

Linux Fast-STREAMS

Frequently Asked Questions

Version 0.7a Edition 4
Updated 2006-01-04
Package streams-0.7a.4

Brian Bidulock <bidulock@openss7.org> for
The OpenSS7 Project <<http://www.openss7.org/>>

Copyright © 2001-2005 OpenSS7 Corporation <<http://www.openss7.com/>>
Copyright © 1997-2000 Brian F. G. Bidulock <bidulock@openss7.org>
All Rights Reserved.

Published by OpenSS7 Corporation
1469 Jefferys Crescent
Edmonton, Alberta T6L 6T1
Canada

This is texinfo edition 4 of the Linux Fast-STREAMS documentation, and is consistent with streams 0.7a. This guide was developed under the **OpenSS7 Project** and was funded in part by **OpenSS7 Corporation**.

Permission is granted to make and distribute verbatim copies of this guide provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this guide under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this guide into another language, under the same conditions as for modified versions.

Short Contents

Acknowledgements	1
Preface	3
1 FAQ	7
A Copying	23
Glossary	37
List of Figures	45
Index	47

Table of Contents

Acknowledgements	1
Sponsors	1
Contributors	1
Preface	3
Document Information	3
Notice	3
Abstract	3
Objective	3
Intent	3
Audience	3
Revisions	4
Version Control	4
ISO 9000 Compliance	4
Disclaimer	5
U.S. Government Restricted Rights	5
1 FAQ	7
1.1 What is STREAMS?	7
1.2 Does Linux have STREAMS?	7
1.2.1 Linux STREAMS (LiS)	7
1.2.2 Linux Fast-STREAMS (LfS)	8
1.2.3 STREAMS for FreeBSD	8
1.2.4 OpenSTREAMS?	8
1.3 Why STREAMS?	8
1.3.1 STREAMS versus Sockets	9
1.3.2 STREAMS Benefits	11
1.3.3 Standardized Service Interfaces	11
1.3.4 Manipulating Modules	12
1.3.4.1 Protocol Portability	12
1.3.4.2 Protocol Substitution	13
1.3.4.3 Protocol Migration	13
1.3.4.4 Module Reusability	14
1.3.5 STREAMS Criticism	15
1.3.6 STREAMS Realities	18
1.4 Why Fast?	18
1.5 Why Linux?	20
1.6 Why Compatibility?	20
1.6.1 Intel Binary Compatibility Suite (iBCS)	20
1.6.1.1 OpenGroup Specifications	21
UNIX 03 Compliance	21
UNIX 98 Compliance	21

UNIX 95 Compliance	21
1.6.2 Device Driver Interface (DDI)	21
Appendix A Copying	23
A.1 GNU General Public License	23
A.1.1 Preamble	23
A.1.2 Terms and Conditions for Copying, Distribution and Modification	24
A.1.3 How to Apply These Terms to Your New Programs	28
A.2 GNU Free Documentation License	29
A.2.1 Preamble	29
A.2.2 Terms and Conditions for Copying, Distribution and Modification	29
A.2.3 How to use this License for your documents	35
Glossary	37
List of Figures	45
Index	47

Acknowledgements

As with most open source projects, this project would not have been possible without the valiant efforts and productive software for the *Free Software Foundation* and the *Linux Kernel Community*.

Sponsors

Funding for completion of the Linux Fast-STREAMS package was provided in part by:

- OpenSS7 Corporation

Additional funding for **The OpenSS7 Project** was provided by:

- OpenSS7 Corporation
- Lockheed Martin
- Performance Technologies
- Motorola
- HOB International
- Comverse
- Sonus Networks
- France Telecom
- SS8 Networks
- Nortel Networks
- Verisign

Contributors

The primary contributor to the OpenSS7 Linux Fast-STREAMS package is **Brian F. G. Bidulock**. The following is a list of significant contributors to **The OpenSS7 Project**:

- Per Berquist
- John Boyd
- Chuck Winters
- Peter Courtney
- Tom Chandler
- Gurol Ackman
- Kutluk Testicioglu
- Others

Acknowledgements

Preface

Document Information

Notice

This package is released and distributed under the *GNU General Public License* (see [Section A.1 \[GNU General Public License\], page 23](#)). Please note, however, that there are different licensing terms for the manual pages and some of the documentation (derived from X/Open publications and other sources). Consult the permission notices contained in the documentation for more information.

This document, is released under the *GNU Free Documentation License* (see [Section A.2 \[GNU Free Documentation License\], page 29](#)) with all sections invariant.

Abstract

This document provides *Frequently Asked Questions* for *Linux Fast-STREAMS*.

Objective

The objective of this document is to provide a guide for the *STREAMS* programmer when developing *STREAMS* modules, drivers and application programs for *Linux Fast-STREAMS*.

This guide provides information to developers on the use of the *STREAMS* mechanism at user and kernel levels.

STREAMS was incorporated in UNIX System V Release 3 to augment the character input/output (I/O) mechanism and to support development of communication services.

STREAMS provides developers with integral functions, a set of utility routines, and facilities that expedite software design and implementation.

Intent

The intent of this document is to act as an introductory guide to the *STREAMS* programmer. It is intended to be read alone and is not intended to replace or supplement the *Linux Fast-STREAMS* manual pages. For a reference for writing code, the manual pages (see *STREAMS(9)*) provide a better reference to the programmer. Although this describes the features of the *Linux Fast-STREAMS* package, **OpenSS7 Corporation** is under no obligation to provide any software, system or feature listed herein.

Audience

This document is intended for a highly technical audience. The reader should already be familiar with *Linux* kernel programming, the *Linux* filesystem, character devices, driver input and output, interrupts, software interrupt handling, scheduling, process contexts, multiprocessor locks, etc.

The guide is intended for network and systems programmers, who use the *STREAMS* mechanism at user and kernel levels for *Linux* and *UNIX* system communication services.

Preface

Readers of the guide are expected to possess prior knowledge of the *Linux* and *UNIX* system, programming, networking, and data communication.

Revisions

Take care that you are working with a current version of this document: you will not be notified of updates. To ensure that you are working with a current version, contact the [Author](#), or check [The OpenSS7 Project](#) website for a current version.

A current version of this document is normally distributed with the *Linux Fast-STREAMS* package.

Version Control

```
STREAMS_FAQ.texi,v
Revision 0.9.2.2  2005/11/20 22:20:19  brian
- still working up documentation

Revision 0.9.2.1  2005/11/14 11:20:03  brian
- working up manual

Revision 0.9.2.43 2005/11/14 04:43:55  brian
- updating manual

Revision 0.9.2.42 2005/11/13 23:04:01  brian
- starting cleanup of SPG

Revision 0.9.2.41 2005/10/07 09:34:00  brian
- more testing and corrections

Revision 0.9.2.40 2005/09/26 10:56:41  brian
- doc updates

Revision 0.9.2.39 2005/09/20 12:53:07  brian
- more doc updates, corrected QFULL handling

Revision 0.9.2.38 2005/09/18 07:38:35  brian
- more doc updates

Revision 0.9.2.37 2005/09/17 11:52:08  brian
- documentation updates

Revision 0.9.2.36 2005/09/17 08:20:57  brian
- more doc updates

Revision 0.9.2.35 2005/09/17 00:46:12  brian
- document updates

Revision 0.9.2.34 2005/09/16 03:06:02  brian
- added glossary

Revision 0.9.2.33 2005/09/15 13:02:52  brian
- added new graphics and updates
```

ISO 9000 Compliance

Only the T_EX, texinfo, or roff source for this document is controlled. An opaque (printed, postscript or portable document format) version of this document is an **UNCONTROLLED VERSION**.

Disclaimer

OpenSS7 Corporation disclaims all warranties with regard to this documentation including all implied warranties of merchantability, fitness for a particular purpose, non-infringement, or title; that the contents of the document are suitable for any purpose, or that the implementation of such contents will not infringe on any third party patents, copyrights, trademarks or other rights. In no event shall *OpenSS7 Corporation* be liable for any direct, indirect, special or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with any use of this document or the performance or implementation of the contents thereof.

OpenSS7 Corporation reserves the right to revise this software and documentation for any reason, including but not limited to, conformity with standards promulgated by various agencies, utilization of advances in the state of the technical arts, or the reflection of changes in the design of any techniques, or procedures embodied, described, or referred to herein. *OpenSS7 Corporation* is under no obligation to provide any feature listed herein.

