# Technische Informatik IV: Praktikum

Protokoll zu Aufgabe A15

Alexander Steen , Max Wisniewski

## Vorbereitung

Zur Vorbereitung haben wir uns das Kapitel *FTP* im *WIP Development Guide* durchgelesen. Dabei war schon festzustellen, dass der grobe Aufbau mit dem des E-Mail Programmes aus Aufgabe A14 recht ähnlich. Wenn man das Framework aus *ftp.c* benutzt unterschied es sich größtenteils in der Erstellung der Verbindung.

## Aufgaben

- Verbinden Sie sich mit dem FTP - Server **wba-dev.de** und laden Sie eine Datei **messgageoftheday.txt** herunter. Geben Sie den Inhalt auf der Konsole aus.
- Laden Sie ihre eigene Nachricht auf den Server. Damit Sie die anderen Gruppen nicht stören, legen Sie die Nachricht in der Datei **mymassage.txt** im Ordnermit dem Namen ihres Arbeitsplatzes (**hwp8**) ab.
- Folgende Einstellungen sind für den FTP-Server zu verwenden: [Aus Sicherheitsgründen werden LogIn Daten nicht mit rein geschrieben]
- Kontrollieren Sie die Ergebnisse

## Dokumentation

**wip_FTPCreateOpts** Mit diesem Befehl wird eine neue Verbindung zu einem FTP-Server geöffnet. Die Syntax dabei ist:
**wip_channel_t wip_FTPCreateOpts(ascii* server_name, wip_eventHandler_f evh, void ctx*, ...)**. Der Rückgabewert ist ein Channel, vom selben Typ, wie wir ihn schon in Aufgabe *A14* beschrieben haben. Als Eingabe gibt man dem Befehl erstens den Namen [Addresse] des Servers zu dem man sich Verbinden möchte. Danach kommt, wie gewohnt der Eventhandler, der auf alle Events (es sind wieder die selben wie in Aufgabe *A14* beschrieben) reagiert. Der ctx Handler verwaltet wieder die Userdaten, die in den Eventhandler gegeben werden. Darüber hinaus, bekommt die optionale Variante

des Befehls noch eine Liste von Parametern, wie immer wird die Liste mit *null* beendet:

- WIP_COPT_TYPE ist ein ascii (einzelner Char), der anzeigt, welches Format die Nachricht hat ('I' image, 'A' ascii, 'E' EBCDIC)
- WIP_COPT_PASSIV ist ein boolean der festlegt, ob die Verbindung passiv, oder aktiv sein soll. In der aktiven Variante wir versucht eine synchronisation mit dem Server aufzubauen. Die passive Variante arbeitet nur, wenn sie den Befehl dazu bekommt. Diese Option bietet sich bei dem begrenzten Volumen, das uns zur Verfügung steht an.
- WIP_COPT_USER ist ein ascii* (String) der den Username enthält. Gibt man keinen Name an, geht man als "anonymous" an de Server heran
- WIP_COPT_PASSWORD ist ein ascii* der das Passwort enthält. Das standardpasswort ist eine E-Mail addresse von Wavecom.
- WIP_COPT_ACCOUNT ist ein ascii* der den Account kennzeichnet über dessen Zugang man an den Server herran tritt. Ohne Angabe ist der String leer und man fragt den Standardzugang des Servers an.

- WIP_COPT_PEER_PORT ist ein unsigned Short (u16) der den Port des Servers speichert. Ohne Eingabe ist es 21 (der Standard FTP Port)
- WIP_COPT_LIST_PLUGIN nimmt ein wip_eventHandler_f. Dieser reagiert auf Events der Funktion LIST. Diese haben wir nicht benutzt, wird hier also nicht näher beschrieben.
- WIP_COPT_KEEPALIVE nimmt ein unsigned Integer n entgegen. Die Option sorgt dafür, dass all $n/10$ Sekunden eine Nachricht an den Server geschickt wird. damit dieser und jeder Knoten auf dem Weg, die Verbindung nicht beendet.
- WIP_COPT_FINALIZER nimmt einen speziellen EventHandler wip_finalizer_f, der sich nach schließen des Channels um Aufraumarbeiten kümmern kann oder z.B. den Channel erneut starten kann. (Bei Verbindungstimeouts oder anderen Fehlern)

