# FLI4L als User-Mode-Linux

TURBO J

27. September 2004

**Zusammenfassung**

Diese Dokument beschreibt, wie man FLI4L als User-Mode-Linux laufen lassen kann. Man benötigt nur den FLI4L (>2.1.4) und das UML-opt. Dazu aber auch ein paar handfeste Linux- Kenntnisse.

## Inhaltsverzeichnis

# 1 Vorbereitungen oder was man so alles braucht

Das Wichtigste zuerst: man benötigt einen Linux- PC als Host. Die Distribution ist eigentlich egal, es muss ein halbwegs aktueller Kernel laufen- 2.2.15 oder höher oder alle 2.4'er, 2.6 sollte auch gehen. Man benötigt die *bridge-utils*[1] und die *uml-utilites*[2], erstes zum Ausetzen der Netzwerkbrücken, letzteres für die UML.

Als fli4l -version kommt z.Z. nur die dev in Frage, da der UML-Client-Kernel zwingend ein 2.4'er sein muß. Da eine UML so gut wie keine Hardware direkt ansprechen kann, fällt ISDN leider weg- das könnte man dann aber auf dem Host hinbekommen. Bei USB-DSL-Modems bin ich mir nicht sicher, ob es geht. Ich habe hier zum Testen nur die "normale(?)" Version mit Ethernet-DSL-Modem. Dafür ist natürlich das dsl-Paket nötig. Bei der Auswahl von weiteren Opts ist darauf zu achten, dass sie keine Kernel-Module selbst mitbringen- das geht mit der UML nicht- aber sonst gibt es kaum Einschränkungen.

## 1.1 Kleine Liste der benötigten Dinge

### 1.1.1 Host

- Kernel 2.2.15 oder neuer, oder 2.4.3 (?) oder neuer
- bridge-tools
- uml-utilities
- Hardware min. ca. P200 mit ca. 64 MB RAM, hier hilft viel viel

### 1.1.2 Fli4l

- Version 2.1.8
- *opt*s nach Wunsch, ISDN oder SCSI geht leider nicht[3]
- opt_uml, natürlich...

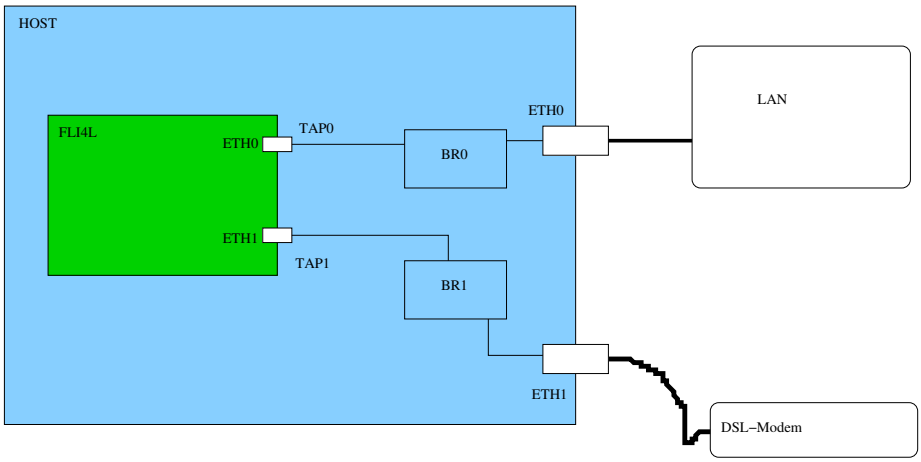
# 2 Konfiguration

So stelle ich mir das vor:

[1] Homepage: http://bridge.sourceforge.net

[2] Homepage: http://user-mode-linux.sourceforge.net; könnten bei SuSE auch dabei sein, selbst kompilieren ist aber beser ;-)

[3] da keine Hardware-Unterstützung der UML




## 2.1 Konfiguation des FLI4L

Den fli[4] konfiguriert man "ganz normal", außer dass man als "Netzwerktreiber" *dummy* einstellt. Im Folgenden gehe ich davon aus, dass man als eth0 eine IP aus dem LAN hat und eth1 für das PPPOE-> Modem benutzt werden soll.

