Linux STREAMS (LiS) Installation and Reference Manual

Version 2.18 Edition 4.rc2
Updated 2006-07-11
Package LiS-2.18.4.rc2

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

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

This is texinfo edition 4.rc2 of the Linux STREAMS (LiS) documentation, and is consistent with LiS 2.18. This manual was developed under the OpenSS7 Project.

Short Contents

Table of Contents

Acknowledgements

Sponsors

Funding for maintenance of the Linux STREAMS (LiS) package was provided in part by:

- OpenSS7 Corporation

Contributors

The current maintainer of the Linux STREAMS (LiS) package is Brian F. G. Bidulock.

Authors

Linux STREAMS, termed LiS, is an SVR4 compatible STREAMS executive which runs in the Linux Kernel as a loadable module. It is the product of a joint effort among the following authors.

Francisco J. Ballesteros `nemo@gsyc.escet.urjc.es`

John Boyd `jaboydjr@netwalk.com`

Denis Froschauer `Denis.Froschauer@hol.fr`

David Grothe `dave@gcom.com`

Ole Husgaard `sparre@login.dknet.dk`

Jrgen Magin `juergen.magin@octogon.de`

Graham Wheeler `gram@cdsec.com`

G Yeganjaiah `yegag@hclt.com`

Brian Bidulock `bidulock@openss7.org`

Brian Bidulock is the principal active maintainer of LiS, so please direct questions to him rather than the others.[1] Ole Husgaard has contributed to the kerneld support and installation procedures. Jrgen Magin contributed patches for Linux SPARC. G Yeganjaiah added interrupt routine support. John Boyd implemented fattach and STREAMS pipes and FIFOs. Brian Bidulock developed a complete set of manual pages for LiS, converted the build process to autoconf, wrapped the source RPMS, updated this manual for texinfo and currently maintains and the package.

See [Index of Authors], page 166, for a complete listing and cross-index of authors to sections of this document.

[1] David Grothe was the previous maintainer of the LiS-2.18 releases; however, David is no longer maintaining any version of LiS. Please do not direct maintenance requests at David.

1 Introduction

This manual documents the design, implementation, installation, operation and future development schedule of the Linux STREAMS (LiS) package.

1.1 Notice

This package is released and distributed under the *GNU General Public License* (see [GNU General Public License], page 133). 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 [Documentation License], page 149) with all sections invariant.

1.2 Overview

This manual documents the design, implementation, installation, operation and future development of the Linux STREAMS (LiS) package.

LiS is a software package that comprises an implementation of SVR4 compatible STREAMS for Linux. It takes the form of a loadable module for the Linux kernel. LiS installs in any directory on your system, not in the kernel source tree. (see Chapter 6 [Installation], page 83)

LiS-2.12 and beyond utilizes aggressive multi-tasking in multiple CPU SMP environments. For further information concerning this implementation, see Section 3.6.2 [LiS SMP Implementation], page 35.

WARNING: This autoconf/RPM release of Linux STREAMS is distributed under the terms of the GNU Public License (GPL) and *not* the GNU Lesser Public License (LGPL).

This means that you *cannot* link proprietary STREAMS drivers with LiS and load the entirety into the Linux kernel without violating license restrictions. OpenSS7 Corporation can remove this restriction for subscribers and sponsors of the OpenSS7 Project.

1.3 Organization of this Document

This document is organized (loosely) into several sections as follows:

Chapter 1 [Introduction], page 3.	This introduction
Chapter 2 [Objective], page 5.	Objective of the package
Chapter 3 [Reference], page 7.	Contents of the package
Chapter 4 [Conformance], page 63.	Conformance of the package
Chapter 5 [Releases], page 67.	Releases of the package
Chapter 6 [Installation], page 83.	Installation of the package
Chapter 7 [Troubleshooting], page 123.	Troubleshooting of the package

1.4 Conventions and Definitions

This manual uses *texinfo* typographic conventions.

2 Objective

3 Reference

3.1 Files

‘`specfs.o`’
‘`streams.o`’
‘`streams-aixcompat.o`’
‘`streams-hpuxcompat.o`’
‘`streams-liscompat.o`’
‘`streams-osfcompat.o`’
‘`streams-suncompat.o`’
‘`streams-svr4compat.o`’
‘`streams-uw7compat.o`’

3.2 Drivers

The LiS package comes with a number of STREAMS drivers and pushable modules in source code form. A number of these drivers and modules are small entities that are used in the testing of LiS. They are included so as to make it easy for any user to run the LiS tests for themselves.

Other drivers are used to implement STREAMS based pipes and FIFOs.

A driver in STREAMS has a major and minor device number associated with it and an entry in the ‘`/dev`’ directory. The driver is opened and closed just like any file.

The driver names used in this document are the declared names that appear in the LiS ‘`Config`’ file for the particular driver.

3.2.1 clone-drvr

Device Name

`/dev/clone_drvr`

Description

This driver is used to assist LiS in implementing the `"clone"` open function. It appears under its own name as '`/dev/clone_drvr`'. By convention, it is allocated the first major number of all the STREAMS drivers.

In order to implement clone opens, one creates a node in the '`/dev`' directory for a device whose major number is set to that of the clone driver, and whose minor number is the major number of the driver to which the clone open is to be directed.

The clone driver's open routine forwards the open call to the target driver, passing a unique flag that informs the driver that a clone open is being requested. The target driver then allocated a minor device number to uniquely associate with this instance of the open operation. The clone driver synthesizes a new major/minor `"device id"` to pass back to LiS. LiS recognizes the change of major/minor from the original open and takes steps to allocate control structures unique to this open.

The `"clone open"` operation is intended to make is easy to open one device from a pool of devices, such as pseudo ttys or logical connections. It saves application programs from having to scan a list of device mnemonics issuing trial opens until one is found that succeeds.

Note that the driver is named '`/dev/clone_drvr`' instead of the more traditional SVR4 '`/dev/clone`'. This is to avoid a conflict with another driver named '`/dev/clone`' on Linux systems.

Author

David Grothe `dave@gcom.com`

3.2.5 fifo

Device Name

```
/dev/fifo (clone device)
/dev/fifo.0
```

Description

The fifo pseudo-driver (which is internal to LiS) provides STREAMS-based fifos as single character special files, and STREAMS-based pipes as pairs of character special files which are interconnected (see pipe(3)).

STREAMS-based fifos differ from typical STREAMS-based character special files in that there are not separate stream head and driver queue pair within the STREAMS-based file. Instead, a fifo is created with only a single queue pair for the stream head. Moreover, in a typical driver queue pair, the write queue is not connected to a next queue. In a fifo, the write queue is directed to the read queue of the pair. A pipe comprises a pair of fifos, with the write queue of each pair directed to the read queue of the other. The two fifos comprising a pipe are referred to as peers, and each somewhat represents a driver to the other. As a degenerate case, a fifo is its own peer.

STREAMS modules may be pushed onto fifos and pipes, but should not expect a driver below them; instead, the SAMESTR() function should be used from the write queue of a pair to determine if the module is the lowest in the STREAMS-based file (this is called the midpoint). The structure of a fifo or pipe is preserved when modules are pushed (and popped); i.e., the write queue at the midpoint will always be directed at the read queue of the peer.

Input and output are handled at a fifo stream head as they would normally be handled at a stream head. In LiS, an fifo open() entry point exists to assign minor device numbers to new opens under the fifo major device number, and a close() entry point is used correspondingly to release them. These functions are kept in a streamtab data struc ture (as they would normally be for any STREAMS driver or module) which is private to the LiS implementation.

Application Usage

In the current Linux kernels, character special major numbers are limited to 16 bits, and major and minor device numbers to 8 bits each. This limits a system to 256 total major device numbers and 256 total minor devices per major device number. This is a rather severe limitation where mechanisms like fifos and pipes are concerned.

However, a driver may handle more than one major device number. The fifo pseudo-driver uses this to overcome this limitation, by supporting the automatic allocation and use of multiple major device numbers for fifos and pipes. Specifying more than 256 minor devices is done in the usual manner, i.e., by specifying the number of "units" in the appropriate '`Config`' file. Enough major device numbers will be allocated to cover the requested number of minor devices (if available, else an error will occur in strconf(8)). The number allocated will include one minor device per major number to be used as a fifo-specific clone minor device (specifically, minor number 0), which exhibits special behavior.

Normally, when cloning is done via the clone pseudo-driver, the clone major device number is used, along with the desired actual major number as the minor device num ber. When an open() is performed on such a device, the clone open() routine in turn calls the appropriate driver's open(), with the sflag parameter set to CLONEOPEN. The driver's open() is expected in this case to allocate an unused minor device number, and return it via an entirely new device number in the devp parameter. In this way, a driver can change the device number to be used for a STREAMS-based file. When minor device 0 for a specified for a fifo major device, the driver will also clone a new minor device number. However, LiS opens fifo devices differently; specifically, when an already-opened fifo-specific clone minor device is reopened, the new and subsequent opens will use the already-opened clone. Thus, using minor device 0 for a fifo when creating a file sys tem node will ensure that all concurrent opens of the associated path name will use the same STREAMS-based file; at the same time, opens of different file system nodes via different paths will open their respectively different STREAMS-based files. This is essentially how kernel-based fifos behave -applications and users of STREAMS-based fifos don't have to keep track of minor numbers to achieve this same behavior when it is desired.

It is in fact recommended that only two forms of file sys tem nodes be used for STREAMS-based fifos: the clone major number as major number with a fifo major number as minor number, to be used when every open of the associated path must clone a new fifo, and a fifo major number as major number with 0 as the minor number, to be used when new opens are to clone a new fifo but subsequent concurrent opens are to use the already opened fifo. These are represented by two device special file paths created when LiS is installed: '`/dev/fifo`' for the former, and '`/dev/fifo.0`' for the latter. It is recommended that these be used, possibly along with the equivalent of stat(2) to determine appropriate major device numbers for the clone and fifo pseudo-drivers, which are also determined when LiS is installed. It can be noted that pipes are actually created as instances of the former, after which the write queues are peer-connected.

The fifo pseudo-driver allocates minor devices in round-robin fashion; i.e., a list of available minor devices is kept, and once a minor number is finally closed, it is put at the end of this list. Thus, a fifo minor device which is opened and closed will not be immediately reused.

Warnings

Because STREAMS-based fifos and pipes are implemented as character special devices, they do not appear as pipe devices when examined with stat(2) or the equivalent (e.g., ls(1)); i.e. the S_IFIFO indication is not set in the mode - S_IFCHR is set instead, and the actual device number is indicated in the st_rdev field of the stat data structure.

Because of the potential use of multiple major numbers, applications should not depend on a fifo or pipe having a specific major device number, nor should an application depend on all fifos and pipes having the same major device number.

See Also

clone(9), connld(9), fifo(4), ls(1), pipe(3), pipemod(9), STREAMS(4), stat(2), strconf(8)

Author

John Boyd, protologos LLC. jaboydjr@netwalk.com

3.2.12 loop-around

Device Name

```
/dev/loop_clone (clone device)
/dev/loop.1
/dev/loop.2
```

Description

This driver is used by LiS and the strtst utility to assist in the regression testing of LiS. It connects two streams together in a manner similar to that of a pipe. Messages written into one stream can be read back from the other.

The driver can be operated as a clone device with the two streams being connected via ioctls. A number of ioctls exist that tailor the operation of the driver. The user codes these ioctls as type I_STR and passes a structure of type struct strioctl to the driver. The ic_cmd field of this structure is decoded according to the following table. the ic_dp and ic_len fields delimit an argument structure which is also passed to the driver. The argument structure differs for each type of ic_cmd.

ic_cmd value	Argument Structure	Description
LOOP_SET	IN: int	Argument is the minor device number of the loop device to use for the other end of the connection. If the loop-around device had been opened by a directed open, such as to '`/dev/loop.1`', then the minor device number is known from the device node. If it was opened via the '`/dev/loop_clone`' device then the minor device can be discovered via the LOOP_GET_DEV ioctl.
LOOP_PUTNXT	None	Set the driver into a mode in which it will perform a direct putnext call on the other stream rather than the default behavior of using the service queue to forward the message.
LOOP_MSGLVL	IN: int	Set to the number of messages to queue in the service queue before forwarding to the other stream. Zero means forward immediately.
LOOP_TIMR	IN: int	Set the number of "ticks" to hold messages before forwarding them to the other stream.
LOOP_MARK	IN: int	Set the MSGMARK flag for each of the next n messages before forwarding them to the other stream.

LOOP_GET_DEV	OUT: int	Return the minor device number of this stream. Useful for finding out the minor number of a clone device.
LOOP_BUFCALL	None	Use the bufcall mechanism to allocate a buffer for copying the next message.
LOOP_CONCAT	IN: int	Concatenate this many messages into a single message and then forward on the other stream. One concatenation resets this value to zero and the ioctl needs to be issued again to repeat the behavior.
LOOP_COPY	None	From this point on, copy messages rather than passing them through to the other stream.

Author

David Grothe `dave@gcom.com` plus others originally.

3.2.16 mini-mux

Device Name

```
/dev/mux_clone (clone device)
/dev/minimux.1
/dev/minimux.2
```

Description

This driver is used by LiS in its testing procedures. It is a small multiplexing driver that allows cascaded multiplexors to be built and torn down. The driver uses a pair of ioctls to establish connectivity between upper streams and lower streams. This allows control over how data flows through the multiplexor.

Both of these ioctls are coded as type I_STR and pass a structure of type struct strioctl to the driver. The ic_cmd field of this structure is decoded according to the following table. the ic_dp and ic_len fields delimit an argument structure which is also passed to the driver. The argument structure may differ for each type of ic_cmd.

ic_cmd value	Argument Structure	Description
MINIMUX_UP	IN: int	The argument is a muxid that was returned from an I_LINK ioctl. This ioctl causes the lower stream indicated by the muxid to be connected to this stream. This is unidirectional linkage and only affects the upstream flow of messages.
MINIMUX_DOWN	IN: int	The argument is a muxid that was returned from an I_LINK ioctl. This ioctl causes this stream to be connected to the lower stream indicated by the muxid. This is unidirectional linkage and only affects the downstream flow of messages.

Author

David Grothe dave@gcom.com

3.2.20 printk

Device Name

`/dev/printk`

Description

This driver accepts messages written to it and prints them from the kernel using the kernel's printk function. It is used by the LiS test software to keep messages from LiS and messages from the test program in sequence.

Author

David Grothe dave@gcom.com

3.2.24 sad

Device Name

`/dev/sad`

Description

The STREAMS Administrative Driver manages the autopush function of LiS. Using ioctls the system administrator can provide a list of modules that are to be automatically pushed onto a given device when that device is opened. The controls are specified via the strapush structure which is defined in <sys/sad.h>.

The ioctl used by the user is of the form:

ioctl(fd, command, arg)

Where fd is the file descriptor of the file that is open to the sad driver, command and arg are described in the following table.

Command	Argument	Description
SAD_SAP	struct strapush *	Set the list of autopushed modules according to the sap_cmd and other arguments contained within the strapush structure.
SAD_GAP	struct strapush *	Get the list of configured autopushed modules associated with the indicated major and minor device number. The sad driver fills in this structure with the names of the modules and the applicable range of minor device numbers.
SAD_VML	struct str_list *	Validates a list of pushable module names to verify that they are installed in LiS. The str_list structure is defined in the file '`<sys/stropts.h>`'.

The *strapush* structure used by the SAD_SAP and SAD_GAP ioctls contains the following fields.

`unsigned sap_cmd`
: This is the autopush command to be executed. The values are as follows.

  `SAP_ONE` Configure one minor device of the driver indicated by sap_major.

  `SAP_RANGE`
  Configure a range of minor devices of the driver indicated by sap_major. The range runs from sap_minor to sap_lastminor, inclusively.

  `SAP_ALL` Configure all minor devices of the driver indicated by sap_major.

  `SAP_CLEAR`
  Undo all autopush configuration for the driver indicated by sap_major.

`major_t sap_major`
: The major device number of the driver which is being configured for autopush.

`minor_t sap_minor`
: The minor device being configured, or the first of a range.

`minor_t sap_lastminor`
: The last minor device of a range to be configured.

`unsigned sap_npush`
: Number of modules to be pushed when the indicated device is opened.

`char sap_list[MAXAPUSH][FMNAMESZ+1]`
: List of module names to be pushed, or list of modules names returned to user.

The ioctl function call returns zero upon success or -1 on failure. Upon failure errno is set to the error number describing the failure, usually either EFAULT or EINVAL.

Note that the sad driver is a standard AT&T STREAMS function. More comprehensive documentation for this driver can be found in the [40]SVR4 Programmer's Guide: STREAMS.

Author

Ole Husgaard `sparre@login.dknet.dk`

3.3 Modules

'`streams-connld.o`'
: Connld module.

'`streams-pipemod.o`'
: Pipe module.

'`streams-sc.o`'
: STREAMS configuration module.

'`streams-sth.o`'
: Stream Head module.

The LiS package comes with a number of STREAMS drivers and pushable modules in source code form. A number of these drivers and modules are small entities that are used in the testing of LiS. They are included so as to make it easy for any user to run the LiS tests for themselves.

A pushable module in STREAMS is an entity that is added to an existing STREAMS file via the I_PUSH ioctl. These modules are known to LiS by mnemonic name, given as an argument to the I_PUSH ioctl. There are no major and minor device numbers or '`/dev`' entries associated with pushable modules.

3.3.1 connld

Module Name

connld

Description

The connld module provides a means to generate multiple unique STREAMS-based pipes from a single existing pipe end. connld may only be pushed (via the STREAMS I_PUSH ioctl) onto a STREAMS-based pipe. When first pushed, connld does nothing; on each subsequent open(2), connld will generate a unique STREAMS-based pipe. One end of each new pipe replaces the original pipe end from the perspective of the open call. The other end of each new pipe is sent, effectively as if by the I_SENDFD ioctl, to the other end of the original pipe, ostensibly to be received by a subsequent I_RECVFD ioctl operation.

Application Usage

The intent of connld is to provide a means to generate unique pipes which separately and independently connect client processes to a server process. The point of access for such clients is expected to be a path name known to all such clients and to which a pipe end may be connected (via fattach(3)) by the server process. The server establishes the original pipe, pushes connld onto the client end, and then listens via I_RECVFD for new connections on the server end. A client wishing to connect to the server will open(2) the path name representing the client end, and can determine via isastream(3) whether or not the server process is active and attached. If it is, the open() call returns one end of a unique new pipe that thus connects the client to the server.

Such a server is responsible both for accepting new connections via I_RECVFD on the original pipe, and for communicating with clients so connected via the received pipe ends. It would also be reasonable for such a server process to invalidate the point of access by calling fdetach(3) before terminating.

It should be noted that the poll(2) primitive may be used to indicate when an `M_PASSFP` representing a newly passed file is available on the original server pipe end. This is reflected by the POLLIN status setting in the events and revents fields of a pollfd structure. Moreover, any attempt to read an `M_PASSFP` message via the data-receiving primitives (i.e., read(2), getmsg(3), and getpmsg(3)) will fail with errno(3) returning an EBADMSG indication without discarding the message.

Even so, it should be reasonable to expect only `M_PASSFP` messages will be received on the original server pipe end, since it is not possible to carry on normal data traffic which has connld on one end, since connld does not support such traffic.

The use of connld can be made entirely free-standing by attaching well-known paths to both ends of the original pipe. The relevant capabilities are implemented in LiS so that the original creator of the pipe can close both ends after attaching paths to them, and the process of passing file descriptors can still be carried out via new open()'s as long as both ends remain attached.

See Also

fattach(3), fattach(8), fdetach(3), fifo(4), fifo(9),
pipe(3), STREAMS(4)

History

Unix System V Release 4 (SVR4)

Author

John Boyd, protologos LLC. jaboydjr@netwalk.com

3.3.8 pipemod

Module Name

pipemod

Description

The pipemod module has the relatively simple task of reversing the sense of the FLUSH flag bits in `M_FLUSH` messages sent in STREAMS-based fifos and pipes. This must happen at the midpoint of a fifo or pipe, so that FLUSHR becomes FLUSHW, and FLUSHW becomes FLUSHR. pipemod does this, and has no other function.

To be used appropriately, then, pipemod must be the first module pushed onto a pipe end or a fifo, but it is only necessary on one end of a pipe.

pipemod is not needed if flush handling need not be supported, or if its function is supported by other means.

See Also

fifo(9), pipe(3), fifo(4), STREAMS(4)

History

Unix System V Release 4 (SVR4)

Author

John Boyd, protologos LLC. `jaboydjr@netwalk.com`

3.3.14 relay, relay2

Module Name

relay relay2

Description

These are two names for the same module. All the module does is forward STREAMS messages along on the stream using putnext. These modules are used in the testing of LiS but are not otherwise useful. One could use the source code as a starting point for coding a pushable STREAMS module.

Author

David Grothe dave@gcom.com

3.4 Libraries

During the installation process of Linux STREAMS (LiS) a subroutine library is built and installed on your system. Three versions of the library are built and installed. They are as follows.

'`libLiS.a`'
: Interface routines to LiS in static library form.

'`libLiS.so`'
: Interface routines to LiS in dynamic library form.

'`libpLiS.so`'
: Like libLiS.so but omits the "pipe" system call.

These three libraries are copied to the directory '`/usr/lib`' when LiS is installed.

In addition, the utility program `ldconfig` is run during the LiS make install. This causes this library to be linked, or searched, ahead of the standard C library. This is necessary because the standard C library contains dummy routines for the STREAMS interface functions, or most of them in the best case. If these dummy routines preempt the LiS versions then STREAMS applications will always perceive error returns from such routines as **getmsg**(2) and **putmsg**(2).

3.4.1 Library Routines

The following routines are present in the libraries '`libLiS.a`' and '`libLiS.so`'. The library '`libpLiS.so`' omits the "pipe" routine.

The routines in these libraries are standard STREAMS interface routines. As such we do not offer detailed descriptions of the functions of these routines. Instead we refer the reader to the AT&T SVR4 STREAMS documentation.

```
int fattach(int fd, const char *path);
int fdetach(const char *path);
int getmsg(int fd, void *ctlptr, void *dataptr, int *flagsp);
int getpmsg(int fd, void *ctlptr, void *dataptr, int *bandp, int *flagsp);
int isastream(int fd);
int pipe(int *fd);
int poll(void *pollfds, long nfds, int timeout);
int putmsg(int fd, void *ctlptr, void *dataptr, int flags);
int putpmsg(int fd, void *ctlptr, void *dataptr, int *bandp, int *flagsp);
```

`int fattach(int fd, const char *path);`
: **fattach**(3)

`int fdetach(const char *path);`
: **detach**(3)

`int getmsg(int fd, void *ctlptr, void *dataptr, int *flagsp);`
: **getmsg**(2)

`int getpmsg(int fd, void *ctlptr, void *dataptr, int *bandp, int *flagsp);`
: **getpmsg**(2s)

`int isastream(int fd);`
: **isastream**(3)

`int pipe(int *fd);`
: **pipe**(2s)

`int poll(void *pollfds, long nfds, int timeout);`
: **poll**(2s)

`int putmsg(int fd, void *ctlptr, void *dataptr, int flags);`
: **putmsg**(2)

`int putpmsg(int fd, void *ctlptr, void *dataptr, int *bandp, int *flagsp);`
: **putpmsg**(2s)

These routines are all very small pieces of code. Most of them simply pass their parameters to LiS via a system call. The **fattach**(3) and fdetach **fdetach**(3) routines use ioctls to LiS if there is no system call available to call directly.

The poll **poll**(2s) routine simply executes the poll system call. It is present for backward compatibility to 2.0 kernels, in which LiS provided the poll system call.

The **pipe**(2s) routine has the same semantics as the standard C library routine. It uses STREAMS FIFOs to implement the pipe instead of the standard Linux pipes.

The '`libpLiS.so`' library, the one that preempts the standard C library, omits the STREAMS pipe routine so that standard Linux pipes are used unless the user explicitly links in '`libLiS`'.

3.4.2 Using the Library

To use one of the LiS libraries you can include the file '`<sys/stropts.h>`' in your program source code. On your compiler command line you can add the option '`-I/usr/include/LiS`' to include the version of '`stropts.h`' that is distributed with LiS, or omit the option to include the system standard header file. The two header files are believed to be compatible enough that it does not matter which one you include in your program.

When linking your program, or performing a final `cc` to build your executable, include one of the following options on your command line.

`/usr/lib/libLiS.a`
: Use '`libLiS.a`' (static, includes "pipe")

`-lLiS`	Use '`libLiS.so`' (dynamic, includes "pipe")
`-lpLiS`	Use '`libpLiS.so`' (dynamic, omits "pipe")

Omit any options

As of '`libc-2.2.1`' the LiS STREAMS interface routines will be used automatically via '`libpLiS.so`'.

3.5 Utilities

The Linux STREAMS (LiS) package contains some user level commands that are used to manage the package and assist the user with STREAMS functions.

These commands are installed in '`/usr/bin`' or '`/usr/sbin`'. They are referred to a "global commands."

A second group is built in the LiS installation directory and left there. This second group is oriented more towards testing of LiS than towards its operation. These commands remain undocumented since they are primarily intended for the use of the authors of the modules that they test.

These are the commands that are installed in '`/usr/bin`' or '`/usr/sbin`', and are thus globally accessible to any user with those directories in his/her path.

3.5.1 fattach

```
/usr/sbin/fattach [-v] [-m|-u|-M mode] [-p|STREAMS-path] path ...
/usr/sbin/fattach -?
```

Description

The fattach program provides a command-line interface to the underlying fattach(3) function. If the -p and/or the -c option is specified, a STREAMS-based pipe is created and its two ends are alternately attached to the path names given. In this mode of usage, at least two path names are required, but there need not be an even number of path names (i.e., the pipe ends need not be attached to the same number of paths).

If the -p and -c options are not specified, the first path name given must identify a STREAMS-based file. That file will be opened, and it will be attached to each of the path names subsequently specified (of which there must be at least one).

Options

-p
: Create a STREAMS-based pipe, to which to attach the subsequently specified path names. The first path will be attached to the first pipe end, the second to the second pipe end, the third to the first pipe end, etc., until the list of path names is exhausted.

  By default, the umask (see umask(2)) is also applied to each end of the pipe after attaching. (See fattach(3)).

-c
: Like -p (both may be given), but additionally pushes the connld module onto the first end of the pipe. This conveniently creates a free-standing pipe-serving pipe (see connld(9), and below).

-m
: Apply the mode of the last-specified path(s) to the attached STREAMS-based file(s) after attaching. (See fattach(3).

-u
: Apply the umask (see umask(2)) of the STREAMS-based file after attaching. (See fattach(3)). This is done by default when a pipe is created via -p.

-M *mode*
: Apply the given *mode* to the STREAMS-based file(s) after attaching. (See fattach(3)).

-v
: Operate in a "verbose" manner. This causes fattach to report its progress via message output to stdout or stderr.

-?
: Provide a usage summary.

Return Value

Upon successful completion, i.e., if all given path names are attached to, fattach returns 0. Upon failure, fattach returns 1. However, the failure of one more attachments does not otherwise affect those that succeed, and the user is responsible for detaching any that may have succeeded if that is the desired behavior in the event of any failures.

Application Usage

The -p and -c options provide a convenient means for creating free-standing mounted pipes. The openers of the paths attached via -p will share a single pipe, while the openers of the paths attached via -c will have access to a pipe-serving pipe. I.e., each open of the first end (e.g., the client end) will generate a new pipe, one end of which will be given to the opener, and the other end of which will be passed as if by the I_SENDFD ioctl to the path attached to the other end (e.g., the server end). Each opener of the server path could poll(2) for input, receive a new pipe end using the I_RECVFD ioctl, and then close the server path, therefter using the new pipe end to communicate with the corresponding opener of the client path (note that the sense of client and server will in fact depend on the application - users of the two paths need only be aware of whether or not an I_RECVFD ioctl must be performed).

See Also

connld(9), fattach(3), fdetach(3), fdetach(8), STREAMS(4), umask(2)

History

An fattach function has been provided for various STREAMS implementations based on SVR4 STREAMS. Not all of these have provided a corresponding utility program of this sort.

Author

John Boyd, protologos LLC `jaboydjr@netwalk.com`

3.5.2 fdetach

```
/usr/sbin/fdetach [-v] path ...
/usr/sbin/fdetach -a
/usr/sbin/fdetach -?
```

Description

The fdetach program provides a command-line interface to the underlying fdetach(3) function.

It is thus intended to provide a convenient means to dismantle so-called mounted STREAMS.

If the -a option is specified, all currently attached STREAMS-based files are detached. If the -a option is not specified, the path names given are taken to identify paths to which STREAMS-based files are currently attached. Those files will be detached from these paths.

Options

-a	Undo all attachments currently in effect.
-v	Operate in a "verbose" manner. This causes fdetach to report its progress via message output to stdout or stderr.
-?	Provide a usage summary.

Return Value

Upon successful completion, i.e., if all given path names identify mounted STREAMS and these are all successfully detached, fdetach returns 0. Upon failure, fdetach returns 1.

Note, however, that a failure indication does not mean that no action is taken; i.e., those detachments that succeed are not affected by those that fail.

Warnings

It should be noted that although the fdetach program implements the -a option, by passing "*" to the fdetach function, this is not at all equivalent to specifying "*" on the command line when executing the program. Normally, "*" specified on the command line will be converted by a shell into a list of all files in the current working directory. By contrast, the -a option causes the fdetach operation to operate not with respect to path names at all, but with respect to STREAMS devices currently active within the STREAMS subsystem. I.e., each active stream head is examined for attachments, and any attachments found are dismantled.

The intended use for the -a option is thus to undo all attachments, e.g., in preparation for unloading the STREAMS subsystem.

See Also

fdetach(3), fattach(8), STREAMS(4)

History

An fdetach function has been provided for various STREAMS implementations based on SVR4 STREAMS. Not all of these have provided a corresponding utility program of this sort.

Author

John Boyd, protologos LLC `jaboydjr@netwalk.com`

3.5.3 polltst

```
/usr/bin/polltst
```

Description

polltst is a simple test program for the poll system call. Using poll, it reads keystrokes from stdin, writes them to one end of the LiS loopback driver, reads them from the other end and then writes them back to stdout.

While performing this operation it configures stdin for `"no echo"` mode, so the appearance of `"echoed"` characters is evidence of the operation of poll involving both a STREAMS and a non-STREAMS file.

Author

David Grothe `dave@gcom.com`

3.5.4 streams

```
/usr/sbin/streams Options
```

Description

The streams program is used to perform several different management functions for the LiS package, including starting and stopping the LiS subsystem.

