Technische Informatik IV: Praktikum

Protokoll zu Aufgabe A16

Alexander Steen , Max Wisniewski

Vorbereitung

Als Vorbereitung haben wir uns mit dem NMEA Datenformat und der Datei *fsa03.h* bzw. *fsa03.c* vertraut gemacht.

Aufgaben

1. Programm schreiben, dass die aktuelle Position (via GPS ermittelt) alle 10 Sekunden an einen Server schickt
2. Erweitern, sodass man via telnet und Eingaben von w,s,a,d (den Tasten) den Modulbetreiber "steuern" kann.

Dokumentation

GPS GPS wurde bereits im Vorprotokoll zu Aufgabenblock 3 beschrieben. Für diese Aufgabe ist die genaue Antwort des GPS-Moduls wichtig. Eine GPS-Antwort sieht folgendermaßen aus:
`$GPRMC,191410,A,4735.5634,N,00739.3538,E,0.0,0.0,181102,0.4,E,A*19`
Dabei bedeuten die Teile der Antwort:
`$GPRMC` - Einleitung der Antwort
`191410` - Uhrzeit (hier: 19:14:10)
`A` - Status / Gültigkeit der Antwort, dabei ist `A` gültig und `V` ungültig
`4735.5634` - Breitengrad
N - Richtung, hier Nord
`00739.3538` - Längengrad
`E` - Richtung, hier Ost
und einige weitere Angaben, die wir hier nicht betrachten.

fsa03_subscribe Registriert eine GPS-Handler-Funktion bei dem GPS-Modul, dabei muss folgender Prototyp eingehalten werden:
`void gpshandler(fsa03_gprmc * data)`
In dem struct `data` stehen dann bereits alle vorverarbeiteten Information der GPS-Antwort.

Durchführung

Zu Beginn legen wir uns globale Variablen an: Die Variable `channel` speichert den zu Beginn des Programmes geöffneten TCP-Kanal zum bekannten Server. Dieser Kanal wird dann von dem GPS-Handler genutzt. Die Zählvariable `counter` zählt die Aufrufe des GPS-Handlers. Dies ist wichtig, da der GPS-Handler zwar jede Sekunde automatisch aufgerufen wird, jedoch nur alle zehn Sekunden Daten schicken soll. In `buffer` speichern wir uns die Nachricht, die an den Server geschrieben werden soll.

```
wip_channel_t channel;
u8 counter = 0;
ascii buffer[64];
```

Die Callback-Funktion `gpshandler` wird nach Registration (siehe main_task) automatisch jede Sekunde anfgerufen und prüft, ob bereits 10 Sekunden vergangen sind (Anzahl der Aufrufe restlos durch zehn teilbar). Falls dies der Fall ist, wird eine Nachricht mit der aktuellen Position an den Server geschickt. Vorher wird

allerdings geprüft, ob die aktuelle GPS-Information schon vollständig (gültig) ist. Dies können wir realisieren, in dem wir auf den Nachrichtenteil der GPS-Antwort überprüfen, in dem ein 'V' steht, falls die Antwort nicht gültig ist.