U.S. Government Restricted Rights

If you are licensing this Software on behalf of the U.S. Government ("Government"), the following provisions apply to you. If the Software is supplied by the Department of Defense ("DoD"), it is classified as "Commercial Computer Software" under paragraph 252.227-7014 of the DoD Supplement to the Federal Acquisition Regulations ("DFARS") (or any successor regulations) and the Government is acquiring only the license rights granted herein (the license rights customarily provided to non-Government users). If the Software is supplied to any unit or agency of the Government other than DoD, it is classified as "Restricted Computer Software" and the Government's rights in the Software are defined in paragraph 52.227-19 of the Federal Acquisition Regulations ("FAR") (or any successor regulations) or, in the cases of NASA, in paragraph 18.52.227-86 of the NASA Supplement to the FAR (or any successor regulations).

1 FAQ

1.1 What is STREAMS?

STREAMS is a facility first presented in a paper by Dennis M. Ritchie in 1984,¹ originally implemented on 4.1BSD and later part of *Bell Laboratories Eighth Edition UNIX*, incorporated into *UNIX System V Release 3.0* and enhanced in *UNIX System V Release 4* and *UNIX System V Release 4.2*. *STREAMS* was used in *SVR4* for terminal input/output, pseudo-terminals, pipes, named pipes (FIFOs), interprocess communication and networking. Since its release in *System V Release 4*, *STREAMS* has been implemented across a wide range of *UNIX*, *UNIX*-like, and *UNIX*-based systems, making its implementation and use an *ipso facto* standard.

STREAMS is a facility that allows for a reconfigurable full duplex communications path, *Stream*, between a user process and a driver in the kernel. Kernel protocol modules can be pushed onto and popped from the *Stream* between the user process and driver. The *Stream* can be reconfigured in this way by a user process. The user process, neighbouring protocol modules and the driver communicate with each other using a message passing scheme closely related to *MOM (Message Oriented Middleware)*. This permits a loose coupling between protocol modules, drivers and user processes, allowing a third-party and loadable kernel module approach to be taken toward the provisioning of protocol modules on platforms supporting *STREAMS*.

On *UNIX System V Release 4.2*, *STREAMS* was used for terminal input-output, pipes, FIFOs (named pipes), and network communications. Modern *UNIX*, *UNIX*-like and *UNIX*-based systems providing *STREAMS* normally support some degree of network communications using *STREAMS*; however, many do not support *STREAMS*-based pipe and FIFOs² or terminal input-output.³

1.2 Does Linux have STREAMS?

No, **Linux** does not have *STREAMS* as part of the kernel (yet). That is rather peculiar, particularly since **Linux** normally follows *SVR4* first and *4BSD* second. (Otherwise, it would just be another BSD.)

Two open source *STREAMS* implementations are available for **Linux**: *Linux STREAMS (LiS)* and *Linux Fast-STREAMS*.

1.2.1 Linux STREAMS (LiS)

Linux STREAMS (LiS) is a *STREAMS* implementation for **Linux** originally released and maintained by **GCOM Inc.**, and recently rereleased under *GPL* by **The OpenSS7 Project**.

A number of attempts were made to move the *Linux STREAMS (LiS)* project into the **Linux** kernel, however, each attempt crashed and burned in a shower of flames from BSD advocates

¹ *A Stream Input-Output System*, AT&T Bell Laboratories Technical Journal 63, No. 8 Part 2 (October, 1984), pp. 1897-1910.

² For example, AIX.

³ For example, HP-UX

on LKML. Arguments against do not appear to be based on valid technical argument. This is discussed more in the next section (see [Section 1.3 \[Why STREAMS?\]](#), page 8).

Nevertheless, the *Linux STREAMS (LiS)* does not conform to **Linux** kernel coding standards and practices and could likely be rejected out of hand for mainline support on that basis. Also, *Linux STREAMS (LiS)* also has a number of problems with conformance and recent releases have bugs and races that make its production use questionable.⁴

Versions of *LiS* are available that supports 2.2, 2.4 and 2.6 **Linux** kernels, however, recent releases only support 2.4 and 2.6 kernels.

1.2.2 Linux Fast-STREAMS (LfS)

Linux Fast-STREAMS is a new implementation of *STREAMS* for **Linux** developed under the [OpenSS7 Project](#) and with development principally funded by [OpenSS7 Corporation](#). This is the implementation that corresponds to this manual.

Linux Fast-STREAMS was developed as a production replacement for *LiS* that was both maintainable and suitable for mainline adoption. The principal behind the implementation was that portability was secondary to performance, conformance, maintainability and suitability for mainline adoption.

1.2.3 STREAMS for FreeBSD

At one point an attempt was made to provide a *STREAMS* implementation for *FreeBSD*, however, it appears that it never matured. If there is sufficient interest, it would be possible to port *Linux Fast-STREAMS* to *FreeBSD* and other BSD variants.

1.2.4 OpenSTREAMS?

Perhaps we should have called *Linux Fast-STREAMS* **OpenSTREAMS**; however, as there already exists suitable *STREAMS* packages for all *UNIX* and *UNIX*-based systems (with the possible exception of BSD variants) and even other operating systems (many RTOS also support *STREAMS*) conformance, compatibility, maintainability and mainline adoption for **Linux** have taken a higher priority than portability.

1.3 Why STREAMS?

So, "Why *STREAMS*" you might ask.

The [OpenSS7 Project](#) selected *STREAMS* as the basis for its telecommunications protocol and networking stacks some years ago because of its ability to encourage reuse of protocol components and its flexibility in configuring protocol stacks. In the telecommunications industry, many protocols exist and tend to be collected into profiles that mix and match protocol layers. *STREAMS* has an innate ability to provide support for flexible configuration of protocol layers.

Also, telecommunications platforms must work with media streams as well as signalling streams. Because of its terminal subsystem background, *STREAMS* is far more suitable

⁴ One of the major shortcomings of *LiS* in the author's opinion is that, because the code was intended on being portable, it is rather obfuscated and difficult to maintain or repair.

when applied to media streams that consist of steady flows of fixed size (small) message blocks. Other mechanisms within the **Linux** kernel are either unsuitable to such an approach (e.g, 'skbuff's) or too application specific (e.g, *ALS*).

The subsections that follow delve into this question deeper.

1.3.1 STREAMS versus Sockets

A basic question that is sometimes asked is: "Why use *STREAMS* when you can just use **Linux**'s NET4 BSD Sockets instead?"

The answer to this question is that *STREAMS* provides capabilities for specialized protocols and streamed input/output requirements (such as media) that are not amenable to the sockets interface or queue mechanisms.

Two examples are SS7 (Signalling System Number 7) which is a specialized Telecommunications protocol used by switching equipment in the Public Switched Telephone Network; and transferring and manipulating voice channels associated with telephone call or other telecommunications services. These are the reasons why the [OpenSS7 Project](#) originally embarked on using *STREAMS*. You will find that a large number of SS7 stack vendors also deliver *UNIX* and even *RTOS* SS7 products on *STREAMS*.

Although the BSD Sockets framework was established to permit arbitrary protocols to be implemented within the framework, it is seldom that BSD Sockets is actually used in this fashion. One reason for this is the tight coupling (function call interface) between layers in BSD sockets. **Linux** has an even tighter coupling between protocol layers and removes entire layers from the BSD model.

The 4.2BSD version of *UNIX* introduced *sockets* [Leffler, McKusick, Karels, Quateman 1988]. The operating system provided an infrastructure in which network protocols could be implemented. It provided memory management facilities, a set of system calls for accessing network software, an object-oriented framework for the network protocols themselves, and a formalized device driver interface. The *sockets* mechanism was primarily used to implement the TCP/IP protocols for the ARPA Internet. The device driver interface made it possible for the operating system to support a wide range of network controllers. *sockets* are widely used for the implementation of TCP/IP on *UNIX* systems and have been ported to many implementations of *UNIX System V*. Although it is possible to implement other protocols within the *sockets* mechanism, it was not often done.

An alternative infrastructure for providing network protocols is *STREAMS*, originally designed by Dennis Ritchie [Ritchie 1984a] and first released in *UNIX System V Release 3.0*. *STREAMS* provides an environment in which communications protocols can be developed. It consists of a set of system calls, kernel functions and data structures. With this environment it is easier to write modular and reusable code. The code is also simpler because many of the support functions the programmer needs are provided by the *STREAMS* infrastructure.

Have you ever seen the *RTP* (*Real-Time Transport Protocol*, *RFC 1889*) implemented under a Socket? Why not? Is not the sockets interface so flexible as to permit such protocols to be implemented?