**wip_putFile** versucht in eine Datei auf dem FTP-Server zu schreiben. Die Syntax ist dabei **wip_channel_t wip_putFile(wip_channel_t ftp_cx, ascii *filename, wip_eventHandler_f evh, void *ctx)**. Der erste Parameter ist der Channel zum FTP-Server auf den man schreiben möchte. Als zweites gibt man den Namen (auch Pfad zu der Datei) an, in die man schreiben möcht. Die letzen beiden Parameter sind wiederum bekannt.
Der Eventhandler kümmert sich diesmal darum, das die Operationen auf dem neuen Channel (der Rückgabewert der Funktion) korrekt ausgeführt werden. In unserem Fall also, das die Datei bei Erstellung geschrieben wird.

**wip_getFile** die Signatur und das Verhalten ist exakt gleich mit dem von **wip_putFile**. Allerdings ist die Intention dieses mal, dass man aus einer Datei lesen kann. (Bei einem Versehen bemerkten wir, das wir auch durch einen mit putFile erstelltem Channel lesen und mit getFile schreiben konnten).
Der Unterschied zu putFile ist bei unserer Benutzung nur gewesen, dass wir auf die Antwort vom Server warten mussten und danach das Ergebniss (den Inhalt der Datei) ausgeben konnten.

## Durchführung

Die Aufgabe stellte einen vor keine großen Probleme, da der größte Teil genau das selbe war, wie in Aufgabe *A14*. Bereiten wir zunächst alles vor.

**Algorithm 1** Öffnen eines Kanals zum FTP-Server und aktivieren der Taster.

```
1  //____VARIABLEN_____________________________________________________________________________
2  wip_channel_t sessionchannel; // Sitzung
3  ascii * nachricht =     "Das ist ein toller Test, wir können via FTP schreiben.
4          \r\nSchönen Tag noch\r\n";
5
6  void ftp_handler(wip_event_t *event, void *context)
7      {
8      if (event->kind == WIP_CEV_ERROR) /* Fehler  */
9          {
10         ERROR("Verbindung zum Mail-Server abgebroche, Neustart erforderlich\r\n");
11         wip_close (sessionchannel);
12         return;
13         }
14     }
15
16 void main_task(void)
17     {
18     // keymeldungen aktivieren
19     adl_atCmdCreate("AT+CMER=,1", FALSE, (adl_atRspHandler_t) NULL, NULL);
20     // keyhandler anlegen
21     adl_atUnSoSubscribe("+CKEV: 0,1", (adl_atUnSoHandler_t) gethandler);
22     adl_atUnSoSubscribe("+CKEV: 1,1", (adl_atUnSoHandler_t) puthandler);
23
24     sessionchannel = wip_FTPCreateOpts( // startet die Sitzung
25         "wba-dev.de", // URL des Server
26         ftp_handler, NULL, WIP_COPT_USER, "praktikum", // Benutzer
27         WIP_COPT_PASSWORD, "praktikum", // Passwort
28         WIP_COPT_PASSIVE, FALSE, // passiver Modus
29         WIP_COPT_TYPE, 'A', // Codierung
30         WIP_COPT_END);
31     }
```

Beim Start des Moduls aktivieren wir zunächst die Tastendrücke. Danach legen wir 2 Handler an. Der erste reagiert nur auf das Herunterdrücken der Taste 0 und der andere nur auf das Herrunterdrücken der Taste 1. Beide bekommen einen Handler, den wir gleich erläutern wollen. Auf Taste 0 wird man eine Nachricht vom Server holen und auf Taste 1 eine auf den Server legen.