Options

'`start`'	Start the LiS subsystem. This amounts to performing the command "modprobe streams".
'`stop`'	Stop the LiS subsystem. This amounts to performing the command "modprobe -r streams".
'`status`'	Reports on the status of the LiS subsystem.
-c *Kbytes*	Print or set the maximum message memory usage for LiS. The value 0 (default) means unlimited.
-C *Kbytes*	Print or set the maximum total memory usage for LiS. The value 0 (default) means unlimited.
-d *mask*	Set the debug mask for LiS. See below for details.
-D *mask*	Set an additional debug mask for LiS. See below for details.
-s	Print STREAMS memory usage statistics.
-L	Print out lock contention statistics. Use debug bit `DEBUG_LOCK_CONTENTION` to enable the lock contention statistics gathering.
-m	Print STREAMS memory allocation to the system messages file (from kernel). This option should be used only for debugging and only when LiS is in a quiescent state. Unpredictable results can occur if this option is used while LiS memory allocations are changing dynamically.
-p	Print the LiS lock trace buffer to the system messages file (from kernel). Used in conjunction with the `DEBUG_SPL_TRACE` debug option.
-q	Print all STREAMS queues to the system messages file (from kernel). This option should be used only for debugging and only when LiS is in a quiescent state. Unpredictable results can occur if this option is used while LiS queue allocations are changing dynamically.
-S	Print out STREAMS queue-runner thread statistics.
-t	Print STREAMS timing statistics. Used inconjunction with the `DEBUG_MEAS_TIME` debug option.
-T	Print the LiS semaphore latency histogram. Use debug bit `DEBUG_SEMTIME` to enable the statistics collection.
-h	Print a command synopsis.
-H	Print a command synopsis including the debug mask mnemonics.

Debug Options

The value that is used with the -d option consists of the logical "or" of the following single bit options.

-d *Options*

`DEBUG_OPEN`	`0x00000001`
`DEBUG_CLOSE`	`0x00000002`
`DEBUG_READ`	`0x00000004`
`DEBUG_WRITE`	`0x00000008`
`DEBUG_IOCTL`	`0x00000010`
`DEBUG_PUTNEXT`	`0x00000020`
`DEBUG_STRRPUT`	`0x00000040`
`DEBUG_SIG`	`0x00000080`
`DEBUG_PUTMSG`	`0x00000100`
`DEBUG_GETMSG`	`0x00000200`
`DEBUG_POLL`	`0x00000400`
`DEBUG_LINK`	`0x00000800`
`DEBUG_MEAS_TIME`	`0x00001000`
`DEBUG_MEM_LEAK`	`0x00002000`
`DEBUG_FLUSH`	`0x00004000`
`DEBUG_FATTACH`	`0x00008000`
`DEBUG_SAFE`	`0x00010000`
`DEBUG_TRCE_MSG`	`0x00020000`
`DEBUG_CLEAN_MSG`	`0x00040000`
`DEBUG_SPL_TRACE`	`0x00080000`
`DEBUG_MP_ALLOC`	`0x00100000`
`DEBUG_MP_FREEMSG`	`0x00200000`
`DEBUG_MALLOC`	`0x00400000`
`DEBUG_MONITOR_MEM`	`0x00800000`
`DEBUG_DMP_QUEUE`	`0x01000000`
`DEBUG_DMP_MBLK`	`0x02000000`
`DEBUG_DMP_DBLK`	`0x04000000`
`DEBUG_DMP_STRHD`	`0x08000000`
`DEBUG_ADDRS`	`0x80000000`

-D *Options*

`DEBUG_SNDFD`	`0x00000001`
`DEBUG_CP`	`0x00000002`
`DEBUG_CACHE`	`0x00000004`
`DEBUG_LOCK_CONTENTION`	`0x00000008`
`DEBUG_REFCNTS`	`0x00000010`
`DEBUG_SEMTIME`	`0x00000020`

Most of these options are intuitive as to their operation from the mnemonics.

The `DEBUG_MEAS_TIME` option causes LiS to use a high precision timer to calculate the execution time of several operations within itself. These timings include the time spent in STREAMS drivers. Thus, under controlled circumstances this option can be used to time STREAMS driver code. It is used in conjunction with the -t option to print out the timing statistics.

The `DEBUG_SAFE` option causes LiS to carefully check for NULL pointers when performing message passing and queueing operations such as putq and putnext.

The `DEBUG_CLEAN_MSG` option causes LiS to clear message data buffers to zero when they are allocated. It is useful for tracking down driver problems relating to using uninitialized areas of messages.

The `DEBUG_SPL_TRACE` option causes LiS to maintain a trace table of all LiS locking operations. It is used in conjunction with the -p option to print out the lock trace table. The locking operations that are traced include calls on the LiS locking primitives from STREAMS drivers.

The options `DEBUG_DMP_QUEUE`, `DEBUG_DMP_MBLK` and `DEBUG_DMP_DBLK` control the verbosity of the printing out of LiS memory areas via the -m option. With these debug mask bits set, LiS will print out the contents of these structures as well as the headers indicating that such a structure was allocated.

The `DEBUG_ADDRS` option causes the -m option to print out the addresses of structures as well as their memory tags and/or contents.

The `DEBUG_MONITOR_MEM` option causes LiS to monitor the guard words surrounding allocated memory areas in an attempt to catch overwriting of these words in a timely fashion. This option comes at a fairly substantial CPU time penalty.

Author

David Grothe `dave@gcom.com`

3.5.5 strmakenodes

```
/usr/sbin/strmakenodes
```

Description

strmakenodes makes all of the '`/dev`' entries that are associated with LiS as a result of the LiS build process. All of the '`Config`' files that contributed to the LiS build are scanned for their `"node"` declarations. strmakenodes performs a mknod system call for each specified `"node"`.

This command must be run before LiS can operate correctly after it is installed. This command is run automatically as a result of the `"make install"` operation of LiS.

This command accepts the option `"-r"` to mean remove nodes instead of making them. The command is run with this option as a result of the `"make uninstall"` operation.

The source code for this command is generated automatically as a side-effect of running the strconf utility.

3.5.6 strtst

```
/usr/bin/strtst
```

Description

strtst is a test program which tests the core functionality of LiS. It is a user level program which uses the built-in drivers that are installed by default with LiS. It performs numerous STREAMS operations and checks the results for correctness. It prints out a voluminous log file whose output is routed to the "messages" file (kernel informational messages).

The output of strtst can be compared to earlier "reference" outputs to see if the behavior of LiS has changed as a result of modifications to the code.

Author

David Grothe dave@gcom.com

3.5.7 timetst

```
/usr/bin/timetst [Iterations]
```

Description

timetst peforms a timing test using the LiS loopback driver. It writes short messages downstream under several different LiS options and measures the round trip time for the messages. The Iterations parameter specified the number of iterations that timetst uses in its timing loop, the default being 100,000.

Author

David Grothe dave@gcom.com

3.6 Development

Linux STREAMS (LiS) provides for an interface between STREAMS drivers and the surrounding kernel environment. This interface has grown over time and is likely to expand in the future.

In the Linux kernel, much of the interface between drivers and other kernel modules and the core kernel services, such as memory allocation and synchronization primitives, is implemented in macros and inline functions declared in kernel header files. This technique was used (probably) out of considerations of efficiency (defined as execution speed) and a consideration that there were no version problems with such constructs because one could always recompile one's drivers in the context of the new kernel. The only "kernel primitives" compatibility that has been attempted from one kernel release to the next is source code compatibility.

The real world of paying customers is quite different. And, as it happens, the world of paying customers seems to impinge upon LiS considerably.

In this world, the customers do not want to rebuild the kernel. They don't want to build the kernel at all. They want to install a distribution with a binary kernel that was configured only at install time. They then want to install add-on binary packages, and they expect these packages to operate correctly with their kernel.

When these add-on packages consist of STREAMS based protocol drivers, LiS is usually the only piece of code that is recompiled from source upon installation into the customer's environment. The STREAMS drivers themselves are typically distributed in binary and linked in with LiS. The resulting module is then typically loaded using "modprobe" or some equivalent command.

In these circumstances it is highly desirable for LiS to "buffer" the interface between the STREAMS drivers and the kernel environment. This allows the STREAMS driver writers to deliver smaller binary packages to their customers and minimizes the number of different versions of those packages that must be maintained by the STREAMS driver writers. Ideally, LiS would be able to present a uniform DKI that would support one version of a user's STREAMS driver across all versions of the Linux kernel.

This ultimate goal is probably not achievable, but it is possible to insulate STREAMS drivers from the Linux kernel to a considerable extent. This is possible in part due to the implied DKI of a STREAMS driver. A STREAMS driver most likely will confine itself to the SVR4 types of DKI calls which have syntax and semantics that do not change over time. The main challenges come from the use of constructs, such as PCI configuration and interrupt service routines, that go outside the SVR4 DKI and must use services of the Linux kernel more-or-less directly.

In general, LiS attempts to replace inline functions and macros with actual subroutine calls to perform kernel operations. This allows the STREAMS driver to be compiled once with references to these routines, with the routines themselves being compiled in the context of the specific kernel version at package installation time. Thus, the STREAMS drivers do not have to be sensitive to differences in kernel versions.

3.6.1 Coding STREAMS Applications

This document is concerned with the include files and compilation techniques for STREAMS application programs. It is not intended to be a tutorial on the subject of writing STREAMS applications. Additional resources are available [17]here for reference material.

3.6.1.1 Header Files

In your STREAMS application program C language source, use the following line to include LiS header files.

```
    #include <sys/stropts.h>
```

This will include all of the STREAMS related information that you need for a user level program.

If your application program uses the poll system call then you need to include one or the other of the following lines depending upon the kernel version that the application is intended to run on. For kernel versions in the 2.0 group, use the following in order to include the poll.h from LiS.

```
    #include <sys/poll.h>
```

For kernel versions in the 2.2 group, use the following in order to include poll.h from the kernel's source tree.

```
    #include <linux/poll.h>
```

3.6.1.2 Compilation Options

When you compile your STREAMS application, put the following compiler option on the gcc (cc) command line for each C language file that contains any of the above include lines.

```
    -I/usr/include/LiS
```

3.6.1.3 Linking Options

When you perform the final link of your application using cc or gcc, add the following to the end of your list of files and libraries to be linked. This links in the system call interface routines for LiS.

```
    -lLiS
```

This library includes the STREAMS based version of the pipe system call. If you want to use the standard STREAMS library routines, such as getmsg and putmsg, but you want to use the standard Linux pipe system call, use the following instead.

```
    -lpLiS
```

3.6.1.4 Other STREAMS Resources

Click [18]here for a list of other locations that you can consult for general information concerning writing STREAMS applications.

3.6.2 LiS SMP Implementation

Beginning with LiS-2.12, LiS makes aggressive use of multiple CPUs in SMP kernels. It is useful for the STREAMS programmer to have some insight into this design in order to know whether, or which, locking techniques must be used in driver code.

3.6.2.1 CPU Scheduling

LiS starts up a kernel thread for each CPU on the system. In the output of a ps display each thread will show as a process with a name such as "LiS-2.12:0". This notation means that an LiS kernel thread is running and is bound to CPU 0 (":0").

LiS maintains a single global list of queues whose service procedures need to be run. A queue is place into this list by calling the function qenable, whether directly or indirectly. A given queue can only be in this list once. The read queue and write queue of a queue-pair are considered two different queues for scheduling purposes and both can be scheduled simultaneously.

At "certain points in time" LiS performs an operation that makes a decision concerning the manner in which service procedures are to be invoked via the list of scheduled queues. There are several factors which influence this decision.

- If the decision is being made just prior to LiS exiting back to user mode from a system call, if the LiS kernel thread for this CPU is inactive and if there are few enough entries in the list of scheduled queues then LiS calls the routine that processes queues directly without waking up any of its kernel threads.
- If the decision is being made from an interrupt routine then the queue processing routine is not to be called directly.
- If the decision is being made from a call on the routine qenable then the queue processing routine is not to be called directly.
- The number of queues in the scheduling list affects the decision making process.
- The number of LiS kernel threads running affects the decision making process.
- Whether or not LiS is executing a system call affects the decision making process. In this case LiS may defer any queue processing action until the system call is about to exit.

In the case that the queue processing routine does not get called directly, LiS needs to decide whether to wake up a kernel thread process or whether to defer queue processing until an LiS system call is about to exit.

LiS tries to enlist one CPU for every four queues that are scheduled. This number is based on considerations of CPU loading and average queue lengths from queueing theory. If the number of CPUs currently processing queues is not enough to meet this target then the scheduling process seeks to enlist more CPUs until the number of CPUs is sufficient to meet this target value. Of course, sometimes there are simply too may queues schedule for processing for the number of available CPUs. In that case, all available CPUs run their queue processing threads.

When making the decision as to whether or not to wake up a kernel thread, LiS gives precedence to the CPU that it is running on. If the scheduling algorithm is called from a point just prior to executing back to the user, and if the kernel thread for the active CPU is sleeping, then LiS will simply call the queue processing routine without waking up the kernel thread. This saves the wakeup and context switch overhead.

The routine that actually runs the queues removes one element at a time from the list of scheduled queues and calls the service procedure pointed to by the queue. The routine continues until the list of scheduled queues is empty. Thus, when a kernel thread is actively processing queues, and if the number of scheduled queues does not exceed the estimated capacity of the running threads, it is quite efficient to simply add a queue to the list and let the already running threads process them in due course.

3.6.2.2 Queue Locking

The queue_t structure in LiS contains a spin lock that is used by LiS to ensure that service procedures are not reentered for the same queue. This lock is not to be used by driver code.

When the LiS queue running routine removes a queue from the list of scheduled queues it acquires this lock prior to calling the service procedure.

LiS also acquires this lock when calling the put procedure associated with a queue. Thus, execution of the put and service procedure are excluded for the same queue.

In a multi-CPU environment, it can happen that one CPU is calling the put procedure while a second CPU is calling the service procedure for the same queue. In this case, one or the other spins until the first CPU finishes the operation and releases the spin lock.

When LiS is about to call the put procedure of a queue from the put or service procedure of a neighboring queue (because the driver called the putnext function), it continues to hold the lock for the calling queue while acquiring the lock for the destination queue. The locks are acquired sequentially as the chain of putnext calls traverse the stream. The locks are released in reverse order as the put procedures return. This has the effect of incrementally locking the entire stream as messages are passed from one module to another.

This behavior is only of interest when modules are I_PUSHed on top of a driver. Otherwise, it is just the stream head write queue and the driver write queue that need to be locked (or other pairwise combinations such as the driver read queue and stream head read queue, or queues involving multiplexors).

The lock that LiS uses has the effect of excluding multiple entries from different threads into the put or service procedure for a given queue. The other queue in the queue pair is unaffected by this locking. Therefore, if there are data structures shared between the read and write put and service procedures of a driver or module, it is up to the driver writer to protect these structures with spin locks.

3.6.2.3 Service Procedure Context

Due to the manner in which service procedures are called, sometimes from the LiS queue runner threads and sometimes from a "borrowed" system call, service procedures may or may not have some user context present when they run. Service procedures should always assume that there is no user context. Even in the cases where there is some user context, the identity of the user process is unpredictable.

LiS does, however, maintain a copy of the credentials of the process that opened the stream when it calls service procedures on the stream. LiS saves the user and group identifiers plus the capability masks (credentials) of the running process in the stream head structure at the time that the STREAMS file is opened. These identifiers are restored to the task structure before calling a service procedure on that stream.

When calling put procedures, however, no such identity restoration occurs. So the credentials in place when a driver or module put procedure is invoked are those of the invoking entity. Because the queue runner theads always begin driver entry with a call to the service procedure, entries into the put procedures of subsequent drivers will have the credentials of the stream whose service procedure was called in the first instance. When a driver's put procedure is entered from a system call the credentials will be that of the user process which issued the system call.

3.6.2.4 Scheduling Statistics

LiS gathers statistics on its queue scheduling algorithm. They can be printed out with the command streams -S. The output looks like the following.

```
N-CPUs N-Qrunners N-Running N-Requested
     2          2         0           0

CPU   Qsched-Cnts Qsched-ISR Svc-Q-Cnts Qrun-Cnts Active Thread-PID
  0     540752204  175753842  459587537 239611835      0        857
  1     540683832  175833424  459150290 239672683      0        858
```

The fields have the following meanings.

'N-CPUs' Number of CPUs on the system.

'N-Qrunners'
: Number of queue runner kernel threads. These are the processes that appear as LiS-2.12:0 in a ps display.

'N-Running'
: Number of qeuue runner threads that are currently active.

'N-Requested'
: The number of queues that are in the list of scheduled queues.

'CPU' The remainder of the statistics are kept on a per-CPU basis.

'Qsched-Cnts'
: This is the number times the routine (lis_setqsched) that decides whether or not to wake up a queue runner process or to directly process scheduled queues has been called. This routine is called whenever a queue is added to the scheduling list. The counter reflects which CPU made the call to the routine.

'Qsched-ISR'
: The number of times lis_setqsched was called from an interrupt routine and from which CPU.

'Svc-Q-Cnts'
: The number of callouts to service procedures on a per-CPU basis.

'`Qrun-Cnts`'
: The number of times the routine (queurun) that removes queues from the schedule list was called. This routine does not return until the queue scheduling list is empty. It can be running on multiple CPUs simultaneously. It is typically called from the queue runner threads but can also be called from an LiS system call either just prior to returning to the user or just prior to sleeping on some event such as the arrival of messages at the stream head.

'`Active`' Displays as 0 or 1 depending upon whether there is a queue runner thread running on the particular CPU at the time that the statistics were sampled.

'`Thread-PID`'
: The process id of the queue runner thread assaigned to eachCPU.

3.6.3 Operating System Interface Routines

In the file <sys/osif.h>, LiS provides insulation routines for a number of commonly used kernel functions. These functions are used with their Linux kernel names, but those names are redefined in <sys/osif.h> to be subroutine calls on functions that are actually defined in the file osif.c within LiS. The osif.c file is compiled at LiS installation time and is sensitive to kernel version information.

To use this interface, you include the header files that you would normally include to use the kernel functions, and then include <sys/osif.h> after all of the kernel include files. This allows for the redefinition of the names.

The kernel functions provided via <sys/osif.h> are as follows, grouped by type of function.

3.6.4 PCI BIOS Interface

These are routines that utilize or simulate the original PCI BIOS interface of the 2.0 series of kernels. The names of these routines are changed via defines. Use them as if the prototypes were as follows. You can use these routines on 2.2 kernels even though they represent the 2.0 style of inteface.

```
#if LINUX_VERSION_CODE < 0x020100       /* 2.0 kernel */
unsigned long pcibios_init(unsigned long memory_start,
                           unsigned long memory_end);
#else   /* 2.1 or 2.2 kernel * /
void pcibios_init(void) ;
#endif
int pcibios_find_class(unsigned int class_code, unsigned short index,
                       unsigned char *bus, unsigned char *dev_fn);
int pcibios_find_device(unsigned short vendor, unsigned short dev_id,
                        unsigned short index, unsigned char *bus,
                        unsigned char *dev_fn);
int pcibios_read_config_byte(unsigned char bus, unsigned char dev_fn,
                             unsigned char where, unsigned char *val);
int pcibios_read_config_word(unsigned char bus, unsigned char dev_fn,
                             unsigned char where, unsigned short *val);
int pcibios_read_config_dword(unsigned char bus, unsigned char dev_fn,
                              unsigned char where, unsigned int *val);
int pcibios_write_config_byte(unsigned char bus, unsigned char dev_fn,
                              unsigned char where, unsigned char val);
int pcibios_write_config_word(unsigned char bus, unsigned char dev_fn,
                              unsigned char where, unsigned short val);
```

```
int pcibios_write_config_dword(unsigned char bus, unsigned char dev_fn,
                               unsigned char where, unsigned int val);
const char *pcibios_strerror(int error) ;
```

3.6.5 PCI Interface

These routines constitute the PCI interface as implemented in the 2.2 series of kernels. Please note that these are filtered calls to the operating system and still depend directly upon the kernel structure "struct pci_dev". LiS provides a more abstract interface to PCI that does not depend upon the direct definition kernel structures. The [61]LiS PCI interface is to be preferred since it provides more insulation against changes in the kernel.

```
struct pci_dev *pci_find_device(unsigned int vendor, unsigned int device,
                                struct pci_dev *from);
struct pci_dev *pci_find_class(unsigned int class, struct pci_dev *from);
struct pci_dev *pci_find_slot(unsigned int bus, unsigned int devfn);
int pci_read_config_byte(struct pci_dev *dev, u8 where, u8 * val);
int pci_read_config_word(struct pci_dev *dev, u8 where, u16 * val);
int pci_read_config_dword(struct pci_dev *dev, u8 where, u32 * val);
int pci_write_config_byte(struct pci_dev *dev, u8 where, u8 val);
int pci_write_config_word(struct pci_dev *dev, u8 where, u16 val);
int pci_write_config_dword(struct pci_dev *dev, u8 where, u32 val);
void pci_set_master(struct pci_dev *dev);
```

3.6.6 IRQ Interface

These are the routines that are used to attach and detach interrupt service routines to hardware interrupts.

```
int request_irq(unsigned int irq,
void (*handler) ((int, void *, void *), unsigned long flags, const char *device,
                void *dev_id);
void free_irq(unsigned int irq, void *dev_id);
void disable_irq(unsigned int irq);
void enable_irq(unsigned int irq);
```

3.6.7 I/O Memory Mapping

These are the routines that are typically used to map PCI bus or physical addresses to CPU virtual addresses. LiS includes some backward compatibility here to older kernel versions.

```
void *ioremap_nocache(unsigned long offset, unsigned long size);
void iounmap(void *addr);
void *vremap(unsigned long offset, unsigned long size);
unsigned long virt_to_phys(volatile void *addr);
void *phys_to_virt(unsigned long addr);
```

3.6.8 I/O Port Access

These are the routines that allow a driver to register I/O ports.

```
int check_region(unsigned int from, unsigned int extent);
void request_region(unsigned int from, unsigned int extent, const char *name);
void release_region(unsigned int from, unsigned int extent);
```

3.6.9 Memory Allocation

These are the kernel routines that can be used to allocate memory. LiS also has a more insulated abstraction for kernel memory allocation. It is recommended that you use the [66]LiS memory allocator versions rather than the direct kernel versions.

```
void *kmalloc(size_t nbytes, int type);
void kfree(const void *ptr);
void *vmalloc(unsigned long size);
void vfree(void *ptr);
```

3.6.10 DMA Routines

These are the routines that are used to allocate a main-board old-style DMA channel for use by your driver. These are not much used anymore. See below for a more elaborate abstraction of DMA routines.

```
int request_dma(unsigned int dma_nr, const char *device_id);
void free_dma(unsigned int dma_nr);
```

3.6.11 Delay Routines

This is the routine that simply spins the CPU for a given number of microseconds. LiS also redefines the symbol "jiffies" to a subroutine call to help insulate STREAMS drivers from changes in the way the kernel keeps track of time. Remember, the redefinition is accomplished using C language defines, so the following declarations describe the effective usage of these symbols, not their literal definition.

```
void udelay(long micro_secs);
unsigned long jiffies;
```

3.6.12 Printing Routines

These are the most commonly used printf-like routines in the kernel. STREAMS drivers would be more portable if they used the cmn_err routine instead of printk.

```
int printk(const char *fmt, ...);
int sprintf(char *bfr, const char *fmt, ...);
int vsprintf(char *bfr, const char *fmt, va_list args);
```

3.6.13 Timer Routines

These are the the routines that start and stop kernel timers. STREAMS drivers would be more portable if they used the standard "[71]timeout" routine.

```
void add_timer(struct timer_list *timer);
int del_timer(struct timer_list *timer);
```

The following routine converts time in micro seconds to system "ticks". The "ticks" value is suitable for use with the timeout routine. Note that if the micro_sec parameter is less than the number of micro seconds in a system tick then the routine returns zero.

```
unsigned lis_usectohz(unsigned micro_sec);
```

The following routine is an LiS abstraction of the C library routine gettimeofday. Note the absence of the time zone parameter.

```
void lis_gettimeofday(struct timeval *tv);
```

The following two kernel routines are called via the LiS osif.c code.

```
void do_gettimeofday(struct timeval *tp);
void do_settimeofday(struct timeval *tp);
```

3.6.14 Sleep and Wakeup Routines

These are the kernel routines for sleeping using wait queues. STREAMS drivers should not be using these since only "open" and "close" routines are allowed to sleep, and for those cases, [73]LiS semaphores would provide better insulation from the kernel. STREAMS "put" and "service" routines should use [74]LiS spin locks for mutual exclusion.

```
void sleep_on(OSIF_WAIT_Q_ARG);
void interruptible_sleep_on(OSIF_WAIT_Q_ARG);
void wake_up(OSIF_WAIT_Q_ARG);
void wake_up_interruptible(OSIF_WAIT_Q_ARG);
```

3.6.15 Thread Creation

A STREAMS driver in LiS can create kernel threads if it so chooses. The following routine simplifies this task. It consolidates all of the kernel manipulations involved with the creation of a kernel thread into one place, thus removing references to these kernel functions from STREAMS driver code.

3.6.15.1 Prototype

```
pid_t lis_thread_start(int (*fcn) (void *), void *arg, const char *name);
int lis_thread_stop(pid_t pid);
```

Arguments

fcn

The function that is to be used as the entry point for the thread.

arg

The argument passed to the function.

name

An ASCII name associated with the thread. This name should be less than 16 characters in length. It will be the name of the thread that displays in a ps listing.

3.6.15.2 Operation

lis_thread_start creates a new thread, performs some operations prior to entering the fcn, and then calls fcn which acts as the "main" routine for the thread. The arg parameter is passed to fcn.

Before fcn is entered, the newly created thread will have shed all user space files and mapped memory. Thus, it is a kernel-only thread.

All signals are still enabled. Note that when the kernel goes down for reboot all processes are first sent a SIGTERM. Once those have been processed, all processes are then sent a SIGKILL. It is the implementor's choice which of these it pays attention to in order to exit prior to a reboot.

The fcn is entered with the "big kernel lock" NOT held, just as it would be for calling the "kernel_thread" function directly. On 2.2 kernels, the fcn should get this lock so that it can utilize kernel services safely.

The user's fcn returns a value when it exits and that value is returned to the kernel. It is not clear that anything actually pays any attention to this returned value. It particular, it is not visible to the thread that started the new thread.

lis_thread_start itself returns the process id of the new thread, or a negative error number. This value can be used to kill the thread.

lis_thread_stop kills a thread started by lis_thread_start. It returns 0 for success or a negative error number for failure.

3.6.16 Major/Minor Device Numbering

Please note that LiS-2.17 changed the internal representation of the major and minor device numbers within the 32 bit `dev_t` structure. The following documents the new format and usage conventions.

In STREAMS the dev_t structure is used to combine a major device number and a minor device number into a single integer length quantity. The Linux kernel restricts these numbers to the range 0 to 255 (8-bit values).

LiS provides a typedef for `dev_t` that results in an unsigned integer quantity. Internal to LiS the high order 12 bits are used for major device number and the low order 20 bits are used for minor device number.

STREAMS drivers include the file '`<sys/stream.h>`' that causes the view of `dev_t` to change from the kernel's 8/8 view to the LiS 12/20 view. To ensure proper operation, STREAMS drivers should use the following functions to manipulate `dev_t` variables. These functions are SVR4 compatible.

`int getmajor(dev_t dev);`
: Extracts the major device number

`int getminor(dev_t dev);`
: Extracts the minor device number

`dev_t makedevice(int maj, int min);`
: Combines a major and minor device number into a dev_t

`int DEV_SAME(dev_t d1, dev_t d2);`
: True if the two devices are the same

`int DEV_TO_INT(dev_t dev);`
: Converts dev_t to an int

The sample drivers that come with LiS now use these constructs to manipulate device structures and can serve as examples for their usage.

Within a STREAMS driver it is occasionally necessary to make a `dev_t` value in the external 8/8 format. This is required, for example, when a driver is using the `lis_mknod()` function to create a device node at driver initialization time. LiS provides the function `UMKDEV(major, minor)` for this purpose.

3.6.17 LiS Memory Allocation

LiS provides for several different styles of memory allocation, all of them insulated from the Linux kernel. These routines allow your driver to allocate memory in several different ways while still maintaining compatibility with different versions of the Linux kernel, with no driver recompilation required.

To use the LiS memory allocation routines include the file <sys/lismem.h> in your STREAMS driver source code.

3.6.18 LiS malloc and free Equivalents

The first group of memory allocation routines are the routines that play the role of "malloc" and "free." These routines keep a master linked list of all allocated memory areas. This list can be printed out via an ioctl to LiS. Each allocated area is tagged with the file name and line number of the code that caused it to be allocated. Each area contains a guard word at the front and back to enable the allocator to detect "off by one" accesses outside the allocated area.

LiS uses this allocator internally for allocating queues, messages and other internal data structures. This would be the allocator of choice for STREAMS drivers to use to allocate instance structures.

Memory allocated in this manner is ultimately allocated by the kernel routine "kmalloc". As such, it is not guaranteed to be DMA-able (in the old style), or to occupy physically contiguous memory locations. [78]See below for routines that can be used to allocate these types of memory areas.

The routines are as follows:

```
void *ALLOC(int nbytes);
void *ALLOCF(int nbytes, char *tag);
void FREE(void *ptr);
```

The ALLOC and FREE routines are analogous to "malloc" and "free". The ALLOCF routine includes a character string which is prepended to the file name stored as the location from which the allocation occurred. It can serve as a tag for the type of memory being allocated.

Usage examples:

```
ptr = ALLOC(456);
FREE(ptr);
ptr = ALLOCF(578, "Instance: ");
FREE(ptr);
```

3.6.19 LiS Kernel Memory Allocators

These routines use the LiS malloc/free internal routines to allow for more flexibility in the options used when calling the kernel allocator. These routines all lead to a call on "kmalloc" with appropriate options. It is worth noting that the numerical value of the constants used in calling the kernel's "kmalloc" routine changed between the 2.2 and 2.4 versions of the kernel. Thus, drivers which called the kernel's "kmalloc" directly have to be recompiled to run in a 2.4 kernel. STREAMS drivers using the memory allocation interface defined here could run without modification and without a recompilation on both kernels, assuming that the drivers otherwise did not use any direct kernel functions.

```
void *lis_alloc_atomic(int nbytes);
void *lis_alloc_kernel(int nbytes);
void *lis_alloc_dma(int nbytes);
void *lis_free_mem(void *mem_area);
```

These routines pass the allocation options GFP_ATOMIC, GFP_KERNEL, and GFP_-DMA, respectively, to "kmalloc" when allocating the memory. LiS takes care of passing the proper values to the kernel routine so that driver code can remain portable.

The routine lis_free_mem returns a NULL pointer for the convenience of the caller.

The kernel's kmalloc is restricted as to the number of bytes that it will allocate. The LiS routines do not have this restriction. If the number of requested bytes is larger than 16K the

LiS allocation routines will call the page allocator to allocate the memory. The lis_free_mem routine knows whether to free pages or to use the kernel's kfree routine.

