Embedded Linux leicht gemacht (6)

Netzwerk und Server-Dienste

Von Benedikt Sauter
[1]

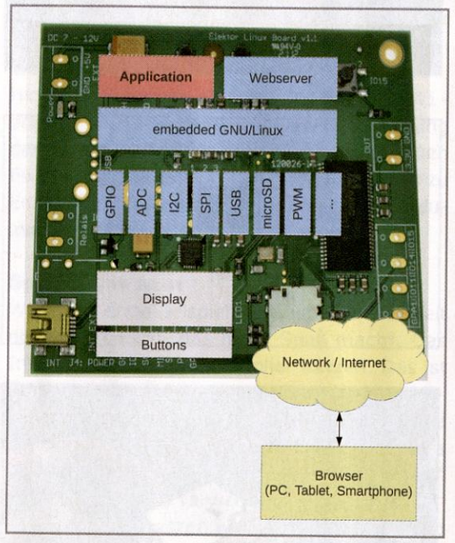

Bild 1.
Übersicht der Anwendung.

Das Baukasten-System Linux bietet sich perfekt als Plattform für Komplettlösungen im Bereich Messen, Steuern und Regeln an. Eine maßgeschneiderte Steuerung kann man in der Programmiersprache der Wahl schreiben, wobei sich die Benutzer-Oberfläche per Webserver zu einem PC, Tablet oder Smartphone schicken lässt. In dieser Folge binden wir das Elektor-Linux-Board in ein typisches Heim-Netzwerk ein. Das Schreiben einer eigenen Server-Anwendung und die Skriptsprache Lua sind weitere Themen.

Vom Bootloader über den Kernel bis hin zur Installation von Gerätetreibern sind in dieser Artikelserie die wesentlichen Linux-Themen behandelt worden. Bis jetzt fehlt aber noch ein Grundkonzept, mit dem man eine größere Anwendung realisieren könnte; beispielsweise mit einer „Logik" im Hintergrund, einer Weboberfläche für die Konfiguration und Bedienung und einer einfachen Bedieneinheit mit Display und Tastern.

Nachrichten zwischen Anwendungen

In **Bild 1** ist ein typisches Szenario dargestellt. In der Mitte befindet sich das Linux-System mit der eigenen Anwendung und dem Webserver, um die Oberfläche auf den verschiedensten Geräten im Browser darstellen zu können. Optional sind noch ein Display und ein paar Taster vorhanden. Für die Ansteuerung des Displays könnte eine eigene Anwendung zuständig sein. Die eigentliche Hauptwendung ist jetzt nur noch ein Prozess, der neben vielen weiteren im Hintergrund läuft. Unter Linux nennt man eine solche Anwendung einen *Daemon* oder Dienst (bzw. Serverdienst). Einmal gestartet, arbeitet diese ohne Eingriff durch den Benutzer. Die Schnittstelle zu weiteren Anwendungen (wie z.B. der Weboberfläche) kann auf verschiedenen Wegen verwirklicht werden (in der Informatik fasst man diesen Austausch zwischen einzelnen Programmen bzw. Prozessen und Threads unter dem Thema Inter-Process-Communication/IPC zusammen). Zur Auswahl stehen Pipes, Fifos, Shared Memory, Sockets, die Standard-Aus- und Eingabe, Dateien, Datenbanken und weitere Hilfsprogramme. Was tatsächlich zum Einsatz kommt, hängt von verschiedenen Faktoren ab. Sind die Anwendungen in der gleichen Programmiersprache geschrieben? Muss die Software auch auf anderen Betriebssystemen laufen können? Welche Datenrate wird bei der Kommunikation zwischen den Anwendungen benötigt?
Eine sehr verbreitete Methode sind klassische Netzwerk-Sockets. Solche Sockets haben den großen Vorteil, dass die Anwendungen sogar über das Netzwerk hinweg Daten und Kommandos austauschen können. Mit Hilfe eines Sockets kann man sehr einfach Nachrichten senden und empfangen, in den verschiedensten Programmiersprachen stehen hierfür Funktionen und Klassen zur Verfügung.
Wir entscheiden uns daher ebenfalls für Sockets. Da das Surfen im Internet auch auf Standard-Sockets basiert, lernen wir zudem alle Bordmittel kennen, um direkt auf Daten im Internet zugreifen zu können. Aus diesem Grund werden wir im nächsten Schritt eine Internetverbindung für unser Linux-Board einrichten.