Danach legen wir noch unseren Kanal zum FTP-Server an. Dieser wird global als *sessionchannel* gespeichert. Damit wir nachher beim Tastendruck darauf zugreifen können. Der Handler dieses Channels macht nichts, außer ihn bei einem Fehler auf dem Kanal zu schließen und eine Fehlermeldung auszugeben (wahlweise hätte man den Kanel erneut öffnen können)

Beschreiben wir zuerst das Lesen vom Server:

**Algorithm 2** Wir legen hier zunächst einen Eventhandler an und reagieren dann auf einen Knopfdruck 0 um zu lesen.

```
1  void ftp_recv_handler(wip_event_t *event, void *context)
2      {
3      s32 length;
4      ascii readbuffer[1024];
5      switch (event->kind)
6          {
7          case WIP_CEV_OPEN:
8              //Lesen
9              break;
10         case WIP_CEV_READ:
11             length = wip_read(event->channel, readbuffer, 1024 - 1);
12             readbuffer[length] = '\0';
13             adl_atSendResponse(ADL_AT_RSP, readbuffer);
14             wip_close (event->channel); // schließt den Übertragung
15             break;
16         default: /* Fehler oder fertig */
17             wip_close (event->channel); // schließt den Übertragung
18             break;
19         }
20     }
21
22 bool gethandler(adl_atUnsolicited_t *paras)
23     {
24     adl_atSendResponse(ADL_AT_RSP, "Taste 0 wurde gedrückt\r\n");
25     wip_getFile(sessionchannel, "messageoftheday.txt", ftp_recv_handler, NULL);
26
27     return FALSE;
28     }
```

Wenn die Taste 0 gedrückt wurde, geben wir eine Debugnachricht aus und öffnen dann einen neuen Kanal zum Lesen. Diesen brauchen wir aber nicht zu speichern, da wir ihn ohnehin nicht wieder verwenden wollen. Damit die Nachricht nicht weiter durch das System geschickt wird, halten wir das weiterleiten mit dem Rückgabewert *false* auf.

Der Eventhandler des Lesekanals springt nun an, wenn der Channel eine Nachricht bekommt. (Bei get wurde die Datei automatisch vom Server angefragt.) Erreicht uns nun die Antwort, geben wir diese auf der Konsole aus. Danach schließen wir den Channel. Absehen vom erstellen, schließen wir den Kanal dann in jedem anderen Fall. Wir schreiben nicht also sollte dieser Fall nicht eintreten und sonst gibt es nur noch Fehler die uns nicht interessieren. Zuletzt schauen wir uns das Schreiben auf den Server an:

Wir geben hier, wie oben erst eine Debugnachricht aus, wenn die Taste gedrückt wurde. Danach wollen wir auf die Datei *mymessage.txt* im Ordner *hwp8* schreiben.

Der Eventhandler tut seine arbeit diesmal beim Öffnen des Kanals, damit wir den Kanal nicht speichern oder extra beschreiben müssen.

Ist der Kanal also erzeugt, nehmen wir uns die oben beschriebene globale Nachricht und schreiben sie in den Kanal. Wir geben noch eine Debugnachricht aus und schließen danach den Channel. Sonst (außer beim schreiben direkt) schließen wir den Channel wieder. Man sieht, dass das Programm wirklich nicht kompliziert war, da ein der größte Teil der arbeit einfach schon abgenommen wird und man sich nicht mit den konkreten Protokoll herum ärgern muss.

## Auswertung

Nach einigen Minuten warten wenn das GPRS endlich angesprungen ist, kann man beim Drücken der Taste 0 folgende Nachricht sehen:

```
+GPRS: Verbindung wurde hergestellt

Taste 0 wurde gedrückt

+CKEV: 0,0

hallo heute ist ein schöner tag
```

**Algorithm 3** Wir legen hier einen Eventhandler an und reagieren dann auf den Knopfdruck 1 um zu schreiben

```
1  void ftp_send_handler(wip_event_t *event, void *context)
2      {
3      switch (event->kind)
4          {
5          case WIP_CEV_OPEN:
6              wip_write(event->channel, nachricht, wm_strlen(nachricht));
7              adl_atSendResponse(ADL_AT_RSP, "Daten hochgeladen\r\n");
8              wip_close (event->channel);
9              break;
10         case WIP_CEV_WRITE:
11
12             break;
13         default: /* Fehler oder fertig */
14             wip_close (event->channel);// schließt den Übertragung
15             break;
16         }
17     }
18
19 bool puthandler(adl_atUnsolicited_t *paras)
20     {
21     adl_atSendResponse(ADL_AT_RSP, "Taste 1 wurde gedrückt\r\n");
22     wip_putFile(sessionchannel, "hwp8/mymessage.txt", ftp_send_handler, NULL);
23
24     return FALSE;
25     }
```

Wir erkennen hier unsere Debugmeldung, wenn die Taste gedrückt wurde und der Channel erstellt wird. Das *+CKEV: 0,0* ist die Information, dass die Taste losgelassen wurde. Diese Nachricht haben wir nicht abgefangen.