Usage Examples:

```
ptr = lis_alloc_kernel(sizeof(structure));
ptr = lis_free_mem(ptr);        /* returns NULL pointer */
```

3.6.20 LiS Page Allocator

These routines allow a STREAMS driver to allocate memory directly from the kernel's page allocator. Memory allocated in this manner occupies physically contiguous locations and is suitable for use with bus master DMA PCI devices.

Unlike the kernel's page allocator, the size that is specified when calling the LiS page allocator is in bytes, not "order", or other encoding of page size. LiS calculates the number of pages based upon the requested size.

Also, LiS does not require you to pass the size of the area when freeing the page.

The routines are as follows:

```
void *lis_get_free_pages(int nbytes);
void *lis_free_pages(void *ptr);
```

The lis_free_pages routine returns a NULL pointer for the convenience of the caller.

Usage Examples:

```
ptr = lis_get_free_pages(1024 * kbytes);
ptr = lis_free_pages(ptr);
```

3.6.21 LiS PCI Interface

To assist in the portability of STREAMS drivers across different versions of the Linux kernel, LiS provides an abstraction of the PCI configuration interface. It defines a data structure that is used to describe a PCI device and a set of routines that perform operations on PCI configuration space.

Using these abstractions, a STREAMS driver can be portable from the 2.2 kernel to the 2.4 kernel with no recompilation required. The LiS structures completely hide the kernel data structures and PCI configuration space operations from the STREAMS driver.

To use this interface include the file <sys/lispci.h> in your STREAMS driver source code.

3.6.22 The LiS PCI Device Structure

This structure is distinct from a similar structure which is defined by the Linux kernel, but which differs significantly between the 2.2 and 2.4 kernels. The LiS version of this structure is oriented towards providing just enough information to allow a driver to operate the PCI device, without being concerned about the details of PCI bus topology.

This structure is used to return information to the STREAMS driver concerning devices that meet certain criteria, such as device class or manufacturer devide identification.

```
#define LIS_PCI_MEM_CNT 12      /* # mem addrs */
typedef struct lis_pci_dev {
        unsigned bus;                   /* bus number */
        unsigned dev_fcn;               /* device/function code */
        unsigned vendor;                /* vendor id */
        unsigned device;                /* device id */
```

```
            unsigned class;                 /* class type */
            unsigned hdr_type;              /* PCI header type */
            unsigned irq;                   /* IRQ number */
            unsigned long mem_addrs[LIS_PCI_MEM_CNT];
            void *user_ptr;                 /* private for user */
    } lis_pci_dev_t;
```

The bus field contains the bus number on which the device is located. LiS obtains this information from the kernel.

The dev_fcn field contains an encoding of the device number on the bus and the function number within the device that this particular structure pertains to. The pair bus and dev_fcn uniquely identifies a device in the PCI subsystem. Devices can be searched for on the PCI bus by bus number and dev_fcn value (see below).

Given a dev_fcn value, a pair of macros will extract the "device" portion and the "function number" portion from it.

`#define LIS_PCI_DEV(devfcn)`
: Extracts the "device" portion

`#define LIS_PCI_FCN(devfcn)`
: Extracts the "function number" portion

`#define LIS_MK_DEV_FCN(dev,fcn)`
: Put dev and fcn together

  Given a device number and a function number, this macro will synthesize a dev_fcn value suitable for use in searching the bus.

The vendor and device fields contain the vendor id (manuracturer code) and the vendor's device identifier for the device. Devices can be searched for on the PCI bus by vendor and device identifier (see below).

The class field contains the class code associated with the device. Devices can be searched for on the PCI bus by class code (see below).

The hdr_type field gives the type information for the PCI configuration space header.

The irq field gives the IRQ number that is assigned to this device. This is the number that is used to attach an interrupt service routine to the device.

The mem_addrs field contains a list of addresses associated with the device. These are raw PCI bus addresses and are not mapped into the address space of the processor. Empty slots contain the value zero.

3.6.23 LiS PCI Search Routines

These routines allow the STREAMS driver to find devices on the PCI bus and obtain a pointer to the lis_pci_dev_t structure for the device.

3.6.23.1 `lis_pci_dev_t *lis_pci_find_device(unsigned vendor, unsigned device, lis_pci_dev_t *previous_struct);`

Find the device by vendor identification and vendor device identification. By passing in the pointer to the previous structure returned it is possible to find all devices of a given type.

The routine returns NULL if there are no (more) devices for the given vender and device identifiers.

Usage example:

```
lis_pci_dev_t *pcip = NULL;
while ((pcip = lis_pci_find_device(0x109e, 0x8474, pcip)) != NULL) {
        pcip points to a unique device from this vendor
}
```

3.6.23.2 `lis_pci_dev_t *lis_pci_find_class(unsigned class, lis_pci_dev_t *previous_struct);`

Find the device by class. The usage is similar to lis_pci_find_device in that you can use a pointer to loop through all devices of a given class.

The function returns NULL if there are no (more) devices of the given class.

3.6.23.3 `lis_pci_dev_t *lis_pci_find_slot(unsigned bus, unsigned dev_fcn);`

Find the device by slot number. If you know the bus number (zero for most simple Intel PC systems) and the dev_fcn, you can obtain the PCI configuration information for that particular "slot". Use the LIS_MK_DEV_FCN macro to synthesize the dev_fcn value from the "device" (slot) number and the function number.

The function returns NULL if there is no device in that slot.

Note that this routine only returns one structure since it is not meaningful to process a list of devices for the same slot.

3.6.24 LiS PCI Configuration Space Routines

The following routines are used to read and write PCI configuration space for a particular device. Configuration space can be accessed by byte, word (16 bit) or dword (32 bit).

Each routine takes a pointer to an lis_pci_dev_t structure as an argument. It also takes an index value which is the byte offset from the base of the configuration space for the device at which the given byte/word/dword is to be read or written.

Care should be exercised when writing to configuration space since many of these values are determined by the PCI BIOS at system boot time.

The lis_pci_set_master routine sets the "bus master DMA" bit for the given device. This is used for devices that perform bus master DMA.

The routines are as follows:

```
int lis_pci_read_config_byte(lis_pci_dev_t *dev, unsigned index,
                             unsigned char *rtn_val);
int lis_pci_read_config_word(lis_pci_dev_t *dev, unsigned index,
                             unsigned short *rtn_val);
int lis_pci_read_config_dword(lis_pci_dev_t *dev, unsigned index,
                              unsigned long *rtn_val);
int lis_pci_write_config_byte(lis_pci_dev_t *dev, unsigned index,
                              unsigned char val);
int lis_pci_write_config_word(lis_pci_dev_t *dev, unsigned index,
                              unsigned short val);
int lis_pci_write_config_dword(lis_pci_dev_t *dev, unsigned index,
                               unsigned long val);
void lis_pci_set_master(lis_pci_dev_t *dev);
```

3.6.25 LiS PCI DMA Routines

These routines are used to allocate memory suitable for use with PCI bus master DMA devices or to map page-allocated memory for those purposes.

To understand what these routines do, please refer to the file '`/usr/src/linux/Documentation/DMA-mapping.txt`' in a fairly recent 2.4 kernel source tree. The kernel provides more functionality than is provided in LiS, so there are more routines documented there than are found in this interface.

You can use these routines in 2.2 kernels but the functions perfomed are simply approximations of the 2.4 semantics and may not work in all cases.

Note that the LiS routines have simplified the kernel interface involving "DMA handles" in such a way as to make these constructs easier to use and less error prone.

The following routines are used to allocate memory which the hardware keeps consistent between CPU access and DMA access.

```
void *lis_pci_alloc_consistent(lis_pci_dev_t *dev, size_t size,
                               lis_dma_addr_t * dma_handle);
void *lis_pci_free_consistent(lis_dma_addr_t * dma_handle);
```

The following routines are used to obtain a DMA address from a returned DMA handle. You need to know whether or not your hardware environment is using 32-bit or 64-bit DMA addresses.

```
u32 lis_pci_dma_handle_to_32(lis_dma_addr_t * dma_handle);
u64 lis_pci_dma_handle_to_64(lis_dma_addr_t * dma_handle);
```

The following routines are usd to map page-allocated memory for DMA purposes. The direction indicator of LIS_SYNC_FOR_CPU means that you intend to use the memory for DMA transfers into memory. The direction indicator of LIS_SYNC_FOR_DMA means that you intend to use the memory for DMA transfers out of memory. If the DMA operation goes both ways then use LIS_SYNC_FOR_BOTH.

```
void lis_pci_map_single(lis_pci_dev_t *dev, void *ptr, size_t size,
                        lis_dma_addr_t * dma_handle, int direction);
void *lis_pci_unmap_single(lis_dma_addr_t * dma_handle);
int lis_osif_pci_map_sg(struct pci_dev *hwdev, struct scatterlist *sg,
int nents, int direction);
void lis_osif_pci_unmap_sg(struct pci_dev *hwdev, struct scatterlist *sg,
int nents, int direction);
```

The direction indicators are as follows:

```
LIS_SYNC_FOR_CPU
LIS_SYNC_FOR_DMA
LIS_SYNC_FOR_BOTH
```

With mapped memory, i.e., non-consistent memory, you need to synchronize the memory whenever the CPU writes into it and the DMA needs to read it, or when the DMA has written into it and the CPU needs to read it. The following routine is used for that purpose.

```
void lis_pci_dma_sync_single(lis_dma_addr_t * dma_handle, size_t size,
                             int direction);
void lis_osif_pci_dma_sync_sg(struct pci_dev *hwdev,
      struct scatterlist *sg,
      int nelems, int direction);
```

The following routines can be used at driver initialization time to discover and control the addressing boundary restrictions of a device.

```
int lis_pci_dma_supported(lis_pci_dev_t *dev, u64 mask);
int lis_pci_set_dma_mask(lis_pci_dev_t *dev, u64 mask);
```

Please consult the file '`<sys/osif.h>`' for additional routines that may be present for DMA support.

3.6.26 LiS Atomic Functions

LiS provides for atomic integers implemented in a portable fashion. To declare an LiS portable atomic integer use the following declaration syntax:

```
lis_atomic_t myatom;
```

LiS then provides the following operations on variables of this type.

```
void lis_atomic_set(lis_atomic_t *atomic_addr, int valu);
int lis_atomic_read(lis_atomic_t *atomic_addr);
void lis_atomic_add(lis_atomic_t *atomic_addr, int amt);
void lis_atomic_sub(lis_atomic_t *atomic_addr, int amt);
void lis_atomic_inc(lis_atomic_t *atomic_addr);
void lis_atomic_dec(lis_atomic_t *atomic_addr);
int lis_atomic_dec_and_test(lis_atomic_t *atomic_addr);
```

Of these, only lis_atomic_dec_and_test needs any explanation. This routine performs an atomic_dec on the variable and returns true if the counter reached zero via that decrement operation. Note that by the time the routine returns some other CPU with access to the same variable may have changed its value. So the return reports only on the instantaneous value of the variable.

3.6.27 LiS Locks

LiS provides an abstraction and an insulated interface to the Linux kernel for spin locks, interrupt disabling and semaphores. If you use this interface in your STREAMS driver you can utilize these kernel services on different versions of the Linux kernel without the necessity of recompiling your driver for each version of the kernel.

The LiS locks are especially useful in consideration of Linux kernels compiled with and without the SMP option set. The spin locks and semaphores of the Linux kernel are implemented using external inline functions. These functions are coded in assembly language and generate different sequences of instructions depending upon the compile time setting of the SMP option. Spin locks and semaphores compiled with SMP reset will not function properly on a multi-CPU system running an SMP kernel.

The LiS locks mechanism solves this problem by abstracting the locking primitives into actual subroutines, not inlines, defined within LiS. Since LiS is compiled from source code when it is installed the subroutines in LiS have the correct setting of SMP for the locking primitives. This allows the STREAMS driver code to be compiled once and the object code reused for multiple installations with varying options.

The following sections document the spin locks, interrupt disabling and semaphore mechanisms offered by LiS. To use these mechanisms include the file <sys/lislocks.h> in your STREAMS driver source code.

In choosing the appropriate type of lock to use, one must bear in mind that STREAMS drivers are not allowed to "sleep" in "put" and "service" procedures, only in "open" and "close" routines. That means that spin locks are the mutual exclusion mechanism of choice for "put" and "service" procedures. It is reasonable to use sleeping semaphores in "open" and "close" routines.

The simple interrupt exclusion mechanism can be used to exclude only interrupt routine execution for a section of code. However, this mechanism does not exclude other "put" or "service" procedures that may be executed on other CPUs. This may not be much of a consideration since LiS acquires a lock in the queue structure before executing the "put" or "service" procedure pointed to by that queue.

However, it could happen that the "read put/service" and "write put/service" procedures get executed simultaneously since there are two different locks in the STREAMS queues, one in the read queue and one in the write queue. In this case, the STREAMS driver code would need to use spin locks to protect data structures shared between the read and write "put" or "service" procedures. See the '`qlock`' option for `strconf` for more information about LiS implicit use of locks to protect put and service procedure entries.

3.6.28 LiS Spin Locks

LiS provides an implementation of spin locks that utilizes the Linux kernel's spin lock mechanism to perform the actual locking functions. The LiS implementation adds features to the kernel spin locks such as the following:

- LiS spin locks are nestable. The same thread can acquire the same lock and release it in nested fashion.
- LiS spin locks are more debuggable. The LiS lock structure contains an ASCII name for the lock which makes it easier to identify in debugging situations.
- LiS maintains a lock trace table. A debugging option for LiS causes it to log all spin lock operations to a trace table which can be printed out via an option to the streams command.
- LiS spin locks are portable. A STREAMS driver can utilize the same LiS lock mechanism across different versions of the Linux kernel. This pushes the kernel differences into LiS and out of the STREAMS driver code.
- LiS spin locks are documented. You don't have to read the kernel source code to figure out how to use them.

For these reasons I highly recommend that STREAMS drivers use the LiS spin lock implementation in place of the direct kernel spin locks. The portability aspect of LiS spin locks cannot be overemphasized. Different Linux kernel compile-time options can lead to a proliferation of STREAMS driver code versions, or the necessity of always compiling the driver from source when it is installed. LiS spin locks allow a STREAMS driver to be compiled independently of kernel options with only the binary needed at driver installation time.

To declare a spin lock, use the typedef lis_spin_lock_t, as in the following:

```
lis_spin_lock_t mylock;
```

LiS spin locks must be initialized before they are used. There is one initialization routine no matter which style of locking you intend to use.

void lis_spin_lock_init(lis_spin_lock_t *lock, const char *name) ;

This routine initializes the spin lock and associates an ASCII string name with it. The pointer name is saved in the lock structure for later use in printing out the lock trace table. It is the caller's responsibility to ensure that the name resides in memory that will persist for the duration of the existence of the lock.

You can also use dynamically allocated spin locks. This technique allows your STREAMS driver to be completely immune from changes in kernel version regarding the size of a spin lock since your driver only has to store a pointer to the allocated lock. The allocation and deallocation routines are as follows.

```
lis_spin_lock_t *lis_spin_lock_alloc(const char *name);
lis_spin_lock_t *lis_spin_lock_free(lis_spin_lock_t *lock, const char *name);
```

The allocation function returns a pointer to the spin lock, or NULL if the memory could not be allocated. The free function returns a NULL pointer for the convenience of the caller.

For further information on spin locks, see the section on [89]debugging spin locks.

To lock and unlock a spinlock, use any of the following pairs of routines. If you use the first routine to lock the spin lock then be sure to use its companion unlock routine. For nesting considerations, [91]see below.

```
void lis_spin_lock(lis_spin_lock_t *lock);
void lis_spin_unlock(lis_spin_lock_t *lock);
int lis_spin_trylock(lis_spin_lock_t *lock);
```

These routines are to be called only from background processing to lock and unlock a spin lock. The trylock routine locks the spin lock if it is available, returning "true", or leaves it unlocked if it is unavailable, returning "false".

Background processing means any STREAMS driver processing that does not occur at interrupt time. These routines lock the lock but do not exclude interrupt routines from execution. Thus, your interrupt service routine can still be called whether or not your driver is holding a spin lock that was locked with one of these routines.

You can nest pairs of calls to these routines from the same thread of execution. [92]See below for more information on lock nesting.

Usage example:

```
lis_spin_lock(&mylock);
...
lis_spin_unlock(&mylock);
```

```
void lis_spin_lock_irq(lis_spin_lock_t *lock);
void lis_spin_unlock_irq(lis_spin_lock_t *lock);
```

This pair of routines locks the spin lock with interrupts disabled for the duration of the holding of the lock. The routine lis_spin_lock_irq re-enables interrupts after unlocking the lock.

You can use this technique to exclude interrupt routine execution. However, it is not advisable for interrupt routines themselves, or any routines called from an interrupt routine, to use this mechanism since the unlock primitive unconditionally enables interrupts, which may not be desirable from inside an interrupt routine.

These routines may be used in nested fashion. Only the outermost unlock routine will actually enable interrupts. [94]See below for more information about lock nesting.

Usage example:

```
lis_spin_lock_irq(&mylock);
...
lis_spin_unlock_irq(&mylock);
```

```
void lis_spin_lock_irqsave(lis_spin_lock_t *lock, int *flags);
void lis_spin_unlock_irqrestore(lis_spin_lock_t *lock, int *flags);
```

This pair of routines is similar to the "spin_lock_irq" routines in that the locking routine disables interrupts. However, it saves the interrupt state in the integer argument whose pointer is passed to the locking routine. The unlock routine then restores the interrupt state after unlocking the lock.

These routines are suitable for use by routines that are called both from interrupt level and from background. They also have the effect, when used in an interrupt routine, of excluding multiple execution of an interrupt routine on multiple CPUs in an SMP system.

These routines may be used in nested fashion. Only the outermost unlock routine will actually restore the interrupt state. [96]See below for more information about lock nesting.

Usage example:

```
lis_spin_lock_t mylock;
int flags;
lis_spin_lock_irqsave(&mylock, &flags);
...
lis_spin_unlock_irqrestore(&mylock, &flags);
```

Note that the unlock routine is passed the address of the flags just as in calling the lock routine.

3.6.29 Lock Nesting

LiS spin locks can be locked and unlocked in nested fashion. When doing so, it is always best to use the same pair of lock and unlock routines at all levels of nesting for the same lock. Mixing different types of locking can lead to unexpected results and non-portable behavior.

LiS allows a single thread to lock spin locks in nested fashion. That is, the second and subsequent calls to the lock routine from a single thread will not spin on the lock because of finding it in a locked state from the first call. Also, every unlock call except the last one, the one that balances the first locking call, does not unlock the lock. Only the outermost unlock call causes the lock to be unlocked.

If the nesting is via lis_spin_lock_irq, then only the outermost unlock call enables interrupts. If the nesting is via lis_spin_lock_irqsave, then only the outermost unlock call restores the interrupt state.

When two or more threads attempt to lock a spin lock "simultaneously" only one thread is allowed to proceed at a time. The other threads "spin", that is, the CPUs executing the other threads are executing a loop that tests the lock repeatedly until it becomes available. Consequently, it is advisable to use locks to protect the execution of fairly short pieces of code if there is any likelihood of contention for the lock. While one thread is holding the lock, other CPUs may be idling waiting for it.

In the context of locking, "simultaneously" means any time from the moment of the first thread locking the spin lock until that thread unlocks the lock. If another thread attempts to lock the spin lock at any point in that interval then it will "spin."

When multiple threads use multiple spin locks to protect multiple resources, it is always a good idea if all threads execute "lock" operations on the multiple spin locks in the same order. It is also highly recommended that they execute "unlock" operations in the exact reverse order as the "lock" operations. This avoids so-called "deadly embrace" situations in which process A acquires spin lock A, process B acquires spin lock B, and then process A waits on B while process B waits on A.

3.6.30 LiS Read/Write Locks

LiS offers an abstraction of the kernel's read/write locks. The LiS abstractions allow STREAMS drivers to use these locks without concern for changes that occur from one version of the kernel to the next.

A read/write lock is declared as a special data object of type lis_rw_lock_t. There are two types of routines to manipulate these locks. One set operates on the lock as a "read" lock. The other set operates on the lock as a "write" lock.

There can be multiple threads owning the lock in read mode. There can only be one thread that owns the lock in write mode. Furthermore, in order to acquire the lock in write mode, all the owners of the read mode lock must give it up.

The locks are used in the obvious way. If you only need to read the protected structure you use the read lock routine. If you need to change the structure you use the write lock routine.

Note that once you have a read lock you must give it up in order to get the same lock as a write lock.

The lock manipulation routines also allow for "regular", "irq" and "irqsave" manipulations of the read/write locks, just as with spin locks.

You must initialize your lock before using it, just as with spin locks. And in parallel to spin locks LiS provides two initialization routines. One operates directly on the read/write lock, and the other allocates memory dynamically for the lock. You can deallocate the dynamically allocated lock by calling the "free" routine.

The following is a listing of the read/write lock routines in LiS. The prototypes are in the file <sys/lislocks.h>.

```
void lis_rw_read_lock(lis_rw_lock_t *lock);
void lis_rw_write_lock(lis_rw_lock_t *lock);
void lis_rw_read_unlock(lis_rw_lock_t *lock);
void lis_rw_write_unlock(lis_rw_lock_t *lock);

void lis_rw_read_lock_irq(lis_rw_lock_t *lock);
void lis_rw_write_lock_irq(lis_rw_lock_t *lock);
void lis_rw_read_unlock_irq(lis_rw_lock_t *lock);
void lis_rw_write_unlock_irq(lis_rw_lock_t *lock);

void lis_rw_read_lock_irqsave(lis_rw_lock_t *lock, int *flags);
void lis_rw_write_lock_irqsave(lis_rw_lock_t *lock, int *flags);
void lis_rw_read_unlock_irqrestore(lis_rw_lock_t *lock, int *flags);
void lis_rw_write_unlock_irqrestore(lis_rw_lock_t *lock, int *flags);

void lis_rw_lock_init(lis_rw_lock_t *lock, const char *name);
lis_rw_lock_t *lis_rw_lock_alloc(const char *name);
lis_rw_lock_t *lis_rw_lock_free(lis_rw_lock_t *lock, const char *name);
```

3.6.31 LiS Interrupt Enable/Disable

LiS provides primitives for enabling and disabling interrupts modelled after the SVR4 SPL mechanism. There is one routine that is used to disable interrupts and another one for enabling interrupts. The routines are as follows:

```
int lis_splstr(void);
void lis_splx(int x);
```

The lis_splstr routine is used to disable interrupts. It returns a value that must be passed to lis_splx when it it desired to restore the interrupt level to its previous state. These two routines are implemented using the primitives lis_spin_lock_irqsave and lis_spin_unlock_irqrestore.

These routines can be used from background code ("put" and "service" procedures, or "open" and "close" routines), or from interrupt level. LiS itself uses these routines to protect STREAMS structures from ill-timed modification by interrupt routines. Many LiS utility routines, such as putq, getq and qenable, call these routines within themselves.

It is safe, and occurs frequently, to use these routines in a nested fashion. When using these routines in a nested fashion be sure that the value returned by the call to lis_splstr at level n is the value passed back to lis_splx at level n. The nesting rules for these routines are otherwise the same as for the pair lis_spin_lock_irqsave and lis_spin_unlock_irqrestore.

Usage examples:

```
int x, y;
x = lis_splstr();
...
y = lis_splstr();
...
lis_splx(y);
...
lis_splx(x);
```

For further information on these routines see the section on [100]debugging spin locks.

3.6.32 LiS Semaphores

LiS provides an implementation of semaphores that is built upon the Linux kernel's semaphores. The LiS implementation adds features to the kernel semaphores such as the following:

- LiS semaphores are more debuggable. The LiS semaphore structure contains fields that save the file name and line number of the semaphore owner. This makes it easier to debug drivers which utilize semaphores.
- LiS semaphores retain error information. When a "down" operation fails, LiS saves the error number in the semaphore structure for post mortem analysis.
- LiS semaphores are portable. A STREAMS driver can utilize the same LiS semaphore mechanism across different versions of the Linux kernel. This pushes the kernel differences into LiS and out of the STREAMS driver code.
- LiS semaphores can be easily allocated dynamically so your driver is completely immune from Linux kernel version considerations.
- LiS semaphores are documented. You don't have to read the kernel source code to figure out how to use them.

For these reasons I highly recommend that STREAMS drivers use the LiS semaphore implementation in place of the direct kernel semaphores. The portability aspect of LiS semaphores cannot be overemphasized. Different Linux kernel compile-time options can lead to a proliferation of STREAMS driver code versions, or the necessity of always compiling the driver from source when it is installed. LiS semaphores allow a STREAMS driver to be compiled independently of kernel options with only the binary needed at driver installation time.

To declare an LiS semaphore, use a declaration similar to the following:

```
lis_semaphore_t mysem;
```

LiS semaphores must be initialized before they are used. Use the following routine to initialize a declared semaphore.

```
void lis_sem_init(lis_semaphore_t *, int);
```

If you initialize the semaphore to 0, then the first "down" operation on the semaphore will wait. If you initialize it to 1, then the first "down" operation will not wait. If you initialize it to n, then the first n "down" operations will not wait.

You can also allocate semaphores dynamically using the following routine.

```
lis_semaphore_t *lis_sem_alloc(int);
```

This routine uses the kernel's memory allocator to allocate space for the semaphore. The lis_sem_destroy routine will deallocate it for you. The advantage of using this routine is that your STREAMS driver only has to have a pointer to the semaphore, not a semaphore structure itself. This adds an extra level of protection of your driver from kernel version considerations.

You can use the semaphore value to manage a pool of resources by initializing a semaphore to the number of items in the resource and having a driver open routine perform a "down" operation on the semaphore. This causes the open operations to be queued until the resource is available.

LiS semaphores should be explicitly destroyed when they are no longer needed, typically from your STREAMS driver close routine. This operation is accomplished via the following routine.

```
lis_semaphore_t *lis_sem_destroy(lis_semaphore_t *,int);
```

This routine returns a NULL pointer for the convenience of the caller.

For further information on semaphores, see the section on [102]debugging semaphores.

The following two routines are used to acquire and release a semaphore.

```
int lis_down(lis_semaphore_t *sem);
void lis_down_nosig(lis_semaphore_t *lsem);
void lis_up(lis_semaphore_t *sem);
```

The routine `lis_down` returns 0 for success and a negative error code for failure. The caller has not acquired the semaphore unless the routine returns zero.

One reason for a negative return could be that the calling task was signalled while waiting for the semaphore to become available. If this has occurred the return code will be set to -EINTR.

The function `lis_down_nosig` waits for the sempahore with signals blocked. It is useful in driver close routines that must use a semaphore to control access to the structures that need to be deallocated. It is common for the driver close routine to be called from a process that has been signalled – for example a process that was killed with a ⟨Ctrl-C⟩ from the keyboard. In this case, `lis_down` will return immediately with `-EINTR`, an undesirable situation. using `lis_down_nosig` in this situation blocks signals so that the close routine can wait on the semaphore even if the process has been signalled.

Semaphores cannot be used in nested fashion. Care must be exercised that a single thread only performs one "down" operation on a given semaphore.

When multiple threads use multiple semaphores to protect multiple resources, it is always a good idea if all threads execute "down" operations on the multiple semaphores in the same order. It is also highly recommended that they execute "up" operations in the exact reverse order as the "down" operations. This avoids so-called "deadly embrace" situations in which process A acquires semaphore A, process B acquires semaphore B, and then process A waits on B while process B waits on A.

Semaphores should be used only in STREAMS driver "open" and "close" routines. STREAMS driver "put" and "service" procedures are not allowed to sleep. They should use spin locks instead of semaphores.

Usage example:

```
if (lis_down(&mysem) == 0) {
        ...
        lis_up(&mysem);
}
```

3.6.33 Debugging Spin Locks

LiS spin lock structures contain fields that assist in the debugging of spin-lock related problems. The LiS spin lock structure contains the following fields.

Field Description

spin_lock_mem An opaque memory area that contains the kernel's spin lock structure.

name

Pointer to an ASCII name for the lock. This allows one to readily identify the function of the lock (assuming that it is aptly named).

taskp

A (void *) which is really a (struct task_struct *) pointer. It points to the task that originally acquired the lock, or is NULL if no task has acquired the lock.

spinner_file, spinner_line

File and line number of the most recent call to one of the lis_spin_lock functions. This tells which line of code most recently tried to get the lock.

owner_file, owner_line

File and line number of the call to one of the lis_spin_lock functions that first acquired the lock. These fields are set at the same time as the taskp field.

unlocker_file, unlocker_line

File and line number of the call to one of the lis_spin_unlock functions that performed the final unlock on the lock, thus making it available for another thread. These fields are set at the same time as the taskp field is set to NULL.

If a thread owns the lock then its value of the current task pointer will be in taskp. If there is no other thread spinning on the lock, and if the lock has not been acquired in a nested fashion, then the spinner and owner fields will indicate the same file and line number.

If the spinner and owner fields are different and if the taskp is non-NULL then if the thread that most recently called one of the lis_spin_lock routines is different from the task that owns the lock, then that other task is spinning on the lock. By examination of the lock you can see which task owns the lock and where in the code it was acquired. This is often enough information to figure out why a deadlock is occurring.

A "deadly embrace" occurs when two threads each need to acquire two spin locks but they acquire them in the opposite order from each other. Under circumstances of contention each process owns the lock that the other is spinning on and will not release the lock until it acquires the other lock. Thus, both threads spin forever.

Note that the LiS splstr and splx functions are written in terms of LiS spin locks. LiS does not use these routines internally. They are provided to the user for backwards compatibility. However, it is important to know that these routines are spin locks in disguise. This means that the order of use of these functions mixed in with explicit spin lock manipulations may also lead to deadly embraces.

An effective technique for troubleshooting these kinds of problems is to use the two-machine kernel debugger, [105]kgdb. With this setup you can break into the target machine and look at memory using high level debugging techniques, including printing out of structures. Using kgdb you can find out where each CPU is executing, look at the corresponding source code lines, observe the locks that are involved, and then print out the lis_spin_lock_t structures for the specific locks. Oftentimes the information contained in the two locks will immediately reveal the nature of the deadly embrace.

It is also possible to have LiS trace all lock and semaphore operations. One of the LiS debug bits enables this function. To set this debug bit use the following command.

```
streams -d0x0x80000
```

This causes LiS to make entries in a global trace buffer named lis_spl_track. The global pointer lis_spl_track_ptr indicates the next location in the table into which an entry is to be placed, which means that it points to the oldest entry in the buffer. Entries in the buffer are of type spl_track_t.

The fields of this structure are as follows. Field Description type The type of entry as follows.

Value	Meaning
1	splstr
2	splx
3	spin lock
4	spin unlock
5	semaphore down
6	semaphore up

cpu

The cpu number of the processor which made this entry. addr The address of the spin lock or semaphore involved in the operation.

tskp The task pointer for the task that made this entry. state Nesting value for spin locks, count field of the semaphore.

file, line

File and line number of the call to the LiS locking or semaphore routine that caused this entry to be made.

The trace buffer contains 4096 of these entries, maintained in a circular fashion. By printing out these entries you can see the history of lock manipulation within LiS. The command streams -p causes LiS to print out this table from within the kernel. The resulting output can be found in '`/var/log/messages`' (typically). However, in practice the system is usually hung when you need this information so you end up printing it from within the debugger.

3.6.34 Lock Semaphore and Queue Contention

The `streams` command can be used to enable the tracking of contetion for locks, semaphores and STREAMS queues. Use the command '`streams -D0x08`' to enable the contention tracking. The command '`streams -L`' then causes the contention tables to be printed out.

Locks and semaphores are in contention when a thread goes to spin on a lock or perform a `down` function on a sempahore, and the thread has to wait because the lock or semaphore is owned by another thread. LiS counts such occurences on a per-lock basis and reports the results with the '`streams -L`' command.

Queues are in contention when the semaphore that controls access to the queue is in contention. However, there are options that affect which semaphore is used to control access to a queue and these options will also have an effect on the reporting of queue contention.

3.6.35 Debugging Semaphores

LiS semaphore structures contain fields that assist in the debugging of semaphore related problems. The LiS semaphore structure contains the following fields.

Field Description

sem_mem An opaque memory area that contains the kernel's semaphore structure.

taskp

A (void *) which is really a (struct task_struct *) pointer. It points to the task that most recently acquired the semaphore, or is NULL if no task has acquired the semaphore. The taskp is set to NULL just prior to calling the kernel's up routine on the semaphore. Thus it stays NULL if no other task is pending on the semaphore.

downer_file, downer_line File and line number of the most recent call to the lis_down function. This tells which line of code most recently tried to get the semaphore.

owner_file, owner_line File and line number of the call to the lis_down function that acquired the semaphore. These fields are set at the same time as the taskp field.

upper_file, upper_line File and line number of the call to the lis_up function. These fields are set at the same time as the taskp field is set to NULL.

If the taskp field is non-NULL then the semaphore is owned by the task so indicated. If it is NULL then the semaphore is unowned. The upper fields show where the semaphore was last released.

If the downer and owner fields both indicate the same file and line number then that is an indication that the semaphore was acquired at that location in the program. If they are different, and if the taskp is non-NULL, that is an indication that there is a task waiting on the semaphore at the downer location. The owner fields show where the semaphore was acquired.

Bear in mind that semaphore acquisitions do not nest as is the case with spin locks. Therefore, if the same thread calls lis_down without calling lis_up on the same semaphore then the thread will be deadlocked. The downer and owner fields will usually offer a clue to this type of deadlock.

You can also use the LiS [107]lock trace buffer mechanism to assist in debugging semaphore usage.

3.6.36 STREAMS Utility Routines

The following routines are available to LiS STREAMS drivers. These are standard AT&T SVR4 utility routines. They (hopefully) have the same semantics in LiS as they do in SVR4 STREAMS.

These routines are presented here in alphabetical order with no description. Please refer to the [109]AT&T SVR4 STREAMS documentation for the descriptions of these routines.

3.6.37 Freezing Streams

There are two sets of routines that can be used to '`freeze`' a stream. They are used in slightly different ways and have slightly different semantics. One set uses the routines `freezestr()` and `unfreezestr()`; the other set uses the routines `qprocesoff()` and `qprocson()`.

3.6.37.1 Freezestr and Unfreezestr

```
void freezestr(queue_t *q);
void unfreezestr(queue_t *q);
```

These routines operate on the entire stream of which the queue is a member. The stream is found by traversing the chain of queues in both directions until encountering a queue that is not linked to another queue. As a simple example it includes all queues from the stream head down through any pushed modules to the driver queue in which one of those queues is the one passed as the parameter to either of these routines.

The process of freezing the stream is to place it into a state such that messages will not flow up and down the stream. That is `put` and `service` procedures will not be called. It `putnext()` is called on a queue within a frozen stream the passed message is placed into a special deferred message list. Messages are removed from this list and passed to the `put` procedure when the stream is later unfrozen.

Drivers that have frozen a stream should refrain from performing queuing operations on queues within the stream, such as `getq` and `putq`. LiS does not enforce this so one must exercise some care when using these routines.

SVR4 STREAMS specification says that the driver's close routine will not be called if the stream is frozen. LiS does not implement this rule and will close a frozen stream.

LiS uses these routines internally at stream close time to stop message flow when the stream is being dismantled. It also uses them during `I_PUSH` and `I_POP` processing to inhibit message flow while replumbing the stream.

Drivers should use these routines with some caution. Because the stream is frozen the driver cannot receive any messages from above or below, including `M_IOCTL`. This may make it tricky deciding when to unfreeze a stream.

3.6.37.2 Qprocsoff and Qprocson

```
void qprocson(queue_t *rdq);
void qprocsoff(queue_t *rdq);
```

These routines are conventionally used in a driver open (`qprocson`) and close (`qprocsoff`) routine. In some STREAMS implementations `qprocson` must be called to enable messages to flow into the queue once open processing has completed. This is not necessary in LiS.

In LiS it does no harm to call `qprocson` in the driver open routine and `qprocsoff` in the driver close routine, though it is not necessary to do so.

The effect of `qprocsoff` is similar to that of `freezestr` except that it applies just to the single queue rather than to the entire stream. Once significant difference is that if a pushable modules is in a '`qprocsoff`' condition and a message flows into the module, STREAMS will route the message to the next module or driver in the chain of queues, looking for one that is enabled. If no such module or driver exists, the messag will be placed into the deferred message list of the queue at the far end of the chain of queues. The messages will be presented to the driver put routine when `qprocon` is called.

It is best to use these routines only at open and close time since that seems to have been the intent of the STREAMS designers.

3.6.38 Flushing Queue Bands

A special note on flushing queue bands is in order. The rules for flushing queues are a bit complex, so we wish to review them here in some detail.

First some definitions and some things that affect all queue flushing. The term "data message" in the context of queue flushing means messages of type `M_DATA`, `M_PROTO`, `M_PCPROTO` or `M_DELAY`. All other message types are considered "non-data messages". You may find it less than intuitive that `M_PCPROTO` is considered a "data message".

The term "ordinary message" in the context of queue flushing means messages of type `M_DATA`, `M_PROTO`, `M_BREAK`, `M_CTL`, `M_DELAY`, `M_IOCTL`, `M_PASSFP`, `M_RSE`, `M_SETOPTS` or `M_SIG`. Please note that `M_PCPROTO` is not on this list.

The flag argument of FLUSHDATA means that only "data messages" are to be flushed. The flag argument of FLUSHALL means that "all" messages are to be flushed. As we shall see, in flushing queue bands whether a message gets flushed or not depends upon what the meaning of the word "all" is.

First, let's take the case of the routine flushq(q,flag). If flag is set to FLUSHDATA then all "data messages" in the entire queue, including all queue bands, are flushed. If the flag is set to FLUSHALL then the entire queue is flushed.

The case of the routine flushband(q,band,flag) is more complicated.

If the band argument is zero then special rules apply. In this case, only "ordinary" messages are flushed from the queue. The value of the flag parameter does not influence the operation. In Solaris STREAMS this behavior does not occur. They flush either "data messages" or "all" messages on band zero. Comments in the Solaris 8 source code indicate that the author of the flush code was somewhat confused on this point.

If the band argument is non-zero then the specific band of the queue is flushed in a manner similar to that of flushq. That is, the flag argument of FLUSHDATA means just flush "data messages" and the value of FLUSHALL means flush "all" messages from the specific band.

One further item needs some attention. Whenever an `M_PCPROTO` (or other "high priority") message is inserted into a STREAMS queue it is queued ahead of all messages in any queue band. This means that an `M_PCPROTO` cannot be directed to a queue band. It also means that flushband can never flush an `M_PCPROTO`, or any other "high priority" message from the queue. In order to flush `M_PCPROTO`s you must call flushq and flush the entire queue of either "data messages" or "all" messages.

3.6.39 Utility Prototypes

```
int adjmsg(mblk_t *mp, int length);
struct msgb *allocb(int size, unsigned int priority);
------------------------------------------

queue_t *backq(queue_t *q);
int bcanput(queue_t *q, unsigned char band);
int bcanputnext(queue_t *q, unsigned char band);
void bcopy(void *src, void *dst, int nbytes);
int bufcall(unsigned size, int priority, void (*function) (long), long arg);
void bzero(void *addr, int nbytes);
------------------------------------------

int canput(queue_t *q);
int canputnext(queue_t *q);
void cmn_err(int err_lvl, char *fmt, ...);
mblk_t *copyb(mblk_t *mp);
mblk_t *copymsg(mblk_t *mp);
------------------------------------------

#define datamsg(type) -- true if msg->b_datap->db_type is data
mblk_t *dupb(mblk_t *mp);
mblk_t *dupmsg(mblk_t *mp);
------------------------------------------

void enableok(queue_t *q);
mblk_t *esballoc(unsigned char *base, int size, int priority, frtn_t *freeinfo);
int esbbcall(int priority, void (*function) (long), long arg);
------------------------------------------

void flushband(queue_t *q, unsigned char band, int flag);
void flushq(queue_t *q, int flag);
void freeb(mblk_t *bp);
void freemsg(mblk_t *mp);
void freezestr(queue_t *q);
void unfreezestr(queue_t *q);
------------------------------------------

int getmajor(dev_t dev);
int getminor(dev_t dev);
mblk_t *getq(queue_t *q);
------------------------------------------

int insq(queue_t *q, mblk_t *emp, mblk_t *mp);
------------------------------------------
```

```
void *kmem_alloc(int siz, int wait_code);
void *kmem_zalloc(int siz, int wait_code);
void kmem_free(void *ptr, int siz);
------------------------------------------

void linkb(mblk_t *mp1, mblk_t *mp2);
------------------------------------------

int msgdsize(mblk_t *mp);
mblk_t *msgpullup(mblk_t *mp, int length);
int msgsize(mblk_t *mp);
------------------------------------------

void noenable(queue_t *q);
------------------------------------------

queue_t *OTHERQ(queue_t *q);
------------------------------------------

int pullupmsg(mblk_t *mp, int length);
int putbq(queue_t *q, mblk_t *mp);
int putctl(queue_t *q, int type);
int putctl1(queue_t *q, int type, int param);
void putnext(queue_t *q, mblk_t *mp);
int putnextctl(queue_t *q, int type);
int putnextctl1(queue_t *q, int type, int param);
int putq(queue_t *q, mblk_t *mp);
------------------------------------------

void qenable(queue_t *q);
void qreply(queue_t *q, mblk_t *mp);
int qsize(queue_t *q);
void qprocsoff(queue_t *rdq);
void qprocson(queue_t *rdq);
------------------------------------------

queue_t *RD(queue_t *q);
queue_t *WR(queue_t *q);
queue_t *OTHERQ(queue_t *q);
mblk_t *rmvb(mblk_t *mp, mblk_t *bp);
void rmvq(queue_t *q, mblk_t *mp);
------------------------------------------

int SAMESTR(queue_t *q);
int strqget(queue_t *q, qfields_t what, unsigned char band, long *val);
int strqset(queue_t *q, qfields_t what, unsigned char band, long val);
------------------------------------------

int testb(int size, unsigned int priority);
------------------------------------------

#define HZ -- ticks per second
typedef void timo_fcn_t (caddr_t arg);
toid_t timeout(timo_fcn_t *timo_fcn, caddr_t arg, long ticks);
toid_t lis_untimeout(toid_t id);
------------------------------------------

void unbufcall(int bcid);
```

```
mblk_t *unlinkb(mblk_t *mp);
int untimeout(int id);
------------------------------------------

queue_t *WR(queue_t *q);
------------------------------------------

int xmsgsize(mblk_t *mp);
```

3.6.40 System Calls from within the Kernel

LiS provides STREAMS drivers with a few system calls that can be made from within the kernel. These calls are intended to allow STREAMS drivers to manage their device special files through which the drivers are accessed. For example, by using the *lis_mknod* function a dynamically loaded driver can register itself with LiS, obtain a major device number and make its "/dev" entries at module load time. Using the lis_unlink function it can remove these "/dev" entries when the module unloads.

The semantics of the following routines are exactly the same as the user level routines of the same names without the "lis_" prefix. This is so because these routines are really just wrappers on a kernel system call. We list the function prototypes here but leave the detailed documentation to "man pages" and other documentation.

The following function prototypes exist in the file <sys/dki.h>.

```
int lis_mknod(char *name, int mode, dev_t dev);
int lis_unlink(char *name);
int lis_mount(char *dev_name, char *dir_name, char *fstype,
              unsigned long rwfla g, void *data);
int lis_umount(char *file, int flags);
```

4 Conformance

4.1 STREAMS Compatibility

Linux STREAMS (LiS) provides some degree of compatibility with other *STREAMS* implementation as follows:

— *SVR 4.2 ES/MP*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *SVR 4.2 ES/MP* to ease portability and common comprehension, see section "SVR 4.2 Compatibility" in *STREAMS Programmer's Guide*.

— *AIX 5L Version 5.1*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *AIX 5L Version 5.1* to ease portability and common comprehension, see section "AIX Compatibility" in *STREAMS Programmer's Guide*.

— *HP-UX 11.0i v2*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *HP-UX 11.0i v2* to ease portability and common comprehension, see section "HP-UX Compatibility" in *STREAMS Programmer's Guide*.

— *OSF/1 1.2/Digital UNIX/True 64*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *OSF/1 1.2/Digital UNIX* to ease portability and common comprehension, see section "OSF/1 Compatibility" in *STREAMS Programmer's Guide*.

— *UnixWare 7.1.3 (OpenUnix 8)*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *UnixWare 7.1.3 (OpenUnix 8)* to ease portability and common comprehension, see section "UnixWare Compatibility" in *STREAMS Programmer's Guide*.

— *Solaris 9/SunOS 5.9*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *Solaris 9/SunOS 5.9* to ease portability and common comprehension, see section "Solaris Compatibility" in *STREAMS Programmer's Guide*.

— *SUPER-UX*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *SUPER-UX* to ease portability and common comprehension, see section "SUX Compatibility" in *STREAMS Programmer's Guide*.

— *UXP/V*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *UXP/V* to ease portability and common comprehension, see section "UXP Compatibility" in *STREAMS Programmer's Guide*.

— *LiS-2.16.18*
: *Linux STREAMS (LiS)* provides some degree of operational compatibility with *LiS 2.16* to ease portability and common comprehension, see section "LiS Compatibility" in *STREAMS Programmer's Guide*.

For additional details, see section "About This Manual" in *STREAMS Programmer's Guide*.

4.2 Porting

— *SVR 4.2 ES/MP*
: *Linux STREAMS (LiS)* provides compatibility functions for source level compatibility with *SVR 4.2 ES/MP* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from SVR 4.2 MP" in *Linux Fast-STREAMS Porting Guide*.

— *AIX 5L Version 5.1*
: *Linux STREAMS (LiS)* provides compatibility functions for source level compatibility with *AIX 5L Version 5.1* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from AIX 5L Version 5.1" in *Linux Fast-STREAMS Porting Guide*.

— *HP-UX 11.0i v2*
: *Linux STREAMS (LiS)* provides compatibility functions for source level compatibility with *HP-UX 11.0i v2* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from HP-UX 11.0i v2" in *Linux Fast-STREAMS Porting Guide*.

— *OSF/1 1.2/Digital UNIX/True 64*
: *Linux STREAMS (LiS)* provides compatibility functions for source level compatibility with *OSF/1 1.2/Digital UNIX/True 64* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from OSF/1 1.2/Digital UNIX" in *Linux Fast-STREAMS Porting Guide*.

— *UnixWare 7.1.3 (OpenUnix 8)*
: *Linux STREAMS (LiS)* provides compatibility functions for source level compatibility with *UnixWare 7.1.3 (OpenUnix 8)* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from UnixWare 7.1.3 (OpenUnix 8)" in *Linux Fast-STREAMS Porting Guide*.

— *Solaris 9/SunOS 5.9*
: *Linux STREAMS (LiS)* provides compatibility functions for source level compatibility with *Solaris 9/SunOS 5.9* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from Solaris 9/SunOS 5.9" in *Linux Fast-STREAMS Porting Guide*.

— *SUPER-UX*
: *Linux STREAMS (LiS)* provides compatibility functions for source level compatibility with *SUPER-UX* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from SUPER-UX" in *Linux Fast-STREAMS Porting Guide*.

— *UXP/V*

Linux STREAMS (LiS) provides compatibility functions for source level compatibility with *UXP/V* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from UXP/V" in *Linux Fast-STREAMS Porting Guide*.

— *LiS-2.16.18*

Linux STREAMS (LiS) provides compatibility functions for source level compatibility with *LiS-2.16.18* and to ease porting of modules and drivers to *Linux STREAMS (LiS)*. Portability considerations are mantained in a separate manual: see section "Porting from Linux STREAMS (LiS) 2.16.18" in *Linux Fast-STREAMS Porting Guide*.

For additional details, see section "About This Manual" in *Linux Fast-STREAMS Porting Guide*.

5 Releases

This is the OpenSS7 Release of the Linux STREAMS (LiS) core, tools, drivers and modules that implement the *Linux STREAMS (LiS)* SVR 4 STREAMS utility for Linux.

The following sections provide information on Linux STREAMS (LiS) releases as well as compatibility information of OpenSS7 release to the original GCOM releases of these modules and drivers, as well as Linux kernel compatibility.

5.1 Prerequisites

Prerequisites for the Linux STREAMS (LiS) package are as follows:

- A fairly LSB compliant GNU/Linux distribution.[1]
- Linux 2.4 or 2.6 kernel (2.4.10 - 2.4.27) or (2.6.3 - 2.6.17)
- glibc2 or better.
- GNU info (for info files).
- GNU groff (for man pages).[2]

If you need to rebuild the package from sources with modifications, you will need a larger GNU toolchain as described in See Section 6.1.6 [Downloading from CVS], page 88.

5.2 Compatibility

This section discusses compatibility with major prerequisites.

5.2.1 GNU/Linux Distributions

Linux STREAMS (LiS) is compatible with the following *Linux* distributions:[3]

- CentOS Enterprise Linux 3.4 (centos34)
- CentOS Enterprise Linux 4.0 (centos4)
- Debian 3.0r2 Woody (deb3.0) – TBD
- Debian 3.1r0a Sarge (deb3.1)
- Fedora Core 1 (FC1) – TBD
- Fedora Core 2 (FC2) – TBD
- Fedora Core 3 (FC3) – TBD
- Fedora Core 4 (FC4)
- Fedora Core 5 (FC5)
- Gentoo 2006.1 (untested)
- Lineox 4.026 (LEL4) – TBD
- Lineox 4.053 (LEL4)
- Mandrakelinux 10.0 (MDK100) – TBD

[1] See Section 5.2.1 [GNU/Linux Distributions], page 67, for more information.

[2] If you are using a Debian release, please make sure to install the groff extension package (‘`groff_ext`’), as it contains the `refer` or `grefer` commands necessary for including references in the manual pages.

[3] Items marked as ‘`TBD`’ are scheduled to have support deprecated. That is, in a future release, the distributions marked ‘`TBD`’ will not longer be validated before release.

- Mandrakelinux 10.1 (MDK101) – TBD
- Mandrakelinux 9.2 (MDK92) – TBD
- Mandriva Linux LE2005 (MDK102) – TBD
- Mandriva Linux LE2006 (MDK103)
- Mandriva One (untested)
- Performance Technlogies *NexusWare24* – TBD
- Performance Technologies NexusWare 8.0
- RedHat Linux 7.2 (RH7)
- RedHat Linux 7.3 (RH7)
- RedHat Linux 8.0 (RH8) – TBD
- RedHat Linux 9 (RH9) – TBD
- RedHat Enterprise Linux 3.0 (EL3)
- RedHat Enterprise Linux 4 (EL4)
- SuSE 8.0 Professional (SuSE8.0) – TBD
- SuSE 9.1 Personal (SuSE9.1) – TBD
- SuSE 9.2 Professional (SuSE9.2) – TBD
- SuSE OpenSuSE (SuSEOSS)
- SuSE 10.0 (SuSE10.0)
- SuSE 10.1 (SuSE10.1) (untested)
- Ubuntu 5.10 (ubu5.10)
- Ubuntu 6.06 LTS (ubu6.06)
- WhiteBox Enterprise Linux 3.0 (WBEL3)
- WhiteBox Enterprise Linux 4 (WBEL4)

When installing from the tarball (see Section 6.4.3 [Installing the Tar Ball], page 111), this distribution is probably compatible with a much broader array of distributions than those listed above. These are the distributions against which the current maintainer creates and tests builds.

5.2.2 Kernel

The *Linux STREAMS (LiS)* package compiles as a *Linux* kernel module. It is not necessary to patch the *Linux* kernel to build or use the package.[4] Nor do you have to recompile your kernel to build or use the package. OpenSS7 packages use `autoconf` scripts to adapt the package source to your existing kernel. The package builds and runs nicely against production kernels from the distributions listed above. Rather than relying on kernel versions, the `autoconf` scripts interrogate the kernel for specific features and variants to better adapt to distribution production kernels that have had patches applied over the official kernel.org sources.

The *Linux STREAMS (LiS)* package is compatible with 2.4 kernel series after 2.4.10 and has been tested up to and including 2.4.27. It has been tested from 2.6.3 up to and including

[4] At a later date, it is possible to move this package into the kernel, however, with continued resistance to STREAMS from within the *Linux* developer community, this is currently unlikely.

2.6.17. Please note that your mileage may vary if you use a kernel more recent than 2.6.17: it is difficult to anticipate changes that kernel developers will make in the future. Many kernels in the 2.6 series now vary widely ny release version and if you encounter problems, try a kernel within the supported series.

UP validation testing for kernels is performed on all supported architectures. SMP validation testing is performed on UP machines, as well as on an Intel 3.0GHz Pentium IV 630 with HyperThreading enabled. Because HyperThreading is not as independent as multiple CPUs, SMP validation testing is limited.

5.2.3 Architectures

The *Linux STREAMS (LiS)* package compiles and installs on a wide range of architectures. Although it is believed that the package will work on all architectures supported by the Linux kernel being used, validation testing has only been performed with the following architectures:

- ix86
- x86_64
- ppc (MPC 860)
- ppc64

32-bit compatibility validation testing is performed on all 64-bit architectures supporting 32-bit compatibility. If you would like to validate an OpenSS7 package on a specific machine architecture, you are welcome to sponsor the project with a test machine.

5.2.3.1 Kernel Version 2.3.x

For LiS version 2.7 and later and for kernel version 2.3.x there are some significant compatibility issues. Version 2.3 of the Linux kernel brings with it some compatibility issues that need to be addressed by the LiS user. The two most important ones concern the file '`<sys/stropts.h>`' and the major device numbers used by LiS.

'`stropts.h`' Compatibility

There are no more compatibility problems with '`<sys/stropts.h>`' with glibc-2.1 and LiS-2.10. The following is more for historical purposes than practical necessity.

Beginning at least with egcs-2.91.66 (egcs-1.1.2 release), which comes with Red Hat 6.0, there is a file in the standard include directory named '`<sys/stropts.h>`'. This file has constant definitions that are incompatible with those used in '`LiS/include/sys/stropts.h`'. If you compile an application against the glibc version of '`stropts.h`', and compile LiS using its own version then certain ioctls may not work correctly. You should be aware of this problem and be sure to include "-I/usr/src/LiS/include" in the compiler options that you use in compiling your STREAMS based applications.

In this version of LiS, some of the constants in '`stropts.h`' have been changed to conform to the values used by UnixWare and Solaris. These are different values than previously used in LiS. When you install LiS the installation procedure will ask you whether you want LiS compiled with the backward-compatible LiS constants, or the UnixWare/Solaris compatible constants. Logically speaking, it does not matter which set you use as long as LiS and your application code are both compiled with the same values.

I highly recommend that you use the UnixWare/Solaris compatible version, however. A future release of egcs, utilizing glibc 2.2, will contain an updated version of its '`stropts.h`' which has constants that are compatible with UnixWare, Solaris and LiS. So by selecting the UnixWare/Solaris compatible version at this time you can ensure that your applications will be fully compatible with these values in the future.

With any luck, these constants will never have to change again.

Major Device Number Compatibility

The second major compatibility issue concerns the major device numbers that LiS assigns to STREAMS devices. In the past LiS based these device numbers at 50, since the Linux kernel did not pre-define many major device numbers. As of kernel version 2.3.x there are major device numbers defined up to 220 and beyone! So starting with LiS-2.12, we have used the major number of 240 as the base for STREAMS device files. This range is supposed to be reserved for "experimental drivers" which should make it safe to use.

What this means is that you must be sure to run the strmakenodes program before running any STREAMS applications after installing LiS-2.12. This need not concern you overly, since doing a "make install" in the '`/usr/src/LiS`' directory causes strmakenodes to be run anyway. This is more a concern if you are compiling LiS on one machine and then loading it onto another for execution. In such cases you may need to load the new strmakenodes program and run it.

I am hoping that the kernel developers will expand the major and minor device number spaces for 2.6. If they do that then LiS should be able to get a block of majors allocated to it.

5.2.3.2 Kernel Version 2.2.x

For LiS version 2.5 and later and for kernel version 2.2.x there are no compatibility issues; there are no kernel patches whatsoever required to install LiS. You will need LiS-2.4 at minimum to run in a 2.2.x kernel.

5.2.3.3 Kernel Version 2.0.36

The latest version of LiS has not been tested on 2.0 kernels. Therefore, do not be surprised if it does not install or execute correctly in these kernels. If you are using an old kernel, you must also use an older version of LiS, perhaps LiS-2.5.

For LiS version 2.5 and later and for kernel version 2.0.36 there are no kernel patches required to run LiS as a "bottom half" process. A one-line patch is required to run LiS as a kernel daemon process. The installation default is to run as a bottom half process in 2.0.36. LiS-1.25 or later should install properly with 2.0.36. The more recent the version of LiS, the less kernel patching is required.

5.2.4 Linux STREAMS

Linux Fast-STREAMS provides a suitable replacement for the (now deprecated) *Linux STREAMS (LiS) 2.18.0* package formerly maintained by Dave Goethe of GCOM.

5.2.4.1 LiS-2.18 Kernel and Driver Compatibility

There are several issues that needed to be addressed for compatibility with the 2.6 Linux kernel. You are encouraged to follow the links in the paragraphs below to see more detailed information on each of these topics.

1. The 2.6 kernel redefined the size of the dev_t structure. LiS has extended its internal dev_t structure to be compatible with the 2.6 method for some time.
2. The 2.6 kernel changed the approach to building and installing kernel modules. This affects LiS as a whole and also affects how you install separate loadable STREAMS drivers. LiS provides a mechanism that allows STREAMS drivers and moduels to be easily installed.
3. The 2.6 kernel offers an option to compile the kernel using machine registers to pass parameters to functions. LiS takes this into account.
4. The 2.6 kernel needs GCC version 3.3.3 (sic) to be compiled properly. LiS needs to be compiled using the same version of the compiler when running with the 2.6 kernel.
5. You may have to edit the file /etc/rc.d/rc.sysinit to get demand loadable modules to work correctly. This is especially true when hosting a 2.6 kernel on a 2.4 distribution.

5.2.4.2 LiS-2.16 Kernel and Driver Compatibility

LiS-2.16 is a small change from LiS-2.15. The change is that it no longer uses Linux system calls to implement getpmsg and putpmsg. Instead it overloads the read and write file system functions with particular values for the count parameter, values that are otherwise invalid.[5]

5.2.4.3 LiS-2.15 Kernel and Driver Compatibility

LiS-2.15 continues to insulate STREAMS drivers from the Linux kernel. It works with 2.2, 2.4, and 2.5 versions of the kernel. Support for 2.0 kernels has been dropped.

Driver writers will need to recompile their drivers against LiS-2.15 include files. You will see the following major changes.

- LiS spin locks and semaphores have been rearranged so that the kernel memory is at the end of the structure instead of the beginning.
- The former change allows for there to be dynamic allocation routines for spin locks and semaphores. (see Section 3.6.28 [LiS Spin Locks], page 49)
- LiS now provides an abstraction for read/write locks, with dynamic allocation. (see Section 3.6.30 [LiS Read/Write Locks], page 52)
- Those experimenting with 2.5 kernels will notice that the "sleep while holding spin lock" problems have been fixed.
- Porting to 2.5 has necessitated some changes to the major/minor device structure handling. (see Section 3.6.16 [Major/Minor Device Numbering], page 42)
- The fattach related functions are functional on kernels version 2.4.7 and later.
- STREAMS pipes and FIFOs are now functional.

[5] This change is far from small because it outdates '`libLiS.a`' and '`libLiS.so`'. A '`libLiS.a`' or '`libLiS.so`' from a previous version will not work correctly. All applications statically linking '`libLiS.a`' must be recompiled to use a '`libLiS.a`' from the more recent version. Unfortunately, LiS did not include versioning on its libraries. This has been corrected with the OpenSS7 release of LiS.

- OS interface code has been added for the kernel's DMA mapping functions.

There is one known bug in LiS-2.15 relative to 2.5 kernels. It has to do with a memory leak involving timer structures, and may prove to be a kernel bug rather than an LiS bug. Since the 2.5 kernel is not suitable for general use I am saving the investigation of this bug for later.

5.2.4.4 LiS-2.14 Kernel and Driver Compatibility

LiS-2.13 was a series of beta releases. LiS-2.14 represents the culmination of this series. There should be enough distribution and kernel compatibility that LiS-2.14 will hold up for some time.

The known fattach and FIFO bugs have still not been fixed. The author of those subsystems has not found the time to put in the fixes, nor have I.

5.2.4.5 LiS-2.13 Kernel and Driver Compatibility

This version of LiS has been tested with 2.4 kernels up to 2.4.16. LiS does not yet support the fattach/fdetach functions on kernel versions 2.4.7 and beyond. There are also known bugs in the LiS pipe/FIFO code. All of these problems are scheduled to be fixed in early 2002.

LiS-2.13 adds the ability for drivers to make their own "/dev" nodes via the *lis_mknod* function (see Section 3.6.40 [System Calls from within the Kernel], page 62). Also provided is an *lis_unlink* function that allows drivers to remove their device files.

There is almost no new functionality added by LiS-2.13. The differences between LiS-2.13 and LiS-2.12 are almost entirely kernel compatibility issues and bug fixes.

5.2.4.6 LiS-2.12 Kernel and Driver Compatibility

This version of LiS is compatible with all 2.2.x versions of the kernel and with early versions of the 2.4.x kernel, at least up to 2.4.2 and perhaps later versions as well.

If you have drivers that have worked with LiS-2.10 or LiS-2.11 (or earlier) please recompile them using the header files from LiS-2.12. This may be the last recompile in quite some time that you will need for your driver code.