Verbindung in das Internet

Voraussetzung ist ein Router im Netzwerk, der eine Verbindung zum Internet hat und diese als Gateway anderen Teilnehmern zur Verfügung stellt. Natürlich müssen auch ein USB/LAN-Adapter an die USB-Schnittstelle des Boards angeschlossen und die Treiber für den Netzwerk-Adapter geladen sein [2].
Um einen Teilnehmer in ein Netzwerk zu integrieren, muss man eine IP-Adresse und weitere Parameter wie Gateway und DNS-Server einstellen. Entweder macht man dies manuell oder per DHCP („Dynamic Host Configuration Procotol"). Verwendet man das DHCP-Protokoll, so kann der zuständige Router im lokalen Netzwerk das Elektor-Linux-Board automatisch konfigurieren (siehe **Bild 2**). Ist die IP-Adresse per DHCP zugewiesen worden, kann man zum Testen z.B. den Webserver unter www.elektor.de mit dem Befehl `ping` ansprechen (siehe **Bild 3**).
Ist kein DHCP-Server verfügbar, muss man die Parameter für die Netzwerkverbindung manuell eingeben.

IP-Adresse

Hierfür gibt man ein:

```
ifconfig eth0 <ipadresse>
```

Der Platzhalter <ipadresse> muss durch eine freie IP-Adresse im Netzwerk ersetzt werden. Um zu testen, ob die IP-Adresse erfolgreich übernommen wurde, kann man den Router mit dem Befehl `ping` ansprechen.

```
ping <routeradresse>
```

Der Platzhalter <routeradresse> muss entsprechend durch die IP-Adresse des Routers ersetzt werden.

```
root@gnublin:~# udhcpc
udhcpc (v1.17.3) started
Setting IP address 0.0.0.0 on eth0
Sending discover...
Sending select for 192.168.0.190...
Lease of 192.168.0.190 obtained, lease time 864000
Setting IP address 192.168.0.190 on eth0
Deleting routers
Adding router 192.168.0.1
Recreating /etc/resolv.conf
 Adding DNS server 192.168.0.1
root@gnublin:~#
```

Bild 2.
DHCP-Konfiguration der IP-Adresse.

```
root@gnublin:~# ping www.elektor.de
PING www.elektor.de (94.236.12.177): 56 data bytes
64 bytes from 94.236.12.177: seq=0 ttl=108 time=89.798 ms
64 bytes from 94.236.12.177: seq=1 ttl=108 time=76.505 ms
```

Bild 3.
Ansprechen eines Servers mit dem ping-Befehl.

Adresse des Routers

Um Netzwerkpakete über den Router senden zu können, muss die Standard-Route für den Netzwerkverkehr eingerichtet werden. Dafür gibt es unter GNU/Linux einen Befehl auf der Kommandozeile:

```
route add default gw <routeradresse>
```

Der Platzhalter <routeradresse> muss wiederum durch die IP-Adresse des Routers ersetzt werden. Möchte man diesen Schritt einzeln überprüfen, so kann man einen beliebigen Computer im Internet mit dem Befehl `ping` ansprechen.

```
ping 94.236.12.177
```

Hier sprechen wir wieder den Elektor-Webserver an. Erhält man ein positives Ergebnis (keine verlorenen Pakete), kann man sich an den letzten Schritt machen.

Adresse des DNS-Servers

Um im Internet per Domain-Namen navigieren zu können, benötigt jeder Netzwerkteilnehmer einen DNS-Server. Der DNS-Server kann zu Domain-Namen wie z.B. „www.elektor.de" die passende IP-Adresse ermitteln.
Um einen DNS-Server für das Elektor-Linux-Board einzustellen, editiert man die Datei „/etc/resolv.conf" (**Bild 4**). Nach Eingabe von ...

```
nano /etc/resolv.conf
```

... öffnet sich der Editor. In der Datei sieht den Eintrag „nameserver". Dort kann man seinen lokalen Router eingeben, der meist auch einen DNS-Dienst anbietet. Ist kein lokaler DNS-Server

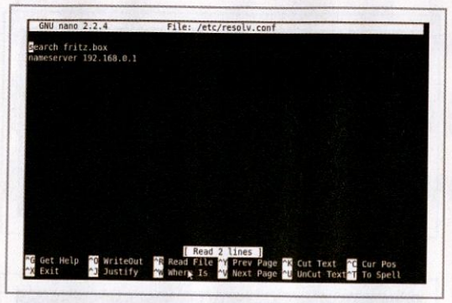

Bild 4.
Konfiguration des DNS-Servers.