There are several reasons that *BSD Sockets* have not been used for other protocol development:

- Although *BSD Sockets* provides a framework for protocol development, it does not provide many utility functions for working with arbitrary protocols. Most of the utilities are DARPA ARPANET specific.⁵
- Protocol to protocol module interfaces are poorly standardized for the *BSD Sockets* system, whereas, protocol to protocol module service interfaces are well defined under OSI for *STREAMS*. (**Linux** discards the protocol to protocol interface anyway.)
- The *BSD Sockets* interface can easily be applied over *STREAMS* transport protocol modules; however, the reverse is not true: the *STREAMS* interface cannot easily be provided over the *BSD Sockets* protocol modules.⁶
- Support in the *BSD Sockets* model for dynamically loaded protocol (kernel) modules and administrative reconfiguration of protocols and interfaces for new protocols are poorly supported.⁷
- The *BSD Sockets* model has almost no support for banded or priority message queues within the model and no systemic approach to flow control.
- The **Linux** implementation of *BSD Sockets* discards much of the general purpose protocol framework, presumably in the pursuit of speed.

BSD sockets consists of a socket that interfaces with the user using file system and socket semantics, a protocol control block that represents the upper-most protocol, a socket to protocol interface, additional protocol control blocks representing lower protocol components, a protocol to protocol interface, a device interface abstraction, and a protocol to device interface.

Linux discards the protocol control block, socket to protocol interface, protocol to protocol interface, and protocol to device interface.

One of the major ramifications of **Linux** discarding the protocol to protocol interface is that it is very difficult to implement layered or tunnelled protocols in the **Linux** kernel.⁸ Layered protocols that run, say, over UDP, such as ‘**econet**’, must internally use the socket interface to a UDP datagram socket to layer the ‘**econet**’ protocol over UDP. Under *STREAMS* it is much easier to either push ‘**econet**’ as a module over the UDP transport provider stream, or to **I_LINK** transport provider streams under an ‘**econet**’ multiplexing driver.

⁵ This should not be suprising as the 4BSD releases were developed for DARPA.

⁶ A case in point is the *iBCS*. You will see in the *iBCS* that, although a basic XTI over Sockets implementation can be provided, none of the *STREAMS* facilities can be supported. In constrast the *STREAMS* INET driver that performs XTI over Sockets with in the *STREAMS* framework is easily implemented as a single device driver and provides both *iBCS* and *STREAMS* capabilities.

⁷ Or not supported in a standardized way. *STREAMS* use of standardized interfaces and facilities permits portability of a module between operating systems with ease.

⁸ That is, even more difficult than on a BSD system.

In commenting on the relative performance of *STREAMS* and *Sockets*, Mitchel Waite had this to say:

Sockets are like pipes with more power. They are bidirectional and may cross network or other machine boundaries. In addition, sockets allow limited control information as well as data.

Streams are more general still, with extensive control information passing capabilities.

On most *UNIX* systems, messages (if available) have the lowest overhead and highest bandwidth, with pipes following close behind. Because they support complex networking facilities, sockets are probably less efficient than streams, but because they rarely appear on the same machine as streams, the question is somewhat academic. They certainly have much lower bandwidth than pipes or messages.⁹

With *Linux Fast-STREAMS* it will be very possible to compare the performance of *STREAMS* in comparison to *Sockets*. It will also be possible to compare the performance of traditional **Linux** pipes and FIFOs with *STREAMS*-based pipes and FIFOs.¹⁰

1.3.2 STREAMS Benefits

STREAMS provides a flexible, portable, and reusable set of tools for development of **Linux** system communication services. *STREAMS* allows an easy creation of modules that offer standard data communications services and the ability to manipulate those modules on a *Stream*. From user level, modules can be dynamically selected and interconnected; kernel programming, assembly, and link editing are not required to create the interconnection.

STREAMS also greatly simplifies the user interface for languages that have complex input and output requirements.

1.3.3 Standardized Service Interfaces

STREAMS simplifies the creation of modules that present a service interface to any neighbouring application program, module, or device driver. A service interface is defined at the boundary between two neighbours. In *STREAMS*, a *service interface* is a set of messages and the rules that allow passage of these messages across the boundary. A module that implements a service interface will receive a message from a neighbour and respond with an appropriate action (for example, send back a request to retransmit) based on the specific message received and the preceding sequence of messages.

In general, any two modules can be connected anywhere in a *Stream*. However, rational sequences are generally constructed by connecting modules with compatible protocol service interfaces. For example, a module that implements an *SS7 MTP2 Signalling Link* protocol layer, as shown in [\[undefined\]](#), presents a protocol service

⁹ *UNIX Papers, for UNIX Developers and Power Users*, (Waite, 1987) pp. 358-359

¹⁰ In fact, such a performance comparison has been performed using *Linux Fast-STREAMS*. On the test system, a *STREAMS*-based pipe delivered 80% of the write/read throughput performance experienced by **Linux** native pipes.

interface at its input and output sides. In this case, other modules should only be connected to the input and output side if they have the compatible *SS7 Signalling Link* service interface.

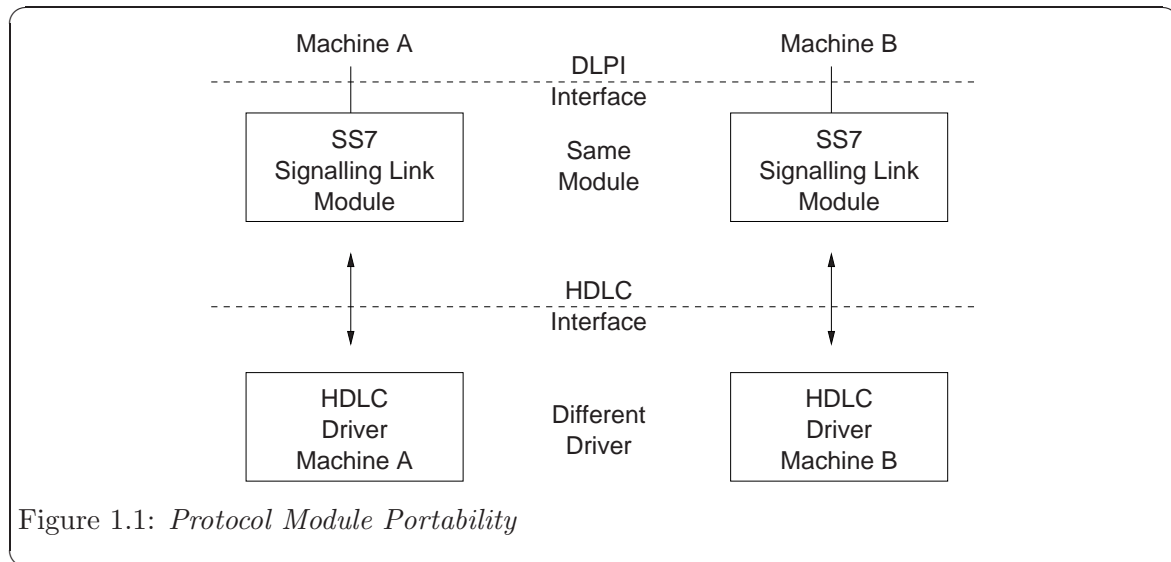
1.3.4 Manipulating Modules

STREAMS provides the ability to manipulate modules from user level, to interchange modules with common service interfaces, and to change the service interface to a *STREAMS* user process. These capabilities yield further benefits when implementing networking services and protocols, including:

- User level programs can be independent of underlying protocols and physical communications media.
- Network architectures and higher level protocols can be independent of underlying protocols, drivers, and physical communications media.
- Higher level services can be created by selecting and connecting lower level services and protocols.

1.3.4.1 Protocol Portability

Figure 1.1 shows how the same *SS7 Signalling Link* protocol module can be used with different drivers on different machines by implementing compatible service interfaces. The *SS7 Signalling Link* protocol module interfaces are *Data Link Provider Interface (DLPI)* and *High-Level Data Link Control (HDLC)*.



This arrangements permits the *SS7 Signalling Link* module to be implemented and conformance tested once, yet applied to many *HDLC* drivers. Each *HDLC* driver can be specific to the hardware that it supports without affecting reuse of the upper layer modules.