Als Antwort auf unsere Anfrage an den Server bekommen wir nun die Nachricht *hallo heute ist ein schöner tag*. Dies war auch die Nachricht, die zu dem Zeitpunkt, an dem wir gearbeitet haben auf dem Server zu sehen war. (Leider haben wir hierfür kein Beweisphoto gemacht.) Als nächstes haben wir die Taste 1 ausprobiert.

```
Taste 1 wurde gedrückt

+CKEV: 1,0

Daten hochgeladen
```

Wir sehen wirder die Debugmeldungen und das loslassen der Taste. Nun sehen wir in diesem Fall nur die Nachricht, dass die Datei geschieben wurde.
Eigentlich wollten wir sehen, dass in der Datei nun steht :

```
Das ist ein toller Test, wir können via FTP schreiben.
Schönen Tag noch
```

Hierfür haben wir nun aber ein Beweisphoto geschossen.

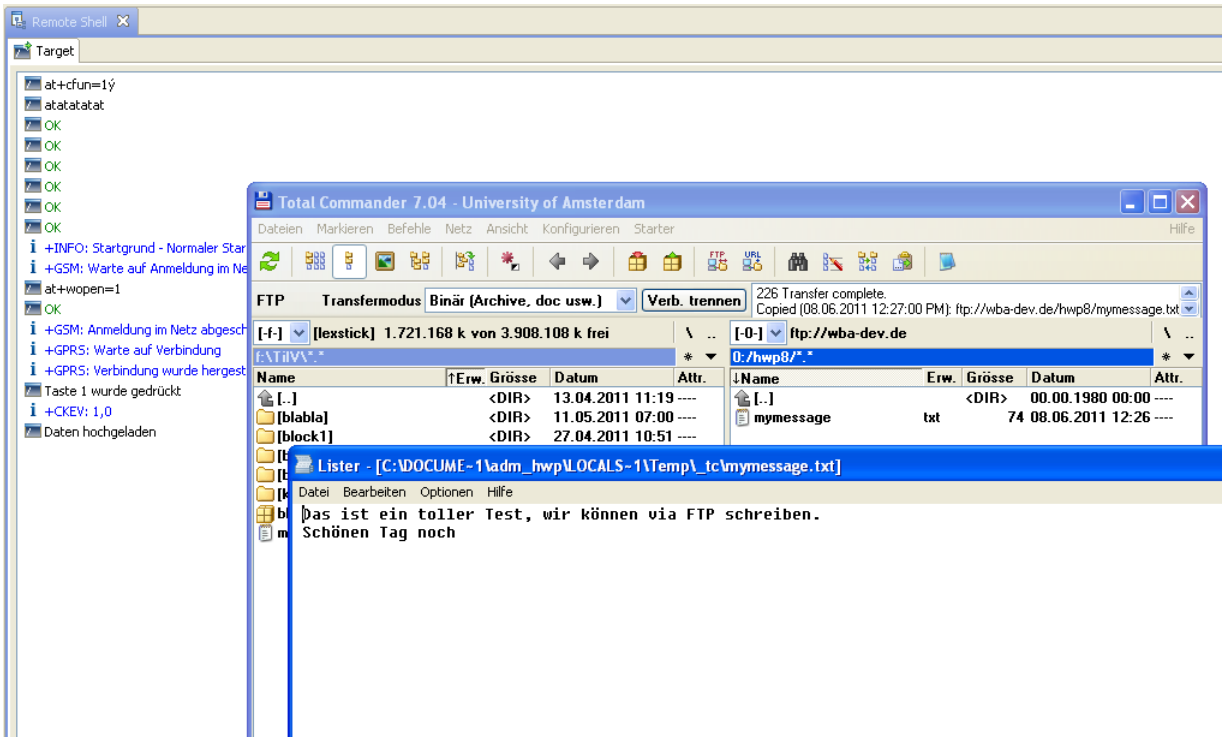


Figure 1: Beweisphoto von der hochgeladenen Datei