LiS-2.12 contains a sufficient Driver/Kernel Interface (DKI), (see Section 3.6 [Development], page 33), that it is straightforward to write a STREAMS driver that can be compiled against LiS-2.12 and the resulting object modules used either on a 2.2 or 2.4 kernel, with only LiS needing recompilation on the target machine.

When run on 2.4 kernels, LiS makes full use of multiple CPUs (see Section 3.6.2 [LiS SMP Implementation], page 35). It forks a queue runner task for each CPU and locks each task onto its CPU. Queue runner tasks are awakened to assist with service procedure processing as the number of scheduled queues increases.

Because of this aggressive use of processors, you may find that your drivers do not function properly when run with LiS-2.12 in a multi-CPU SMP environment. You should expect that drivers that worked in single-CPU environments will continue to work as before.

Making your drivers MP safe involves the use of spin locks. The DKI documentation contains advice on the use of these locks. See Section 3.6.28 [LiS Spin Locks], page 49.

This version of LiS also contains a rewrite of the flushing code and tests added to strtst for flushing. In particular the details of the rules for flushing queue bands are now adhered to. See Section 3.6.38 [Flushing Queue Bands], page 59. Be advised, however, that Solaris STREAMS does not adhere strictly to these rules so there may be some subtle differences in behavior between LiS and Solaris when flushing queue bands.

Speaking of queue bands, the queue band handling code has been debugged a bit more and a test added to strtst to illustrate its correct behavior.

5.2.4.7 LiS-2.10 Kernel and Driver Compatibility

This version of LiS is compatible with all 2.2.x versions of the Linux kernel. It may work with 2.4.x kernels, but you should probably wait for LiS-2.11 for that.

If you have drivers that worked with LiS-2.8 or earlier, you must recompile your drivers in the context of the LiS-2.10 header files. The queue_t structure has changed in size since LiS-2.8 which means that the old RD and WR macros will not compute the correct addresses.

LiS-2.10 contains features that are intended to greately reduce the necessity of recompiling STREAMS driver code in future versions of LiS or future versions of the kernel. The goal is to be able to compile STREAMS drivers against LiS-2.10 header files and use the resulting object code on both 2.2.x kernels and 2.4.x kernels.

For more details about the interface between STREAMS drivers and the kernel, see the Driver/Kernel Interface documentation, (see Section 3.6 [Development], page 33).

5.2.5 Linux Fast-STREAMS

The *Linux STREAMS (LiS)* package is no longer receiving active development or support. The *Linux STREAMS (LiS)* package is so fraught with bugs that it is unusable as far as The OpenSS7 Project is concerned. *Linux Fast-STREAMS* is the preferred replacement for *Linux STREAMS (LiS)*.

5.3 Release Notes

The sections that follow provide information on OpenSS7 releases of the
Linux STREAMS (LiS) package.

5.3.1 Release LiS-2.18.4.rc2

Added '`--enable-devel`' `configure` option for embedded targets. Added `send-pr` script for automatic problem report generation.

5.3.2 Release LiS-2.18.2

Corrections for and testing of 64-bit clean compile and test runs on x86_64 architecture. Some bug corrections resulting from gcc 4.0.2 compiler warnings.

Corrected build flags for Gentoo and 2.6.15 kernels as reported on mailing list. Builds on FC4 2.6.15 kernel and with gcc 4.0.2.

Added in many of Paul's 64-bit corrections.

The *Linux STREAMS (LiS) 2.18.3* is still largely unusable on 64-bit or SMP kernels. **Linux STREAMS (LiS) package is deprecated. Do not use it. The package contains many unresovled bugs. Use *Linux Fast-STREAMS* instead.**

5.3.3 Release LiS-2.18.2

Cross-build support for newer *NexusWare* releases. Build support for (recent FC4) 2.6.14 kernel. Corrected installation support (init scripts) for SuSE 9.2.

Binary compatibility backwards compatible to GCOM 2.18.0 included. This includes exported symbols changed to not generate versioned symbols on 2.4 kernels. Also, exported symbols are always compiled `attribute((regparm(0)))` on regparm capable architectures, regardless of the kernel version or compile options.[6] For actual binary compatibilty packaging, see the '`strcompat`' package.

A major change for 2.18.2 is the port-back of POSIX/SUSv3 XSR/XSI conformance test suites and performance programs from Linux Fast-STREAMS. The purpose of porting back theses tests suites and supporting modules and drivers is to provide the ability to do comparison tests between LiS and Linux Fast-STREAMS.

Another change is a module unloading safe vstrlog hook register and unregister functions register_strlog() and unregister_strlog().

Some bug corrections and fixes for glaring SMP errors reported by Kutluck.

This might be the last OpenSS7 release of *Linux STREAMS (LiS)*. You should seriously consider using the Linux Fast-STREAMS package (streams-0.7a.4 or later) instead. If you need comptibility to LiS (or other STREAMS implementation), investigate the strcompat package, which provides some binary compatibility to LiS under Linux Fast STREAMS.

5.3.4 Release LiS-2.18.1

Initial autoconf/RPM packaging of the `LiS` release.

This is a port forward of most of the build and patches from 2.16.19 forward and applied over 2.18.0. This is our first LiS-2.18 release. All further development on 2.16.19 will now cease. 2.18.1 is maintained on both 2.4 and 2.6 kernels. No active development will be performed on 2.18.1, only maintenance. For an active development release, see the Linux Fast-STREAMS releases.

Major changes from LiS-2.18.0 include all of the autoconf build system, manual pages and texi/pdf manual for LiS that were applied on the 2.16.19 release. This includes a number of 64 bit, HPPA, PARISC, printf, atomic stats, HZ calculations for 64bit machines, DMA patch for mblk buffer alignment, flush handling patch, panic patch, smp patch, parisc syscall patch, appq patch, and multithreaded test program patches, POSIX threads compilant library functions.

Additional changes made to support later 2.6 kernels and distributions. Switched putpmsg()/getpmsg() to use ioctl for system call emulation instead of read()/write(), primarily because 2.6.11 kernels check for a valid count before calling the driver's read()/write() file operations. Updates to the build system to support a wider range of kernels and distributions. See the installation and reference manual for a complete list of supported kernels and distributions.

Please note that the entire package is released under GPL.

5.3.5 Release LiS-2.16.19

Not publicly released.

[6] Regparm capable architectures are really just `__i386__` and similar such as `__x86_64__` and `k8`.

5.3.6 Release LiS-2.16.18-22

Replaced m4 and automake files with common equivalents. This allows the same set of m4 macros and automake fragments to be used with all of the OpenSS7 release packages. Maintenance is easier as one correction will propagate across all items. Performed similar function with texinfo documentation pieces.

5.3.7 Release LiS-2.16.18-21

Removed all XTI/TLI and Linux networking code, headers and documentation from LiS distribution and bumped epoch to 2. Linux networking code has been migrated to the **strxns**, **strxnet**, **strinet** and **strsctp** packages. The purpose for doing this was to allow the Linux networking to build against Linux Fast-STREAMS as well as *Linux STREAMS (LiS)* and is a preparation for phasing out LiS and phasing in LfS.

Added missing '`configure.nexusware`' to distribution. LiS cache options now default to 'no' because of instabilities with timers.

Not publicly released.

5.3.8 Release LiS-2.16.18-20

Minor corrections: made conflicting manpage '`xti_sctp.3`' dependent on OpenSS7 SCTP kernel.

Not publicly released.

5.3.9 Release LiS-2.16.18-19

Changes to compile, install and builds rpms for Fedora Core 1 (FC1), Whitebox Enterprise Linux (WBEL) and RedHat Enterprise Linux 3 (EL3). Included explicit epoch in internal dependencies in spec file for RPM versions 4.2.1, 4.2.2 and higher. Added hugemem kernel detection and moved getpmsg and putpmsg manual pages.

Correction to symbolic linking and system map file location during non-rpm autoconf installation.

Correction to zero `maxlen` behavior in `t_rcvconnect()`.

5.3.10 Release LiS-2.16.18-18

Added check for `CONFIG_REGPARM`, addition of `-mregparm=3 CFLAGS`, addition of `regparm_` prefix for exported ksyms.

Minor corrections to separate build directory install of devices and caching of detected ksyms.

5.3.11 Release LiS-2.16.18-17

Added option `--disable-k-modversions` to supress versioning of LiS exported symbols.

A couple of corrections to the build process reported by Gurol. Changed order of build in 'make rebuild' target to build tools last so that the rpm debug package is built correctly on RH9.

Change `MODULE_PARM` to static so that `make install-strip` does not strip module parameter symbols.

Added `lis_check_mem_region()`, `lis_release_mem_region()` and `lis_request_mem_region()` for memory mapped io instead of just io.

Added `printk` patches discussed on linux-stream mailling list. Added gcc `printf` checking and corrected errors in LiS debugging `printk` statements.

Added HP patches. There are a couple of questionable components in the HP patches that I reversed. They include;

- modification of `lis_msgsize` to `lis_msgdsize`. This would change the calculation of queue counts. Queue counts aren't `M_DATA` counts, they are "data" message counts. LiS probably doesn't have this the right way, but `lis_msgdsize` is not correct either.
- addition of `qi_mstat->ms_pcnt` increment on `lis_safe_putmsg`. Same for `ms_scnt`, `ms_ocnt`, `ms_ccnt`. STREAMS is not supposed to increment counts. It is the module writer's responsibility to increment counts in their own queue procedures.

Added HP `ldl` patches.

Made modifications to `putq()`, `putbq()`, `insq()` and `appq()` discussed on linux-streams mailing list. These do not free messages on failure. Modified all ocurrences internal to LiS to free the message on error to ensure old behavior.

Added HP dejagnu patches to `strtst` and added dejagnu testsuite directory and file. Added the `make check` target. Use `DEJATOOLS` on the make command line to invoke the tests, such as '`make DEJATOOLS=strtst check`' to invoke the tests. Because a patched `netperf` is not commonly available and `netperf` will not be distributed with the package, GNU `autotest` might be a better choice. But that's for a later release.

5.3.12 Release LiS-2.16.18-16

General updates to the build process, optimization options, build options. Corrected library linkage. Synced TLI modules and INET driver to Linux Fast-STREAMS. Removed deadlock from INET driver and loosened locking. Unfortunately suitable libraries must be installed before distcheck will clear.

Smoother and more reliable stripping of kernel symbols, starts with /proc/ksyms if applicable then System.map then modversions.h to attempt to choose symbols most closely synced with an installed or running kernel.

Improvements to autoconf installation of manpages (autocompressed now) and info and pdf manuals are distributed. install-strip target will actually properly strip kernel modules.

Included an option to build and install only kernel or user parts of package to speed rpm rebuild process for multiple kernel. Added 'rebuild' target to rebuild the rpms from srpm for multiple kernel and architectures. Added a 'sign' and 'resign' target to sign srpm and rebuilt rpms respectively.

Greatly enhanced cross-build and cross-compile support, primarily in support of the NexusWare embedded target. Added NexusWare helper script and documentation. DESTDIR is now a blessed environment variable used by configure to set the cross-build root as well as the install root. Try adding –with-k-optimize='size' to configure to optimize for size for embedded targets. Builds clean against NexusWare24 (810p0674.10-rc4).

Added start of an option to build as linkable object for embedded targets rather than loadable kernel module.

5.3.13 Release LiS-2.16.18-15

Fixed several symbol errors that made -13 and -14 unusable. Corrected error in calculation of kernel debug flags.

5.3.14 Release LiS-2.16.18-14

A few more enhancements to the build process to work with autoconf 2.59.

5.3.15 Release LiS-2.16.18-13

Enhanced build process for autoconf-2.59, automake-1.8.3, gettext-0.14.1, and libtool-1.5.6.

5.3.16 Release LiS-2.16.18-12

Added defaults for SK_WMEM_MAX and SK_RMEM_MAX for lastest 2.4.25 and 2.4.26 kernel builds.

Enhanced build process.

All kernel symbols exported by LiS are versioned on kernels that have versioned symbols. This makes it safer to compile kernel modules against kernel/LiS combinations. This is in preparation for splitting off the strxnet package, and the technique was imported from the Linux Fast-STREAMS build.

5.3.17 Release LiS-2.16.18-11

Ripped three additional kernel symbols in support of INET driver that were missing in -10 release.

5.3.18 Release LiS-2.16.18-10

Added support for cooked manpages both for non-rpm systems and for rpm systems. It is still better to leave manpages uncooked for rpm releases because they are much smaller that way. Give the –with-cooked-manpages flag to configure if you want cooked manpages. You still need grefer on the build system.

Updates to all manual pages in man7, and some others (xti) in man3. Removed unused .macros and .refs files.

Moved automake fragments into separate directory. Cleaned up automake fragments.

Rearranged header files in the xti subdirectory to install in LiS package include directory instead. Reworked xti, tihdr and tiuser file groups to include properly from kernel or user space independent of order. tiuser and xti still cannot be included together. Added older TLI interface <tiuser.h> that is still consistent with newer XTI interface. Changed references in man pages to XTI/TLI instead of just XTI.

Added ticlts.h, ticots.h and ticotsord.h header files. Updated dlpi.h and npi.h header files. Removed sys/LiS/tpicommon.h because it is largely replaced by sys/tli.h and sys/tpi.h. Removed the, now redundant, xti header file subdirectory.

A series of bug fixes to xnet.c (libxnet) that resulted from discussions with Gurol Akman on openss7-develop mailing list. Mostly surrounding t_alloc and t_getinfo behaviour and the behavior when NULL pointers are passed to various XTI/TLI library calls. Updated xti documentation as well.

Many changes to the inet.c INET driver. Wildcard IP addresses can now be bound and wildcard addresses will be assigned with no address is passed to most providers. (/dev/rawip still requires an address or TNOADDR is returned.) Option management has been extensively rewritten to be more conformant to XNS documentation. Test programs test-inet_raw, test-inet_udp, test-inet_tcp have been upgraded and converted to multiple child processes. A number of fixes to SMP lock behavior and M_FLUSH have beend added as reported by Dave Grothe. Corrected all level and TBADOPT behavior on negotiation.

Although this driver is now closer to expected behavior, it has not been tested for XNS 5.2 compliance, nor will it be until someone has the time to extend the test programs to handle all test cases in a similar manner as was done for the library. Your mileage many vary. Remember, there is no warranty.

5.3.19 Release LiS-2.16.18-9

Changes primarily in support of builds on HPPA (PARISC) architectures. LiS doesn't build too well on PARISC so some modifications where used from the Linux Fast-STREAMS package to correct deficiencies. Better building on recent 2.4 kernels (2.4.23, 2.4.24, 2.4.25) is also provided.

5.3.20 Release LiS-2.16.18-8

Changes to permit better builds on recent RedHat kernels, and especially kernel-2.4.20-30.9.

5.3.21 Release LiS-2.16.18-7

Fixed a module loading bug in LiS. Previously modules would not demand load.

5.3.22 Release LiS-2.16.18-6

Fixed a possible null pointer dereference in libxnet. Corrected t_bind to return TNOADDR instead of TBADADDR on wildcard bind attempt. Module loading bug patched.

5.3.23 Release LiS-2.16.18-5

Fixes a t_open and t_bind problem in libxnet. Fixes alignemnt of data portion of mblks. Adds (untested) ticots_ord, ticots and ticlts devices over UNIX domain sockets.

5.3.24 Release LiS-2.16.18-4

Adds back in missing strms_up/down/status scripts to distribution and install.

5.3.25 Release LiS-2.16.18-3

Not publicly released.

5.3.26 Release LiS-2.16.18-2

Not publicly released.

5.3.27 Release LiS-2.16.18-1

This OpenSS7 release of LiS-2.16.18 updates the previous LiS-2.16.16 rpm release to the lastest LiS-2.16 release level.

5.3.28 Release LiS-2.16.16-1

This OpenSS7 release of LiS-2.16.16 includes autoconf for configuration, complete manual pages and documentation, and packaging in source and binary RPMs for ease and repeatability of installation. The package also builds and installs properly versioned LiS shared object libraries.

Before the OpenSS7 release of LiS, it was necessary to have a significant working knowledge of the Linux kernel, kernel source, headers and other intricacies. This made it difficult to distribute software based on LiS to users not proficient in those areas. The OpenSS7 release removes the configuration and installation tasks from the user and permits distribution of applications, modules and driver software based on LiS to users without sufficient kernel expertise to install the package.

This OpenSS7 release fixes few of the outstanding bugs and deficiencies of the LiS software. This release is intended to package and distribute LiS in an efficient manner and, for the most part, does not address LiS deficiencies or errors.

This OpenSS7 release is compatible with Linux 2.4 kernels only and will refuse to configure for older or newer kernels.

Following are the new features of the OpenSS7 release of LiS:

- adds configuration using the GNU tools, autoconf, automake and autotest.

  These tools greatly enhance the ability to maintain a repeatable and testable release cycle as well as being compatible with most major package managers such as Redhat's RPM.
- adds long options to all LiS utilities.

  This change was necessitated because we use GNITS (the strictest level) of configuration with autoconf that requires for distribution checking that all utility programs support the '`--help`' and '`--version`' long options wtihout side-effects.
- provides a source and binary release mechanism using both autoconf distributions as well as RedHat source and binary RPMs.

  Use of the RPM mechanism for release permits add-on packages to ensure that they have sufficient level of support and verionsing of the LiS load during their build and installation process. It is now also possible to ensure that add on binaries are compatible with a loaded LiS during installation.
- includes a complete set of kernel programmer manual pages for all LiS exported kernel functions for use by STREAMS module and driver developers.
- includes a complete set of user manual pages for all libLiS functions and separate administrative utilities.
- includes the OpenSS7 strinet driver providing XTI/TLI access to the Linux native NET4 IP stack including TCP, UDP, IP and (and OpenSS7 SCTP if your kernel is so equipped).
- includes functional `tirdwr` and `timod` modules for use with the included XTI/TLI library.[7]
- includes complete, thread-safe XNS 5.2 XTI/TLI library support with the '`libxnet`' library, complete manual pages and documentation released under the LGPL (see [GNU Lesser General Public License], page 139).

The next release may include some *strss7* software.

[7] The `tirdwr` module included with the Gcom LiS-2.16.18 (and even more current) releases is almost completely disfunctional and has been replaced in entirety.

5.4 Maturity

The *OpenSS7 Project* adheres to the following release philosophy:

- pre-alpha release
- alpha release
- beta release
- gamma release
- production release

5.4.1 Pre-Alpha Releases

Pre-alpha releases are releases that have received no testing whatsoever. Code in the release is not even known to configure or compile. The purpose of a pre-alpha release is to make code and documenation available for insepection only, and to solicit comments on the design approach or other characteristics of the software package.

Pre-alpha release packages ship containing warnings recommending that the user not even execute the contained code.

5.4.2 Alpha Releases

Alpha release are releases that have received little to no testing, or that have been tested and contains known bugs or defects that make the package unsuitable even for testing. The purpose for an *alpha* release are the same as for the pre-alpha release, with the additional purpose that it is an earily release of partially functional code that has problems that an external developer might be willing to fix themselves and contribute back to the project.

Alpha release packages ship containing warnings that executing the code can crash machines and might possibly do damage to systems upon which it is executed.

5.4.3 Beta Releases

Beta releases are releases that have received some testing, but the testing to date is not exhaustive. *Beta* release packages do not ship with known defects. All known defects are respolved before distribution; however, as exhaustive testing has not been performed, unknown defects may exist. The purpose for a *beta* release is to provide a baseline for other organizations to participate in the rigorous testing of the package.

Beta release packages ship containing warnings that the package has not been exhaustively tested and that the package may cause systems to crash. Suitability of software in this category for production use is not advised by the project; however, as always, is at the discretion of the user of the software.

5.4.4 Gamma Releases

Gamma release are releases that have received exhaustive testing within the project, but external testing has been minimal. *Gamma* release packages do not ship with known defects. As exhaustive internal testing has been performed, unknown defects should be few. Please remember that there is NO WARRANTY on public release packages.

Gamma release packages typically resolve problems in previous *beta* releases, and might not have had full regression testing performed. Suitability of software in this category for production use is at the discretion of the user of the software. *The OpenSS7 Project* recommends that the complete validation test suites provided with the package be performed and pass on target systems before considering production use.

5.4.5 Production Releases

Production releases are releases that have received exhaustive testing within the project and validated on specific distributions and architectures. *Production* release packages do not ship with known defects. Please remember that there is NO WARRANTY on public release packages.

Production packages ship containig a list of validated distributions and architecutres. Full regression testing of any maintenance changes is performed. Suitability of software in this category for production use on the specified target distributions and architectures is at the discretion of the user. It should not be necessary to preform validation tests on the set of supported target systems before considering production use.

5.5 Bugs

5.5.1 Defect Notices

Linux STREAMS (LiS), both releases from *OpenSS7* and *GCOM*, contany many known bugs. These are unstable releases. Although there are no bugs known to be harmful, the *OpenSS7 Project* has tested the release and found defects that cause the kernel to lock or crash. Use at your own risk. Remember that there is **NO WARRANTY**[8] and that the package is no longer actively maintained.

This software has tested *unstable*. As such, it can crash your kernel. Installation of the software could irreparably mangle your header files or Linux distribution in such a way as to make it unusable. Crashes can lock your system and rebooting the system might not repair the problem. You can loose all the data on your system. Because this software locks or crashes under a large number of test cases in the test suite, simply running the test cases can cause locks or crashes. The resulting unstable system could destroy computer hardware or peripherals making them unusable. You could void the warranty on any system on which you run this software. YOU HAVE BEEN WARNED.

5.5.2 Known Defects

With the exception of packages not originaly created by the *OpenSS7 Project*, the *OpenSS7 Project* software does not ship with known bugs in any release stage except *pre-alpha*. *Linux STREAMS (LiS)* had many known bugs at the time of release.

Linux STREAMS (LiS) has many known bugs. Under some architectures, the test cases in the conformance test suite cause the kernel to lock or crash. *Linux STREAMS (LiS)* contains many races and defects and is unsuitable for production environments. This section provides a summary of some (but not all) known defects.

1. A substantial group of test cases in the *POSIX* conformance test suite fail. This is largely due to non-fatal defects in LiS.
2. A number of test cases fail under any architecture and result in a kernel lock. In particular if a significant number of modules are pushed onto a Stream using I_PUSH,

[8] See section **NO WARRANY** under [GNU General Public License], page 133.

the kernel will lock when the Stream is closed. The number of modules pushed to cause a crash depends on the speed of the machine upon which the test case is run. The test case in the test suite pushes 64 modules and has always resulted in a kernel lock regardless of the machine speed upon which it was tested. Pushing 20 modules results in a kernel lock on some of the *OpenSS7 Project* 2.57GHz test machines.

As a result, any test case that pushes a number of modules, and the performance tests (that push modules for measurement) will cause the kernel to lock.

3. A large number of test cases fail when running under an SMP kernel, regardless of the number of processors on the test system. These test cases cause the kernel to lock. The kernel locks are apparentlhy due to locking defects in the implementation. None of these implementation defects have been repaired.
4. The original '`LiS-2.18.0`' release from *GCOM* has a large number of defects that were repaired in the *OpenSS7* '`LiS-2.18.1`' release.
5. The original *LiS-2.18.0* release has defects in the description of data types and handling under 64-bit architectures. 32-bit compatibility for 64-bit architectures is all but non-existent in the LiS-2.18.0 release. Because of binary compatibility issues, many of these defects persist in the *OpenSS7* '`LiS-2.18.1`' and '`LiS-2.18.2`' releases. LiS is largely unusable in a 64-bit and almost completely unusable in a mixed architecture.

The work-around for these defects is to not use LiS at all: use the *OpenSS7 Linux Fast-STREAMS* release instead. The *OpenSS7 Linux Fast-STREAMS*, being a completely independent implemenation, does not suffer from this extensive set of LiS defects.

5.6 Schedule

5.7 History

6 Installation

6.1 Downloading

The Linux STREAMS (LiS) package releases can be downloaded from the downloads page of The OpenSS7 Project. The package is available as a binary RPM (for popular architectures) a source RPM, Debian binary DEB and source DSC, or as a tar ball. If you are using a browsable viewer, you can obtain the OpenSS7 release of `LiS` from the links in the sections that follow.

By far the easiest form for installing and using `LiS-2.18.4.rc2` is to download and install binary RPM. If a binary RPM is not available for your distribution, but your distribution supports RPM, the next best method for installing and using `LiS-2.18.4.rc2` is to download and rebuild the source RPM. If your architecture does not support RPM at all, or you have special needs (such as cross-compiling for embedded targets), the final resort method is to download, configure, build and install from the source tarball.

6.1.1 Downloading the Binary RPM

To install from binary RPM, you will need several of the RPM for a complete installation. Binary RPM fall into several categories. To download and install a complete package requires the appropriate RPM from each of the several categories below.

To install from Binary RPM, you will need all of the following kernel indepdendent packages for your architecture, and one of the kernel-depdendent packages from the next section.

Independent RPM

Independent RPM are not dependent on the Linux kernel version. For example, the source package '`LiS-source-2.18.4.rc2-1.FC5.noarch.rpm`', is not dependent on kernel.

All of the following kernel independent RPM are required for your architecture. Binary RPMs listed here are for example only: additional binary RPMs are available from the downloads site. If your architecture is not available, you can build binary RPM from the source RPM (see see Section 6.3.1 [Building from the Source RPM], page 105).

Architecture Independent

LiS-dev-2.18.4.rc2-1.FC5.noarch.rpm
: The '`LiS-dev`' package contains the device definitions necessary to run applications programs developed for Linux STREAMS (LiS).[1]

LiS-doc-2.18.4.rc2-1.FC5.noarch.rpm
: The '`LiS-doc`' package contains this manual in plaintext, postscript, PDF and HTML forms, along with the meta-information from the '`LiS`' package. It also contains all of the manual pages necessary for developing Linux STREAMS (LiS) applications and Linux STREAMS (LiS) STREAMS modules or drivers.

[1] Not all distributions support the '`%dev`' RPM macro: a case in point is the SuSE 8.0 distribution which uses an older version of `rpm`. Distributions that do not support the '`%dev`' macro will build devices as a '`%post`' operation. Note also that not all release packages contain devices. Only packages that provide STREAMS character device drivers need devices, and then only when the '`specfs`' or '`devfsd`' is not being used.

LiS-init-2.18.4.rc2-1.FC5.noarch.rpm
: The '`LiS-init`' package contains the init scripts and provides the postinst scripts necessary to create kernel module preloads and modules definitions for all kernel module '`core`' subpackages.

LiS-source-2.18.4.rc2-1.FC5.noarch.rpm
: The '`LiS-source`' package contains the source code necessary for building the Linux STREAMS (LiS) release. It includes the `autoconf` configuration utilities necessary to create and distribute tarballs, rpms and deb/dscs.

Architecture Dependent

LiS-devel-2.18.4.rc2-1.FC5.i686.rpm
: The '`LiS-devel`' package contains library archives for static compilation, header files to develop Linux STREAMS (LiS) modules and drivers. This also includes the header files and static libraries required to compile Linux STREAMS (LiS) applications programs.

LiS-lib-2.18.4.rc2-1.FC5.i686.rpm
: The '`LiS-lib`' package contains the run-time shared libraries necessary to run application programs and utilities developed for the '`LiS`' package.

LiS-util-2.18.4.rc2-1.FC5.i686.rpm
: The '`LiS-util`' package provides administrative and configuration test utilities and commands associated with the Linux STREAMS (LiS) package.

Kernel-Dependent RPM

Kernel-Dependent RPM are dependent on specific Linux Kernel Binary RPM releases. Packages are provided for popular released *RedHat* kernels. Packages dependent upon *RedHat* or other kernel RPM will have the '`_kversion`' kernel package version in the package name.

One of the following Kernel-Dependent packages is required for your architecture and kernel version. If your architecture or kernel version is not on the list, you can build binary RPM from the source RPM (see see Section 6.3.1 [Building from the Source RPM], page 105).[2]

LiS-core-2.6.17-1.2139˙FC5-2.18.4.rc2-1.FC5.i686.rpm
: The '`LiS-core`' package contains the loadable kernel modules that depend only on the kernel. This package is heavily tied to the kernel for which it was compiled. This particular package applies to kernel version '`2.6.17-1.2139_FC5`'.[3]

LiS-info-2.6.17-1.2139˙FC5-2.18.4.rc2-1.FC5.i686.rpm
: The '`LiS-info`' package[4] contains the module symbol version information for the '`core`' subpackage, above. It is possible to load this subpackage and com-

[2] Note that on *Mandrakelinux*, unlike other RPM kernel distributions, kernel packages for the ix86 architectures are always placed in i586 architecture packages regardless of the true processor architecture of the kernel package. `configure` detects this and builds the appropriate packages.

[3] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example. Note also that only release packages that contain kernel modules will contain a '`core`' subpackage.

[4] Note that only release packages that contain kernel modules and that export versioned symbols will contain a '`info`' subpackage. Also, this subpackage is only applicable to 2.4 series kernels and is not necessary and not built for 2.6 series kernels.

pile modules that use the exported symbols without loading the actual kernel modules (from the '`core`' subpackage above). This package is heavily tied to the kernel for which it was compiled. This particular package applies to kernel version '`2.6.17-1.2139_FC5`'.[5]

Configuration and Installation

To configure, build and install the binary RPM, See Section 6.2.1 [Configuring the Binary RPM], page 90.

6.1.2 Downloading the Debian DEB

To install from binary DEB, you will need several of the DEB for a complete installation. Binary DEB fall into several categories. To download and install a complete package requires the appropriate DEB from each of the several categories below.

To install from Binary DEB, you will need all of the following kernel indepdendent packages for your architecture, and one of the kernel-depdendent packages from the next section.

Independent DEB

Independent DEB are not dependent on the Linux kernel version. For example, the source package '`LiS-source_2.18.4.rc2-0_i386.deb`', is not dependent on kernel.

All of the following kernel independent DEB are required for your architecture. Binary DEBs listed here are for example only: additional binary DEBs are available from the downloads site. If your architecture is not available, you can build binary DEB from the Debian DSC (see see Section 6.3.2 [Building from the Debian DSC], page 106).

Architecture Independent

LiS-dev_2.18.4.rc2-0_all.deb
: The '`LiS-dev`' package contains the device definitions necessary to run applications programs developed for Linux STREAMS (LiS).[6]

LiS-doc_2.18.4.rc2-0_all.deb
: The '`LiS-doc`' package contains this manual in plaintext, postscript, PDF and HTML forms, along with the meta-information from the '`LiS`' package. It also contains all of the manual pages necessary for developing Linux STREAMS (LiS) applications and Linux STREAMS (LiS) STREAMS modules or drivers.

LiS-init_2.18.4.rc2-0_all.deb
: The '`LiS-init`' package contains the init scripts and provides the postinst scripts necessary to create kernel module preloads and modules definitions for all kernel module '`core`' subpackages.

LiS-source_2.18.4.rc2-0_all.deb
: The '`LiS-source`' package contains the source code necessary for building the Linux STREAMS (LiS) release. It includes the `autoconf` configuration utilities

[5] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example.