Da die GPS-Antwort nicht kompatibel mit der Schnittstelle des Server ist (die GPS-Antwort hat jeweils eine Nachkommastelle mehr bei den Positionsangaben), schneiden wir einfach die letzte Nachkommastelle der Antwort ab. Die Genauigkeit wird davon nur maginal beeinflusst.

```
void gpshandler(fsa03_gprmc * data)     {
    counter = (counter + 1) % 10;
    if (counter != 1)       return;

    // Erzwinge Schnittstellen-Kompatibilität:
    // (Jeweils eine Nachkommastelle löschen)
    ascii latitude[10];
    wm_strncpy(latitude, data->latitude, 9);
    latitude[9] = '\0';
    ascii longitude[11];
    wm_strncpy(latitude, data->latitude, 10);
    latitude[10] = '\0';

    // Baue  Ausgabe (für die Nachricht an den Server) zusammen:
    wm_sprintf(buffer, "%s%s,%s%s;", latitude, data->lat_dir, longitude, data->long_dir);
    // Kontrollausgabe:
    adl_atSendResponse(ADL_AT_RSP, "Daten: \r\n");
    adl_atSendResponse(ADL_AT_RSP, buffer);
    adl_atSendResponse(ADL_AT_RSP, "\r\n");

    // Nur schicken, falls Daten komplett,
    // also "status"-Wert nicht void
    if (wm_strcmp(data->status, "V") == 0)
        return;

    // Existiert der channel noch?
    // Falls nein, neu erstellen und in nächster Runde schicken
    if (channel) {
        adl_atSendResponse(ADL_AT_RSP, "Schicken Daten auf TCP: \r\n");
        wip_write(channel, buffer, 25);
    } else {
        channel = wip_TCPClientCreateOpts("hwp.mi.fu-berlin.de",
                50008,
                tcphandler,
                NULL, WIP_COPT_PORT, 13337, WIP_COPT_END);
    }
}
```

In der main-Funktion wird der TCP-Channel geöffnet und GPS aktiviert. Nachdem dies geschehen ist, registrieren wir unseren gpshandler.

```
void main_task(void) {
    // Erstelle TCP-Stream zum Server
    channel = wip_TCPClientCreateOpts("hwp.mi.fu-berlin.de",
                50008,
                tcphandler,
                NULL, WIP_COPT_PORT, 13337, WIP_COPT_END);
    // GPS aktivieren und Handler registrieren
    fsa03_enable(TRUE);
    fsa03_subscribe(gpshandler);
}
```

Um die Steuerung durch die Telnet-Konsole zu realisieren, reagieren wir einfach bei jeder eingehenden Nachricht, wie es in der Aufgabenstellung verlangt ist. Wir lesen zunächst ein Zeichen aus, überprüfen ob es entweder w,s,a oder d ist und schalten dann die jeweiligen LEDs an bzw. aus. Wird keine gültige Eingabe gelesen (ein anderes Zeichen), so schalten wir alle LEDs aus.

```
void tcphandler(wip_event_t *event, void *ctx) {
    ascii read_buffer[2];
    switch (event->kind) {
        case WIP_CEV_OPEN:
            break;
        case WIP_CEV_READ:
            // Nachricht bekommen: Analysieren
            wip_read(event->channel, read_buffer, 2);
            // Falls es s,w,a oder d entspricht, entsprechende
            // Lampen anschalten, sonst alles aus.
            if (wm_strcmp(read_buffer, "s") == 0)               {
                led_on(0); led_off(1); led_off(2);led_off(3);
            } else if (wm_strcmp(read_buffer, "a") == 0) {
                led_on(1); led_off(2); led_off(3); led_off(0);
            } else if (wm_strcmp(read_buffer, "d") == 0) {
                led_on(2); led_off(3); led_off(0); led_off(1);
            } else if (wm_strcmp(read_buffer, "w") == 0) {
                led_on(3); led_off(0); led_off(1);  led_off(2);
            } else {
                led_off(3); led_off(0); led_off(1); led_off(2);
            }
            break;
        case WIP_CEV_WRITE:
            break;
        case WIP_CEV_PEER_CLOSE:
            // Falls von der Gegenseite geschlossen wird,
            // neu öffnen.
            channel = wip_TCPClientCreateOpts("hwp.mi.fu-berlin.de",
                50008,
                tcphandler,
                NULL, WIP_COPT_PORT, 13337, WIP_COPT_END);
            break;
        case WIP_CEV_ERROR:
            wip_close(channel);
            channel = NULL;
            break;
        default:
            wip_close(channel);
            channel = NULL;
            ERROR("ERROR:␣Unkown␣TCP␣Event");
            break;
    }
}
```

Testlauf:

Um unser Programm zu testen, mussten wir einige Minuten draußen auf der Wiese warten, bis wir Empfang zu genügend GPS-Satelliten hatten. Danach funktionierte die Ortung sehr gut und die Nachricht wurde an den Server geschickt. Hier ein Screenshot der Ortung: