

# „Cheaper Packet“ mit Linux

---

Thomas Sailer, HB9JNX

16. März 1996

## 1 Einleitung

Das Betriebssystem Linux erfreut sich – gerade auch im Amateurfunk – zunehmender Beliebtheit. Das Spektrum an Packet-Radio-Hardware, welche unterstützt wird, ist aber verglichen mit MS-DOOF sehr bescheiden. Keine der üblichen Lösungen eignet sich gut für Mobilbetrieb mit einem Laptop.

Bis anhin sind mir folgende Packet-Radio-Lösungen bekannt:

- TNC, RMNC oder Rechner an serieller Schnittstelle (Ankopplung mittels KISS)
- Rechner mittels Ethernet und AXIP<sup>1</sup>
- (U)SCC Karte

## 2 Baycom SER12 Modem

Das Baycom SER12 Modem [4] ist mittlerweile weit verbreitet. Die SMD-Ausführung dieses Modems ist sehr klein (Steckergehäuse) und wird ausserdem über die serielle Schnittstelle gespiesen, sodaß eine externe Stromversorgung entfällt. Es ist daher durchaus auch für Mobilbetrieb geeignet.

Es wurde oft behauptet, daß eine Ansteuerung dieses Modems unter einem modernen Multitasking-Betriebssystem wie Linux nicht möglich sei, weil die Interruptbelastung viel zu hoch sei usw. Dies trifft nicht zu. Im Vergleich zum DOS/EMM386-Gespann dürfte Linux wohl ausgesprochen gut dastehen, was die Interrupt-Verarbeitungsgeschwindigkeit betrifft.

Der Standard-Linux-Treiber für die serielle Schnittstelle kann natürlich nicht verwendet werden. Dieser Treiber ist ja für den eigentlich vorgesehenen Verwendungszweck als asynchrone serielle Schnittstelle gedacht, während das SER12-Modem

Vor einem Jahr habe ich in diesem Forum [3] Soundkarten mit programmierbarem DSP vorgestellt. Diese können auch recht einfach unter Linux benutzt werden, da der DSP auf der Soundkarte alle zeitkritischen Aufgaben übernimmt und die Ansteuerung der Karte daher einfach aus dem Usermode erfolgen kann. Auch diese Lösung kommt wie die USCC-Karte für Mobilanwendungen nicht in Frage, da Laptops über keine ISA-Slots verfügen.

Dieses Paper untersucht nun folgende Ansätze:

- Baycom SER12 Modem
- „Dummsoundkarte“

die Schnittstelle als PIO-Port mit Timer zweckentfremdet. Das Baycom-SER12-Modem lässt sich daher nur von einem Kernel-Modul ansteuern. Nur ein Kernel-Modul kann unter Linux direkt auf die Hardware zugreifen<sup>2</sup> und Interrupt-Handler installieren.

Abbildung 1 zeigt die von mir verwendete Lösung. Sie besteht aus zwei Teilen, einem Treiber im Kernel-space und dem Modul im Userspace, das das KISS-Protokoll implementiert.

Das Modul im Kernelspace – ich habe es `ser12` getauft – übernimmt die Ein- und Ausgabe der Pakete. Es wird als char device mit major 60, minor 0, angesprochen (`mknod /dev/ser12 c 60 0`). 60 ist die erste major number aus dem local/experimental range. Gegebenenfalls sollte der Treiber koordiniert werden und eine offiziell zugeteilte major number verwenden. Der Treiber verwendet ein „proprietäres“ (d.h. entgegen allen sonst üblichen Gepflogenheiten)

---

<sup>1</sup>Dieses Setup verwende ich am Heim-QTH, ein 286er mit PC/FlexNet dient der Linux-Kiste als TNC

<sup>2</sup>dies geht zwar auch noch vom Usermode aus, mittels `iopl()` oder `ioperm()`

Interface, das nur `ioctl`, nicht aber `read` oder `write`, unterstützt. Die Paket-Ein/Ausgabe erfolgt paketweise ebenfalls mittels `ioctl`.

`ser12` spricht direkt die Hardware, also den UART-Baustein 8250, 16450 oder 16550<sup>3</sup>, an. Dies hat nun zwei Konsequenzen:

- `ser12` unterstützt nur Standard-Schnittstellen. Wer das Baycom-Modem an einer `stallion`- oder `cyclades`-Karte anschliessen will, hat verloren...
- `rs`, der Standard-Async-Treiber, darf die UART-Bausteine nicht für sich reservieren. Dies tut er normalerweise aber, auch wenn der entsprechende Port gar nicht gebraucht

wird. Dies kann man ihm abgewöhnen mit `setserial /dev/ttyS□ port 0`.