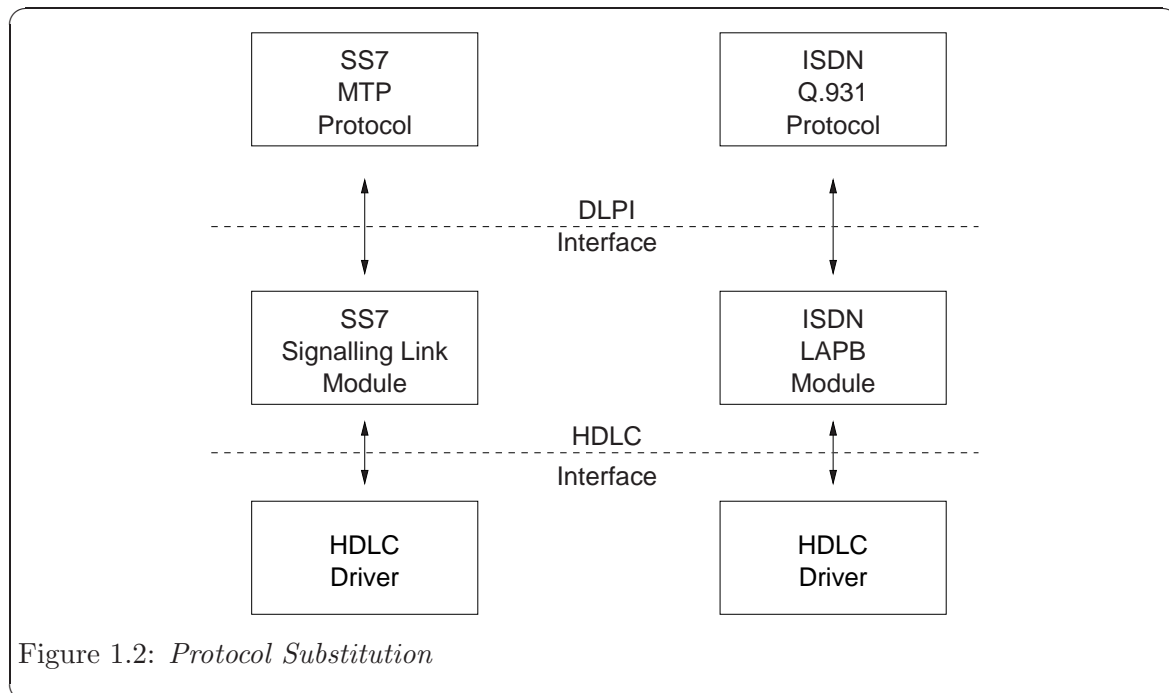
For the **OpenSS7** SS7 stacks, this permits the same *SS7 Signalling Link* module to be used and only a hardware specific *HDLC* driver to be written for each hardware interface device.

It also allows the same *SS7 Signalling Link* module to be used with non-*HDLC* links (such as UDP-based pseudo links).

1.3.4.2 Protocol Substitution

Alternate protocol modules (and device drivers) can be exchanged on the same machine if they are implemented to an equivalent service interface.

Figure 1.2 illustrates how *STREAMS* can substitute upper layer protocol modules to implement a different protocol stack over the same *HDLC* driver. As each module and driver support the same service interface at each level, it is conceivable that the resulting modules could be recombined to support, for example, *SS7 MTP* over an *ISDN LAPB* channel.¹¹

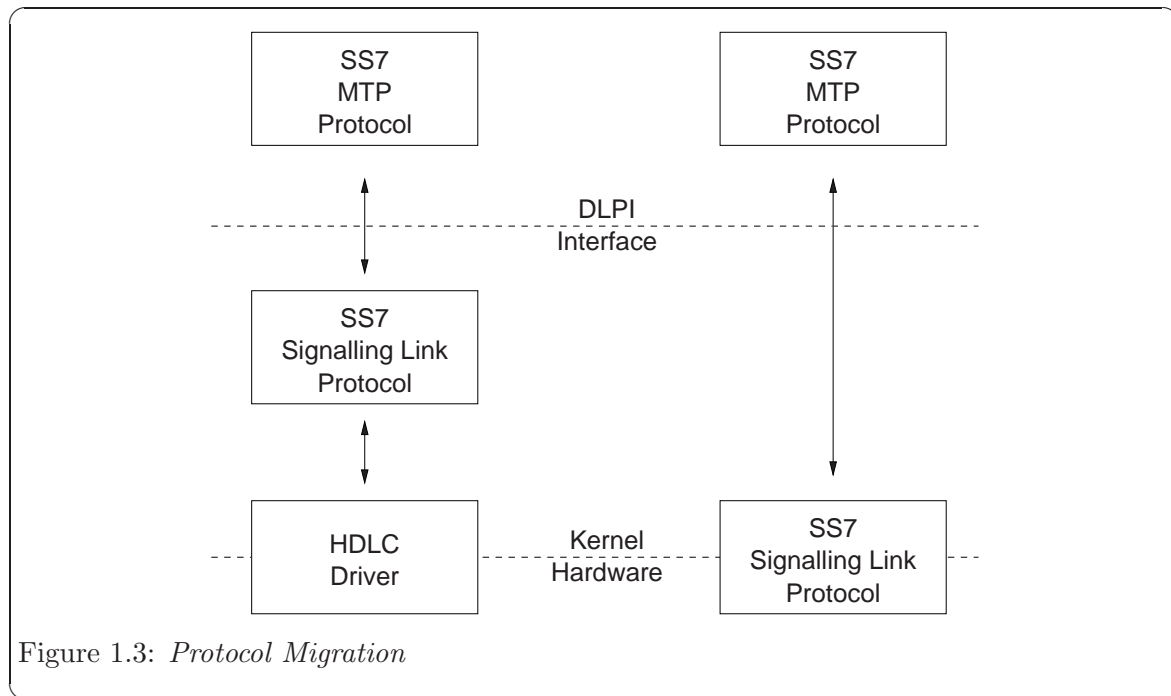


1.3.4.3 Protocol Migration

Figure 1.3 illustrates how *STREAMS* can move functions between kernel software and front end firmware. A common downstream service interface allows the transport protocol module to be independent of the number or type of modules below. The same transport module will connect without modification to either an *SS7 Signalling Link* module or *SS7 Signalling Link* driver that has the same service interface.

By shifting functions between software and firmware, you can produce cost-effective, functionally equivalent systems over a wide range of configurations. They can rapidly incorporate technological advances. The same transport protocol module can be used on a lower capacity machine, where economics may preclude the use of front-end hardware, and also on a larger scale system where a front-end is economically justified.

¹¹ SS7 MTP over ISDN LAPB was originally defined under ISDN as an E-Channel.