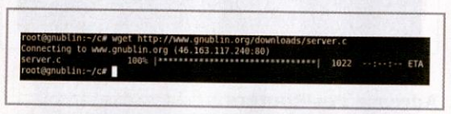

Bild 5.
Download der Datei „server.c".

vorhanden, kann man einen aus dem Internet nutzen. Einfach zu merken und für Testzwecke ideal ist der DNS-Server von Google mit der IP-Adresse 8.8.8.8. Die Datei speichert man mit „Strg-o". „Strg-x" schließt den Editor wieder.
Jetzt sollte jeder Leser erfolgreich beliebige Server im Internet per URL und dem ping-Befehl ansprechen können.

Eigenen Server-Dienst schreiben

Nachdem jetzt die Verbindung vom Board in das Internet steht, können wir uns die Beispieldatei für einen eigenen Serverdienst direkt aus dem Internet herunterladen.
Mit dem Programm „wget", das wir bereits vom PC-Linux her kennen, lassen sich vom Board aus beliebige Dateien aus dem Netz ziehen. Der Download des Beispiel-Serverdienstes wird in **Bild 5** gezeigt. Der Serverdienst ist in der Programmiersprache C geschrieben. Für erste Tests verwenden wir den Compiler auf dem Board. Möchte man mit der Entwicklung selbständig weitermachen, sollte man unbedingt mit der Toolchain auf dem Entwicklungs-PC arbeiten. In **Listing 1** sieht man den Quelltext der Anwendung. Im Wesentlichen wird ein einfacher Server gestartet, der auf Port 5000 lauscht. Meldet sich dort ein Teilnehmer, dann erhält er automatisch die aktuelle Uhrzeit auf dem System als Antwort. Später kann man hier einen Kommandointerpreter einbauen, der abhängig von dem vom Client versendeten Befehl verschiedene Antworten geben bzw. Aktionen durchführen kann.

Um den Server zu testen muss man die C-Datei übersetzen:

```
gcc server.c
```

Ist der Compiler fertig, kann man den Server starten mit:

```
./a.out
```

Jetzt blockiert die Anwendung die aktuelle Konsole und es scheint, als ob man nichts mehr machen kann (außer die Anwendung mit „Strg-c" abzubrechen). Doch das stimmt nicht. Man kann die Server-Anwendung mit „Strg-z" schlafen legen, so dass man auf der Konsole wieder weitere Programme oder Befehle aufrufen kann.
Auf der Konsole erscheint dann:

```
[1]+  Stopped                 ./a.out
```

Gibt man jetzt...

```
fg
```

...ein (fg bedeutet „Foreground"), so ist die Konsole wieder blockiert. Man kann diese jedoch abermals mit „Strg-z" freigeben. Danach kann man...

```
bg
```

...eingeben, der Server-Prozess läuft jetzt aktiv im Hintergrund weiter.

Wie schon erwähnt, schickt unser Test-Socketserver einem Client nach einer Anfrage die aktuelle Server-Uhrzeit. Testen kann man dies auf der Konsole mit dem Programm „telnet", als Parameter werden die IP-Adresse des Servers und der Port angegeben:

```
root@gnublin:~/c# telnet 127.0.0.1 5000

Tue Sep 27 21:34:49 2011

Connection closed by foreign host
```

Möchte man den Server wieder in den Vordergrund holen (um ihn beispielsweise zu beenden), gibt man einfach wieder...

```
fg
```

...auf der Konsole ein.

Natürlich kann man nun auch von einer Weboberfläche aus eine Anfrage an den Server stellen. Und statt die Uhrzeit zurückzugeben, könnte der Server auch eine andere Anwendung steuern (die zum Beispiel eine LED an- und ausschaltet). Im letzten Teil wurde ja schon gezeigt, wie das geht [2].