Das Programm im Userspace, `s2kiss`, konvertiert nun das "proprietäre" `ioctl`-Interface des `ser12`-Treibers in ein Standard-KISS-Interface, das dann über ein `pty/tty`-Paar<sup>4</sup> mit einem AX25-Stack wie `Wampes` verbunden werden kann.

Die obige, aus zwei Modulen bestehende Lösung ist leider recht kompliziert. Es wäre deutlich eleganter, wenn `ser12` direkt KISS sprechen würde. Leider habe ich keine Ahnung, wie man einen `tty`-Driver<sup>5</sup> schreiben muß, mir fehlt dazu jegliche Dokumentation<sup>6</sup>.

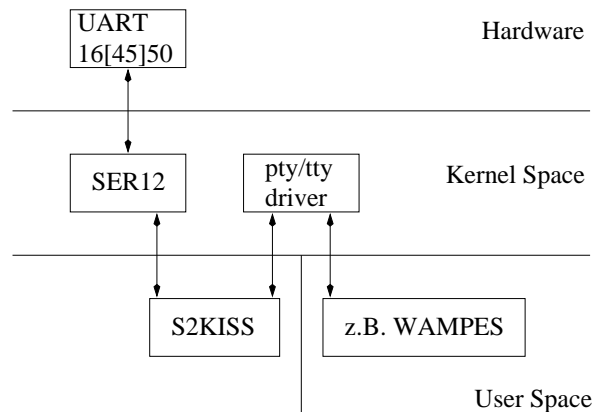


Abbildung 1: Treiber zur Ansteuerung des SER12-Modems unter Linux

## 2.1 Die Installation des Treibers

`setserial /dev/ttyS□ port 0` Freigabe des UART-Bausteines durch den Linux Async-Treiber. □ muß durch die Nummer des Ports ersetzt werden, der den Baustein belegt.

ist `/dev/ttyS0`, was dem COM1 von DOS entspricht).

`s2kiss` Usermode-Modul laden

`insmod ser12.o iobase=0x2f8 irq=3`  
Kerneltreiber laden. Die Angabe der Adresse und des IRQs des Ports ist optional (Default

Das Laden von `ser12` kann ggf. mittels `kernelld` automatisiert werden.

<sup>3</sup>Die in diesem Baustein eingebauten FIFOs können nicht verwendet werden und werden abgeschaltet

<sup>4</sup>Dies kann man sich als Software-Emulation zweier miteinander verbundenen seriellen Schnittstellen vorstellen

<sup>5</sup>Der *Kernel Hackers Guide* 0.6 [1] schweigt sich darüber aus

<sup>6</sup>Einige Leute vertreten die Meinung, der Sourcecode sei ja verfügbar, damit sei es dokumentiert. Es gibt wohl für den Leser angenehmere Formen der Dokumentation, als sich durch Megabytes an Sourcecode zu kämpfen

Linux-Pfad	DOS-Äquivalent	iobase	irq
/dev/ttyS0	COM1	0x3f8	4
/dev/ttyS1	COM2	0x2f8	3
/dev/ttyS2	COM3	0x3e8	4
/dev/ttyS3	COM4	0x2e8	3

Tabelle 1: Adressen und IRQ's der Standardschnittstellen

### 3 “Dummsoundkarte”

Unter “Dummsoundkarte” verstehe ich eine Soundkarte ohne eigene Rechenleistung, die nur Samples ein- und ausgeben kann.

Seit Linux 1.3.69, welche TASD<sup>7</sup> 3.5-beta10 enthält, ist es mir erstmals gelungen, den Linux-Sounddriver zur Arbeit mit meiner Soundkarte (im Desktop-PC) zu überreden, obwohl die angeblich

schon seit mehr als einem Jahr unterstützt wird. Ich konnte daher der Versuchung nicht widerstehen, damit ein Packet-Modem zu implementieren.

Wie in Abbildung 2 zu sehen ist befindet sich lediglich die Ansteuerung der Soundkarte im Kernel-space, die ganze Verarbeitung der Audiodaten findet im Userspace statt.