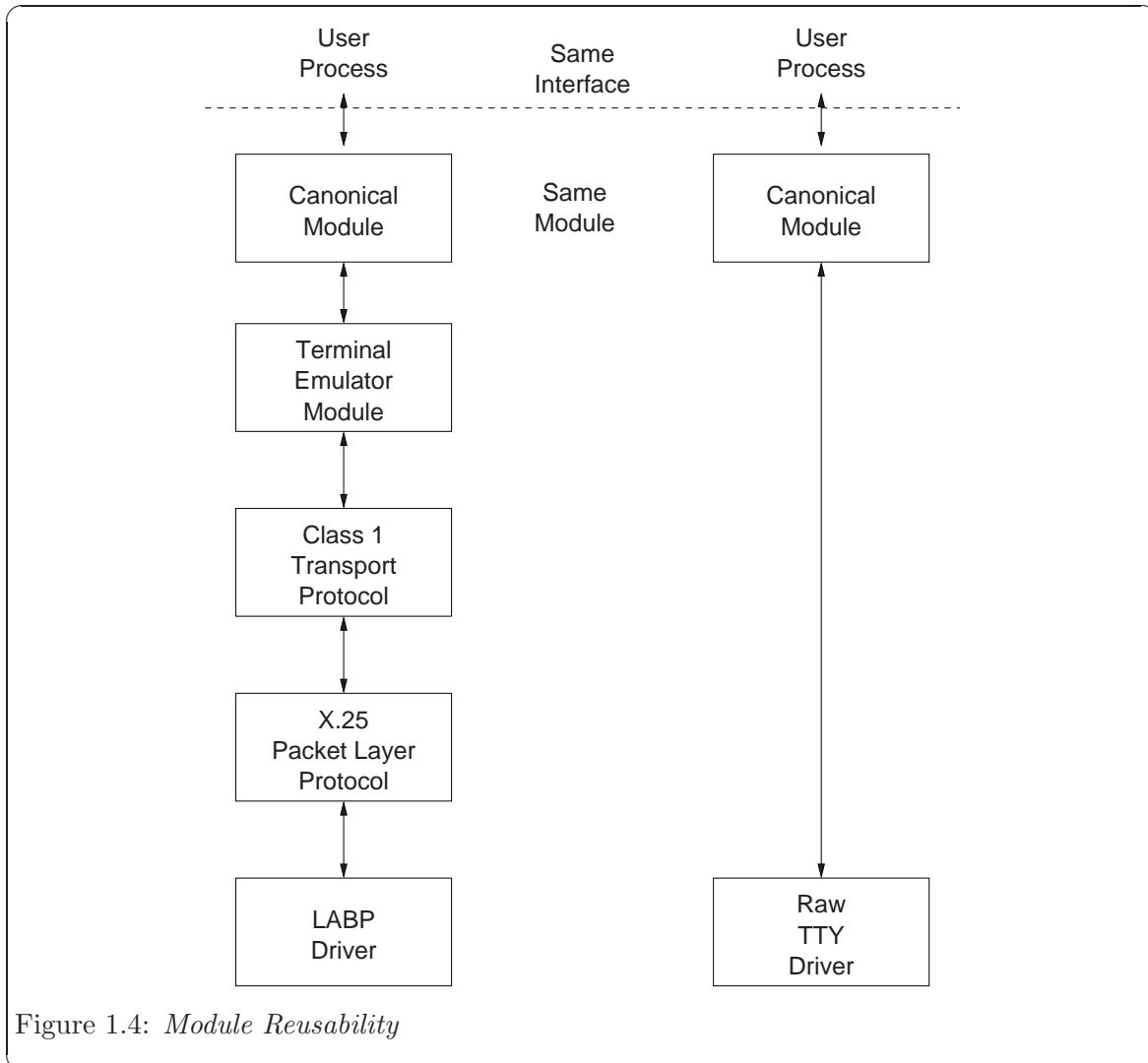


The **OpenSS7** *SS7 Protocol Stack* uses this capability of *STREAMS* to provide for hardware that contains varying levels of support for *SS7-HDLC* or *SS7 Signalling Data Terminal* functions (such as *AERM* and *SUERM* capabilities). The implementation goes one layer lower than illustrated in Figure 1.3, and provides an *HDLC* module that runs over a bare bearer channel (i.e, accepts a bit-stream at its lower service interface). This supports any channel interface hardware that can be placed in a raw mode making initial hardware driver support easier, following the left side of Figure 1.3. Hardware that supports *HDLC* functions, *Signalling Data Terminal* functions or even *Signalling Link* functions in firmware can use an integrated driver as illustrated on the right side of Figure 1.3.

Although, in practice, the performance cost of breaking a protocol layer into independent protocol modules within *STREAMS* is low, a firmware implementation as that shown on the right performs somewhat better, but typically at a higher monetary cost. *STREAMS* provides a wide range of scalable cost/performance options to the system designer in this regard.

1.3.4.4 Module Reusability

Figure 1.4 shows the same canonical module (for example, one that provide delete and kill processing on character strings) reused in two different *Streams*. This module would typically be implemented as a filter, with no downstream service interface. In both cases, a *tty* interface is presented to the *Stream*'s user process since the module is nearest the *Stream head*.



1.3.5 STREAMS Criticism

Following are some excerpts from Dennis M. Ritchie's original (1984) Bell Technical Journal paper on the stream I/O system. These excerpts are the limitations of the system as were perceived by Dennis M. Ritchie at the time. Strangely enough, although every limitation listed by Dennis was fixed even as early as *UNIX System V Release 3.0* and even in the *UNIX Eighth Edition*, some BSD advocates will use these limitations as a reason for not using *STREAMS* in BSD. Also, note that BSD'ers will also say that *STREAMS* was a *UNIX Eighth Edition* (Bell Laboratories Research version of UNIX) thing; however, Dennis' paper clearly states that the base system for the initial *Stream Input-Output System* was *4.1BSD*. Also note that *4.1BSD* already had *sockets* and that some of Ritchie's work was taken from *sockets*. It took until *4.2BSD* for *BBN* to add the *DARPA* protocol stack to *sockets*.

Perhaps it is not so surprising why *BSD*'ers hark back to Ritchie's original problem list for *STREAMS*: because it was at that point that *BSD* decided to not follow the *STREAMS* work too closely, except as regards IPC, and *UNIX* domain mechanisms. It is likely that

BSD would have used *STREAMS*, however, it was included in *UNIX System V Release 3.0* and this was the first release that *AT&T* was allowed to aggressively market under the terms of the *Modified Judgement*.

Although the new organization performs well, it has several peculiarities and limitations. Some of them seem inherent, some are fixable, and some are the subject of current work.

I/O control calls turn into messages that require answers before a result can be returned to the user. Sometimes the message ultimately goes to another user-level process that may reply tardily or never. The stream is write-locked until the reply returns, in order to eliminate the need to determine which process gets which reply. A timeout breaks the lock, so there is an unjustified error return if a reply is late, and a long lockup period if one is lost. The problem can be ameliorated by working harder on it, but it typifies the difficulties that turn up when direct calls are replaced by message-passing schemes.

This problem was never really fixed I suppose because most *STREAMS* specifications say that only one `ioctl` can be outstanding for a given stream. Nevertheless, an `ioctl` identifier was added to the `M_IOCTL` message that uniquely identifies the `ioctl`; but, a timer is still used. With the `I_STR` `ioctl`, however, the caller has control over the duration of the timeout. Strange, but this unfixed problem is the one that seldom gets raised as a reason for not using *STREAMS*.

Several oddities appear because time spent in server routines cannot be assigned to any particular user or process. It is impossible, for example, for devices to support privileged `ioctl` calls, because the device has no idea who generated the message. Accounting and scheduling becomes less accurate; a short census of several systems showed that between 4 and 8 per cent of non-idle CPU time was being spent in server routines. Finally, the anonymity for server processing most certainly makes it more difficult to measure the performance of the new I/O system.

This problem with privileged `ioctl` calls was easily fixed by adding the credentials of the caller to the `M_IOCTL` message. This limitation is also not mentioned by *STREAMS* critics.

In its current form the stream I/O system is purely data-driven. That is, data is presented by a user's `write` call, and passes through to the device; conversely, data appears unbidden from a device and passes to the top level, where it is picked up by `read` calls. Wherever possible flow control throttles down fast generators of data, but nowhere except at the consumer end of a stream is there knowledge for precisely how much data is desired. Consider a command to execute possibly interactive program on another machine connected to a stream. The simplest such common sets up the connection and invokes the remote program, and then copies characters from its own standard input to the stream, and from the stream to its standard output. The scheme is adequate in practise, but breaks when the user types more than the remote program expects. For example, if the remote program reads no input at all, any typed-ahead characters are sent to the remote system and lost. This demonstrates

a problem, but I know of no solution inside the stream I/O mechanism itself; other ideas will have to be applied.

Back-enabling of queues and the use of the `M_READ` message makes it possible for the consumer end of the stream to signal its desire for data downstream. Also, in the example that Ritchie gives here, the network protocol (TCP) is of no help either. This limitation is also not mentioned by *STREAMS* critics.

Streams are linear connections; by themselves, they support no notion of multiplexing, fan-in or fan-out. Except at the ends of a stream, each invocation of a module has a unique "next" and "previous" module. Two locally-important applications of streams testify to the importance of multiplexing: Blit terminal connections, where the multiplexing is done well, though at some performance costs, but a user program, and remote execution of commands over a network, where it is desired, but not now easy, to separate the standard output from error output. It seems likely that a general multiplexing mechanism could help in both cases, but again, I do not yet know how to design it.

This was, of course, solved, even in *UNIX System V Release 3.0* with the `I_LINK`, `I_PLINK`, `I_UNLINK` and `I_PUNLINK` *STREAMS* `ioctl` commands and the concept of a multiplexing pseudo-device driver. This fixed limitation you will see mentioned below.

The following excerpt shows how *BSD*'ers like to misinterpret the situation:

Original work on the flexible configuration of IPC processing modules was done at Bell Laboratories in *UNIX Eighth Edition* [Presotto & Richie, 1985]. This *stream I/O system* was based on *UNIX* character I/O system. It allowed a user process to open a raw terminal port and then to insert appropriate kernel-processing modules, such as one to do normal terminal line editing. Modules to process network protocols also could be inserted. Stacking a terminal-processing module on top of a network-processing module allowed flexible and efficient implementation of *network virtual terminals* within the kernel. A problem with streams modules, however, is that they are inherently linear in nature, and thus they do not adequately handle the fan-in and fan-out associated with multiplexing in datagram-based networks; such multiplexing is done in device drivers, below the modules proper. The Eighth Edition stream I/O system was adopted in System V, Release 3 as the *STREAMS* system.¹²

Well, the *UNIX Eighth Edition Stream Input-Output System* may have been included in *UNIX System V Release 3.0* as stated, however, Ritchie's *Stream Input-Output System* was implemented on *4.1BSD* for the October 1984 paper.