Das Root für UML erzeugt man mit dem mkuml-root.sh, welches leider root-Rechte benötigt, da es ext2-Filesystem(e) mounten muss. Einen "Bug" hat es: Das erzeugte Root würde genau 1 Mal laufen, danach bringt es Fehler beim Booten. Aus diesem Grund wird ein COW-Filesystem (Spezialität von UML) benuzt, das Änderungen in einer extra Datei speichert, die dann einfach gelöscht wird- so ist bei jedem Booten das Root-FS "frisch".

Ich sollte hier vielleicht erwähnen, dass der fli eine *andere* IP-Adresse benötigt als der Host, sonst geht es nicht. Hat der Host die 192.168.0.99, so kann man dem fli z.B die 192.168.0.100 geben- sofern die frei ist.

Fast hätte ich es vergessen: BOOT_TYPE (in der base.txt) muss auf "integrated" oder "attached" stehen. Falls man PPPOE verwendet: PPPOE_TYPE='sync' geht auch nicht (fehlendes Modul), 'async' oder 'in_kernel' geht. Dasselbe Problem (fehlendes Modul) hat PPTP_MODEM_TYPE='xdsl'.

## 2.2 Konfiguration des Host

Manchmal muss man das tun-Modul nachladen mit *modprobe tun.*Wichtig:chmod 666 /dev/net/tun, wenn die UML als normaler user laufen soll, *ifconfig tap0 promisc*, damit ALLE Pakete drübergehen. Ach ja, dies und die folgenden Schritte benötigen "root"- Rechte auf dem Linux-Host.

Das eigentliche Problem sind aber die Brücken: hier greift man in die Netzwerkkonfiguation ein, was bei jeder Distibution ein bisschen anders aussieht. Zuerst ezeugt man die Brücken: *brctl addbr br0* und *brctl addbr br1*, dann kommen die tapX- Devices: *ifconfig tap0 up promisc* und *ifconfig tap1 up promisc*

[4] fli = FLI4L = fli4l

(promisc steht für den Modus, wo das Ethernet-Device alle Pakete durchlässt, also "ohne Filter" ). Jetzt mal mit *ifconfig* ohne Parameter nachschauen, ob alle Devices "oben" sind, und dann weiter mit der Bridge: *brctl addif br0 tap0* und *brctl addif br1 tap1.* Danach löscht man die IP-Adresse der Ethernet-Karte eth0 mit *ifconfig eth0 0.0.0.0 promisc* und *ifconfig eth1 0.0.0.0 promisc*, aber jetzt hat der Rechner keine Netzverbindung mehr. Deshalb schnell: *brctl addif br0 eth0* und *ifconfig br0 up 1.2.3.4*, wobei 1.2.3.4 die IP-Adresse der Ethernetkarte eth0 ist. Zum Schluss kommt noch *brctl addif br1 eth1* und *ifconfig br1 up* für die andere Bridge. Damit man die Behle nicht umständlich von Hand eigeben muss, empfielt sich ein Skript. Eine beliebte Stolperstelle ist hier der Fall, dass eth0 die IP-Adresse von einem DHCP-Server bekommt. Wenn möglich, sollte eine statische IP benutzt werden, ansonsten kann man einer Brücke auch die IP über DHCP verpassen- das wird hier jetzt aber nicht beschrieben[5].

Für ganz Faule stelle ich ein Skript für SuSE 8.2 bereit, was dies alles gleich beim Start erledigen kann. Es müsste ohne große Änderungen auch unter 9.0 und 9.1 funktionieren.

# 3 Fli4L als UML laufen lassen

## 3.1 Das "run-uml.sh" Skript

Obwohl nur ein Beispiel-Skript, kann man- sofern man sich an die obige Anleitung besonders bezüglich tapX-Devices gehalten hat- das Skript ohne grosse Änderungen laufen lassen. Man kann es sogar als normaler User laufen lassen, was noch einmal etwas zusätzliche Sicherheit bringt. Ein Problem wird auf der UML-WS beschrieben: Bei einer bestimmten Koniguration des Kernels (sog. 2G/2G -Konfiguration) gibt es sofort nach dem Start ein "Segmantation fault". Abhilfe gibt es in 4.1.1 auf Seite 6- leider muss man selbst einen UML-Kernel kompilieren oder sich einen Passenden (incl. Module) suchen.