[6] Note that not all release packages contain devices. Only packages that provide STREAMS character device drivers need devices, and then only when the '`specfs`' or '`devfsd`' is not being used.

necessary to create and distribute tarballs, rpms and deb/dscs. !ignore[7] !end ignore

Architecture Dependent

LiS-devel_2.18.4.rc2-0_i386.deb
: The '`LiS-devel`' package contains library archives for static compilation, header files to develop Linux STREAMS (LiS) modules and drivers. This also includes the header files and static libraries required to compile Linux STREAMS (LiS) applications programs.

LiS-lib_2.18.4.rc2-0_i386.deb
: The '`LiS-lib`' package contains the run-time shared libraries necessary to run application programs and utilities developed for the '`LiS`' package.

Kernel-Dependent DEB

Kernel-Dependent DEB are dependent on specific Linux Kernel Binary DEB releases. Packages are provided for popular released *RedHat* kernels. Packages dependent upon *RedHat* or other kernel DEB will have the '`_kversion`' kernel package version in the package name.

One of the following Kernel-Dependent packages is required for your architecture and kernel version. If your architecture or kernel version is not on the list, you can build binary DEB from the source DEB (see see Section 6.3.2 [Building from the Debian DSC], page 106).[8]

LiS-core-2.6.17-1.2139˙FC5_2.18.4.rc2-0_i386.deb
: The '`LiS-core`' package contains the loadable kernel modules that depend only on the kernel. This package is heavily tied to the kernel for which it was compiled. This particular package applies to kernel version '`2.6.17-1.2139_FC5`'.[9]

LiS-info-2.6.17-1.2139˙FC5_2.18.4.rc2-0_i386.deb
: The '`LiS-info`' package[10] contains the module symbol version information for the '`core`' subpackage, above. It is possible to load this subpackage and compile modules that use the exported symbols without loading the actual kernel modules (from the '`core`' subpackage above). This package is heavily tied to the kernel for which it was compiled. This particular package applies to kernel version '`2.6.17-1.2139_FC5`'.[11]

Configuration and Installation

To configure, build and install the Debian DEB, See Section 6.2.2 [Configuring the Debian DEB], page 91.

---

[7] Note that not all releases have source DEB packages. Release packages that do not contain kernel modules do not generate a source DEB package.

[8] Note that on *Mandrakelinux*, unlike other DEB kernel distributions, kernel packages for the ix86 architectures are always placed in i586 architecture packages regardless of the true processor architecture of the kernel package. `configure` detects this and builds the appropriate packages.

[9] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example. Note also that only release packages that contain kernel modules will contain a '`core`' subpackage.

[10] Note that only release packages that contain kernel modules and that export versioned symbols will contain a '`info`' subpackage. Also, this subpackage is only applicable to 2.4 series kernels and is not necessary and not built for 2.6 series kernels.

[11] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example.

6.1.3 Downloading the Source RPM

If you cannot obtain a binary RPM for your architecture, or would like to roll you own binary RPM, download the following source RPM.

LiS-2.18.4.rc2-1.src.rpm
: This is the source RPM for the package. From this source RPM it is possible to build binary RPM for any supported architecture and for any 2.4 or 2.6 kernel.

Configuration

To configure the source RPM, See Section 6.2.3 [Configuring the Source RPM], page 91.

6.1.4 Downloading the Debian DSC

If you cannot obtain a binary DEB for your architecture, or would like to roll your own DEB, download the following Debian DSC.

LiS_2.18.4.rc2-0.dsc
LiS_2.18.4.rc2-0.tar.gz
: This is the Debian DSC for the package. From this Debian DSC it is possible to build binary DEB for any supported architecture and for any 2.4 or 2.6 kernel.

Configuration

To configure the source RPM, See Section 6.2.4 [Configuring the Debian DSC], page 96.

6.1.5 Downloading the Tar Ball

For non-RPM architectures, such as *NexusWare* embedded target, download the tarball as follows:

LiS-2.18.4.rc2.tar.gz
LiS-2.18.4.rc2.tar.bz2
: These are the `tar` balls for the release. These `tar` balls contain the `autoconf` distribution which includes all the source necessary for building and installing the package. These tarballs will even build Source RPM and Binary RPM on RPM architectures and Debian DSC and DEB on DPKG architectures.

The tar ball may be downloaded easily with `wget` as follows:

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
```

or

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.gz
```

Unpacking the Archive

After downloading one of the tar balls, unpack the archive using one of the following commands:

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.gz
% tar -xzvf LiS-2.18.4.rc2.tar.gz
```

or

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
```

Either will create a subdirectory name '`LiS-2.18.4.rc2`' containing all of the files and subdirectories for the `LiS` package.

Configuration

To configure and install the tar ball, See Section 6.2.5 [Configuring the Tar Ball], page 96.

6.1.6 Downloading from CVS

If you are a subscriber or sponsor of The OpenSS7 Project with CVS archive access privileges then you can download release or mid-release versions of the '`LiS`' package from the project CVS archive.

The Linux STREAMS (LiS) package is located in the '`LiS`' subdirectory of '`/var/cvs`'. For release tag information, see Chapter 5 [Releases], page 67.

To access the archive from the project CVS pserver, use the following commands to check out a version from the archive:

```
% export CVSROOT='-d:pserver:username@cvs.openss7.com:2401/var/cvs'
% cvs login
Password: *********
% cvs co -r LiS_2.18.4.rc2 LiS
% cvs logout
```

It is, of course, possible to check out by date or by other criteria. For more information, see section "cvs(1)" in *The Manual Pages*.

Preparing the CVS Working Directory

Although public releases of the '`LiS`' package do not require reconfiguration, creating a configurable directory from the CVS archive requires tools not normally distributed with the other releases.

The build host requires the following GNU tools:

- `autoconf 2.59`
- `automake 1.9.6`
- `libtool 1.5.22`
- `gettext 0.14.5`

It should be stressed that, in particular, the `autoconf` and `automake` must be at version releases `2.59` and `1.9`. *The versions normally distributed in mainstream GNU/Linux distributions are, in fact, much older than these versions.*[12] GNU version of these packages

[12] A notable exception is Debian.

configured and installed to default directories will install in '`/usr/local/`' allowing them to coexist with distribution installed versions.

For building documentation, the build host also requires the following documentation tools:

- `gs` 8.15
- `tetex` 3.0
- `texinfo` 4.8
- `transfig` 3.2.5
- `imagemagick` 6.2.4
- `groff` 1.17.2

Most desktop GNU/Linux distributions will have these tools; however, some server-style installations (e.g. *Ubuntu*-server) will not and they must be installed separately.

For uncooked manual pages, the entire `groff` package is required on *Debian* and *Ubuntu* systems (the base package does not include grefer which is used extensively by uncooked manual pages). The following will get what you need:

```
Debian: % apt-get install groff_ext
Ubuntu: % apt-get install groff
```

In addition, the build host requires a complete tool chain for compiling for the target host, including kernel tools such as `genksyms` and others.

If you wish to package rpms on an `rpm` system, or debs on a `dpkg` system, you will need the appropriate tool chain. Systems based on `rpm` typically have the necessary toolchain available, however, `dpkg` systems do not. The following on a *Debian* or *Ubuntu* system will get what you need:

```
% apt-get install debhelper
% apt-get install fakeroot
```

To generate a configuration script and the necessary scriptlets required by the GNU `autoconf` system, execute the following commands on the working directory:

```
% autoreconf -fiv LiS
```

where, '`LiS`' is the name of the directory to where the working copy was checked out under the previous step. This command generates the `configure` script and other missing pieces that are normally distributed with the release Tar Balls, SRPMs and DSCs.

Make sure that '`autoreconf --version`' returns '`2.59`'. Otherwise, you may need to perform something like the following:

```
% PATH="/usr/local/bin:$PATH"
% autoreconf -fiv LiS
```

After reconfiguring the directory, the package can then be configured and built using the same instructions as are used for the Tar Ball, see Section 6.2.5 [Configuring the Tar Ball], page 96, and Section 6.3.3 [Building from the Tar Ball], page 107.

Do note, however, that `make` will rebuild the documentation that is normally released with the package. Additional tools may be necessary for building the documentation.

When configuring the package in a working directory and while working a change-compile-test cycle that involves configuration macros or documentation, I find it of great advantage to invoke the GNU `configure` options `--enable-maintainer-mode` and `--enable-dependency-tracking`. The first of these two options will add maintainer-specific targets to any generated '`Makefile`', and the later will invoke automatic dependency tracking within the '`Makefile`' so rebuilds after changes to macro, source or documentation files will be automatically rebuilt.

6.2 Configuration

6.2.1 Configuring the Binary RPM

In general the binary RPM do not require any configuration, however, during installation it is possible to relocate some of the installation directories. This allows some degree of customization. Relocations that are available on the binary RPM are as follows:

'`LiS-core-2.6.17-1.2139_FC5-2.18.4.rc2-1.FC5.i686.rpm`'

> '`/lib/modules/2.6.17-1.2139_FC5`'
>
> > This relocatable directory contains the kernel modules that provide the LiS core, drivers and modules.[13]

'`LiS-info-2.6.17-1.2139_FC5-2.18.4.rc2-1.FC5.i686.rpm`'

> '`/usr/include/LiS/2.6.17-1.2139_FC5`'
>
> > This relocatable directory contains the kernel module exported symbol information that allows other kernel modules to be compiled against the correct version of the LiS package.[14]

'`LiS-dev-2.18.4.rc2-1.FC5.i686.rpm`'

> (not relocatable)

'`LiS-devel-2.18.4.rc2-1.FC5.i686.rpm`'

> '`/usr/lib`'
>
> > This relocatable directory contains LiS libraries.
>
> '`/usr/include/LiS`'
>
> > This relocatable directory contains LiS header files.

'`LiS-doc-2.18.4.rc2-1.FC5.i686.rpm`'

> '`/usr/share/doc`'
>
> > This relocatable directory contains all package specific documentation (including this manual). The subdirectory in this directory is the '`LiS-2.18.4.rc2`' directory.

[13] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example.

[14] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example. Also, note that the '`info`' subpackage is only applicable to the 2.4 kernel series.

'/usr/share/info'
: This relocatable directory contains info files (including the info version of this manual).

'/usr/share/man'
: This relocatable directory contains manual pages.

'LiS-lib-2.18.4.rc2-1.FC5.i686.rpm'

'/usr/lib'
: This relocatable directory contains the run-time shared libraries necessary to run applications programs and utilities developed for Linux STREAMS (LiS).

'/usr/share/locale'
: This relocatable directory contains the locale information for shared library files.

'LiS-source-2.18.4.rc2-1.FC5.i686.rpm'

'/usr/src'
: This relocatable directory contains the source code.

'LiS-util-2.18.4.rc2-1.FC5.i686.rpm'

'/usr/bin'
: This relocatable directory contains binary programs and utilities.

'/usr/sbin'
: This relocatable directory contains system binary programs and utilities.

'/usr/libexec'
: This relocatable directory contains test programs.

'/etc'
: This relocatable directory contains init scripts and configuration information.

Installation

To install the binary RPM, See Section 6.4.1 [Installing the Binary RPM], page 110.

6.2.2 Configuring the Debian DEB

In general the binary DEB do not require any configuration.

Installation

To install the Debian DEB, See Section 6.4.2 [Installing the Debian DEB], page 111.

6.2.3 Configuring the Source RPM

When building from the source RPM (see Section 6.3.1 [Building from the Source RPM], page 105), the rebuild process uses a number of macros from the user's '`.rpmmacros`' file as described in section "rpm(8)" in *The Manual Pages*.

Following is an example of the '`~/.rpmmacros`' file that I use for rebuilding RPMS:

```
#
# RPM macros for building rpms
#

%_topdir /usr/src/openss7.rpms

%vendor OpenSS7 Corporation
%distribution OpenSS7
%disturl http://www.openss7.org/
%packager Brian Bidulock <bidulock@openss7.org>
%url http://www.openss7.org/

%_signature gpg
%_gpg_path /home/brian/.gnupg
%_gpg_name openss7@openss7.org
%_gpgbin /usr/bin/gpg

%_source_payload w9.bzdio
%_binary_payload w9.bzdio

%_unpackaged_files_terminate_build 1
%_missing_doc_files_terminate_build 1
%_enable_debug_packages 1

#
# Template for debug information sub-package.
# with our little addition of release
#
%debug_package \
%ifnarch noarch\
%global __debug_package 1\
%package debug\
Summary: Debug information for package %{name}\
Group: Development/Debug\
AutoReqProv: 0\
%{?fullrelease:Release: %{fullrelease}}\
%description debug\
This package provides debug information for package %{name}.\
Debug information is useful when developing applications that use this\
package or when debugging this package.\
%files debug -f debugfiles.list\
%defattr(-,root,root)\
%endif\
%{nil}
```

When building from the source RPM (see Section 6.3.1 [Building from the Source RPM], page 105), it is possible to pass a number of additional configuration options to the `rpmbuild` process.

The additional configuration options are described below.

Note that distributions that use older versions of rpm do not have the '`--with`' or '`--without`' options defined. To acheive the same effect as:

```
--with someparm=somearg
```

do:

```
--define "_with_someparm --with-someparm=somearg"
```

`--define "_kversion $PACKAGE_KVERSION"`
: Specifies the kernel version other than the running kernel for which to build. If `_kversion` is not defined when rebuilding, the environment variable *PACKAGE_KVERSION* is used. If the environment variable *PACKAGE_KVERSION* is not defined, then the version of the running kernel (i.e. discovered with '`uname -r`') is used as the target version for kernel-dependent packages. This option can also be defined in an '`.rpmspec`' file using the macro name '`_kversion`'.

`--with checks`
`--without checks`
: Enable or disable preinstall checks. Each packages supports a number of preinstall checks that can be performed by invoking the '`check`' target with `make`. These currently consist of checking each kernel module for unresolved kernel symbols, checking for documentation for exported kernel module symbols, checking for documentation for exported library symbols, checking for standard options for build and installable programs, checking for documentation for built and installable programs. Normally these checks are only run in maintainer mode, but can be enabled and disabled with this option.

`--with k-optimize=HOW`
`--without k-optimize`
: Specify '`HOW`' optimization, *normal*, *size*, *speed* or *quick*. *size* compiles kernel modules `-Os`, *speed* compiles kernel modules `-O3`, and *quick* compiles kernel modules `-O0`. The default is *normal*. Use with care.

`--with cooked-manpages`
`--without cooked-manpages`
: Some systems do not like `grefer` references in manpages.[15] This option will cook `soelim`, `refer`, `tbl` and `pic` commands from the manpages and also strip `groff` comments. The default is to leave manpages uncooked: they are actually smaller that way.

`--with public`
`--without public`
: Release public packages or private packages. This option has no effect on the '`LiS`' package. The default is to release public packages.

`--with k-debug`
`--without k-debug`
: Specifies whether kernel debugging is to be performed on the build kernel modules. Mutually exclusive with `test` and `safe` below. This has the effect of removing static and inline attributes from functions and invoking all debugging macros in the code. The default is to not perform kernel debugging.

[15] In particular, some *Debian* systems do not load the `groff` extensions package and do not have `grefer` installed. Although this is an oversight on the configuration of the particular *Debian* system, we accomodate such misconfiguration with this feature.

`--with k-test`
`--without k-test`
: Specifies whether kernel testing is to be performed. Mutually exclusive with `debug` above and `safe` below. This has the effect of removing static and inline attributes from functions and invoking most debugging macros in the code. The default is to not perform kernel testing.

`--with k-safe`
`--without k-safe`
: Specifies whether kernel saftey is to be performed. Mutually exclusive with `debug` and `test` above. This has the effect of invoking some more pedantic assertion macros in the code. The default is not to apply kernel safety.

`--with k-inline`
`--without k-inline`
: Specifies whether kernel `inline` functions are to be place inline. This has the effect of adding the `-finline-functions` flag to *CFLAGS* for compiling kernel modules. Linux 2.4 kernels are normally compiled `-O2` which does not respect the `inline` directive. This compiles kernel modules with `-finline-functions` to get closer to `-O3` optimization. For better optimization controls, See Section 6.2.5 [Configuring the Tar Ball], page 96.

`--with k-modversions`
`--without k-modversions`
: Specifies whether kernel symbol versioning is to be applied to symbols exported by package kernel modules. The default is to version exported module symbols. This package does not export symbols so this option has no effect.

`--with devfs`
`--without devfs`
: Specifies whether the build is for a device filesystem daemon enabled system with autoloading, or not. The default is to build for devfsd autoloading when CONFIG_DEVFS_FS is defined in the target kernel. The `reuild` target uses this option to signal to the RPM spec file that the '`dev`' subpackage need not be built. This option does not appear when the package has no devices.

`--with devel`
`--without devel`
: Specifies whether to build development environment packages such as those that include header files, static libraries, manual pages and texinfo documentation. The default is to build development environment packages. This option can be useful when building for an embedded target where only the runtime components are desired.

`--with tools`
`--without tools`
: Specifies whether user space packages are to be built. The default is to build user space packages. This option can be useful when rebuilding for multiple architectures and target kernels. The `rebuild` automake target uses this feature when rebuilding for all available architectures and kernels, to rebuild user packages once per architecture instead of once per kernel.

`--with modules`
`--without modules`
: Specifies whether kernel modules packages are to be built. The default is to build kernel module packages. This option can be useful when rebuilding for multiple architectures and target kernels. The `rebuild` automake target uses this feature to rebuild for all available architectures and kernels.

In addition, the following `rpm` options, specific to the Linux STREAMS (LiS) package are available:

`--with broken-cpu-flags`
: Specify `int` instead of `long` interrupt flags. Linux uses `long` interrupt flags exclusively. The default is to use `long` interrupt flags. This option defaults to '`disabled`'.

`--without lis-development`
: This option is new to *LiS-2.18.2*. Disable reporting of source code path traces in debug logs. Linux STREAMS (LiS) passes file name and line number arguments to many of the exported functions. This is exhaustive of stack resources and requires the passing of more than three arguments on the stack in many circumnstances. Note that disabling code path traces breaks binary compatibility. The default is perform source code path traces, unless optimized for size or speed, in which case, the default is to not perform source code path traces. This option defaults to '`enabled`'.

`--with k-timers`
: Invoke Linux kernel cahcing on timer structures. This increases speed and efficiency but is unsafe on SMP architectures and for buggy drivers. The default is to perform kernel caching on timer structures. This option defaults to '`disabled`'.

`--with k-cache`
: Invoke Linux kernel caching on primary structures. This incresase speed and efficiency. The default is to perform kernel caching on primary structures. This option defaults to '`disabled`'.

`--with atomic-stats`
: Specify `atomic_t` element types for elements of the `module_stat` structure instead of `int` element types. The default is to use `int` elements types for the `module_stat` structure. This option defaults to '`disabled`'.

`--with lis-regparms=NUMBER`
`--without lis-regparms`
: This option is new to *LiS-2.18.2*. Specifies the number of parameters to pass in registers to Linux STREAMS (LiS) exported functions. Specifying other than '`0`' here may break binary compatibility. Specifying '`--without lis-regparms`' will default to the number used for kernel exported functions. Specifying greater than '`3`' will cause the build to fail. The default is to pass no arguments in registers. This option defaults to '`0`'.

`--without solaris-consts`
: Attempt to be compatible with Solaris constants. The default is to attempt to be compatible with Solaris constants. This option defaults to '`enabled`'.

`--without solaris-cmn_err`
: Specifies whether Solaris-style `cmn_err` is used. The Solaris style prints a newline at the end of each statement. The SVR 4.2 style prints a newline at the beginning of each statement unless `CE_CONT` is specified. Linux kernel logs use the former, so this defaults to Solaris-style. This option defaults to '`enabled`'.

In general, the default values of these options are sufficient for most purposes and no options need be provided when rebuilding the Source RPMs.

Build

To build from the source RPM, See Section 6.3.1 [Building from the Source RPM], page 105.

6.2.4 Configuring the Debian DSC

The Debian DSC can be configured by passing options in the environment variable *BUILD_DEBOPTIONS*. The options placed in this variable take the same form as those passed to the `configure` script, See Section 6.2.5 [Configuring the Tar Ball], page 96. For an example, See Section 6.3.2 [Building from the Debian DSC], page 106.

Build

To build from the Debian DSC, See Section 6.3.2 [Building from the Debian DSC], page 106.

6.2.5 Configuring the Tar Ball

All of the normal GNU `autoconf` configuration options and environment variables apply. Additional options and environment variables are provided to tailor or customize the build and are described below.

6.2.5.1 Configure Options

Following are the additional `configure` options, their meaning and use:

`--enable-checks`
`--disable-checks`
: Enable or disable preinstall checks. Each packages supports a number of preinstall checks that can be performed by invoking the '`check`' target with `make`. These currently consist of checking each kernel module for unresolved kernel symbols, checking for documentation for exported kernel module symbols, checking for documentation for exported library symbols, checking for standard options for build and installable programs, checking for documentation for built and installable programs. Normally these checks are only run in maintainer mode, but can be enabled and disabled with this option.

`--disable-compress-manpages`
: Compress manpages with '`gzip -9`' or '`bzip2 -9`' or leave them uncompressed. The default is to compress manpages with '`gzip -9`' or '`bzip2 -9`' if a single compressed manpage exists in the target installation directory (`--mandir`). This disables automatic compression.

`--disable-public`
: Disable public release. Has no effect on the '`LiS`' release. No private components exist in '`LiS`' releases.

`--disable-initscripts`
: Disables the installation of init scripts. The default is to configure and install init scripts and their associated configuration files.

`--disable-devel`
: Disables the installation of development environment components such as header files, static libraries, manual pages and texinfo documentation. The default is to install development environment components. This option can be useful when configuring for an embedded target where only the runtime components are desired.

`--enable-tools`
: Specifies whether user space programs and libraries are to be built and installed. The default is to build and install user space programs and libraries. This option can be useful when rebuilding for multiple architectures and target kernels, particularly under rpm. The **`rebuild`** target uses this feature when rebuilding RPMs for all available architectures and kernels, to rebuild user packages once per architecture instead of once per kernel.

`--enable-modules`
: Specifies whether kernel modules are to be built and installed. The default is to build and install kernel modules. This option can be useful when rebuilding for multiple architectures and target kernels, particularly under rpm. The **`rebuild`** automake target uses this feature to rebuild for all available architectures and kernels.

`--enable-arch`
: Specifies whether architectural dependent package components are to be built and installed. This option can be useful when rebuilding for multiple architectures and target kernels, particularly under dpkg. The default is to configure, build and install architecture dependent package components.

`--enable-indep`
: Specifies whether architecture independent package components are to be built and installed. This option can be useful when rebuilding for multiple architectures and target kernels, particularly under dpkg. The default is to configure, build and install architecture independent package components.

`--enable-k-inline`
: Enable kernel inline functions. Most Linux kernels build without `-finline-functions`. This option adds the `-finline-functions` and `-Winline` flags to the compilation of kernel modules. Use with care.

`--enable-k-safe`
: Enable kernel module run-time safety checks. Specifies whether kernel safety is to be performed. This option is mutually exclusive with `--enable-k-test` and `--enable-k-debug` below. This has the effect of invoking some more pedantic assertion macros in the code. The default is not to apply kernel safety.

`--enable-k-test`
: Enable kernel module run-teim testing. Specifies whether kernel testing is to be performed. This option is mutually exclusive with `--enable-k-safe` above and `--enable-k-debug` below. This has the effect of remove `static` and `inline` attributes from functions and invoking most non-performance affecting debugging macros in the code. The default is not to perform kernel testing.

`--enable-k-debug`
: Enable kernel module run-time debugging. Specifies whether kernel debugging is to be performed. This option is mutuallly exclusive with `--enable-k-safe` and `--enable-k-test` above. This has the effect of removing `static` and `inline` attributes from functions and invoking all debuggin macros in the code (including performance-affecting debug macros). The default is to not perform kernel debugging.

`--enable-devfs`
`--disable-devfs`
: Specifies whether the build is for a device filesystem daemon enabled system with autoloading, or not. The default is to build for devfsd autoloading when CONFIG_DEVFS_FS is defined in the target kernel. The `reuild` target uses this option to signal to the RPM spec file that the '`dev`' subpackage need not be built. This option does not appear when the package has no devices.

`--with-gpg-user=GNUPGUSER`
: Specify the `gpg` '`GNUPGUSER`' for signing RPMs and tarballs. The default is the content of the environment variable *GNUPGUSER*. If unspecified, the `gpg` program will normally use the user name of the account invoking the `gpg` program. For building source RPMs, the RPM macro '`_gpg_name`' will override this setting.

`--with-gpg-home=GNUPGHOME`
: Specify the '`GNUPGHOME`' directory for signing RPMs and tarballs. The default is the user's '`~/.gpg`' directory. For building source RPMs, the RPM macro '`_gpg_path`' will override this setting.

`--with-pkg-epoch=EPOCH`
: Specifies the epoch for the package. This is neither used for RPM nor Debian packages, it applies to the tarball release as a whole. The default is the contents of the '`.pkgepoch`' file in the source directory or, if that file does not exist, zero (0).

`--with-pkg-release=RELEASE`
: Specifies the release for the package. This is neither used for RPM nor Debian packages, it applies to the tarball release as a whole. The default is the contents of the '`.pkgrelease`' file in the source directory or, if that file does not exist, one (1). This is the number after the last point in ther package version number.

`--with-pkg-distdir=DIR`
: Specifies the distribution directory for the package. This is used by the maintainer for building distributions of tarballs. This is the directory into which archives are copied for distribution. The default is the top build directory.

`--with-cooked-manpages`
: Convert manual pages to remove macro dependencies and `grefer` references. Some systems do not like `grefer` references in manpages.[16] This option will cook `soelim`, `refer`, `tbl` and `pic` commands from the manpages and also strip `groff` comments. The default is to leave manpages uncooked (they are actually smaller that way).

`--with-rpm-epoch=PACKAGE_EPOCH`
: Specify the '`PACKAGE_EPOCH`' for the RPM spec file. The default is to use the RPM epoch conatined in the file '`.rpmepoch`'.

`--with-rpm-release=PACKAGE_RPMRELEASE`
: Specify the '`PACKAGE_RPMRELEASE`' for the RPM rspec file. The default is to use the RPM release contained in the file '`.rpmrelease`'.

`--with-rpm-extra=PACKAGE_RPMEXTRA`
: Specify the '`PACKAGE_RPMEXTRA`' extra release information for the RPM spec file. The default is to use the RPM extra release information contained in the file '`.rpmextra`'. Otherwise, this value will be determined from automatic detection of the RPM distribution.

`--with-rpm-topdir=PACKAGE_RPMTOPDIR`
: Specify the '`PACKAGE_RPMTOPDIR`' top directory for RPMs. If specified with a null '`PACKAGE_RPMTOPDIR`', the default directory for the RPM distribution will be used. If this option is not provided on the command line, the top build directory will be used as the RPM top directory as well.

`--with-deb-epoch=EPOCH`
: Specify the '`PACKAGE_DEBEPOCH`' for the DEB control file. The default is to use the DEB epoch contained int he file '`.debepoch`'.

`--with-deb-release=RELEASE`
: Specify the '`PACKAGE_DEBRELEASE`' for the DEB control file. The default is to use the DEB release contained in the file '`.debrelease`'.

`--with-deb-topdir=DIR`
: Specify the '`PACKAGE_DEBTOPDIR`' top directory for DEBs. If specified with a null '`PACKAGE_DEBTOPDIR`', the default directory for the DEB distribution will be used. If this option is not provided on the command line, the top build directory will be used as the DEB top directory as well.

`--with-k-release=PACKAGE_KRELEASE`
: Specify the '`PACKAGE_KRELEASE`' release of the Linux kernel for which the build is targeted. When not cross compiling, if this option is not set, the build will be targeted at the kernel running in the build environment (e.g., '`uname -r`'). When cross-compiling this option must be specified or the configure script will generate an error and terminate.

[16] In particular, some *Debian* systems do not load the `groff` extensions package and do not have `grefer` installed. Although this is an oversight on the configuration of the particular *Debian* system, we accomodate such misconfiguration with this feature.

`--with-k-linkage=PACKAGE_KLINKAGE`
: Specify the '`PACKAGE_KLINKAGE`' for kernel module linkage. This can be one of the following:

  - '`loadable`' – loadable kernel modules
  - '`linkable`' – linkable kernel objects

  The default is to build loadable kernel modules.

`--with-k-modules=K-MODULES-DIR`
: Specify the '`K-MODULES-DIR`' directory to which kernel modules will be installed. The default is based on the option `--with-k-release`, `--with-k-prefix` and `--with-k-rootdir`. The default is '`DESTDIR`'/'`K-MODULES-DIR`' which is typically '*`DESTDIR`*`/lib/modules/`*`PACKAGE_KRELEASE`*`/`'. This directory is normally located by the `configure` script and need only be provided for special cross-build environments or when requested by a `configure` script error message.

`--with-k-build=K-BUILD-DIR`
: Specify the '`K-BUILD-DIR`' base kernel build directory in which configured kernel source resides. The default is '*`DESTDIR`*`/`*`K-MODULES-DIR`*`/build`'. This directory is normally located by the `configure` script and need only be provided for special cross-build environments or when requested by a `configure` script error message.

`--with-k-source=K-SOURCE-DIR`
: Specify the '`K-SOURCE-DIR`' base kernel build directory in which configured kernel source resides. The default is '*`DESTDIR`*`/`*`K-MODULES-DIR`*`/source`'. This directory is normally located by the `configure` script and need only be provided for special cross-build environments or when requested by a `configure` script error message.

`--with-k-modver=K-MODVER-FILE`
: Specify the '`K-MODVER-FILE`' kernel module versions file. The default is '*`K-BUILD-DIR`*`/Module.symvers`'. This file is normally located by the `configure` script and need only be provided for special cross-build environments or when requested by a `configure` script error message.

`--with-k-sysmap=K-SYSMAP-FILE`
: Specify the '`K-SYSMAP-FILE`' kernel system map file. The default is '*`K-BUILD--DIR`*`/System.map`'. This file is normally located by the `configure` script and need only be provided for special cross-build environments or when requested by a `configure` script error message.

`--with-k-archdir=K-ARCHDIR`
: Specify the '`K-ARCHDIR`' kernel source architecture specific directory. The default is '*`DESTDIR`*`/`*`K-SOURCE-DIR`*`/arch`'. This directory is normally located by the `configure` script and need only be provided for special cross-build environments or when requested by a `configure` script error message.

`--with-k-machdir=K-MACHDIR`
: Specify the '`K-MACHDIR`' kernel source machine specific directory. The default is '*`DESTDIR`*`/`*`K-SOURCE-DIR`*`/`*`target_cpu`*'. This directory is normally located

by the `configure` script and need only be provided for special cross-build environments or when requested by a `configure` script error message.

