NAME

log – interface to STREAMS error logging and event tracing

SYNOPSIS

#include <sys/strlog.h>
#include <sys/log.h>

DESCRIPTION

log is a STREAMS software device driver that provides an interface for console logging and for the STREAMS error logging and event tracing processes (see **strerr**(1M), and **strace**(1M)). **log** presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit **log** messages; and a set of **ioctl**(2) requests and STREAMS messages for interaction with a user level console logger, an error logger, a trace logger, or processes that need to submit their own **log** messages.

Kernel Interface

log messages are generated within the kernel by calls to the function **strlog( )**:

strlog(short *mid***, short** *sid***, char** *level***, ushort** *flags***, char** **fmt***, unsigned** *arg1* **. . .);**

Required definitions are contained in **<sys/strlog.h>**, **<sys/log.h>**, and **<sys/syslog.h>**. *mid* is the STREAMS module id number for the module or driver submitting the **log** message. *sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of **SL_ERROR** (the message is for the error logger), **SL_TRACE** (the message is for the tracer), **SL_CONSOLE** (the message is for the console logger), **SL_FATAL** (advisory notification of a fatal error), and **SL_NOTIFY** (request that a copy of the message be mailed to the system administrator). *fmt* is a **printf**(3S) style format string, except that **%s**, **%e**, **%E**, **%g**, and **%G** conversion specifications are not handled. Up to **NLOGARGS** (in this release, three) numeric or character arguments can be provided.

User Interface

log is opened through the **/dev/log** instance of the **clone** driver. Each open of **/dev/log** obtains a separate stream to **log**. In order to receive **log** messages, a process must first notify **log** whether it is an error logger, trace logger, or console logger using a STREAMS **I_STR ioctl** call (see below). For the console logger, the **I_STR ioctl** has an **ic_cmd** field of **I_CONSLOG**, with no accompanying data. For the error logger, the **I_STR ioctl** has an **ic_cmd** field of **I_ERRLOG**, with no accompanying data. For the trace logger, the **ioctl** has an **ic_cmd** field of **I_TRCLOG**, and must be accompanied by a data buffer containing an array of one or more struct **trace_ids** elements.

```
struct trace_ids {
        short ti_mid;
        short ti_sid;
        char  ti_level;
};
```

Each **trace_ids** structure specifies a *mid*, *sid*, and *level* from which messages will be accepted. **strlog( )** will accept messages whose *mid* and *sid* exactly match those in the **trace_ids** structure, and whose level is less than or equal to the level given in the **trace_ids** structure. A value of −1 in any of the fields of the **trace_ids** structure indicates that any value is accepted for that field.

Once the logger process has identified itself using the **ioctl** call, **log** will begin sending up messages subject to the restrictions noted above. These messages are obtained using the **getmsg**(2) function. The control part of this message contains a **log_ctl** structure, which specifies the *mid*, *sid*, *level*, *flags*, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, a sequence number, and a priority. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of **log** messages can be determined.

```
struct log_ctl {
        short    mid;
        short    sid;
        char     level;       /* level of message for tracing */
        short    flags;       /* message disposition */
        clock_t ltime;        /* time in machine ticks since boot */
        time_t   time;        /* time in seconds since 1970 */
        long     seq_no;      /* sequence number */
        int      pri;         /* priority = (facility | level) */
};
```

The priority consists of a priority code and a facility code, found in **<sys/syslog.h>**. If **SL_CONSOLE** is set in *flags*, the priority code is set as follows: If **SL_WARN** is set, the priority code is set to **LOG_WARNING**; If **SL_FATAL** is set, the priority code is set to **LOG_CRIT**; If **SL_ERROR** is set, the priority code is set to **LOG_ERR**; If **SL_NOTE** is set, the priority code is set to **LOG_NOTICE**; If **SL_TRACE** is set, the priority code is set to **LOG_DEBUG**; If only **SL_CONSOLE** is set, the priority code is set to **LOG_INFO**. Messages originating from the kernel have the facility code set to **LOG_KERN**. Most messages originating from user processes will have the facility code set to **LOG_USER**.

Different sequence numbers are maintained for the error and trace logging streams, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by **NLOGARGS** words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to **log**, even if it is not an error or trace logger. The only fields of the **log_ctl** structure in the control part of the message that are accepted are the *level*, *flags*, and *pri* fields; all other fields are filled in by **log** before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to **NLOGARGS**) must be packed, one word each, on the next word boundary following the end of the format string.

ENXIO is returned for **I_TRCLOG** ioctls without any **trace_ids** structures, or for any unrecognized **ioctl** calls. The driver silently ignores incorrectly formatted **log** messages sent to the driver by a user process (no error results).

Processes that wish to write a message to the console logger may direct their output to **/dev/conslog**, using either **write**(2) or **putmsg**(2).

EXAMPLES

Example of **I_ERRLOG** registration:

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;                /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I_TRCLOG registration:

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = −1;               /* any sub-id will be allowed */
tid[1].ti_level = −1;             /* any level will be allowed */

ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a **log** message (no arguments):

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk
          on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
```

```
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERROR | SL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

FILES

/dev/log
/dev/conslog

SEE ALSO

strace(1M), **strerr**(1M), **intro**(2), **getmsg**(2), **putmsg**(2), **write**(2),
STREAMS Programming Guide