Beim Laufen des Skripts sollte man die Meldungen des Linux-Kernels sehen, als ob er ganz normal booten würde (mach er ja auch). Hier sieht man auch Fehlermeldungen. Wenn er sich öfters über ppp-Module beschwert, ist womöglich eth1 (auf dem fli) nicht "oben". Ein ifconfig hilft auf dem FLI um zu sehen, ob eth0 und eth1 "da" sind- einloggen konnt ich mich fast immer in den fli, selbst wenn sonst fast nix mehr ging. Ein *halt* oder *poweroff* fährt den fli herunter und beendet die UML. Es ist normal, wenn im *ps aux* viele Prozesse names "linux" auftauchen- jeder Prozess einer UML hat auch ein Gegenstück auf dem Host[6]. Zu Glück laufen beim fli normalerweise nur sehr wenige Prozesse.

Es ist möglicherweise nötig, das run-uml.sh Skript auf die jeweiligen Gegebenheiten des Host abzustimmen. Stichworte sind hier Netzwerk- und Speicherzuordnung.

[5]Ich bin dafür nicht nur zu faul: Das ist bei fast jeder Distribution ein kleines bisschen anders...

[6]Dies kann man mit dem SKAS- Patch auf der UML-Webseite ändern, was auch gleich noch mehr Performance bringt

### 3.2 Start von Hand- optional?

Für den händischen Testlauf kann man einfach das run-uml.sh Skript als Beispiel benutzten. Die auszuführende Datei heisst "linux" und liegt in "./img/". Sie enthält den Linux-Kernel von fli4l und kann wie ein gewöhnliches Programm gestartet werden. Mann kann sogar *./img/linux –help* eingeben um eine kleine Übersicht über die Optionen zu bekommen. Das Rootfs liegt in ./img/root_fs.uml bzw. ./img/root_fs.cow, wobei das .cow -file "sparse" ist - es erscheint viel grösser als es Platz auf der Platte belegt. Einfahch mal *ls -ls img/* eingeben. Auf (V)FAT gibt es leider keine Sparse-Dateien, deshalb sollte man die für eine Fli4l-UML vermeiden- oder das Rootfs ohne .COW-Datei als Ramdisk laden. Wenn jemand Lust hat, kann er die Doku dazu schreiben ;-)

### 3.3 Funktioniert alles?!

Wenn man sich in den fli eingeloggt hat, kann man mal die Netzwerkverbindungen testen. Ein Ping sagt mehr als tausend Worte- nee, das ging dann doch irgenwie anders...

Ich habe selber PPPOE über eine Bridge und UML ausprobiert, das funktioniert ohne Probleme[7]. Interessanter sind in diesem Zusammenhang USB-DSL-Modems - es gibt AFAIK ein USB-Modul für UML, was Geräte benutzen kann, die NICHT vom Host angesprochen werden (wo also im Host keine Treiber für geladen sind). Wie dieser in ein UML-Fli eingefügt und aktiviert werden kann, muss leider jemand anders testen. Normalerweise sollte die Bridge funktionieren, USB-DSL-Modems sehen AFAIK wie Netzwerkkarten aus (aus Sicht der Software betrachtet).

## 4 Sollbruchstellen oder wo es so alles klemmen kann

Einiges habe ich schon erwähnt. Kommen wir zunächst zu

### 4.1 Host - Probleme

Ob die Brücken richtig oben sind, verrät ein *ifconfig* gefolgt von einem *brctl show* -bei letzterem sieht man die Zuordnung von Netzwerk-Devices zu den Brücken und ob da was fehlt. Ein laufender dhcp-client kann hier auch schon mal das Ethernet-Device umkonfigurieren, so dass man plötzlich den fli nicht mehr ansprechen kann. Am Besten abschalten und statische IP verwenden.