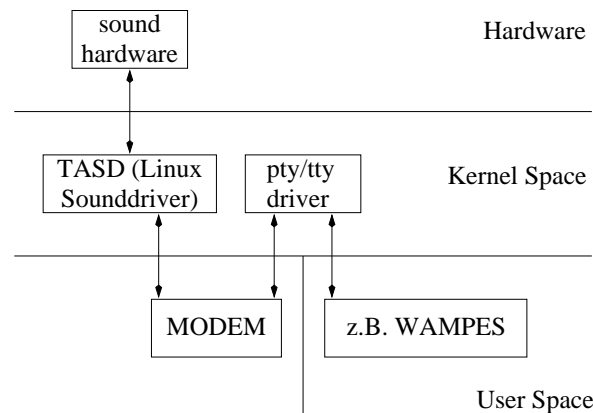


Abbildung 2: Packet Radio mit dem Linux-Sounddriver

#### 3.1 Probleme

Das Modem im Userspace bietet einige große Vorteile:

- Der Code lässt sich einfach entwickeln und debuggen
- Es findet kein direkter Zugriff auf die Hardware statt, alle Zugriffe gehen durch den Sounddriver. Soll eine neue Soundhardware unterstützt werden, ist dies lediglich ein Problem des Sounddrivers, nicht des Modems

Dieses Konzept hat aber auch einige gravierende Nachteile:

- Die Echtzeitfähigkeit ist beschränkt. Der Linux-Scheduler arbeitet mit 10ms-Timeslices, daher kann es durchaus passieren, daß das Modem  $\frac{1}{10}$ s warten muß, was die Umschaltzeiten zwischen Senden und Empfangen in die Höhe treibt. Man

<sup>7</sup>Temporarily Anonymous Sound Driver; er hiess früher mal VoxWare, mußte aber umbenannt werden, da es offenbar eine Firma gleichen Namens gibt

kann zwar mittels `nice`<sup>8</sup> und `mlockall()`<sup>9</sup> schon dafür sorgen, daß man recht häufig drankommt, aber eine Garantie ist dies noch nicht. Ausserdem sind die Echtzeitfähigkeiten des Linux Sounddrivers recht beschränkt. Er wurde ja auch dazu entwickelt, die lückenlose Krachuntermalung von Spielen<sup>10</sup> zu erlauben, und nicht zur Implementation von Modems.

- Die Dokumentation des Sounddrivers [2] ist recht alt und unvollständig. Ich konnte es ihm z.B. nicht abgewöhnen, das Eingangssignal zum Ausgang hinzuzumischen. Dies verhindert ziemlich zuverlässig das Senden...

Dieses Projekt ist, der oben beschriebenen Probleme wegen, noch nicht so weit gediehen wie die Ansteuerung des `ser12`-Modems. Der Receiver läuft

(1200 Baud AFSK). Die Prozessorbelastung ist mit 20% auf einem 486DX2/66 noch recht hoch, lässt sich aber noch drücken, denn das ganze Modem ist derzeit in C geschrieben und benötigt noch sehr viele Integer-Multiplikationen, was auf dieser Rechnerklasse lange dauert<sup>11</sup>.

Ob ich das Projekt fortsetze, hängt noch von verschiedenen Faktoren ab, nämlich der eigenen Motivation, der angekündigten Verbesserung der Sounddriver-Dokumentation und dem Interesse der 'Fachwelt'.

Sowohl die hier beschriebene Software, wie auch dieser Artikel sind unter grossem Zeitdruck in weniger als einer halben Woche entstanden, es ist daher durchaus möglich, daß der Artikel zum Zeitpunkt seiner Drucklegung wieder veraltet ist.

## Literatur

- [1] Michael K. Johnson, *Linux Kernel Hackers' Guide*, Linux Documentation Project, <ftp://sunsite.unc.edu/pub/Linux/docs/LDP>
- [2] Hannu Savolainen, *Hacker's Guide to VoxWare 2.4 (second draft)*
- [3] *11. Internationale Packet-Radio Tagung, Scriptum*, Darmstadt, 1995
- [4] Johannes Kneipp, DG3RBU, Florian Radlherr, DL8MBT, ◇ *BayCom SMD-Modem*, 2. überarbeitete Version, Februar 1992

---

<sup>8</sup>Priorität festlegen

<sup>9</sup>Auslagern des Prozesses verhindern. Meine `libc` unterstützt dies aber noch gar nicht

<sup>10</sup>"Killerapplikationen" wie DOOM lassen grüssen

<sup>11</sup>Nicht-Intel-Prozessorbesitzer sind hier im Vorteil