Debugging D

Overview of Debugging Facilities

This appendix describes some of the tools available to assist in debugging STREAMS-based applications.

The basic categories available for debugging can be broken into these following areas:

1. Kernel debug printing
   This consists of kernel facilities for printing from inside drivers
2. STREAMS error logging
   This is a STREAMS-supported model of generating error messages and allowing them to be received by one of three different types of loggers.
3. Kernel-examination tools
   These include the tools bundled with the operating system

D

Kernel Debug Printing

Console Messages

The kernel routine cmn_err() allows printing of formatted strings on a system console. It displays a specified message on the console and/or stores it in the putbuf that is a circular array in the kernel and contains output from cmn_err(). Its format is:

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

void cmn_err (int level, char *fmt, int ARGS)
```

where *level* can take the following values:

- CE_CONT - may be used as simple printf(). It is used to continue another message or to display an informative message not associated with an error.
- CE_NOTE - report system events. It is used to display a message preceded with NOTICE:. This message is used to report system events that do not necessarily require user action, but may interest the system administrator. For example, a sector on a disk needing to be accessed repeatedly before it can be accessed correctly might be such an event.
- CE_WARN - system events that require user action. This is used to display a message preceded with WARNING:. This message is used to report system events that require immediate attention, such as those where if an action is not taken, the system may panic. For example, when a peripheral device does not initialize correctly, this level should be used.
- CE_PANIC - system panic. This is used to display a message preceded with PANIC:. Drivers should specify this level only under the most severe conditions. A valid use of this level is when the system cannot continue to function. If the error is recoverable, not essential to continued system operation, do not panic the system. This level halts all processing.

fmt and *ARGS* are passed to the kernel routine printf() that runs at splhi() and should be used sparingly. If the first character of *fmt* begins with *!* (an exclamation point) output is directed to putbuf. putbuf can be accessed with the crash(1M) command. If the destination character begins with ^ (a caret) output goes to the console. If no destination character is specified, the message is directed to both the putbuf array and the console.

cmn_err() appends each *fmt* with "\n", except for the CE_CONT level, even when a message is sent to the putbuf array. *ARGS* specifies a set arguments passed when the message is displayed. Valid specifications are *%s* (string), *%u* (unsigned decimal), *%d* (decimal), *%o* (octal), and *%x* (hexadecimal). cmn_err() does not accept length specifications in conversion specifications. For example, *%3d* is ignored.

STREAMS Error Logging

Error and Trace Logging

STREAMS error and trace loggers are provided for debugging and for administering STREAMS modules and drivers. This facility consists of log(7), strace(1M), strclean(1M) strerr(1M), and the strlog function.

Any module or driver in any Stream can call the STREAMS logging function strlog, described in log(7). strlog is also described in Appendix C. When called, strlog will send formatted text to the error logger strerr(1M), the trace logger strace(1M), or the console logger.

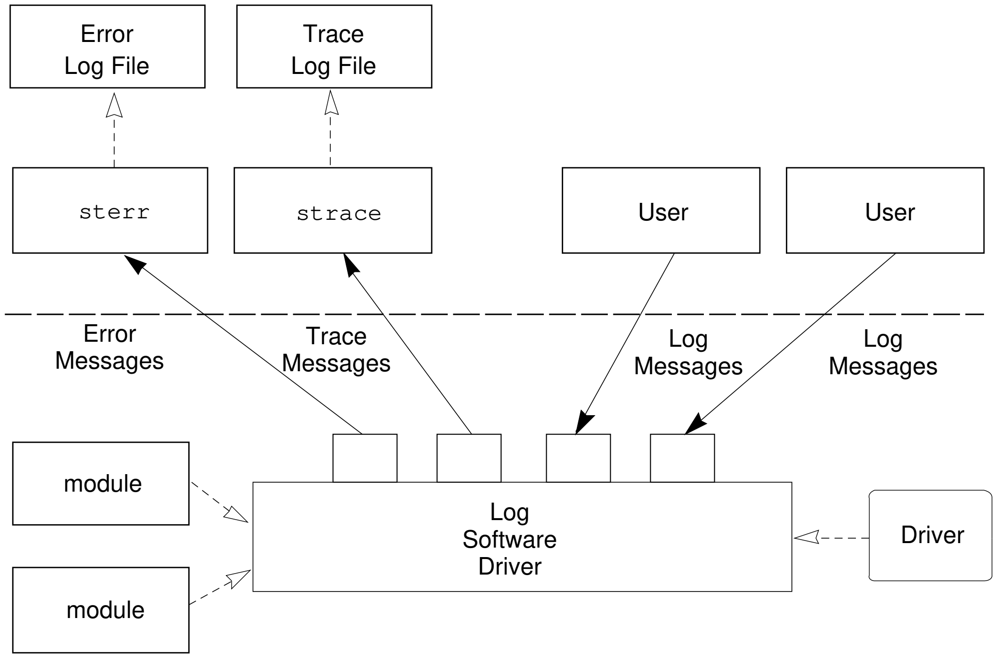

Figure D-1 Error and Trace Logging

`strerr` is intended to operate as a daemon process initiated at system startup. A call to `strlog` requesting an error to be logged causes an `M_PROTO` message to be sent to `strerr`, which formats the contents and places them in a daily file. The utility `strclean`(1M) is provided to purge daily log files that have not been modified for three days.

A call to `strlog` requesting trace information to be logged causes a similar `M_PROTO` message to be sent to `strace`(1M), which places it in a user designated file. `strace` is intended to be initiated by a user. The user can designate the modules/drivers and severity level of the messages to be accepted for logging by `strace`.

A user process can submit its own `M_PROTO` messages to the log driver for inclusion in the logger of its choice through `putmsg`(2). The messages must be in the same format required by the logging processes and will be switched to the logger(s) requested in the message.

The output to the log files is formatted, ASCII text. The files can be processed by standard system commands such as grep(1) or ed(1), or by developer-provided routines.

Kernel Examination Tools

The crash(1M) Command

The crash command is used to examine kernel structures interactively. It can be used on a system dump and on an active system.

The following crash functions are related to STREAMS:

- dbfreePrint data block header free list
- dblockPrint allocated STREAMS data block headers
- linkblkPrint the linkblk table
- mbfreePrint free STREAMS message block headers
- mblockPrint allocated STREAMS message block headers
- ptyPrint pseudo ttys now configured. The -l option gives information on the line discipline module ldterm, the -h option provides information on the pseudo-tty emulation module ptem, and the -s option gives information on the packet module pckt.
- qrunPrint a list of scheduled queues
- queuePrint the STREAMS queues
- streamPrint the stdata table
- strstatPrint STREAMS statistics
- ttyPrint the tty table. The -l option prints details about the line discipline module.

The crash funtions dblock, linkblk, mblock, queue, and stream take an optional table entry argument or address that is the address of the data structure. The strstat command gives information about STREAMS event cells and linkblks in addition to message blocks, data blocks, queues, and streams. On the output report, the CONFIG column represents the number of structures currently configured. It may change because resources are allocated as needed.

The adb(1) Command

adb is an interactive general-purpose debugger. It can be used to examine files and provides a controlled environment for the execution of programs. It has no support built in for any STREAMS functionality.

The kadb(1M) Command

kadb is an interactive debugger with a user interface similar to adb, but runs in the same virtual address space as the program begin debugged. It also has no specific STREAMS support.