In größeren Netzwerken kann auch noch das Problem von 2 gleichen MAC-Adressen auftauchen. Wenn auch noch - bei SuSE 8.2 standard- die tapX-Devices in der /etc/modules.conf auf z.B. `alias tap0 ethertap` stehen wird, wenn man etwas Pech hat, statt des tun/tap *ethertap* ausgewählt. Ethertap sucht sich normalerweie keine eigene MAC aus, sondern nutzt per Default "FE:FD:00:00:00:00".

[7] Falls nicht gerade das DSL-Modem die Hufe hochgerissen hat..

Wenn das auf 2 Rechnern im selben Netz passiert, hat man ein Problem. Abhilfe: auf einem Rechner entweder "`ifconfig tapX hw ether FE:FD:00:xx:xx:xx`" eingeben (oder `rmmod ethertap`;-))- oder in der /etc/modules.conf `alias tapX tun` eingeben (X-0.. Anzahl TAP-Devices). Das "2- gleiche- MACs" -Problem ist ziemlich schwer zu diagnostizieren, wenn man nicht auf die Idee kommt die MACs zu vergleichen, da dies alle möglichen Netzprobleme nach sich ziehen kann.

#### 4.1.1 2G/2G

Falls der fli -Kernel gleich beim Start eine Schutzverletzung auslöst ist u.U. der Host- Kernel mit der (IMHO non-stardard) Option 2G/2G kompiliert worden. Wenn man das nicht ändern möchte/ kann so kann man den Fli-Kern selbst kompilieren. Man geht in das Verzeichnis src/kernel-2.4/ und sichert die Datei dot-config-uml erstmal (z.B. indem man sie als dot-config-uml.orig kopiert). Dann kopiert man die dot-config-uml-2G2G über die dot-config-uml und startet das Skript *./mkkernel-uml.sh -all* was den Kernel (2.4.26, ca 29MB) per ftp herunterlädt, auspackt, patcht, konfiguriert und ein neues Kernel-Image samt dazugehöriger Module erstellt. Ausreichend Speicherplatz auf der Platte als Voraussetzung: ca. 250 MB werden benötigt. Ein Start des "linux"-Programms sollte dann klappen, ohne Optionen gibts allerdings (im Fli) einen Kernel Panic- weil er kein rootfs findet.

Wer die SuSE 9.1 benutzt, hat u.U. ein Problem: Deren GCC schafft es nicht, den Kernel fehlerfrei zu bauen. Aber: Die SuSE-Kernels sind nicht mit 2G/2G übersetzt, man barucht ihn also auch nicht zu kompilieren. Darüberhinaus enthalten die SuSE-Kernel auch noch den "SKAS-Patch", siehe 5.1.

### 4.2 Probleme auf dem FLI4L

Falls die Netzwerk- Devices auf dem fli nur "permission denied" beim Hochfahren (ifconfig ethX up) melden, so hat der fli nicht genügende Zugriffsrechte auf /dev/net/tun. Es muss sowohl Lesen als auch Schreiben möglich sein. Zum Testen genügt "chmod 666 /dev/net/tun", wer etwas paranoid ist (also Erfahrung hat) stellt sich die Rechte so ein, dass nur der fli drauf zugreifen kann.

## 5 Tuning

### 5.1 Host- Tuning

Laud der Webseite von UML kann man mit dem SKAS-Patch noch mehr aus dem System herausholen, da dem UML-Fli dann ein eigener Adressraum zugestanden wird. Das macht dann den Fli schneller, günstig wenn die CPU knapp ist. Der große Nachteil: Das ist ein Patch für den Host-Kernel, und nicht jeder will sich seine Kernel selber patchen. Da aber der Fli normalerweise sehr wenig Resourcen verbraucht, ist es eigentlich auch nicht nötig. Mit der etwas höheren CPU-Last muss man dann halt leben können.

Anmerkung: Offenbar ist der Patch bei den Kernels von SuSE 9.0 und 9.1 bereits enthalten.

# 6 Anhang

## 6.1 Haftungsausschluss

Soweit es geltende Gesetze erlauben ist jede Haftung duch den/die Autor(en) ausgeschlossen.

Wer dieses Dokument liest und seine Anweiseungen befolgt, sollte besser wissen was er/sie da tut.

