Technische Informatik IV: Praktikum

Protokoll zu Aufgabe A8

von Max Wisniewski, Alexander Steen

Vorbereitug

Zur Vorbereitung haben wir uns mit dem Kapitel *3.15 SMS Service* und dem Kapitel *3.17 Call Service* im *ADL_USER_GUIDE* beschäftigt. Dort lies sich alles finden, was wir brauchten, zusammen genommen mit dem Wissen der voran gegangenen Aufgaben.

Aufgaben

- Per Programm soll eine SMS empfangen werden, geben Sie Absender Uhrzeit und Inhalt auf der Konsole aus
- Erweitern Sie die Funktionalität in der Form, dass bei einer SMS mit dem Inhalt *call* der Absender zurückgerufen wird
- Diese SMS soll im Gegensatz zu den anderen Mitteilungen nicht gespeichert werden.

Dokumentation

adl_smsSubscribe Mit Hilfe dieses Kommandos kann man einen Eventlistener auf das Empfangen neuer SMS setzten. Die Syntax sieht dabei folgender Maßen aus:

```
s8      adl_smsSubscribe(adl_smsHdlr_f SmsHandler
                adl_smsCtrlHdlr_f SmsCtrlHandler
                u8 mode);
```

Als erstes geben wir das **struct** für unseren eigentlichen Handler an.

```
typedef bool    (* adl_smsHdlr_f) (ascii* SmsTel
                ascii* SmsTimeLength
                ascii* SmsText);
```

Dieser bekommt zunächst die Telefonnummer übergeben, danach (im Textmode, den wir verwendet haben) den Timestamp, wann die Nachricht ankam. Zuletzt wird der eigentliche Inhalt der SMS übergeben.

Über den Rückgabewert, lässt ich festlegen, ob die Nachricht danach weiter ans Modul gesendet werden soll. Gibt man *true* zurück, so wird das Event für eine SMS weiter geworfen und die SMS in den Speicher geladen. Gibt man *false* zurück, speichert man weder die Nachricht, noch wird das *+CMTI* Event geworfen.

Der zweite Handler bezeichnet einen ControlHandler für das SMS schreiben. Dieser reagiert auf Bestätigungsnachrichten oder Fehler die beim senden von SMS geworfen werden. Da wir uns in dieser Aufgabe nur für das Empfangen interessieren, gehen wir auf diesen Handler nicht weiter ein.

Damit wir hier keinen unnützen Handler eintragen, schreiben wir uns im Programm eine leere Funktion, da das übergeben von *NULL* in diese Fall einen Fehler wirft.

Als letzten kann man der smsSubscribe Funktion noch übergeben, ob diese auf den Textmode *ADL_SMS_MODE_TEX* oder auf den PDUmode (*ADL_SMS_MODE_PDU*) reagieren soll.

adl_callSetup Mit Hilfe dieser Funktion kann man einen Anruf tätigen.

```
s8      adl_callSetup(ascii*    phoneNumber
                u8 mode);
```

Man gibt zunächst die Rufnummer an, die man gerne Anrufen möchte. Als nächstes gibt man noch den Modus mit, den man gerne benutzen will. Dies ist einmal *ADL_CALL_MODE_VOICE* für normale Sprachübertragungen und *ADL_CALL_MODE_DATA* für Datenübertragungen.

Diese Funktion nimmt im Gegensatz zur Erweiterten Funktion als Port einfach *ADL_PORT_OPEN_AT_VIRTUAL_*

Durchführung

Zu Begin haben wir uns einen SMS Handler angelegt.

```
1  void main_task(void)
2  {
3          s8 ret = adl_smsSubscribe(smshandler, smscontrolhandler, ADL_SMS_MODE_TEXT);
4          if (ret == ADL_RET_ERR_PARAM)
5                  {
6                  adl_atSendResponse(ADL_AT_RSP, "\r\nFehler beim registrieren:
7                          parameter falsch\r\n");
8                  }
9          if (ret >= 0)
10                 {
11                 adl_atSendResponse(ADL_AT_RSP, "\r\nregistrierung erfolgreich\r\n");
12                 }
13         }
```

Wir tragen mit der oben beschriebenen Methode *adl_smsSubsribe* einen neuen SMS Handler ein und überprüfen dann den Rückgabewert der Funktion. Haben wir die Konstante *ADL_RET_ERR_PARAM* zurück bekommen, so haben wir in die Methode einen falschen Paramter hereingesteckt.

Ist der Rückgabewert größer oder gleich Null, so hat die Registierung funktioniert. Darüber geben wir zu Protokollzwecken noch auskunft.

Da der *smsSubscribe* Befehl an der Stelle für den controlhandler keine *NULL* Funktion akzeptiert, müssen wir eine leere Funktion schreiben und hinein stecken.

```
1  void smscontrolhandler(u8 event, u16 nb)
2          {
3          return;
4          }
```

Diese Funktion implementiert den nötigen Aufruf, tut aber sonst nichts.