The design of the networking facilities for *4.2BSD* took a different approach, based on the *socket* interface and a flexible multilayer network architecture. This design allows a single system to support multiple sets of networking protocols with stream datagram, and other types of access. Protocol modules may deal with multiplexing of data from different connection onto a single transport medium, as well as with demultiplexing of data for different protocols

¹² *The Design and Implementation of the 4.4BSD Operating System*, McKusick, et. al., (Addison-Wesley, 1996) pp. 15-16

and connection received from each network device. The 4.4BSD release made small extensions to the socket interface to allow the implementation of the *ISO* networking protocols.¹³

1.3.6 STREAMS Realities

The realities of *STREAMS* are as follows:

- *STREAMS* is implemented on every major “Big Iron” *UNIX*.
Even Sun Microsystems chose to abandon *BSD Sockets* as an internal kernel networking implementation and moved to the *UNIX System V Release 4 STREAMS* subsystem instead.¹⁴ *STREAMS* is implemented (to list a few) in *AIX 5L Version 5.1 PSE*, *HP-UX 11.0i v2 STREAMS/UX*, *OSF/1 1.2/Digital UNIX*, *UnixWare 7.1.3 (OpenUnix 8)*, *Solaris 9/SunOS 5.9*, *Super-UX*, *UXP/V* and *MacOS OT*.
- *STREAMS* is implemented on many Real-Time Operating Systems (RTOS) based on *UNIX*.
Examples include *WindRiver*, *PSOS*, *VxWorks*, etc.
- *STREAMS* implementations are widely standardized on the *UNIX System V Release 4.2* specifications.
- *STREAMS* provides standardized (POSIX, OpenGroup, SVID) Transport Library Interface to communications networking suites.
- *STREAMS* has many portable implementations.

1.4 Why Fast?

Linux Fast-STREAMS includes the word *fast* in the name because of the original roots of the *Linux Fast-STREAMS* development effort. *Linux Fast-STREAMS* was originally developed by the **OpenSS7 Project** as a production replacement for the *Linux STREAMS (LiS)* package previously available from **GCOM**. One of the reasons for contemplating a replacement for *Linux STREAMS (LiS)* was the dismal performance provided by *Linux STREAMS (LiS)*. Other reasons included:

- Mainline Adoption instead of Portability
LiS attempts to maintain portability across a number of operating systems. The goals of portability and mainline adoption are usually at cross-purposes. *Linux Fast-STREAMS* proposes mainline adoption in contrast to portability. Many *STREAMS* implementations are available for other operating systems.
- Production Grade
LiS attempts to always provide debugging facilities¹⁵ and does not trust the driver or module writer. This leads to poor performance and in many cases the propagation of

¹³ *The Design and Implementation of the 4.4BSD Operating System*, McKusick, et. al., (Addison-Wesley, 1996) pp. 15-16

¹⁴ BSD'er will tell you that Sun Microsystems just made a bad decision.

¹⁵ In fact, these debugging facilities always point at the driver or module writer when a bug is encountered in *LiS* itself!

bugs to the field by failing to panic the kernel. *Linux Fast-STREAMS* aims at a production grade environment that implicitly trusts the driver or module while providing optional debugging facilities (both compile-time options as well as run-time options).

- SVR 4.2 MP Compatibility

LiS only provides *SVR 4* uniprocessor capabilities. *Linux Fast-STREAMS* provides *SVR 4.2 MP* capabilities.

- Portability

LiS forces ported drivers and modules from other implementations to use the *LiS* DDI and configuration mechanisms. *Linux Fast-STREAMS* provides compatibility functions for all major implementations of *STREAMS* as well as providing a rich DDI based on *SVR 4.2 MP*, *Solaris*, and other implementations.

- Bug Circumvention

LiS implementation approach has resulted in *LiS* masking its own internal bugs while pointing at the third-party module or driver (with log statements) whenever it fails to mask its own bugs. In the process *LiS* masks bugs even in the module or driver permitting defects to propagate to the field. *Linux Fast-STREAMS* uses industry standard programming by assertions approaches in its implementation avoiding these difficulties.

- New Implementation

Linux Fast-STREAMS is a completely new implementation of *STREAMS* independent from *LiS* and not derived from *LiS*. *LiS* is a collection of software available under various licenses with little or no commercial licensing alternatives available. *Linux Fast-STREAMS* is released in entirety under the *GPL*, but the implementation can be commercially licensed from [OpenSS7 Corporation](#).

- Scalable

LiS, particularly on 2.4 kernels, has difficulty with the scalability of major and minor device numbers. This is a limitation related to some **Linux** 2.4 kernels themselves, however, *Linux Fast-STREAMS* surmounts this difficulty by providing a *Shadow Special Filesystem (specfs)* that permits an almost unlimited number of major and minor devices on even the limited **Linux** 2.4 kernels.

- Soft Real Time Performance

LiS avoids use of high-performance **Linux**-specific facilities because of its aims at portability. *Linux Fast-STREAMS* being aimed at only **Linux** uses the highest-performance techniques available in the **Linux** kernel for implementation. This includes kernel memory caches and other techniques.

- Maintainability

Due to its initial objective to be portable to a number of environments, *LiS* has a large volume of source code that never gets executed in the **Linux** environment. Also, coding style does not seem to follow any mainstream practices. Aside from log messages and a few debugging flag facilities, *LiS* does not provide any debugging facilities.

Linux Fast-STREAMS provides a fully functional *STREAMS* logging facility and **strerr** and **strace** utilities. This permits the logging and tracing of live modules

and also permits diagnosis of a running (fielded) system. Additional support for panic recovery (without system failure or reboot) is also possible.

Linux Fast-STREAMS is also resplendent with the same maintenance features based on `autoconf`, `rpm` and `deb` that are used by all *OpenSS7* packages, including check scripts and installation conformance and verification test suites.

1.5 Why Linux?

Well, **Linux** is the only *SVR 4* based system that does not provide *STREAMS*, although *STREAMS* is an essential part of *SVR 4*. Without *STREAMS*, **Linux** is just another *BSD*, and perhaps a bad one.

1.6 Why Compatibility?

Linux Fast-STREAMS is designed and implemented to be compatible with as many *SVR 4.2 MP* based implementations of *STREAMS* as possible. This is done for several reasons:

1. *Porting legacy drivers to Linux:*

Many legacy *STREAMS* drivers have been written and developed for *SVR 4.2 MP* or *UNIX* systems based on *SVR 4.2 MP*. Remaining compatible with as many implementation as possible permits these legacy drivers to be easily ported from their native *UNIX* variant to the *Linux Fast-STREAMS* environment, thus quickly porting these legacy drivers to **Linux**.

2. *Leverage of knowledge base:*

Many developers are familiar one or another of the mainstream *UNIX* implementations of *SVR 4.2 MP STREAMS*. By remaining as compatible as possible with all these implementations of *STREAMS* permits knowledge and expertise in the *UNIX* variant of *STREAMS* to be transferred and applied to *Linux Fast-STREAMS* on **Linux**.

3. *Reverse portability:*

Because it is as compatible as possible with other *STREAMS* implementations, *STREAMS* drivers and modules developed on *Linux Fast-STREAMS* can easily be ported to other implementations if a set of compatibility and portability guidelines are followed. This allows *STREAMS* drivers and modules developed on the **Linux** operating system to be used on branded *UNIX* systems (or commercially available *RTOS* systems) with minimal porting and modification.

4. *Standardization:*

By being as compatible as possible with as many *STREAMS* implementations as possible, *Linux Fast-STREAMS* implements an *ipso facto* standard. Unfortunately, the *OpenGroup* and *POSIX* have been very lacking in the standardization of internal kernel interfaces such as *STREAMS*. Maximum compatibility moves close to providing a standard for such interfaces.

1.6.1 Intel Binary Compatibility Suite (iBCS)

The *Intel Binary Compatibility Suite* provides binary compatibility on the *Intel* architecture for systems conforming to *SVR 4.2*. *RedHat* has released an *iBCS* module for their distributions of **Linux** and the **Linux Kernel** for some time.

1.6.1.1 OpenGroup Specifications

OpenGroup and POSIX specifications have never directly addressed *STREAMS* implementation within the operating system. I suppose that this is primarily because 4BSD based systems have seldom included *STREAMS*. Perhaps it was due to some ideological upheaval from BSD advocates that did not want to see *STREAMS* become part of a standard.

Nevertheless, the *STREAMS* subsystem has been an optional part of the **OpenGroup** specifications for some time. It is my opinion that the **OpenGroup** has missed a rich opportunity for standardization of kernel level interfaces.

UNIX 03 Compliance

UNIX 03 compliance to *Open Group Extensions* requires that XTI/TLI networking support be provided. (See *XNS 5.2*). As the *iBCS* has proven, this does not require full *STREAMS* support, however, it is an easier thing to accomplish with *STREAMS* support. Even though the *XNS 5.2* specification does not describe *STREAMS*, the *SUSv3* does. The **OpenGroup** has never defined the internals of the *STREAMS* facility in their *CAE* specifications; however, they are described and the user-space facilities and system calls are completely defined and described.