`--with-k-config=K-CONFIG`
: Specify the '`K-CONFIG`' kernel configuration file. The default is '*`BOOT`*`/config-`*`K-RELEASE`*'. This configuration file is normally located by the `configure` script and need only be provided forspecial cross-build environments or when requested by a `configure` script error message.

`--with-k-optimize=HOW`
`--without-k-optimize`
: Specify '`HOW`' optimization, *normal*, *size*, *speed* or *quick*. *size* compiles kernel modules `-Os`, *speed* compiles kernel modules `-O3`, and *quick* compiles kernel modules `-O0`. The default is *normal*. Use with care.

`--with-strconf-master=STRCONF_CONFIG`
: Specify the '`STRCONF_CONFIG`' file name to which the configuration master file is written. The default is '`Config.master`'.

`--with-base-major=STRCONF_MAJBASE`
: Start numbering for major devices at '`STRCONF_MAJBASE`'. The default is '`230`'.

In addition, the following `configure` options, specific to the Linux STREAMS (LiS) package are available:

`--enable-user-mode`
: Enable user mode build. The option defaults to '`disabled`'. Don't use this option, it hasn't been tested.

`--enable-broken-cpu-flags`
: Specify `int` instead of `long` interrupt flags. Linux uses `long` interrupt flags exclusively. The default is to use `long` interrupt flags. This option defaults to '`disabled`'.

`--disable-lis-development`
: This option is new to *LiS-2.18.2*. Disable reporting of source code path traces in debug logs. Linux STREAMS (LiS) passes file name and line number arguments to many of the exported functions. This is exhaustive of stack resources and requires the passing of more than three arguments on the stack in many circumnstances. Note that disabling code path traces breaks binary compatibility. The default is perform source code path traces, unless optimized for size or speed, in which case, the default is to not perform source code path traces. This option defaults to '`enabled`'.

`--enable-k-timers`
: Invoke Linux kernel cahcing on timer structures. This increases speed and efficiency but is unsafe on SMP architectures and for buggy or high-speed drivers. The default is to **not** perform kernel caching on timer structures. This option defaults to '`disabled`'.

`--enable-k-cache`
: Invoke Linux kernel caching on primary structures. This incresase speed and efficiency. The default is to **not** perform kernel caching on primary structures. This option defaults to '`disabled`'.

`--enable-atomic-stats`
: Enable atomic statistics elements for STREAMS statistics. The changes STREAMS statistics elements from the standard `int` to `atomic_t` to permit atomic increment of statistics structures. This is not recommended and the default is to **not** use `atomic_t` for statistics structures. This option defaults to '`disabled`'.

`--with-lis-regparms=NUMBER`
`--without-lis-regparms`
: This option is new to *LiS-2.18.2*. Specifies the number of parameters to pass in registers to Linux STREAMS (LiS) exported functions. Specifying other than '`0`' here may break binary compatibility. Specifying '`--without-lis-regparms`' will default to the number used for kernel exported functions. Specifying greater than '`3`' will cause the build to fail. The default is to pass no arguments in registers. This option defaults to '`0`'.

`--without-solaris-consts`
: Attempt to be compatible with Solaris constants. The default is to attempt to be compatible with Solaris constants. This option defaults to '`enabled`'.

`--without-solaris-cmn_err`
: Specifies whether Solaris-style `cmn_err` is used. The Solaris style prints a newline at the end of each statement. The SVR 4.2 style prints a newline at the beginning of each statement unless `CE_CONT` is specified. Linux kernel logs use the former, so this defaults to Solaris-style. This option defaults to '`enabled`'.

6.2.5.2 Environment Variables

Following are additional environment variables to `configure`, their meaning and use:

GPG
: GPG signature command. This is used for signing distributions by the maintainer. By default, `configure` will search for this tool.

GNUPGUSER
: GPG user name. This is used for signing distributions by the maintainer.

GNUPGHOME
: GPG home directory. This is used for signing distributions by the maintainer.

GPGPASSWD
: GPG password for signing. This is used for signing distributions by the maintainer. This environment variable is not maintained by the `configure` script and should only be used on an isolated system.

SOELIM
: Roff source elimination command. This is only necessary when the option `--with-cooked-manpages` has been specified and `configure` cannot find the proper `soelim` command. By default, `configure` will search for this tool.

REFER Roff references command. This is only necessary when the option `--with-cooked-manpages` has been specified and `configure` cannot find the proper `refer` command. By default, `configure` will search for this tool.

TBL Roff table command. This is only necessary when the option `--with-cooked-manpages` has been specified and `configure` cannot find the proper `tbl` command. By default, `configure` will search for this tool.

PIC Roff picture command. This is only necessary when the option `--with-cooked-manpages` has been specified and `configure` cannot find the proper `pic` command. By default, `configure` will search for this tool.

GZIP Default compression options provided to `GZIP_CMD`.

GZIP_CMD
Manpages (and kernel modules) compression commands. This is only necessary when the option `--without-compressed-manpages` has *not* been specified and `configure` cannot find the proper `gzip` command. By default, `configure` will search for this tool.

BZIP2 Default compression options provided to `BZIP2_CMD`

BZIP2_CMD
Manpages compression commands. This is only necessary when the option `--without-compressed-manpages` has *not* been specified and `configure` cannot find the proper `bzip2` command. By default, `configure` will search for this tool.

MAKEWHATIS
Manpages apropros database rebuild command. By default, `configure` will search for this tool. By default, `configure` will search for this tool.

CHKCONFIG
Chkconfig command. This was used for installation of init scripts. All pacakges now come with `init_install` and `init_remove` scripts used to install and remove init scripts on both RPM and debian systems.

RPM Rpm command. This is only necessary for RPM builds. By default, `configure` will search for this tool.

RPMBUILD
Build RPM command. This is only necessary for RPM builds. By default, `configure` will search for this tool. `rpm` will be used instead of `rpmbuild` only if `rpmbuild` cannot be found.

DPKG Dpkg comand. This command is used for building debian packages. By default, `configure` will search for this tool.

DPKG_SOURCE
Dpkg-source command. This command is used for building debian dsc packages. By default, `configure` will search for this tool.

DPKG_BUILDPACKAGE
Dpkg-buildpackage command. This command is used for building debian deb packages. By default, `configure` will search for this tool.

DEB_BUILD_ARCH
: Debian build architecture. This variable is used for building debian packages. The default is the autoconf build architecutre.

DEB_BUILD_GNU_CPU
: Debian build cpu. This variable is used for building debian packages. The default is the autoconf build cpu.

DEB_BUILD_GNU_SYSTEM
: Debian build os. This variable is used for building debian packages. The default is the autoconf build os.

DEB_BUILD_GNU_TYPE
: Debian build alias. This variable is used for building debian packages. The default is the autoconf build alias.

DEB_HOST_ARCH
: Debian host architecture. This variable is used for building debian packages. The default is the autoconf host architecture.

DEB_HOST_GNU_CPU
: Debian host cpu. This variable is used for building debian packages. The default is the autoconf host cpu.

DEB_HOST_GNU_SYSTEM
: Debian host os. This variable is used for building debian packages. The default is the autoconf host os.

DEB_HOST_GNU_TYPE
: Debian host alias. This variable is used for building debian packages. The default is the autoconf host alias.

LDCONFIG
: Configure loader command. Command used to configure the loader when libraries are installed. By default, `configure` will search for this tool.

DESTDIR
: Cross build root directory. Specifies the root directory for build and installation. For example, for *NexusWare* cross-builds, this is set to environment variable *NEXUSWARE_PREFIX* on configuration to point to the root of the cross-build tree for both configuration and installation.

DEPMOD
: Build kernel module dependencies command. This is used during installation of kernel modules to a running kernel to rebuild the modules dependency database. By default, `configure` will search for this tool.

MODPROBE
: Probe kernel module dependencies command. This is used during installation of kernel modules to a running kernel to remove old modules. By default, `configure` will search for this tool.

LSMOD
: List kernel modules command. This is used during installation of kernel modules to a running kernel to detect old modules for removal. By default, `configure` will search for this tool.

LSOF List open files command. This is used during installation of kernel modules to a running kernel to detect old modules for removal. Processes owning the old kernel modules will be killed and the module removed. If the process restarts, the new module will be demand loaded. By default, `configure` will search for this tool.

GENKSYMS
Generate kernel symbols command. This is used for generating module symbol versions during build. By default, `configure` will search for this tool.

KGENKSYMS
Linux 2.6 generate kernel symbols command. This is used for generating module symbol version during build. By default, `configure` will search for this tool.

OBJDUMP
Object dumping command. This is used for listing information about object files. By default, `configure` will search for this tool.

NM Object symbol listing command. This is used for listing information about object files. By default, `configure` will search for this tool.

MODPOST_CACHE
Cache file for modpost. The version of the `modpost.sh` script that ships with each package can cache information to a cache file to speed multiple builds. This environment variable is used to specify a cache file.

AUTOM4TE
Autom4te command. This is the executable used by autotest for pre- and post-installation checks. By default, `configure` will search for this tool.

AUTOTEST
Autotest macro build command. This is the executable used by autotest for pre- and post-installation checks. By default, `configure` will search for this tool.

6.2.5.3 Build

To build from the tar ball, See Section 6.3.3 [Building from the Tar Ball], page 107.

6.3 Building

6.3.1 Building from the Source RPM

If you have downloaded the necessary source RPM (see Section 6.1.3 [Downloading the Source RPM], page 87), then the following instructions will rebuild the binary RPMs on your system. Once the binary RPMs are rebuilt, you may install them as described above (see Section 6.4.1 [Installing the Binary RPM], page 110).

The source RPM is rebuilt to binary RPMs as follows:

```
% wget http://www.openss7.org/rpms/SRPMS/LiS-2.18.4.rc2-1.src.rpm
% rpmbuild --rebuild -vv LiS-2.18.4.rc2-1.src.rpm
```

The rebuild process can also recognize a number of options that can be used to tweak the resulting binaries, See Section 6.2.3 [Configuring the Source RPM], page 91. These options are provided on the `rpm` command line. For example:

```
% rpmbuild --rebuild -vv --target athlon-redhat-linux \
  --define "_kversion 2.6.17-1.2139_FC5" \
  -- LiS-2.18.4.rc2-1.src.rpm
```

will rebuild binary RPM for the '`2.6.17-1.2139_FC5`' kernel for the '`athlon`' architecture.[17]

Installation

To install the resulting binary RPM, See Section 6.4.1 [Installing the Binary RPM], page 110.

6.3.2 Building from the Debian DSC

If you have downloaded the necessary Debian DSC (see Section 6.1.4 [Downloading the Debian DSC], page 87), then the following instructions will rebuild the binary DEBs on your system. Once the binary DEBs are rebuilt, you may install them as described above (see Section 6.4.2 [Installing the Debian DEB], page 111).

The Debian DSC is rebuilt to binary DEBs as follows:

```
% wget http://www.openss7.org/debian/LiS_2.18.4.rc2-0.dsc
% wget http://www.openss7.org/debian/LiS_2.18.4.rc2-0.tar.gz
% dpkg-buildpackage -v LiS_2.18.4.rc2-0.dsc
```

The rebuild process can also recognize a number of options that can be used to tweak the resulting binaries, See Section 6.2.4 [Configuring the Debian DSC], page 96. These options are provided in the environment variable *BUILD_DPKGOPTIONS* and have the same form as the options to `configure`, See Section 6.2.5 [Configuring the Tar Ball], page 96. For example:

```
% BUILD_DEBOPTIONS='
        --with-k-release=2.6.17-1.2139_FC5
        --host=athlon-debian-linux-gnu'
  dpkg-buildpackage -v \
  LiS_2.18.4.rc2-0.dsc
```

will rebuild binary DEB for the '`2.6.17-1.2139_FC5`' kernel for the '`athlon`' architecture.[18]

Installation

To install the resulting binary DEB, See Section 6.4.2 [Installing the Debian DEB], page 111.

[17] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example.

[18] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example.

6.3.3 Building from the Tar Ball

If you have downloaded the tar ball (see Section 6.1.5 [Downloading the Tar Ball], page 87), then the following instructions will rebuild the package on your system. (Note that the build process does not required `root` privilege.)

6.3.3.1 Native Build

Folowing is an example of a native build against the running kernel:

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
% pushd LiS-2.18.4.rc2
% ./configure
% make
% popd
```

6.3.3.2 Cross-Build

Following is an example for a cross-build. The kernel release version must always be specified for a cross-build.[19] If you are cross-building, specify the root for the build with environment variable *DESTDIR*. The cross-compile host must also be specified if different from the build host. Either the compiler and other tools must be in the usual places where GNU `autoconf` can find them, or they must be specified with declarations such as '`CC=/u5/NexusWare24/ppc-linux/gcc`' on the `configure` command line. Look in the file '`configure.nexusware`' in the release package for an example.

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
% pushd LiS-2.18.4.rc2
% ./configure DESTDIR="/some/other/root" \
--with-k-release=2.4.18 --host sparc-linux
% make
% popd
```

6.3.3.3 NexusWare Build

Additional support is provided for cross-building for the *Performance Technologies Inc. NexusWare* embedded target for the CPC-384, CPC-388 and CPC-396 cards. A configuration script wrapper ('`configure.nexusware`') is provided to simplify the cross-build operation for these targets. The following steps describe the process:

1. Follow the normal *NexusWare* instructions for rebuilding a '`generic`' kernel and flash image as follows: (Note that I keep my *NexusWare* build in '`/u5/NexusWare24`'.)

[19] Because it *is* a cross-build, the kernel version on the build machine is unlikely to be the kernel version of the target machine, except by coincidence.

```
% pushd /u5/NexusWare24
% source SETUP.sh
% make
% popd
```

For more recent *NexusWare* releases, the method for rebuilding a kernel is a little different as follows:

```
% pushd /u5/NexusWare80
% ./nexus 2.4
% ./nexus 8260
% ./nexus quick
% . SETUP.sh
% popd
```

2. Next download, unpack (see Section 6.1.5 [Downloading the Tar Ball], page 87) and configure (see Section 6.2.5 [Configuring the Tar Ball], page 96) using the provided '`configure.nexusware`' wrapper for `configure`. This wrapper simply tells the `configure` script where to find the *NexusWare* sources and which *NexusWare* cross-building tools to use for a cross-compile.[20]

   Any of the normal `configure` script options (see Section 6.2.5 [Configuring the Tar Ball], page 96) can be used on the same line as '`./configure.nexusware`'. One of particular interest to embedded targets is '`--with-k-optimize=size`' to attempt to reduce the size of the kernel modules.

   You must specify the kernel version of the kernel for which you are configuring. Add the –with-k-release=2.4.18 option for older *NexusWare* releases, –with-k-release=2.4.25 or –with-k-release=2.6.12 for more current *NexusWare* releases.

3. Install as normal (see Section 6.4.3 [Installing the Tar Ball], page 111), however, for embedded targets the `install-strip` target should be used instead of the `install` target. The `install-strip` target will strip unnecessary symbols from kernel modules and further reduce the size in the root file system flash image.

Following is what I use for configuration and installation: (My *NexusWare* tree is rooted at '`/u5/NexusWare`'.)

[20] Although I have not tried it, because we use GNU `autoconf` for configuration, these instructions should work equally well for the Solaris *NexusWare* cross-building environment as it does for the Linux *NexusWare* cross-building environment.

```
% pushd /u5/NexusWare80
% ./nexus 2.4
% ./nexus 8260
% ./nexus quick
% . SETUP.sh
% popd
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
% pushd LiS-2.18.4.rc2
% ./configure.nexusware --with-k-release=2.4.25 --with-k-optimize=size
% make
% make DESTDIR="$NEXUSWARE_PREFIX" install-strip
% popd
```

Once built and installed in the *NexusWare* directory, you will have to (currently) hand edit a '`.spec`' file to include the components you want in the *NexusWare* root file system. If you are cross-building for *NexusWare* you should already know what that means. Objects that you might be interested in copying to the root file system are kernel modules that were installed in '*$NEXUSWARE_PREFIX*`/lib/modules/2.4.18/lis`', libraries installed in '*$NEXUSWARE_PREFIX*`/usr/lib`' and utility functions installed in '*$NEXUSWARE_PREFIX*`/usr/bin`' and '*$NEXUSWARE_PREFIX*`/usr/sbin`' and test programs in '*$NEXUSWARE_PREFIX*`/usr/libexec`'. If you would prefer that these programs be installed in '*$NEXUSWARE_PREFIX*`/lib`', '*$NEXUSWARE_PREFIX*`/bin`', '*$NEXUSWARE_PREFIX*`/sbin`' and '*$NEXUSWARE_PREFIX*`/libexec`', (say because you want to remote mount the '`/usr`' directory after boot), then specify the '`--exec-prefix=/`' option to '`./configure.nexusware`'.

In addition, because *NexusWare* does not include an '`/etc/modules.conf`' file by default, it will be necessary to add one or edit your '`rc.4`' file to `insmod` the necessary '`LiS`' modules at boot time.

Also, *NexusWare* does not configure its kernels for *CONFIG_KMOD*, so any kernel modules must be loaded by the '`rc.4`' init script at boot. On more recent *NexusWare* releases, the init scripts will be installed in '*$NEXUSWARE_PREFIX*`/etc/rc.d/init.d/`' but you must manually edit your '`rc.4`' script to invoke these scripts.

Once you have completed the necessary '`.spec`' and '`rc.4`' file entries, you need to rebuild the '`generic`' kernel flash image once more for these objects to be included in the flash file system. It is important that this second build of the kernel image be the same as the first.

When modifying and rebuilding a *NexusWare* kernel, it will be necessary to rebuild and install '`LiS`'. Simply perform the last '`make install-strip`' stage or start again with '`./configure.nexusware`'. You can place the unpacked tarball in '*$NEXUSWARE_PREFIX*`/usr/src/LiS`', and add the following to the top-level *NexusWare* '`Makefile`' to make the build process a single step process instead of dual pass:

```
all:
...
        (cd kernels/generic; $(MAKE) depend)
        (cd usr/src/pcmcia-cs-3.2.1; $(MAKE) config)
        (cd kernels/generic; $(MAKE))
        (cd usr/src/pcmcia-cs-3.2.1; $(MAKE) pti)
        (cd usr/src/pti; $(MAKE))
        (cd drivers; $(MAKE))
        (cd utility; $(MAKE))
#       uncomment for LiS build
#       (cd usr/src/LiS; ./configure.nexusware; $(MAKE) install-strip)
        (cd build/generic; $(MAKE))
...
```

Another, perhaps simpler approach, is to make the necessary edits to the *NexusWare* top-level '`Makefile`' and '`.spec`' and '`rc.4`' files, download and unpack the tar ball into the *NexusWare* directory, and build the *NexusWare* flash image as normal:

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% pushd /u5/NexusWare24
% source SETUP.sh
% pushd usr/src
% tar -xjvf ${DIRSTACK[2]}/LiS-2.18.4.rc2.tar.bz2
% ln -sf LiS-2.18.4.rc2 LiS
% popd
% make
% popd
```

The situation is a little more complex for recent *NexusWare* releases.

6.4 Installing

6.4.1 Installing the Binary RPM

If you have downloaded the necessary binary RPMs (see Section 6.1.1 [Downloading the Binary RPM], page 83), or have rebuilt binary RPMs using the source RPM (see Section 6.3.1 [Building from the Source RPM], page 105), then the following instructions will install the RPMs on your system. For additional information on **rpm**, see section "rpm(8)" in *The Manual Pages*.

```
% pushd RPMS/i686
% rpm -ihv LiS-*-2.18.4.rc2-1.FC5.i686.rpm
```

You must have the correct binary RPMs downloaded or built for this to be successful.

Some of the packages are relocatable and can have final installation directories altered with the '`--relocate`' option to **rpm**, see section "rpm(8)" in **manpages**. For example, the following will relocate the documentation and info directories:

```
% pushd RPMS/i686
% rpm -ihv \
        --relocate '/usr/share/doc=/usr/local/share/doc' \
        --relocate '/usr/share/info=/usr/local/share/info' \
        -- LiS-doc-2.18.4.rc2-1.FC5.i686.rpm
```

The previous example will install the '`LiS-doc`' package by will relocate the documentation an info directory contents to the '`/usr/local`' version.

6.4.2 Installing the Debian DEB

If you have downloaded the necessary Debian DEBs (see Section 6.1.2 [Downloading the Debian DEB], page 85), or have rebuild binary DEBs using the Debian DSC (see Section 6.3.2 [Building from the Debian DSC], page 106), then the following instructions will install the DEBs on your system. For additional information on `dpkg`, see section "dpkg(8)" in *The Manual Pages*.

```
% pushd debian
% dpkg -iv LiS-*_2.18.4.rc2-0_*.deb
```

You must have the correct '`.deb`' files downloaded or build for this to be successful.

6.4.3 Installing the Tar Ball

After the build process (see Section 6.3.3 [Building from the Tar Ball], page 107), installation only requires execution of one of two `make` targets:

'`make install`'
: The '`install`' `make` target will install all the components of the package. Root privilege is required to successfully invoke this target.

'`make install-strip`'
: The '`install-strip`' `make` target will install all the components of the package, but will strip unnecessary information out of the objects and compress manual pages. Root privilege is required to successfully invoke this target.

6.5 Removing

6.5.1 Removing the Binary RPM

To remove an installed version of the binary RPMs (whether obtained from the OpenSS7 binary RPM releases, or whether created by the source RPM), execute the following command:

```
% rpm -evv `rpm -qa | grep '^LiS-'`
```

For more information on `rpm`, see section "rpm(8)" in *The Manual Pages*.

6.5.2 Removing the Debian DEB

To remove and installed version of the debian DEB (whether obtained from the OpenSS7 binary DEB releases, or whether created by the Debian DSC), execute the following command:

```
% dpkg -ev `dpkg -l | grep '^LiS-'`
```

For more information on dpkg, see section "dpkg(8)" in *The Manual Pages*.

6.5.3 Removing the Source RPM

To remove all the installed binary RPM build from the source RPM, see Section 6.5.1 [Removing the Binary RPM], page 111. Then simply remove the binary RPM package files and source RPM file. A command such as:

```
% find / -name 'LiS-*.rpm' -type f -print0 | xargs --null rm -f
```

should remove all 'LiS' RPMs from your system.

6.5.4 Removing the Debian DSC

To remove all the installed binary DEB build from the Debian DSC, see Section 6.5.2 [Removing the Debian DEB], page 111. Then simply remove the binary DEB package files and Debian DSC file. A command such as:

```
% find / \( -name 'LiS-*.deb' \
         -o -name 'LiS-*.dsc' \
         -o -name 'LiS-*.tar.* \
         \) -type f -print0 | xargs --null rm -f
```

should remove all 'LiS' DEBs, DSCs and TARs from your system.

6.5.5 Removing the Tar Ball

To remove a version installed from tar ball, change to the build directory where the package was built and use the 'uninstall' make target as follows:

```
% cd /usr/src/LiS
% make uninstall
% cd ..
% rm -fr LiS-2.18.4.rc2
% rm -f LiS-2.18.4.rc2.tar.gz
% rm -f LiS-2.18.4.rc2.tar.bz2
```

If you have inadvertently removed the build directory and, therefore, no longer have a configured directory from which to execute 'make uninstall', then perform all of the steps for configuration and installation (see Section 6.4.3 [Installing the Tar Ball], page 111) except the final installation and then perform the steps above.

6.6 Loading

6.6.1 Normal Module Loading

When '`LiS`' installs, modules and drivers are normally configured for demand loading. The '`install`' and '`install-strip`' `make` targets will make the necessary changes to the '`/etc/modules.conf`' file and place the modules in an appropriate place in '`/lib/modules/2.6.17-1.2139_FC5/lis`'. The '`make install`' process should have copied the kernel module files '`streams-*.o`' to the directory '`/lib/modules/2.6.17-1.2139_FC5/lis`'. This means that to load any of these modules, you can simply execute, for example, '`modprobe stream-`*`somedriver`*'.[21]

6.6.1.1 Linux STREAMS Module Loading

The '`LiS`' demand load system supports both the old kerneld and the new kmod mechanisms for demand loading kernel modules.

The convention for '`LiS`' kernel loadable object files is:

- Their name start with "streams-".
- They are placed in '`/lib/modules/2.6.17-1.2139_FC5/streams/`', where '`2.6.17-1.2139_FC5`' is an example kernel version.

If your kernel has been built using the '`kerneld`' daemon, then '`LiS`' kernel modules will automatically load as soon as the STREAMS module is pushed or the driver is opened. The '`make install`' process makes the necessary changes to the '`/etc/modules.conf`' file. After the install, you will see lines like the following added to your '`/etc/modules.conf`' file:

```
prune modules.lis
if -f /lib/modules/‘uname -r‘/modules.lis
include /lib/modules/‘uname -r‘/modules.lis
endif
```

which will provide for demand loading of the modules if they have been built and installed for the running kernel. The '`/lib/modules/‘uname -r‘/modules.lis`' file looks like this:

```
alias char-major-245  streams-some_driver
alias char-major-246  streams-other_driver
```

Note that STREAMS modules are not listed in this file, but will be loaded by name using '`kerneld`' if available.

6.6.1.2 Linux Fast-STREAMS Module Loading

Linux Fast-STREAMS has a wider range of kernel module loading mechanisms than is provided by *LiS*. For mechanisms used for kernel module loading under *Linux Fast-STREAMS*, See section "Top" in *Linux Fast-STREAMS Reference Manual*.

[21] Note that the '`_kversion`' of '`2.6.17-1.2139_FC5`' is only an example.

6.6.2 NexusWare Module Loading

Under exceptional circumstances, such as a *NexusWare* build, it is necessary to hand-edit a '`.spec`' and '`rc.4`' file to load the modules at boot time.[22]

6.7 Maintenance

6.7.1 Makefile Targets

Automake has many targets, not all of which are obvious to the casual user. In addition, *OpenSS7* automake files have additional rules added to make maintaining and releasing a package somewhat easier. This list of targets provides some help with what targets can be invoked, what they do, and what they hope to acheive. The available targets are as follows:

6.7.1.1 User Targets

The following are normal targets intended to be invoked by installers of the package. They are concerned with compiling, checking the compile, installing, checking the installation, and uninstalling the package.

'`[all]`' This is also the default target. It compiles the package. This is performed after configuring the source with '`configure`'. A makefile stub is provided so that if the package has not had autoreconf run (such as when checked out from CVS, the package will attempt to run '`autoreconf -fiv`'.

All *OpenSS7 Project* packages are configured without maintainer mode and without dependency tracking by default. This speed compilation of the package for one-time builds. This also means that if you are developing using the source package (edit-compile-test-cycle), changes made to source files will not cause the automatic rebuilding due to dependencies. There are two ways to enable dependency tracking: specify '`--enable-maintainer-mode`' to `configure`; or, specify '`--enable-dependency-tracking`' to `configure`. I use the former during my edit-compile-test cycle.

This is a standard *GNU* `automake` makefile target. This target does not require root privilege.

'`check`' All *OpenSS7 Project* packages provide check scripts for the check target. This step is performed after compiling the package and will run all of the check programs against the compiled binaries. Which checks are performed depends on whether '`--enable-maintainer-mode`' was specified to configure. If in maintainer mode, checks that assist with the release of the package will be run (such as checking that all manual pages load properly and that they have required sections.) We recommend running the check stage before installing, because it catches problems that might keep the installed package from functioning properly.

Another way to enable the greater set of checks, without invoking maintainer mode, is to specify '`--enable-checks`' to `configure`.

[22] At some time I expect to create an '`install-nexusware`' target that will make the necessary modifications to the '`.spec`' and '`rc.4`' files automatically.

This is a standard *GNU* `automake` makefile target, although the functions performed are customized for the *OpenSS7 Project*. This target does not require root privilege.

'`install`'
'`install-strip`'
: The '`install`' target installs the package. This target also performs some actions similar to the pre- and post-install scripts used by packaging tools such as `rpm` or `dpkg`. The '`install-strip`' target strips unnecessary symbols from executables and kernel modules before installing.

  This is a standard *GNU* `automake` makefile target. This target requires root privilege.

'`installcheck`'
: All *OpenSS7 Project* packages provide test scripts for the '`installcheck`' target. Test scripts are created and run using autotest (part of the autoconf package). Which test suites are run and how extensive they are depends on whether '`--enable-maintainer-mode`' was specified to `configure`. When in maintainer mode, all test suites will be run. When not in maintainer mode, only a few post-install checks will be performed, but the test suites themselves will be installed in '`/usr/libexec/LiS`'[23] for later use.

  This is a standard *GNU* `automake` makefile target. This target might require root privilege. Tests requiring root privilege will be skipped when run as a regular user. Tests requiring regular account privileges will be skipped when run as root.

'`retest`'
: To complement the '`installcheck`' target above, all *OpenSS7 Project* packages provide the '`retest`' target as a means to rerun failed conformance testsuite test cases. The retest target is provided because some test cases in the testsuites have delicate timing considerations that allow them to fail sporadically. Invoking this target will retest the failed cases until no cases that are not expected failures remain.

  This is an *OpenSS7 Project* specific makefile target. As with '`installcheck`', this target might require root privilege. Tests requiring root privilege will be skpped when run as a regular user. Tests requiring regular account privileges will be skipped when run as root.

'`uninstall`'
: This target will reverse the steps taken to install the package. This target also performs pre- and post- erase scripts used by packaging tools such as *rpm* or *dpkg*. You need to have a configured build directory from which to execute this target, however, you do not need to have compiled any of the files in that build directory.[24]

  This is a standard *GNU* `automake` makefile target. This target requires root privilege.

---

[23] '`/usr/libexec/LiS`' is just an example, the actual location is '`${libexecdir}/${PACKAGE}`', which varies from distribution to distribution (as some distributions such as Mandriva do not have a libexec directory).

[24] Therefore, it is possible to download the package, configure it, and then uninstall it. This is handy if you do not have the sources used to build and install the package immediately available.

'`remove`' This target is like '`uninstall`' with the exception that it uninstalls in the reverse order that installation was performed.[25]

This is an *OpenSS7 Project* specific makefile target.

6.7.1.2 Maintainer Targets

The following targets are targets intended for use by maintainers of the package, or those responsible for rerelease and packaging of a derivative work of the package. Some of these targets are only effective when maintainer mode has been invoked ('`--enable-maintainer-mode`' specified to `configure`.)

'`dist`' Creates a distribution package (tarball) in the top level build directory. *OpenSS7 Project* packages distribute two archives: a '`gzip tar`' archive and a '`bzip tar`' archive. These archives will have the name '`LiS-2.18.4.rc2.tar.gz`' and '`LiS-2.18.4.rc2.tar.bz2`'.

This is a standard *GNU* `automake` makefile target. This target does not require root privilege.

'`distcheck`'

This target is intended for use when releasing the package. It creates the `tar` archives above and then unpacks the tarball in a source directory, configures in a separate build directory, compiles the package, installs the package in a separate install directory, tests the install package to ensure that some components work, and, finally, uses the unpacked source tree to build another tarball. If you have added or removed files from the package, this is a good way to ensure that everything is still stable for release.

This is a standard *GNU* `automake` makefile target. This target does not require root privilege.

6.7.1.3 Clean Targets

'`mostlyclean`'

Cleans out most of the files from the compile stage. This target is helpful if you have not enabled dependency tracking and need to recompile with changes.

This is a standard *GNU* `automake` makefile target. This target does not require root privilege.

'`clean`' Cleans all the files from the build directory generated during the '`make [all]`' phase. It does not, however, remove files from the directory left there from the `configure` run. Use the '`distclean`' target to remove those too.

This is a standard *GNU* `automake` makefile target. This target might require root privilege if the '`installcheck`' target or the `testsuite` was invoked with root privilege (leaving files belonging to root).

'`distclean`'

This target cleans out the directories left behind by '`distcheck`' and removes all the `configure` and generated files from the build directory. This will effectively

[25] This is useful from the *OpenSS7 Master Package*.

remove all the files in the build directory, with the except of files that belong to you or some other process.

This is a standard *GNU* `automake` makefile target. This target might require root privilege if the '`installcheck`' target or the `testsuite` was invoked with root privilege (leaving files belonging to root).

'`maintainer-clean`'
: This target not only removes files from the build directory, it removes generated files from the source directory as well. Care should be taken when invoking this target, because it removes files generated by the maintainer and distributed with the archive that might require special tools to regenerate. These special tools might only be available to the maintainer (but they aren't). It also means that you probably need a full blown Linux system to rebuild the package.

  This is a standard *GNU* `automake` makefile target. This target might require root privilege if the '`installcheck`' target or the `testsuite` was invoked with root privilege (leaving files belonging to root).

'`check-clean`'
: This target removes log files left behind by the '`check`' target. By default, the check scripts append to log files in the top level build directory. This target can be used to clean out those log files before the next run.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

6.7.1.4 Release Targets

The following are targets used to generate complete releases into the package distribution directory. These are good for unattended and NFS builds, which is what I use them for. Also, when building from atop multiple packages, these targets also recurse down through each package.

'`release`'
: Build all of the things necessary to generate a release. On an `rpm` system this is the distribution archives, the source rpm, and the architecture dependent and architecture independent binary rpms. All items are placed in the package distribution directory that can be specified with the '`--with-pkg-distdir=DIR`' option to `configure`.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`forced-release`'
: The '`release`' target will not regenerate any files that already exist in the package distribution directory. This forced target will.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`release-sign`'
: You will be prompted for a password, unless to specify it to make with the *GNUPGPASS* variable. For unattended or non-interactive builds with signing, you can do that as: '`make GNUPGPASS=mypasswd release-sign`'

This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`forced-release-sign`'
: The '`release-sign`' target will not regenerate any files that already exist in the package distribution directory. This forced target will.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`release-clean`'
: This target will remove all distribution files for the current package from the package distribution directory.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

6.7.1.5 Logging Targets

For convenience, to log the output of a number of targets to a file, log targets are defined. The log file itself is used as the target to make, but make invokes the target minus a '`.log`' suffix. So, for example, to log the results of target '`foo`', invoke the target '`foo.log`'. The only target that this does not apply to is '`compile.log`'. When you invoke the target '`compile.log`' a simple `make` is invoked and logged to the file '`compile.log`'. The '`foo.log`' rule applies to all other targets. This does not work for all targets, just a selected few.[26] Following are the logging targets:

Common Logging Targets

Common logging targets correspond to normal user `automake` makefile targets as follows:

'`compile.log`'
: This is an *OpenSS7 Project* specific makefile target, but it invokes the standard *GNU* `automake` makefile target '`[all]`'.

'`check.log`'
: This is an *OpenSS7 Project* specific makefile target, but it invokes the standard *GNU* `automake` makefile target '`check`'.

'`install.log`'
: This is an *OpenSS7 Project* specific makefile target, but it invokes the standard *GNU* `automake` makefile target '`install`'.

'`installcheck.log`'
: This is an *OpenSS7 Project* specific makefile target, but it invokes the standard *GNU* `automake` makefile target '`installcheck`'.

'`uninstall.log`'
: This is an *OpenSS7 Project* specific makefile target, but it invokes the standard *GNU* `automake` makefile target '`uninstall`'.

---

[26] Note that because logging targets invoke a pipe, `make` does not return the correct return status (always returns success if the `tee` operation is successful). Therefore, these targets should not be invoked by scripts that need to use the return value from `make`.

'`remove.log`'
: This is an *OpenSS7 Project* specific makefile target, that invokes the *OpenSS7 Project* '`remove`' target.

Maintainer Logging Targets

Maintainer logging targets correspond to maintainer mode `automake` makefile targets as follows:

'`dist.log`'
: This is an *OpenSS7 Project* specific makefile target, but it invokes the standard *GNU* `automake` makefile target '`dist`'.

'`distcheck.log`'
: This is an *OpenSS7 Project* specific makefile target, but it invokes the standard *GNU* `automake` makefile target '`distcheck`'.

'`srpm.log`'
: This is an *OpenSS7 Project* specific makefile target, that invokes the *OpenSS7 Project* '`remove`' target.

'`rebuild.log`'
: This is an *OpenSS7 Project* specific makefile target, that invokes the *OpenSS7 Project* '`remove`' target.

'`resign.log`'
: This is an *OpenSS7 Project* specific makefile target, that invokes the *OpenSS7 Project* '`remove`' target.

'`release.log`'
: This is an *OpenSS7 Project* specific makefile target, that invokes the *OpenSS7 Project* '`remove`' target.

'`release-sign.log`'
: This is an *OpenSS7 Project* specific makefile target, that invokes the *OpenSS7 Project* '`remove`' target.

If you want to add one, simply add it to *LOGGING_TARGETS* in '`Makefile.am`'.

6.7.1.6 Problem Report Targets

To ease problem report generation, all logging targets will automatically generate a problem report suitable for mailing in the file '*`target`*`.pr`' for target '*`target`*`.log`'. This problem report file is in the form of an email and can be sent using the included `send-pr` script or by invoking the '`send-pr`' makefile target.

There are two additional problem report targets:

'`pr`'
: The '`pr`' target is for independently generating a problem report outside of the build or installation process. The target will automatically generate a problem report skeleton suitable for editting and mailing in the file '`problem.pr`'. This problem report file is in the form of an email and can be editted and sent directly, or sent using the included `send-pr` script or by invoking the '`send-pr`' target.

This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`send-pr`' The '`send-pr`' target is for finalizing and mailing a problem report generated either inside or outside the build and installation process. The target will automatically finalize and mail the '`problem.pr`' problem report if it has changed since the last time that '`send-pr`' was invoked.

This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege (unless the problem report file was generated as root).

6.7.1.7 Release Archive Targets

The following targets are used to generate and clean distribution archive and signature files. Whereas the '`dist`' target affects archives in the top build directory, the '`release-archive`' targets affects archives in the package distribution directory (either the top build directory or that specified with '`--with-pkg-distdir=DIR`' to `configure`).

You can change the directory to which packages are distributed by using the '`--with-pkg-distdir=DIR`' option to `configure`. The default directory is the top build directory.

'`release-archives`'

This target creates the distribution archive files if they have not already been created. This not only runs the '`dist`' target, but also copies the files to the distribution directory, which, by default is the top build directory.

The files generated are named:

'`LiS-2.18.4.rc2.tar.gz`' and '`LiS-2.18.4.rc2.tar.bz2`'

You can change this distribution directory with the '`--with-pkg-distdir`' option to `configure`. See '`./configure --help`' for more details on options.

This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`release-sign-archives`'

This target is like '`release-archives`', except that it also signs the archives using a *GPG* detached signature. You will be prompted for a password unless you pass the *GNUPGPASS* variable to make. For automated or unattended builds, pass the *GNUPGPASS* variable like so:

'`make GNUPGPASS=mypasswd release-sign-archives`'

Signature files will be named:

'`LiS-2.18.4.rc2.tar.gz.asc`' and '`LiS-2.18.4.rc2.tar.bz2.asc`'

These files will be moved to the package distribution directory with the plaintext archives.

This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`release-clean-archives`'

This target will clean the release archives and signature files from the package distribution directory.

This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

6.7.1.8 RPM Build Targets

On `rpm` systems, or systems sporting rpm packaging tools, the following targets are used to generate `rpm` release packages. The epoch and release number can be controlled by the contents of the '`.rpmepoch`' and '`.rpmrelease`' files, or with the '`--with-rpm-epoch=EPOCH`' and '`--with-rpm-release=RELEASE`' options to `configure`. See '`configure --help`' for more information on options. We always use release number '`1`'. You can use release numbers above '`1`'.

'`srpm`'
: This target generates the source rpm for the package (without signing the source rpm). The source rpm will be named: '`LiS-2.18.4.rc2-1.srpm`'.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`rpms`'
: This target is responsible for generating all of the package binary rpms for the architecture. The binary rpms will be named:

  '`LiS-*-2.18.4.rc2-1.*.rpm`'

  where the stars indicate the subpackage and the architecture. Both the architecture specific subpackages (binary objects) and the architecture independent ('`.noarch`') subpackages will be built unless the the former was disabled with the option '`--disable-arch`', or the later with the option '`--disable-indep`', passed to `configure`.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`sign`'
'`srpm-sign`'
: These two targets are the same. When invoked, they will add a signature to the source rpm file, provided that the file does not already have a signature. You will be prompted for a password if a signature is required. Automated or unattended builds can be acheived by using the `emake` expect script, included in '`${srcdir}/scripts/emake`'.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`rebuild`'
: This target accepts searches out a list of kernel names from the '`${DESTDIR}/lib/modules`' directory and builds rpms for those kernels and for each of a set of architectures given in the *AM_RPMTARGETS* variable to make. This is convenience target for building a group of rpms on a given build machine.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`resign`'
: This target will search out and sign, with a *GPG* signature, the source rpm, and all of the binary rpms for this package that can be found in the package distribution directory. This target will prompt for a *GPG* password. Automated or

unattended builds can be acheived with the `emake` expect script loccated here: '`${srcdir}/scripts/emake`'.

This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

6.7.1.9 Debian Build Targets

On debian systems, or systems sporting debian packaging tools, the following targets are used to generate debian release packages. The release number can be controlled by the contents of the '`.debrelease`' file, or with the '`--with-debrelease=RELEASENUMBER`' option to `configure`. See '`configure --help`' for more information on options.

'`dsc`'
: This target will build the debian source change package ('`.dsc`' file). We use release number '`0`' so that the entire tarball is included in the '`dsc`' file. You can use release number '`1`' for the same purposes. Release numbers above '`1`' will not include the entire tarball. The '`.dsc`' file will be named: '`LiS_2.18.4.rc2-0.dsc`'.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`sigs`'
: This target signs the '`.deb`' files. You will be prompted for a password, unless to specify it to make with the *GNUPGPASS* variable.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`debs`'
: This target will build the debian binary package ('`.deb`' file) from the '`.dsc`' created above. (This target will also create the '`.dsc`' if it has not been created already.) The subpackage '`.deb`' files will be named: '`LiS-*_2.18.4.rc2-0_*.deb`', where the stars indicate the subpackage and the architecture.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

'`csig`'
: This target signs the '`.dsc`' file. You will be prompted for a password, unless to specify it to make with the *GNUPGPASS* variable.

  This is an *OpenSS7 Project* specific makefile target. This target does not require root privilege.

7 Troubleshooting

7.1 Test Suites

7.1.1 Pre-installation Checks

Most *OpenSS7* packages, including the *Linux STREAMS (LiS)* package, ship with pre-installation checks integral to the build system. Pre-installation checks include check scripts that are shipped in the '`scripts`' subdirectory as well as specialized `make` targets that perform the checks.

When building and installing the package from *RPM* or *DEB* source packages (see Section 6.3.1 [Building from the Source RPM], page 105; and Section 6.3.2 [Building from the Debian DSC], page 106), a fundamental set of post-compile, pre-installation checks are performed prior to building binary packages. This is performed automagically and does not require any special actions on the part of the user creating binary packages from source packages.

When building and installing the package from *tarball* (see Section 6.3.3 [Building from the Tar Ball], page 107; and Section 6.4.3 [Installing the Tar Ball], page 111), however, pre-installation checks are only performed if specifically invoked by the builder of the package. Pre-installation checks are invoked after building the package and before installing the package. Pre-installation checks are performed by invoking the '`check`' or '`check.log`' target to `make` when building the package, as shown in Example 7.1.

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
% pushd LiS-2.18.4.rc2
% ./configure
% make
% make check  # <------- invoke pre-installation checks
% popd
```

Example 7.1: *Invoking Pre-Installation Checks*

Pre-installation checks fall into two categories: *System Checks* and *Maintenance Checks*.

7.1.1.1 Pre-Installation System Checks

System Checks are post-compilation checks that can be performed before installing the package that check to ensure that the compiled objects function and will be successfully installed. When the '`--enable-maintainer-mode`' option has not been passed to `configure`, only *System Checks* will be performed.

For example, the steps shown in Example 7.2 will perform *System* checks.

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
% pushd LiS-2.18.4.rc2
% ./configure
% make
% make check  # <------ invokes System pre-installation checks
% popd
```

Example 7.2: *Invoking System Checks*

7.1.1.2 Pre-Installation Maintenance Checks

Maintenance Checks include all *System Checks*, but also checks to ensure that the kernel modules, applications programs, header files, development tools, test programs, documentation, and manual pages conform to *OpenSS7* standards. When the '`--enable-maintainer-mode`' option has been passed to `configure`, *Maintenance Checks* will be performed.

For example, the steps shown in Example 7.3 will perform *Maintenance* checks.

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
% pushd LiS-2.18.4.rc2
% ./configure --enable-maintainer-mode
% make
% make check  # <------ invokes Maintenance pre-installation checks
% popd
```

Example 7.3: *Invoking Maintenace Checks*

7.1.1.3 Specific Pre-Installation Checks

A number of check scripts are provided in the '`scripts`' subdirectory of the distribution that perform both *System* and *Maintenance* checks. These are as follows:

`check_commands`

This check performs both *System* and *Maintainance* checks.

When performing *System* tests, the following tests are performed:

Unless cross-compiling, or unless a program is included in `AM_INSTALLCHECK_STD_OPTIONS_EXEMPT` every program in `bin_PROGRAMS`, `sbin_PROGRAMS`, and `libexec_PROGRAMS` is tested to ensure that the '`--help`', '`--version`', and '`--copying`' options are accepted. When cross-compiling is is not possible to execute cross-compiled binaries, and these checks are skpped in that case.

Script executables, on the other hand, can be executed on the build host, so, unless listed in `AM_INSTALLCHECK_STD_OPTIONS_EXEMPT`, every program in `dist_bit_SCRIPTS`, `dist_sbin_SCRIPTS`, and `pkglibexec_SCRIPTS` are tested to ensure that the '`--help`', '`--version`', and '`--copying`' options are accepted.

When performing *Maintenance* tests, `check_commands` also checks to ensure that a manual page exists in section 1 for every executable binary or script that will be installed from `bin_PROGRAMS` and `dist_bin_SCRIPTS`. It also checks to ensure that a manual page exists in section 8 for every executable binary or script that wil be installed from `sbin_PROGRAMS`, `dist_sbin_SCRIPTS`, `libexec_PROGRAMS`, and `pkglibexec_SCRIPTS`.

`check_decls`

This check only performs *Maintenance* checks.

It collects the results from the `check_libs`, `check_modules` and `check_headers` check scripts and tests to ensure every declaration of a function prototype or external variable contained in installed header files has a corresponding exported symbol from either a to be installed shared object library or a to be installed kernel module. Declarations are exempted from this requirement if their identifiers have been explicitly added to the `EXPOSED_SYMBOL` variable. If `WARN_EXCESS` is set to '`yes`', then the check script will only warn when excess declarations exist (without a corresponding exported symbol); otherwise, the check script will generate an error and the check will fail.

`check_headers`

This check only performs *Maintenance* checks.

When performing *Maintenance* tests, it identifies all of the declarations included in to be installed header files. It then checks to ensure that a manual page exists in sections 2, 3, 7 or 9, as appropriate, for the type of declaration. It also checks to see if a manual page source file exists in the source directory for a declaration that has not been included in the distribution. Function or prototype declarations that do not have a manual page in sections 2, 3, or 9 will cause the check to fail. Other declarations (variable, externvar, macro, enumerate, enum, struct, union, typedef, member, etc.) will only warn if a manual page does not exist, but will not fail the check.

`check_libs`

This check only performs *Maintenance* checks.

When performing *Maintenance* tests, it checks that each exported symbol in each to be installed shared object library has a manual page in section 3. It also checks that each exported symbol has a function, prototype or externvar declaration in the to be installed header files. A missing declaration or manual page will cause this check to fail.

`check_mans`

This check only performs *Maintenance* checks.

When performing *Maintenance* tests, it checks that to be install manual pages can be formatted for display without any errors or warnings from the build host `man` program. It also checks that required headings exist for manual pages according to the section in which the manual page will be installed. It warns if recommended headings are not included in the manual pages. Because some *RPM* distributions have manual pages that might conflict with the package manual pages, this check script also checks for conflicts with installed manual pages on the build host. This check script also checks to ensure that all to be installed manual pages are used in some fashion, that is, they have a declaration, or exported symbol, or are the name of a kernel module or STREAMS module or driver, possibly capitalized.

Note that checking for conflicts with the build host should probably be included in the *System* checks (because *System* checks are performed before the source *RPM* `%install` scriptlet).

`check_modules`

This check performs both *System* and *Maintenance* checks.

When performing *System* tests, it checks each to be installed kernel module to ensure that all undefined symbols can be resolved to either the kernel or another module. It also checks whether an exported or externally declared symbol conflicts with an exported or externally declared symbol present in the kernel or another module.[1]

When performing *Maintenance* tests, this check script tests that each to be installed kernel module has a manual page in section 9 and that each exported symbol that does not begin with an underscore, and that belongs to an exported function or exported variable, has a manual page in section 9. It also checks to ensure that each exported symbol that does not begin with an underscore, and that blongs to an exported function or exported variable, has a function, prototype or externvar declaration in the to be installed header files.

`check_streams`

This check performs only *Maintenance* checks.

When performing *Maintenance* tests, it checks that for each configured *STREAMS* module or driver, or device node, that a manual page exists in section 4 or section 7 as appropriate.

The output of the pre-installation tests are fairly self explanatory. Each check script saves some output to '`name.log`', where *name* is the name of the check script as listed above. A summary of the results of the test are display to standard output and can also be captured to the '`check.log`' file if the '`check.log`' target is used instead of the '`check`' target to `make`.

Because the check scripts proliferate '`name.log`' files throughout the build directory, a '`make check-clean`' `make` target has be provided to clean them out. '`make check-clean`' should be run before each successive run of '`make check`'.

7.1.2 Post-installation Checks

Most OpenSS7 packages ship with a combatibility and conformance test suite built using the '`autotest`' capabilities of '`autoconf 2.59`'. These test suites act as a wrapper for the compatibility and conformance test programs that are shipped with the package.

Unlike the pre-installation checks, the post-installation checks are always run complete. The only check that post-installation test scripts perform is to test whether they have been invoked with root privileges or not. When invoked as root, or as a plain user, some tests might be skipped that require root privileges, or that require plain user privileges, to complete successfully.

[1] This particular check has caught some namespace polution that has occurred in the 2.6.11 kernel.

7.1.2.1 Running Test Suites

There are several ways of invoking the conformance test suites:

1. The test suites can be run after installation of the package by invoking the '`make installcheck`' or '`make installcheck.log`' target. Some packages require that root privileges be acquired before invoking the package.
2. The test suites can be run from the distribution subdirectory after installation of the package by invoking the `testsuite` shell script directly.
3. The test suites can be run standalone from the '`libexec`' ('`/usr/libexec`') installation directory by invoking the `testsuite` shell script directly.

Typical steps for invoking the testsuites directly from `make` are shown in Example 7.4.

```
% wget http://www.openss7.org/LiS-2.18.4.rc2.tar.bz2
% tar -xjvf LiS-2.18.4.rc2.tar.bz2
% pushd LiS-2.18.4.rc2
% ./configure
% make
% make check  # <------ invokes System pre-installation checks
% make install
% sudo make installcheck # <------- invokes post-installation tests
% popd
```

Example 7.4: *Invoking System Checks*

When performing post-installation checks for the purposes of generating a problem report, the checks should always be performed from the build directory, either with '`make installcheck`' or by invoking `testsuite` directly from the '`tests`' subdirectory of the build directory. This ensures that all of the information known to `configure` and pertinent to the configuration of the system for which a test case failed, will be collected in the resulting '`testsuite.log`' file depositied upon test suite failure in the '`tests`' directory. This '`testsuite.log`' file can then be attached as part of the problem report and provides rich details to maintainers of the package. See also See Section 7.2 [Problem Reports], page 127, below.

Typical steps for invoking and installed `testsuite` standalone are shown in Example 7.5.

```
% [sudo] /usr/libexec/LiS/testsuite
```

Example 7.5: *Invoking* `testsuite` *Directly*

When invoked directly, `testsuite` will generate a '`testsuite.log`' file in the current directory, and a '`testsuite.dir`' directory of failed tests cases and debugging scripts. For generating a problem report for failed test cases, see Section 7.2.4 [Stand Alone Problem Reports], page 130.

7.2 Problem Reports

7.2.1 Problem Report Guidelines

Problem reports in the following categories should include a log file as indicated in the table below:

'`./configure`'
: A problem with the configuration process occurs that causes the '`./configure`' command to fail. The problem report must include the '`config.log`' file that was generated by `configure`.

'`make compile.log`'
: A problem with the build process occurs that causes the '`make`' command to fail. Perform '`make clean`' and then '`make compile.log`' and attach the '`config.log`' and '`compile.log`' files to the problem report.

'`make check.log`'
: A problem occurs with the '`make check`' target that causes it to fail. Perform '`make check-clean check.log`' and attach the '`config.log`', '`compile.log`' and '`check.log`' files to the problem report.

'`sudo make install.log`'
: A problem occurs with '`sudo make install`' that causes it to fail. Perform '`sudo make uninstall`' and '`sudo make install.log`' and attach the '`config.log`', '`compile.log`', '`check.log`', and '`install.log`' files to the problem report.

'`[sudo] make installcheck.log`'
: A problem occurs with the '`make installcheck`' target that causes the test suite to fail. Attach the resulting '`tests/testsuite.log`' and '`installcheck.log`' file to the problem report. There is no need to attach the other files as they are included in '`tests/testsuite.log`'.

'`[sudo] make uninstall.log`'
: A problem occurs with the '`make uninstall`' target that causes the test suite to fail. Perform '`sudo make uninstall.log`' and attach the '`config.log`', '`compile.log`', '`check.log`', '`install.log`', '`installcheck.log`', '`tests/testsuite.log`' and '`uninstall.log`' file to the problem report.

'`[sudo] make remove.log`'
: A problem occurs with the '`make remove`' target that causes the test suite to fail. Perform '`sudo make remove.log`' and attach the '`config.log`', '`compile.log`', '`check.log`', '`install.log`', '`installcheck.log`', '`tests/testsuite.log`' and '`remove.log`' file to the problem report.

For other problems that occur during the use of the *Linux STREAMS (LiS)* package, please write a test case for the test suite that recreates the problem if one does not yet exist and provide a test program patch with the problem report. Also include whatever log files are generated by the kernel (`cmn_err(9)`) or by the `strerr(8)` or `strace(1)` facilities (`strlog(9)`).

7.2.2 Generating Problem Reports

The OpenSS7 Project uses the *GNU GNATS* system for problem reporting. Although the '`send-pr`' tool from the *GNU GNATS* package can be used for bug reporting to the project's

GNATS database using electronic mail, it is not always convenient to download and install the *GNATS* system to gain access to the 'send-pr' tool.

Therefore, the *Linux STREAMS (LiS)* package provides the 'send-pr' shell script that can be used for problem reporting. The 'send-pr' shell script can invoked directly and is a workalike for the *GNU* 'send-pr' tool.

The 'send-pr' tool takes the same flags and can be used in the same fashion, however, whereas 'send-pr' is an interactive tool[2], 'send-pr' is also able to perform batch processing. Whereas 'send-pr' takes its field information from local databases or from using the 'query-pr' C-language program to query a remote database, the 'send-pr' tool has the field database internal to the tool.

Problem reports can be generate using make, See Section 6.7.1.6 [Problem Report Targets], page 119. An example of how simple it is to generate a problem report is illustrated in Example 7.6.

```
% make pr
SEND-PR:
SEND-PR: send-pr:  send-pr was invoked to generate an external report.  An
SEND-PR: automated problem report has been created in the file named
SEND-PR: 'problem.pr' in the current directory.  This problem report can
SEND-PR: be sent to bugs@openss7.org by calling this script as
SEND-PR: '/home/brian/os7/scripts/send-pr --file="problem.pr"'.
SEND-PR:
SEND-PR: It is possible to edit some of the fields before sending on the
SEND-PR: problem report.  Please remember that there is NO WARRANTY.  See
SEND-PR: the file 'COPYING' in the top level directory.
SEND-PR:
SEND-PR: Please do not send confidential information to the bug report
SEND-PR: address.  Inspect the file 'problem.pr' for confidential
SEND-PR: information before mailing.
SEND-PR:
% vim problem.pr  # <--- follow instructions at head of file
% make send-pr
```

Example 7.6: *Invoking Problem Report Generation*

Using the 'make pr' target to generate a problem report has the advantages that it will assemble any available '*.log' files in the build directory and attach them to the problem report.

7.2.3 Automatic Problem Reports

The *Linux STREAMS (LiS)* package also provides a feature for automatic problem report generation that meets the problem report submission guidelines detailed in the preceding sections.

Whenever a logging makefile target (see Section 6.7.1.5 [Logging Targets], page 118) is invoked, if the primary target fails, the send-pr shell script is invoked to automatically

[2] 'send-pr' launches the user's *EDITOR* to edit the problem report before submitting it.

generate a problem report file suitable for the corresponding target (as described above under see Section 7.2.1 [Problem Report Guidelines], page 128). An example is shown in Example 7.8.

```
% make compile.log
...
...
make[5]: *** [libXNSdrvs_a-ip.o] Error 1
make[5]: Leaving directory ‘/u6/buildel4/strxns’
make[4]: *** [all-recursive] Error 1
make[4]: Leaving directory ‘/u6/buildel4/strxns’
make[3]: *** [all] Error 2
make[3]: Leaving directory ‘/u6/buildel4/strxns’
make[2]: *** [all-recursive] Error 1
make[2]: Leaving directory ‘/u6/buildel4’
make[1]: *** [all] Error 2
make[1]: Leaving directory ‘/u6/buildel4’
SEND-PR:
SEND-PR: send-pr:  Make target compile.log failed in the compile stage.  An
SEND-PR: automated problem report has been created in the file named
SEND-PR: ’problem.pr’ in the current directory.  This problem report can
SEND-PR: be sent to bugs@openss7.org by calling ’make send-pr’.
SEND-PR:
SEND-PR: It is possible to edit some of the fields before sending on the
SEND-PR: problem report.  Please remember that there is NO WARRANTY.  See
SEND-PR: the file ’COPYING’ in the top level directory.
SEND-PR:
SEND-PR: Please do not send confidential information to the bug report
SEND-PR: address.  Inspect the file ’problem.pr’ for confidential
SEND-PR: information before mailing.
SEND-PR:
% vim problem.pr  # <--- follow instructions at head of file
% make send-pr
```

Example 7.7: *Problem Report from Failed Logging Target*

7.2.4 Stand Alone Problem Reports

The *Linux STREAMS (LiS)* package installs the `send-pr` script and its configuration file ‘`send-pr.config`’ in ‘`${libexecdir}/LiS`’ along with the validation `testsuite`, see See Section 7.1 [Test Suites], page 123. As with the `testsuite`, this allows the `send-pr` script to be used for problem report generation on an installed system that does not have a build directory.

An example of invoking the package `testsuite` and then generating a problem report for failed cases is shown in Example 7.8.

```
% [sudo] /usr/libexec/LiS/testsuite
% # test cases failed...
% /usr/libexec/LiS/send-pr
SEND-PR:
SEND-PR: send-pr:  send-pr was invoked to generate an external report.  An
SEND-PR: automated problem report has been created in the file named
SEND-PR: 'problem.pr' in the current directory.  This problem report can
SEND-PR: be sent to bugs@openss7.org by calling this script as
SEND-PR: '/usr/libexec/LiS/send-pr --file problem.pr'.
SEND-PR:
SEND-PR: It is possible to edit some of the fields before sending on the
SEND-PR: problem report.  Please remember that there is NO WARRANTY.  See
SEND-PR: the file 'COPYING' in the top level directory.
SEND-PR:
SEND-PR: Please do not send confidential information to the bug report
SEND-PR: address.  Inspect the file 'problem.pr' for confidential
SEND-PR: information before mailing.
SEND-PR:
% vim problem.pr  # <--- follow instructions at head of file
% /usr/libexec/LiS/send-pr --file problem.pr
```

Example 7.8: *Invoking* `send-pr` *Directly*

The advantage of the approach shown in the example is that the `send-pr` script is capable of collecting the '`testsuite.log`' file and the failed test cases and debugging scripts from the '`testsuite.dir`' directory and including them in the problem report, as well as all package pertinent information from the installed '`send-pr.config`'.

7.3 Known Problems

The OpenSS7 Project does not ship software with known bugs. All bugs are unknown.

Verified behaviour is that behaviour that has been verified by conformance test suites that are shipped with the *Linux STREAMS (LiS)* package.

Unverified behaviour may contain unknown bugs.

Please remember that there is **NO WARRANTY**.

See also Section 5.5 [Bugs], page 81.

Copying

GNU General Public License

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.

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

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.

GNU Lesser General Public License

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, 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 and use pieces of it in new free programs; and that you are informed that you can do these things.

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

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the *Lesser* General Public License because it does *Less* to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

Terms and Conditions for Copying, Distribution and Modification

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, 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 library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete 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 distribute a copy of this License along with the Library.

   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.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, 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. The modified work must itself be a software library.

   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, 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 Library, 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 Library.

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

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you 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.

   If distribution of 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 satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
   b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
   c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
   d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
   e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be 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.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library 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.

9. 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 Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library 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 with this License.

11. 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 Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library 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 Library.

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. 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 library's name and an idea of what it does.
Copyright (C) year  name of author

This library is free software; you can redistribute it and/or modify it
under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2.1 of the License, or (at
your option) any later version.

This library 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
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307,
USA.
```

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

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

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the library
'Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

Documentation License

GNU Free Documentation License

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.

Indices

Index of Concepts

Index of Data Types

Index of Functions and Macros

Index of Variables and Constants

Index of Files and Programs

Index of Configuration Options

Index of Makefile Targets

Index of Authors