Listing 1: Eine einfache Server-Anwendung.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>

int main(int argc, char *argv[])
{
  int listenfd = 0, connfd = 0;
  struct sockaddr_in serv_addr;

  char sendBuff[1025];
  time_t ticks;

  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  memset(&serv_addr, '0', sizeof(serv_addr));
  memset(sendBuff, '0', sizeof(sendBuff));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);

    bind(listenfd, (struct sockaddr*)&serv_addr,
sizeof(serv_addr));

    listen(listenfd, 10);

    while(1)
    {
      connfd = accept(listenfd, (struct sockaddr*)
NULL, NULL);

      ticks = time(NULL);
      snprintf(sendBuff, sizeof(sendBuff),
"%.24s\r\n", ctime(&ticks));
      write(connfd, sendBuff, strlen(sendBuff));

      close(connfd);
      sleep(1);
     }
}
```

Fernwartung per Netzwerk

Aktuell verbinden wir uns immer mit „picocom" oder einem anderen Terminalprogramm zu der Konsole auf dem Board. Da wir jetzt aber über eine Netzwerkverbindung verfügen, können wir beliebig viele weitere Konsolen einfach über das Netzwerk öffnen. Dies hat vor allem während der Entwicklung von Anwendungen einen enormen Vorteil. In einer der Konsolen kann man beispielsweise den Editor mit dem Programm öffnen, während sich in einer weiteren Konsole das Programm immer wieder aufrufen lässt, um es im Live-Betrieb zu testen. Um eine Netzwerkverbindung zum Elektor-Linux-Board aufbauen zu können, verwenden wir SSH („Secure Shell"). Diese Konsole bietet einen automatisch verschlüsselten Kanal zum Board an. Dies hat den großen Vorteil, dass kein anderer Teilnehmer im Netzwerk mitlesen kann, was man zu dem Board sendet.

SSH von Linux aus

Unter Linux gehört „ssh" zu den Standard-Tools. Auf der Konsole am Host-Rechner gibt man folgende Zeile ein:

```
ssh root@192.168.0.190
```

Die IP-Adresse muss hier durch diejenige ersetzt werden, die für das Board vergeben wurde. Wenn die Verbindung zum Board steht, wird man aufgefordert, das Wort „yes" einzugeben (**Bild 6**). Und schon sieht man die Standardkonsole, auf der man sich mittlerweile schon heimisch fühlen sollte.

SSH von Windows aus

Unter Windows kann man das Programm PuTTY

Listing 2: Start-Skript.

```
#!/bin/sh

if [ ! -d /var/log/lighttpd ]
then
   mkdir /var/log/lighttpd
   chown -R lighttpd:lighttpd /var/log/lighttpd
   chmod 777 /var/log/lighttpd/
   touch /var/log/lighttpd/error.log
fi

modprobe asix
udhcpc

echo 3 > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio3/direction
chown lighttpd:lighttpd /sys/class/gpio/gpio3/value


