Technische Informatik IV: Praktikum

Protokoll zu Aufgabe A12

Max Wisniewski, Alexander Steen

Vorbereitung

Zur Vorbereitung haben wir den Abschnitt Socket Layer des WIP Development Guide gelesen und die Datei tcp.c aus dem Rahmenwerk studiert. Um die GPRS-Funktionalitäten nutzen zu können, wurde in der config.h INIT_GPRS aktiviert und dann ein clean build des Projekts gemacht.

Aufgaben

1. Befehl AT+ECHO erstellen
2. Erweitern, sodass Tastenbetätigungen via TCP verschickt werden

Dokumentation

wip_TCPClientCreateOpts

wip_read

wip_write

Durchführung und Auswertung

Um den TCP-Kanal, den wir bei einem ECHO-Kommando öffnen werden, global ansprechen zu können, speichern wir diesen global in der Variable channel. Da wir nicht für jedes ECHO-Kommando einen neuen Socket erstellen wollen, merken wir uns außerdem über die globale boolesche Variable channel_open, ob bereits ein Socket erstellt wurde. So können wir den bereits erzeugten Socket benutzen. Außerdem arbeiten wir mit einem globalen Sendepuffer, da wir eventuell nicht direkt im Kommando-Handler von ECHO senden können (genaueres später). Die Größe des Puffers haben wir eher willkürlich gewählt.

```
wip_channel_t channel;
ascii sendbuffer[256];
bool channel_open;
```

Wird die Applikation gestartet, so wird das Befehl AT+ECHO angelegt und mit dem Callback echoHandler vverknüpft. Das Kommando wird Parameter akzeptieren - nämlich genau eins: Den String, der vom Host zurückgeschickt werden soll.

```
void main_task(void) {
    channel_open = false;
    // Kommando anlegen
    adl_atCmdSubscribe("AT+ECHO", echoHandler,
                       ADL_CMD_TYPE_PARA | 0x0011);
}
```

Der Handler für das ECHO-Kommando holt sich nun den Parameter (also den zu sendenden String). Ist schon ein TCP-Socket geöffnet, so können wir einfach den Befehl wip_write nutzen, um den String in den Socket zu schreiben. Ist das nicht der Fall, so muss ein Socket geöffnet werden. Dies setzen wir mit dem Befehl wip_TCPClientCreateOpts (siehe Dokumentation oben) um, und setzen die Parameter entsprechend

unseres Hostes und Ports. Dem Socket übergeben wir einen Eventhandler echo_response, der auf TCP-Events reagiert. Da wir nach dem Aufruf von wip_TCPClientCreateOpts noch nicht sicher sein können, dass der Kanal bereits geöffnet ist, speichern wir am Anfang der Funktion den String in den globalen Buffer. Nun kann von dem TCP-Eventhandler der Sendepuffer verschickt werden, sobald der Socket offen ist.

```
void echoHandler(adl_atCmdPreParser_t *param) {
    if (param->Type == ADL_CMD_TYPE_PARA) {
        wm_strGetParameterString(sendbuffer, param->StrData, 1);
        // Socket erstellen
        if(!channel_open){
            channel = wip_TCPClientCreateOpts (
                        "hwp.mi.fu-berlin.de",  // Host
                        50008,                  // Port
                        echo_response,          // Handler
                        NULL,
                        WIP_COPT_PORT, 13338,
                        WIP_COPT_END);
        }else{
            // Bereits erstellt, direkt senden
            wip_write(channel, sendbuffer, 256);
        }
        adl_atSendResponse(ADL_AT_RSP, "\r\nOK\r\n");
    }
}
```

Bei dem TCP-Eventhandler haben wir uns an dem Beispielcode orientiert und den Handler als großen switch-Block entworfen, in dem wir unterscheiden, was für ein Event gerade passiert ist. Wurde der Kanal gerade geöffnet, so muss noch die im globalen Puffer wartende Nachricht verschickt werden. Bekommen wir eine Nachricht, so speichern wir diese Zeichen für Zeichen in einem lokalen Puffer und geben diese dann auf der Konsole aus. Sollte ein Fehler auftreten oder der Kanal von der Gegenseite geschlossen werden, schließen wir den Socket.

```
void echo_response(wip_event_t *event, void *ctx){
    s32 answer;
    ascii buffer[256];

    switch(event->kind){
        case WIP_CEV_OPEN:
            // Socket frisch geöffnet, wartende Nachricht schicken
            channel_open = true;
            wip_write(channel, sendbuffer, 256);
            break;
        case WIP_CEV_READ:
            // Wir erhalten eine Nachricht, ausgeben:
            do {
                answer = wip_read (event->channel, buffer, sizeof (buffer) - 1);
                buffer[answer] = '\0';
                // Terminieren, falls es ein String ist
                adl_atSendResponse(ADL_AT_RSP,buffer);
            } while (answer == sizeof (buffer) - 1);
            break;
        case WIP_CEV_WRITE:
            break;
        case WIP_CEV_PEER_CLOSE:
        case WIP_CEV_ERROR:
            wip_close(channel);
            channel_open = false;
            break;
        default:
            wip_close(channel);
            channel_open = false;
```

```
            ERROR("ERROR: Unkown TCP Event");
            break;
    }
}
```

Damit ist die Grundfunktionalität hergestellt und AT+ECHO verschickt einen String, der wieder ausgegeben wird, sobald er wieder empfangen wird.

Nun modifizieren wir das Programm noch so, dass bei einem Tastendruck eine entsprechende Nachricht via Socket geschickt wird. Dafür modifizieren wir die main so, dass Key-Nachrichten aktiviert werden und ein Handler auf die Nachrichten reagiert.

```
void main_task(void) {
    channel_open = false;
    // Kommando anlegen
    adl_atCmdSubscribe("AT+ECHO", echoHandler,
                    ADL_CMD_TYPE_PARA | 0x0011);
    // Key-Nachrichten aktivieren
    adl_atCmdCreate("AT+CMER=,1", FALSE, (adl_atRspHandler_t) NULL, NULL);
    // Keyhandler anlegen
    adl_atUnSoSubscribe("+CKEV:", (adl_atUnSoHandler_t) keyhandler);
}
```

Dieser Keyhandler liest einfach aus, welche Taste gedrückt wurde und schickt dann einen String der Form "Taste <Nr>" via Socket. Das Vorgehen ist dabei analog zu echoHandler.

```
bool keyhandler(adl_atUnsolicited_t *paras) {
    ascii param1[1];
    // parameter auslesen
    wm_strGetParameterString(param1, paras->StrData, 1);
    // String zusammenbauen
    wm_sprintf(sendbuffer, "Taste %s",param1);
    if(!channel_open){
            //Socket erstellen
            channel = wip_TCPClientCreateOpts (
                        "hwp.mi.fu-berlin.de",  // Host
                        50008,                  // Port
                        echo_response,          // Handler
                        NULL,
                        WIP_COPT_PORT, 13338,
                        WIP_COPT_END);
    }else{
        // Bereits erstellen, direkt senden
        wip_write(channel, sendbuffer, 256);
    }
    return FALSE;
}
```