Als nächstes haben wir uns dem eigentlichen Handler zugewandt. Als erstes haben wir uns darum gekümmert, dass wir bei einem Anruf die später benötigten Informationen auslesen.

```
1  bool smshandler(ascii * smstel, ascii * smstime, ascii * smstext)
2          {
3          adl_atSendResponse(ADL_AT_RSP, "\r\nAbsender:\r\n");
4          adl_atSendResponse(ADL_AT_RSP, smstel);
5          adl_atSendResponse(ADL_AT_RSP, "\r\nUhrzeit:\r\n");
6          adl_atSendResponse(ADL_AT_RSP, smstime);
7          adl_atSendResponse(ADL_AT_RSP, "\r\nInhalt:\r\n");
8          adl_atSendResponse(ADL_AT_RSP, smstext);
9          if (wm_strcmp("call", smstext) == 0)
10                 {
11                 return false;
12                 }
13         else
14                 {
15                 return true;
16                 }
17         }
```

Zu beachten schon an der ersten Version, wenn wir die Nachricht call bekommen, geben wir *false* zurück. Bei allen anderen Nachrichten *true*. Dies hat, wie sich später zeigt, den Erfolg, dass wir das ankommen der Nachricht nicht mehr mitgeteilt bekommen.

In der nächsten Version haben wir nun angefangen, wenn wir eine Nachricht mit dem Inhalt *call* erhalten, den Absender zurück zu rufen.

```
bool smshandler(ascii * smstel, ascii * smstime, ascii * smstext)
        {
        adl_atSendResponse(ADL_AT_RSP, "\r\nAbsender:\r\n");
        adl_atSendResponse(ADL_AT_RSP, smstel);
        adl_atSendResponse(ADL_AT_RSP, "\r\nUhrzeit:\r\n");
        adl_atSendResponse(ADL_AT_RSP, smstime);
        adl_atSendResponse(ADL_AT_RSP, "\r\nInhalt:\r\n");
        adl_atSendResponse(ADL_AT_RSP, smstext);
        if (wm_strcmp("call", smstext) == 0)
                {
                if (adl_callSetup(smstel, ADL_CALL_MODE_VOICE) != OK)
                        {
                        adl_atSendResponse(ADL_AT_RSP, "\r\nERROR beim anrufen\r\n");
                        }
                else
                        {
                        adl_atSendResponse(ADL_AT_RSP,"\r\nNun sollte angerufen werden\r\n");
                        }
                return false;
                }
        else
                {
                return true;
                }
        }
```

Haben wir die richige Nachricht erhalten, so benutzen wir *callSetup* um an die *smstel*, die vorher ermittelte Nummer, einen Anrufen abzusetzen. Bekommen wir *OK* zurück geben wir noch auf der Konsole aus, dass wir eben dieses tun. Sollte der Anruf aus unbekannten Gründen nicht funktionieren, geben wir einen entsprechenden Fehler zurück.

Auswertung

Betrachten wir erste einmal unser erstes Programm. Nachdem die bekannten Startnachrichten durch waren, haben wir uns zwei *SMS* schreiben lassen. Eine mit dem Inhalt *Bla* und eine andere mit dem Inhalte *call*.

```
Absender:
+49176--------
Uhrzeit:
11/04/27,12:29:08+08
Inhalt:
Bla
+CMTI: "SM",3
```

In der ersten Nachricht sehen wir die Absenderaddresse (hier auskommentiert), die Uhrzeit und den Inhalt. Wie oben schon angekündigt, wird diese Nachricht mit einem *+CMTI* quitiert, das uns anzeigt, dass die SMS normal empfangen wurde.

```
Absender:
+49176-------
Uhrzeit:
11/04/27,12:29:48+08
Inhalt:
call
```

Die zweite Nachricht, mit dem behandelten Inhalt *call*, wird uns vom Modul keine solche Nachricht zurück gegeben.

Mit dem erweiterten Programm erhalten wir die folgende Ausgabe

```
Absender:
+4917696704595
Uhrzeit:
11/04/27,12:54:39+08
Inhalt:
call
Nun sollte angerufen werden
```

Die Nachrichten legen nahe, dass nun der Absender angerufen werden sollte. Die letze Nachricht sollte auch nur ausgegeben werden, wenn der Anruf klappt. Allerdings haben wir auf dem Handy keinen Anruf erhalten. Dies ist merkwürdig, da ein explizites Anrufen (an dieser Stelle nicht gezeigt, aber am Anfang eingebaut) funktioniert.

Wir sahen aber kein Fehler in unserem Code sondern tippten an dieser Stelle auf ein Fehler im Modul.