chown lighttpd:lighttpd /dev/lpc313x_adc
echo 1 > /dev/lpc313x_adc

chmod 666 /var/log/lighttpd/error.log
/etc/init.d/lighttpd start
```

```
sauter@sauter-Inspiron-530:~$ ssh root@192.168.0.190
The authenticity of host '192.168.0.190 (192.168.0.190)' can't be established.
RSA key fingerprint is bf:7f:52:57:86:59:f0:c6:a6:a9:e3:cb:97:75:36:81.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.190' (RSA) to the list of known hosts.
root@gnublin:~#
```

Bild 6.
Zugang per SSH unter Linux.

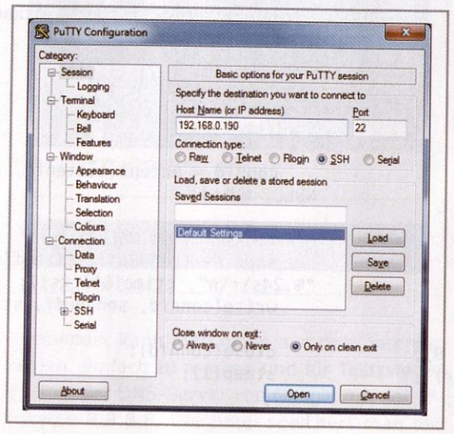

Bild 7.
Zugang per SSH unter Windows.

[3] verwenden. In **Bild 7** sieht man das Hauptfenster dieser Software; hier muss ebenfalls die IP-Adresse für das Board eingegeben werden. Nach einem Klick auf „Open" wird man gefragt, ob der *Fingerprint* des Teilnehmers (ein Teil des SSH-Schlüssels zur Identitäts-Prüfung des Teilnehmers) korrekt ist. Hier kann man auf „Yes" klicken. Bei dem Prompt „login as" gibt man wie gewohnt „root" ein. Nach einer Bestätigung mit der Enter-Taste ist man auf dem System angemeldet.

Skriptsprache Lua für einfache Steuerungen

Im Beispiel zuvor haben wir C als Programmiersprache verwendet. C ist sehr verbreitet, schnell und beliebt. Neben C können wir in einem Linux-System aber viele weitere Programmiersprachen nutzen. Jede Sprache hat ihre Vor- und Nachteile und eignet sich für verschiedene Anwendungszwecke.

Eine sehr flexible, schnelle und einfach zu verwendende Sprache ist Lua. Es handelt sich um eine interpretierte Sprache, die vor allem als Hilfsprogrammiersprache für Spiele bekannt geworden ist. Der Lua-Interpreter ist gerade mal 120 KB groß und bietet dennoch modernste Programmierkonstrukte an. Zugriff auf Dateien – und das können unter Linux ja auch Geräte oder Hardware-Funktionen sein – hat man mit den Befehlen io.open, io.read, io.write und io.close. Für das Elektor-Board hat der Autor eine Lua-Bibliothek geschrieben, um die wichtigsten Hardware-Funktionen (GPIO und A/D-Konverter) einfach nutzen zu können.

Möchte man schnell eine kleine einfache Steuerung realisieren, ist der Vorteil solch einer Interpreter-Sprache gegenüber C enorm. Denn auf dem Board kann man mit dem integrierten Editor (nano oder vi) direkt Programme schreiben und sofort ausführen lassen.

```
require "elektor"

-- initialize hardware

initLED()

-- start main loop

print("blinking ...")

while 1 do
  setLED()
  wait(1)
  clearLED()
  wait(1)
end
```

Bild 8.
Beispielprogramm „blink.lua".

Über die neu eingerichtete Internetverbindung übertragen wir zuerst die Lua-Beispiele auf die SD-Karte:

```
mkdir lua
cd lua
wget http://www.gnublin.org/downloads/elektor.lua
wget http://www.gnublin.org/downloads/blink.lua
wget http://www.gnublin.org/downloads/button.lua
wget http://www.gnublin.org/downloads/adc_relay.lua
```

Das Lua-Listing eines einfachen Blinklichts sieht man in **Bild 8**. Zu Beginn muss die Elektor-Bibliothek geladen werden. Die LED wird mit einem Aufruf von `initLED()` initialisiert und anschließend in einer Endlosschleife ein- und ausgeschaltet. Auf der Konsole startet man das Programm so:

```
lua blink.lua
```

Ein einfaches Ansteuern des Relais mit dem IO15-Taster auf dem Board zeigt die Anwendung „button.lua". Und in **Bild 9** ist der Code des Beispiels „adc_relay.lua" zu sehen. In der Endlosschleife wird über den A/D-Wandler zuerst ein Spannungswert eingelesen. Liegt das Ergebnis über dem Grenzwert von 500, dann wird das Relais geschlossen, ansonsten wird es geöffnet. Zum Testen schließt man ein Poti an den Analog-Eingang an, so wie in der letzten Ausgabe beschrieben [2].