## 6.2 Das Startskript für SuSE

### 6.2.1 bridge und net-bridge

Ich habe das Skript -welches sich unter `unix/uml/bridge` befindet -für SuSE8.2 geschrieben und getestet. Man kopiert es nach `/etc/init.d` und setzt die Dateirechte auf +x für user `root`. Die Konfiguration - ein Beispel steht unter `unix/uml/net-bridge` - kopiert man nach `/etc/sysconfig/` und ändert es nach den jeweiligen Gegebenheiten und Vorlieben ab. Die Standard-Konfig stellt eine Brückenkonfiguration wie im Bild her und stellt als IP für die Brücke br0 die 192.168.6.2 ein. Die restlichen Devices- br1, eth0 und eth1, tap0 und tap1 erhalten KEINE IP-Adresse. Für die Brücken kann man- so man braucht- IP-Adressen einstellen- allerdings nur 1 pro Brücke. In dem (fiktiven ?) Beispiel benötigt man br1 aber nur zum Durchleiten der PPPOE- Pakete vom/zum DSL-Modem, und dafür benötigt man keine IP-Adresse. Brücken arbeiten auf der sog. MAC-Ebene, also mit "rohen" Ethernet-Paketen. Deswegen kann man JEDES Protokoll (TCP/IP, IPX, ...) durch eine Bridge durchleiten.

Für andere Distributionen kann man das Skript als groben Anhaltspunkt zur Erstellung eigener Skripte benutzen. Ältere SuSE- Distributionen haben IMHO ein anderes System zur Netzwerkkonfiguration. Wann dies umgestellt wurde ist mir nicht in Erinnerung geblieben, es ist möglich dass das Skript auch für SuSE 8.1 läuft. Neuere SuSE sollten allerdings funktionieren- falls SuSE nicht schon wieder was geändert hat. SuSE 9.0 hat eine gute Chance, ohne Änderungen im Skript zu laufen. Für 9.1 muss man mindestens den Pfad zu "brctl" in 'bridge' ändern: Dieses Programm steht in /sbin statt in /usr/sbin.

Falls man das Skript im laufenden Betrieb startet, gibt es eine kleine Unterbrechung des netzwerkverkehrs. Ich glaube das dies daran liegt, dass die Bridge erst einmal "lernen" muss, wo sie welche Pakete hinzuschicken hat (MAC-Adressen benachbarter Rechner). Das dauert bei mir so ungefähr 10 bis 20 Sekunden.

### 6.2.2 fli4l.init

Das fli4l.init- Skript kopiert man nach "/etc/init.d/fli4l". Vorsicht! Das ist noch 'ne Alpha-Version. Es wird davon ausgegangem, dass ein User 'fli4l' existiert, dessen $HOME das Verzeichnis ist, in dem 'run-uml.sh' steht.

## 6.3 Weitere Gemeinheiten

Die IP-Adresse auf der br0 des Host ist zum Betrieb des Fli nicht unbedingt notwendig, man kann "ohne" den Host völlig unsichtbar machen. Jedoch muss ich an dem Sinn der Aktion zweifeln, denn was will man mit einem Linux-PC, der keine eigene IP hat? Man kann dann ja gar nicht mehr darauf zugreifen! Da könnte man den Fli doch gleich direkt ohne UML laufen lassen. Bei vielen Netzwerken und -karten könnte eine ähnliche Konfiguration allerdings sehr wohl sinvoll sein.

Falls man nicht mehr an den Fli dran kommt, kann man ihn einfach auf dem Host mit "*killall linux*" abschiessen. Notfalls hilft ein "killall -9 linux", dabei könnten aber UML-Hilfsprogramme aus dem Tritt kommen. Somit hat eine UML auch eine Art von Reset-Taster...

# 7 Danksagungen

Danke vor allem an das FLI-Team für ein großartiges (und doch kleines ;-) Stück Software.

Danke auch an Arno für das mkintagrated.sh -Skript, aus dem ich das mkuml-root.sh Skript gebaut habe.

Danke auch an Damian Philipp für ein paar gute Anregungen in der NG und fürs Testen.

Danke an alle die ich jetzt wieder vergessen habe...