mit **STRG+C** kann ein laufendes Programm im Terminal gestoppt werden.

Aufgabe 1:

Importieren Sie das mitgelieferte Projekt **Blatt3_Aufgabe1** aus dem **Vorlagen-Workspace** als “Existing Projects into Workspace“. Kompilieren Sie das Projekt und führen Sie es aus. Werfen Sie nun erneut einen Blick auf das Terminal-Fenster mit dem geöffneten CanLogger-Programm, dort ist nun eine einzelne Nachricht zu sehen.

Aufgabe 2: Programmierübungen, Prog_2_CAN

Erzeugen Sie ein neues Projekt mit dem Namen **Blatt3_Aufgabe2** und kopieren Sie die Quelldateien *.**cpp** bzw. ***.h** aus **Blatt3_Aufgabe1** in das neue Projekt.

Wechseln Sie in das neu angelegte Verzeichnis und bearbeiten Sie in Eclipse das Projekt:

Schauen Sie sich das vorgegebene Beispiel in **main.c** genauer an. Hartgesottene können auch einen Blick auf die Datei **can.cpp** im Verzeichnis **lib** werfen. Experimentieren Sie mit der **can_frame**-Datenstruktur, indem Sie einige Werte ändern und sich die Ausgaben im **CanLogger** ansehen.

1. Senden Sie eine zweite CAN-Nachricht mit 8 Datenbytes.

2. Lassen Sie Ihr Programm eine Sekunde zwischen der ersten und zweiten CAN-Nachricht schlafen. Welche C-Befehle können dazu genutzt werden?

3. Geben Sie CAN-Nachrichten in einer Endlosschleife aus. Reduzieren Sie dabei die Wartezeit zwischen den Nachrichten auf 200 Millisekunden.

4. Verändern Sie bei jedem Schleifendurchlauf den Inhalt der CAN-Nachrichten.

5. Setzen Sie sich mit den Posix-Befehlen **open**(), **write**(), **read**(); **close**() im Detail auseinander. Welcher Wert wird bei einem **write**()-Fehler zurückgegeben? Welcher Wert bei Erfolg?

6. Geben Sie einen möglichen Fehlerfall aus.

7. Wie genau spielt keine Rolle, experimentieren Sie!

Aufgabe 3: Programmierübungen, CAN_2_Prog

Erzeugen Sie ein neues Projekt mit dem Namen **Blatt3_Aufgabe3** und kopieren Sie die Quelldateien *.**cpp** bzw. ***.h** aus **Blatt3_Aufgabe1** in das neue Projekt.

Wechseln Sie in das neu angelegte Verzeichnis und bearbeiten Sie in Eclipse das Projekt:

1. Verändern Sie das Programm so, dass Sie Frames vom CAN-Bus lesen und die Daten in ihrem Programm anzeigen können.

2. Studieren Sie dafür den Posix-**read**() Befehl.

Hinweis: Wenn Sie Ihr Programm Prog_2_CAN aus Aufgabe 2 im Hintergrund laufen lassen, dann sollten Sie Ihre CAN-Frames in Ihrer Ausgabe auf der Konsole sehen. Experimentieren Sie!

Aufgabe 4: Dateien lesen und schreiben

Als Einführung für die nächste Aufgabe werden wir die **can_frame**-Strukturen, die wir in den letzten Aufgaben noch an das CAN-Interface gesendet haben, in eine Datei schreiben und danach wieder auslesen.

Aufgabe 4.1: In Dateien schreiben, Prog_2_File

1. Erzeugen Sie ein neues Projekt mit dem Namen **Blatt3_Aufgabe4-1** und kopieren Sie die Quelldateien *.**cpp** bzw. ***.h** aus **Blatt3_Aufgabe1** in das neue Projekt.

2. Beschäftigen Sie sich mit dem Befehl **open**().

3. Erstellen Sie mit **open**() die Datei **./test.bin** mit Schreibrechten.

> **Hinweis**: Nutzen Sie Ihre Fehlererkennung aus der vorherigen Aufgabe! Bei Problemen können Sie so sehen, was genau schiefgelaufen ist.
> **Hinweis**: Verwenden Sie als drittes Argument von **open**() den Wert **0660**. Siehe Flags mode

4. Schreiben Sie eine einzelne CAN-Nachricht in die Datei. Was muss an dem **write**()-Befehl dafür geändert werden? Öffnen Sie die Datei mit dem Hex-Editor **Bless**. Dieser lässt sich über das Quickmenü von Linux finden.

5. Schauen Sie sich den Inhalt der Datei an. Wo befinden sich die Datenbytes aus Ihrem can_frame?

6. Schreiben Sie eine (oder mehrere) weitere Nachricht(en) mit jeweils verändertem Inhalt in die Datei. Wie sind die Nachrichten in der Datei angeordnet? Was passiert, wenn Sie ihr Programm mehrmals hintereinander ausführen?