```
require "elektor"

-- initialize hardware

initButton()
initLED()
initRelay()
initADC(1)

-- start main loop

print("Please tune the resitor")

while 1 do
  value = getADC()
  print(value)
  if value > 500 then setRelay() else clearRelay() end
  wait(1)
end
```

Bild 9.
Beispielprogramm „adc_relay.lua".

Literatur für Lua findet man ausreichend im Internet [4] und es gibt auch gute Bücher [5].

Webserver und Anwendung automatisch starten

Es ist ein wenig umständlich, wenn man jedes Mal nach dem Hochfahren des Boards eine Server-Anwendung von Hand starten (und eventuell noch die Hardware konfigurieren) muss. Deshalb fassen wir die entsprechenden Befehle in einem Start-Skript zusammen. **Listing 2** zeigt ein solches Skript für unsere Webserver-Anwendung aus der letzten Ausgabe [2]. Die Datei „start.sh" laden wir uns ebenfalls aus dem Internet herunter (siehe **Bild 10**).

Soll das Skript automatisch beim Start des Linux-Systems aufgerufen werden, kann man hierfür beispielsweise in die Skript-Datei „/etc/rcS.d/S55bootmisc.sh" einen Eintrag aufnehmen (im Ordner „/etc/rcS.d" stehen die Start-Skripts). Zuerst öffnen wir die Datei mit dem Editor:

```
nano /etc/rcS.d/S55bootmisc.sh
```

Am Ende der Datei findet man bereits die Einträge:

```
/bin/mkdir /var/run/sshd
/usr/sbin/sshd
```

Nun ergänzt man dort:

```
/home/root/start.sh
```

Jetzt muss man die Datei wieder mit „Strg-o" abspeichern und den Editor mit „Strg-x" beenden. Damit die Datei „start.sh" gefunden wird, muss diese im Verzeichnis „/root" liegen. Ebenso muss noch das Executable-Flag „x" für die Datei gesetzt werden:

```
chmod +x /home/root/start.sh
```

Ab jetzt sollte sich das Board automatisch nach dem Hochfahren mit dem Netzwerk verbinden, außerdem wird der Webserver gestartet.

Zugriff per Smartphone und Tablet

In der letzten Ausgabe haben wir gezeigt, wie man per PC-Browser auf eine Demo-Webanwendung zugreift. Natürlich geht das mit einem Smartphone oder Tablet genauso. Im Netzwerk, in dem das Elektor-Linux-Board hängt, muss man nur einen WLAN-Zugang für mobile Geräte eingerichtet haben. **Bild 11** zeigt die HTML-Oberfläche auf dem iPhone. Mit dem HTML-Button kann die LED auf dem Board ein- und ausgeschaltet werden.

```
root@gnublin:~# wget http://www.gnublin.org/downloads/start.sh
Connecting to www.gnublin.org (46.163.117.240:80)
start.sh            100% |**************************|   544  --:--:-- ETA
root@gnublin:~#
```

Bild 10.
Download des Start-Skripts für den Webserver.

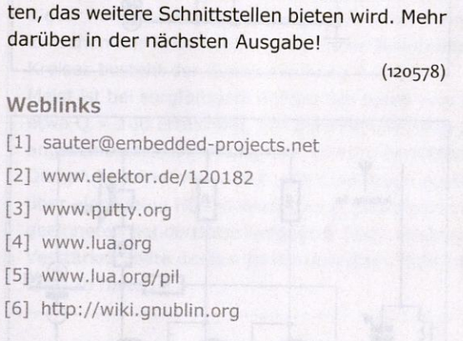

Bild 11.
Zugriff vom iPhone aus.

An dieser Stelle noch ein Tipp für interessierte Leser: Im Community-Wiki [6] des Gnublin-Projekts werden die verschiedensten Hard- und Softwarethemen behandelt. Unter anderem findet man dort Anleitungen, wie man über die I2C-, SPI- und UART-Schnittstelle Daten empfangen und senden (und in den User-Space übertragen) kann.

Für die nächste Folge konnten sich unsere Leser Themen wünschen (zum Redaktionsschluss dieses Artikels lief die Umfrage noch). Wir sind schon gespannt! Und schon jetzt können wir ankündigen, dass wir an einem Erweiterungsboard arbeiten, das weitere Schnittstellen bieten wird. Mehr darüber in der nächsten Ausgabe!

(120578)

Weblinks

[1] sauter@embedded-projects.net
[2] www.elektor.de/120182
[3] www.putty.org
[4] www.lua.org
[5] www.lua.org/pil
[6] http://wiki.gnublin.org