UNIX 98 Compliance

UNIX 98 compliance to *X/Open Extensions* requires that XTI/TLI networking support be provided. (See *XNS 5*). As the *iBCS* has proven, this does not require full *STREAMS* support, however, it is an easier thing to accomplish with *STREAMS* support. Even though the *XNS 5* specification does not describe *STREAMS*, the *XSI 5* and *SUSv2* does. The **OpenGroup** has never defined the internals of the *STREAMS* facility in their *CAE* specifications; however, they are described and the user-space facilities and system calls are completely defined and described.

UNIX 95 Compliance

UNIX 95 compliance to *X/Open Extensions* requires that XTI/TLI networking support be provided. (See *XNS 4.2*). As the *iBCS* has proven, this does not require full *STREAMS* support, however, it is an easier thing to accomplish with *STREAMS* support. Even though the *XNS 4.2* specification does not describe *STREAMS*, the *XSI 4.2* and *SUS* does. The **OpenGroup** has never defined the internals of the *STREAMS* facility in their *CAE* specifications; however, they are described and the user-space facilities and system calls are completely defined and described.

1.6.2 Device Driver Interface (DDI)

Appendix A Copying

A.1 GNU General Public License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

A.1.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in

effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

A.1.2 Terms and Conditions for Copying, Distribution and Modification

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 - c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or

distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries

not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. **BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.**
13. **IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**

END OF TERMS AND CONDITIONS

A.1.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) 19yy name of author
```

```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'. This is free software, and you are welcome
to redistribute it under certain conditions; type 'show c'
for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program 'Gnomovision'
(which makes passes at compilers) written
by James Hacker.
```

```
signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

A.2 GNU Free Documentation License

GNU FREE DOCUMENTATION LICENSE

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

A.2.1 Preamble

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2.2 Terms and Conditions for Copying, Distribution and Modification

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related

matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers

that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgments" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement

made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgments”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s

Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

END OF TERMS AND CONDITIONS

A.2.3 How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being list their titles, with the
Front-Cover Texts being list, and with the Back-Cover Texts being list.
A copy of the license is included in the section entitled ‘‘GNU
Free Documentation License’’.
```

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being *list*”; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Glossary

<i>anchor</i>	A <i>STREAMS</i> locking mechanism that prevents the removal of <i>STREAMS</i> modules with the <code>I_POP</code> <code>ioctl</code> . Anchors are placed on <i>STREAMS</i> modules by adding the ‘[<code>anchor</code>]’ flag to <code>autopush(8)</code> configuration files or directly with the <code>I_ANCHOR</code> <code>ioctl</code> .
<i>autopush</i>	A <i>STREAMS</i> mechanism that enables a pre-specified list of modules to be pushed automatically onto a <i>Stream</i> when a <i>STREAMS</i> device is opened. This mechanism is used only for administrative purposes.
<i>back-enable</i>	To enable (by <i>STREAMS</i>) a preceding blocked queue’s <code>service</code> procedure when <i>STREAMS</i> determines that a succeeding queue has reached its low-water mark.
<i>blocked</i>	A queue’s <code>service</code> procedure that cannot be enabled due to flow control.
<i>clone device</i>	A <i>STREAMS</i> device that returns an unused major/minor device number when initially opened, rather than requiring the minor device to be specified by name in the <code>open</code> call.
<i>close procedure</i>	A routine that is called when a module is popped from a <i>Stream</i> or when a driver is closed. A pointer to this procedure is specified in the <code>qi_qopen</code> member of the <code>queue(9)</code> structure associated with the read side of the module’s queue pair.
<i>control Stream</i>	A <i>Stream</i> above a multiplexing driver used to establish lower multiplexer connections. Multiplexed <i>Stream</i> configurations are maintained through the controlling <i>Stream</i> to a multiplexing driver.
<i>Device Driver Interface</i>	An interface that facilitates driver portability across different <i>UNIX</i> system versions.
<i>device driver</i>	A <i>Stream</i> component whose principle functions are handling an associated physical device and transforming data and information between the external interface and the <i>Stream</i> .

<i>Driver Kernel Interface</i>	An interface between the <i>UNIX</i> system kernel and different types of drivers. It consists of a set of driver defined functions that are called by the kernel. These functions are entry points into a driver.
<i>downstream</i>	A direction of data flow going from the <i>Stream head</i> toward a driver. Also called the <i>write-side</i> and <i>output-side</i> .
<i>driver</i>	A module that forms the <i>Stream end</i> . It can be a device driver or a pseudo-device driver. It is a required component in <i>STREAMS</i> (except in <i>STREAMS</i> -based pipes and FIFOs), and is physically identical to a module. It typically handles data transfer between the kernel and a device and does little or no processing of data.
<i>enable</i>	A term used to describe scheduling of a queue's service procedure.
<i>FIFO</i>	<i>First In, First Out</i> . A term used in <i>STREAMS</i> for named pipes. This term is also used in queue scheduling.
<i>flow control</i>	A <i>STREAMS</i> mechanism that regulates the rate of message transfer within a <i>Stream</i> and from user space into a <i>Stream</i> .
<i>hardware emulation module</i>	A module required when the terminal line discipline is on a <i>Stream</i> but there is no terminal driver at the <i>Stream end</i> . This module recognizes all termio(7) ioctl s necessary to support terminal semantics specified by termio(9) and termios(9) .
<i>input side</i>	A direction of data flow going from a driver toward the <i>Stream head</i> . Also called <i>read-side</i> and <i>upstream</i> .
<i>line discipline</i>	A <i>STREAMS</i> module that performs termio(7) canonical and non-canonical processing. It shares some termio(7) processing with a driver in a <i>STREAMS</i> terminal subsystem.
<i>lower Stream</i>	A <i>Stream</i> connected beneath a multiplexing pseudo-device driver, by means of an I_LINK or I_PLINK ioctl . The far end of a lower <i>Stream</i> terminates at a device driver or another multiplexer driver.
<i>master driver</i>	A <i>STREAMS</i> -based device supported by the pseudo-terminal subsystem. It is the controlling part of the pseudo-terminal subsystem (also called ' ptm ').

<i>message</i>	One or more linked message blocks. A message is referenced by its first message block and its type is defined by the message type of that block.
<i>message block</i>	A triplet consisting of a data buffer and associated control structures, a <code>msgb(9)</code> structure, a <code>datab(9)</code> structure. It carries data or information, as identified by its message type, in a <i>Stream</i> .
<i>message queue</i>	A linked list of zero or more messages connected together.
<i>message type</i>	A enumerated set of values identifying the contents of a message.
<i>module</i>	A defined set of kernel-level routines and data structure used to process data, status, and control information on a <i>Stream</i> . It is an optional element, but there can be many modules in one <i>Stream</i> . It consists of a pair of queues (read queue and write queue), and it communicates to other components in a <i>Stream</i> by passing messages.
<i>multiplexer</i>	A <i>STREAMS</i> mechanism that allows message to be routed among multiple <i>Streams</i> in the kernel. A multiplexing configuration includes at least one multiplexing pseudo-device driver connected to one or more upper <i>Streams</i> and one or more lower <i>Streams</i> .
<i>named Stream</i>	A <i>Stream</i> , typically a pipe, with a name associated with it by way of a call to <code>fattach(3)</code> (that is, a <code>mount(2)</code> operation). This is different from a named pipe (FIFO) in two ways: a named pipe (FIFO) is unidirectional while a named <i>Stream</i> is bidirectional; a name <i>Stream</i> need not refer to a pipe, but can be another type of <i>Stream</i> .
<i>open routine</i>	A procedure in each <i>STREAMS</i> driver and module called by <i>STREAMS</i> on each <code>open</code> system call made on the <i>Stream</i> . A module's <code>open</code> procedure is also called when the module is pushed.
<i>packet mode</i>	A feature supported by the <i>STREAMS</i> -based pseudo-terminal subsystem. It is used to inform a process on the master side when state changes occur on the slave side of a pseudo-TTY. It is enabled by pushing a module called 'pckt' on the master side.

<i>persistent link</i>	A connection below a multiplexer that can exist without having an open controlling <i>Stream</i> associated with it.
<i>pipe</i>	See <i>STREAMS</i> -based pipe.
<i>pop</i>	A term used when a module that is immediately below the <i>Stream</i> head is removed.
<i>pseudo-device driver</i>	A software driver, not directly associated with a physical device, that performs functions internal to a <i>Stream</i> such as a multiplexer or <code>log(4)</code> driver.
<i>pseudo-terminal subsystem</i>	A user interface identical to a terminal subsystem except that there is a process in place of a hardware device. It consists of at least a master device, slave device, line discipline module, and hardware emulation module.
<i>push</i>	A term used when a module is inserted in a <i>Stream</i> immediately below the <i>Stream head</i> .
<i>pushable module</i>	A module put between the <i>Stream head</i> and driver. It performs intermediate transformations on messages flowing between the <i>Stream head</i> and driver. A driver is a non-pushable module.
<i>put procedure</i>	A routine in a module or driver associated with a queue that receives messages from the preceding queue. It is the single entry point into a queue from a preceding queue. It may perform processing on the message and will then generally either queue the message for subsequent processing by this queue's <code>service</code> procedure, or will pass the message to the <code>put</code> procedure of the following queue (using <code>putnext(9)</code>).
<i>queue</i>	A data structure that contains status information, a pointer to routines processing message, and pointers for administering a <i>Stream</i> . It typically contains pointer to <code>put</code> and <code>service</code> procedures, a message queue, and private data.
<i>read-side</i>	A direction of data flow going from a driver toward the <i>Stream head</i> . Also called <i>upstream</i> and <i>input-side</i> .
<i>read queue</i>	A message queue in a module or driver containing messages moving <i>upstream</i> . Associated with the <code>read(2)</code> system call and input from a driver.

<i>remote mode</i>	A feature available with the pseudo-terminal subsystem. It is used for applications that perform the canonical and echoing functions normally done by line discipline module and TTY driver. It enables applications on the master side to turn off the canonical processing.
<i>STREAMS Administrative Driver</i>	A <i>STREAMS</i> Administrative Driver that provides an interface to the <code>autopush(8)</code> mechanism.
<i>schedule</i>	To place a queue on the internal list of queues that will subsequently have their service procedure called by the <i>STREAMS</i> scheduler. <i>STREAMS</i> scheduling is independent of <i>Linux</i> process scheduling.
<i>service interface</i>	A set of primitives that define a service at the boundary between a service user and a service provider and the rules (typically represented by a state machine) for allowable sequences of the primitives across the boundary. At a <i>Stream</i> /user boundary, the primitives are typically contained in the control part of a message; within a <i>Stream</i> , in <code>M_PROTO</code> or <code>M_PCPROTO</code> message blocks.
<i>service procedure</i>	A module or driver routine associated with a queue that receives messages queue for it by the <code>put</code> procedure is called by the <i>STREAMS</i> scheduler. It may perform processing on the message and generally passes the message to the <code>put</code> procedure of the following queue.
<i>service provider</i>	An entity in a service interface that responds to request primitives from the service user with response and event primitives.
<i>service user</i>	An entity in a service interface that generates request primitives for the service provider and consumes response and event primitives.
<i>slave driver</i>	A <i>STREAMS</i> -based device supported by the pseudo-terminal subsystem. It is also called ‘ <code>pts</code> ’ and works with a line discipline module and hardware emulation module to provide an interface to a user process.
<i>standard pipe</i>	A mechanism for the unidirectional flow of data between two processes where data written by one process becomes data read by the other process.

<i>Stream</i>	A kernel level aggregate created by connecting <i>STREAMS</i> components, resulting from an application of the <i>STREAMS</i> mechanism. The primary components are the <i>Stream head</i> , the driver (or <i>Stream end</i>), and zero or more pushable modules between the <i>Stream head</i> and driver.
<i>STREAMS-based pipe</i>	A mechanism used for bidirectional data transfer implemented using <i>STREAMS</i> , and sharing the properties of <i>STREAMS</i> -based devices.
<i>Stream end</i>	A <i>Stream</i> component furthest from the user process that contains a driver.
<i>Stream head</i>	A <i>Stream</i> component closest to the user process. It provides the interface between the <i>Stream</i> and the user process.
<i>STREAMS</i>	A kernel mechanism that provides the framework for network services and data communication. It defines interface standards for character input/output within the kernel, and between the kernel and user level. The <i>STREAMS</i> mechanism includes integral functions, utility routines, kernel facilities, and a set of structures.
<i>TTY driver</i>	A <i>STREAMS</i> -based device used in a terminal subsystem.
<i>upper stream</i>	A <i>Stream</i> that terminates above a multiplexing driver. The beginning of an upper <i>Stream</i> originates at the <i>Stream head</i> or another multiplexing driver.
<i>upstream</i>	A direction of data flow going from a driver toward the <i>Stream head</i> . Also called <i>read-side</i> and <i>input side</i> .
<i>water mark</i>	A limit value used in flow control. Each queue has a high-water mark and a low-water mark. The high-water mark value indicates the upper limit related to the number of bytes contained on the queue. When the queued character reaches its high water mark, <i>STREAMS</i> causes another queue that attempts to send a message to this queue to become blocked. When the characters in this queue are reduced to the low-water mark value, the other queue is unblocked by <i>STREAMS</i> .

<i>write queue</i>	A message queue in a module or driver containing messages moving downstream. Associated with the <code>write(2)</code> system call and output from a user process.
<i>write-side</i>	A direction of data flow going from the <i>Stream head</i> toward a driver. Also called downstream and output side.

List of Figures

Figure 1.1: <i>Protocol Module Portability</i>	12
Figure 1.2: <i>Protocol Substitution</i>	13
Figure 1.3: <i>Protocol Migration</i>	14
Figure 1.4: <i>Module Reusability</i>	15

List of Figures

Index

A

AIX 18
 anchor 37
 ARPANET 9
 autopush 37
 autopush(8) 37, 41

B

back-enable 37
 benefits, STREAMS 11
 blocked 37
 BSD 20

C

clone device 37
 close procedure 37
 compliance, UNIX 03 21
 compliance, UNIX 95 21
 compliance, UNIX 98 21
 contributors 1
 control Stream 37
 credits 1

D

datab(9) 39
 DDI 21
 device driver 37
 Device Driver Interface 37
 Digital UNIX 18
 document abstract 3
 document audience 3
 document disclaimer 5
 document information 3
 document intent 3
 document notice 3
 document objective 3
 document revisions 4
 Does Linux have STREAMS? 7
 downstream 38
 driver 38
 Driver Kernel Interface 38

E

enable 38

F

FAQ 7

fattach(3) 39
 FIFO 38
 flow control 38
 FreeBSD 8

G

GCOM 18

H

hardware emulation module 38
 HP-UX 18

I

I_ANCHOR 37
 I_LINK 10, 17, 38
 I_PLINK 17, 38
 I_POP 37
 I_PUNLINK 17
 I_STR 16
 I_UNLINK 17
 iBCS 10
 input side 38
 Intel Binary Compatibility Suite 10
 Intel Binary Compatibility Suite (iBCS) 21

L

license, FDL 29
 license, GNU Free Documentation License 29
 license, GNU General Public License 23
 license, GPL 23
 licensing 3
 line discipline 38
 Linux Fast-STREAMS (Lfs) 8, 20
 Linux STREAMS (Lis) 7, 18
 log(4) 40
 lower Stream 38

M

M_IOCTL 16
 M_PCPROTO 41
 M_PROTO 41
 M_READ 17
 MacOS 18
 Manipulating Modules 12
 master driver 38
 message 39
 message block 39

Index

message queue 39
message type 39
module 39
mount(2) 39
msgb(9) 39
multiplexer 39

N

named Stream 39

O

open routine 39
OpenGroup 20
OpenGroup Specifications 21
OpenSS7 Project 9
OpenSTREAMS 8
OpenUnix 18
OSF/1 18

P

packet mode 39
persistent link 40
pipe 40
pop 40
POSIX 20
pseudo-device driver 40
pseudo-terminal subsystem 40
push 40
pushable module 40
put procedure 40
putnext(9) 40

Q

qi_qopen 37
queue 40
queue(9) 37

R

read queue 40
read(2) 40
read-side 40
remote mode 41

S

schedule 41
service interface 41
service procedure 41
service provider 41

service user 41
slave driver 41
sockets 9
Solaris 18
sponsors 1
standard pipe 41
Standardized Service Interfaces 11
Stream 42
Stream end 42
Stream head 42
STREAMS 42
STREAMS Administrative Driver 41
STREAMS Criticism 15
STREAMS Realities 18
STREAMS versus Sockets 9
STREAMS(9) 3
STREAMS, benefits 11
STREAMS-based pipe 42
SunOS 18
Super-UX 18
SVR 4 20
SVR 4.2 MP 19, 20
SVR 4.2 MP, *STREAMS* 20

T

TCP/IP 9
termio(7) 38
termio(9) 38
termios(9) 38
TTY driver 42

U

UNIX 03 compliance 21
UNIX 95 compliance 21
UNIX 98 compliance 21
UNIX System V Release 3.0 7
UNIX System V Release 4 7
UNIX System V Release 4.2 7
UnixWare 18
upper stream 42
upstream 42
UXP/V 18

W

water mark 42
Why Compatibility? 20
Why Fast? 18
Why Linux? 20
Why STREAMS? 8
write queue 43
write(2) 43
write-side 43