Aufgabe 4.2: Aus Dateien lesen, File_2_Prog

1. Erzeugen Sie ein neues Projekt mit dem Namen **Blatt3_Aufgabe4-2** und kopieren Sie die Quelldateien *.**cpp** bzw. ***.h** und die eben erstelle **test.bin** aus **Blatt3_Aufgabe1** in das neue Projekt.

2. Öffnen Sie mit open() die Datei ./test.bin mit Leserechten.

> **Hinweis**: Verwenden Sie als drittes Argument von **open**() den Wert **0660**. Siehe Flags mode

3. Lesen Sie ein einzelnes **can_frame** mit dem Befehl **read**() aus. Schauen Sie sich den Inhalt des ausgelesenen can_frame mit dem Debugger an oder senden Sie die Nachricht an das CAN-Interface.

4. Führen Sie den **read**() mehrmals hintereinander aus. Vergleichen Sie die Reihenfolge der ausgelesenen Nachrichten mit Aufgabe 4.1.

5. Setzen Sie sich mit dem Befehl **read**() im Detail auseinander, insbesondere mit den Rückgabewerten. Woran kann man erkennen, dass das Ende einer Datei erreicht wurde?

6. Programmieren Sie eine Schleife, die so lange CAN-Nachrichten ausliest, bis das Ende der Datei erreicht wurde.

Aufgabe 5: Daten aus Log-Datei lesen, File_2_CAN, CAN_Player

Unter den in dieser Übung mitgelieferten Dateien findet sich auch eine Log-Datei, die eine Aufzeichnung der CAN-Nachrichten mit zusätzlichen Zeitinformationen in einem Fahrzeug beinhaltet. Dadurch lassen sich die verschiedenen Ereignisse wie z.B. die Drehzahl und die Geschwindigkeit des Fahrzeugs rekonstruieren. Ihre Aufgabe ist es, einen Parser zu schreiben, der diese Log-Datei ausliest und die CAN-Nachrichten zeitrichtig ausgibt.

1. Importieren Sie **Blatt3_Aufgabe5** als "Existing Projects into Workspace". Schauen Sie sich das vorgegebene Beispiel in **main.c** genauer an, speziell die Struktur **log_entry** ist hier von Interesse.

2. Implementieren Sie das Öffnen der LOG_FILE Log-Datei mit dem Befehl **open**().

3. Kompilieren Sie die Aufgabe, starten Sie Ihr Programm und den CanLogger und schauen sich das Ergebnis an. Von den unten dargestellten 6 CAN-Nachrichten sollte nun die erste Nachricht zu sehen sein.

```
0x00000199 6   0x08 0xfc 0x0d 0x7f 0x00 0x20
0x0000019a 6   0x3a 0xfc 0xcf 0x80 0x00 0x20
0x0000019f 6   0x66 0xfc 0xde 0x80 0x00 0x20
0x0000018e 3   0xff 0xff 0xff
0x000001e1 6   0x16 0xfb 0x00 0xff 0xff 0xff
0x000002a7 5   0x7a 0xfe 0xfe 0xff 0x14
```

4. Programmieren Sie eine Endlosschleife, die so lange Datensätze ausliest, bis das Ende der Datei erreicht wurde. Zur Selbstkontrolle die letzten 6 Datensätze:

```
0x000000a5 8   0x2c 0xfa 0x35 0x38 0x83 0x2c 0x0c 0xf1
0x0000008f 8   0xdd 0x12 0x7b 0x7e 0xa2 0x00 0x90 0x10
0x000000f3 8   0x0b 0xea 0x04 0xc0 0xf0 0xff 0xff 0xff
0x000000d9 8   0x8f 0x1a 0x00 0x10 0x00 0xe0 0x7f 0xf0
0x00000199 6   0x82 0xfd 0x26 0x7c 0x00 0x20
0x0000019a 6   0xce 0xfd 0xab 0x7f 0x00 0x20
```

5. Die originale Aufzeichnung der CAN-Nachrichten dauerte etwa 6-8 Minuten. Wie lange benötigt das Programm, um durchzulaufen? Passt der Zeitraum zur Aufzeichnung?

6. Setzen Sie sich mit der **timeval**-Datenstruktur innerhalb der **log_entry**-Datensätze auseinander. Welche Zeitinformationen können damit rekonstruiert werden?

7. Programmieren Sie eine Zeitverzögerung zwischen den Nachrichten. Nutzen Sie dazu die Informationen in der **timeval**-Datenstruktur.

Viel Spaß!