This file contains brief information about the SCSI tape driver.
The driver is currently maintained by Kai Mäkisara (email
Kai.Makisara@kolumbus.fi)

Last modified: Sun Feb 24 21:59:07 2008 by kai.makisara

BASICS

The driver is generic, i.e., it does not contain any code tailored
to any specific tape drive. The tape parameters can be specified with
one of the following three methods:

1. Each user can specify the tape parameters he/she wants to use
directly with ioctls. This is administratively a very simple and
flexible method and applicable to single-user workstations. However,
in a multiuser environment the next user finds the tape parameters in
state the previous user left them.

2. The system manager (root) can define default values for some tape
parameters, like block size and density using the MTSETDRVBUFFER ioctl.
These parameters can be programmed to come into effect either when a
new tape is loaded into the drive or if writing begins at the
beginning of the tape. The second method is applicable if the tape
drive performs auto-detection of the tape format well (like some
QIC-drives). The result is that any tape can be read, writing can be
continued using existing format, and the default format is used if
the tape is rewritten from the beginning (or a new tape is written
for the first time). The first method is applicable if the drive
does not perform auto-detection well enough and there is a single
"sensible" mode for the device. An example is a DAT drive that is
used only in variable block mode (I don't know if this is sensible
or not :-).

The user can override the parameters defined by the system
manager. The changes persist until the defaults again come into
effect.

3. By default, up to four modes can be defined and selected using the minor
number (bits 5 and 6). The number of modes can be changed by changing
ST_NBR_MODE_BITS in st.h. Mode 0 corresponds to the defaults discussed
above. Additional modes are dormant until they are defined by the
system manager (root). When specification of a new mode is started,
the configuration of mode 0 is used to provide a starting point for
definition of the new mode.

Using the modes allows the system manager to give the users choices
over some of the buffering parameters not directly accessible to the
users (buffered and asynchronous writes). The modes also allow choices
between formats in multi-tape operations (the explicitly overridden
parameters are reset when a new tape is loaded).

If more than one mode is used, all modes should contain definitions
for the same set of parameters.

Many Unices contain internal tables that associate different modes to

supported devices. The Linux SCSI tape driver does not contain such tables (and will not do that in future). Instead of that, a utility program can be made that fetches the inquiry data sent by the device, scans its database, and sets up the modes using the ioctls. Another alternative is to make a small script that uses mt to set the defaults tailored to the system.

The driver supports fixed and variable block size (within buffer limits). Both the auto-rewind (minor equals device number) and non-rewind devices (minor is 128 + device number) are implemented.

In variable block mode, the byte count in write() determines the size of the physical block on tape. When reading, the drive reads the next tape block and returns to the user the data if the read() byte count is at least the block size. Otherwise, error ENOMEM is returned.

In fixed block mode, the data transfer between the drive and the driver is in multiples of the block size. The write() byte count must be a multiple of the block size. This is not required when reading but may be advisable for portability.

Support is provided for changing the tape partition and partitioning of the tape with one or two partitions. By default support for partitioned tape is disabled for each driver and it can be enabled with the ioctl MTSETDRVBUFFER.

By default the driver writes one filemark when the device is closed after writing and the last operation has been a write. Two filemarks can be optionally written. In both cases end of data is signified by returning zero bytes for two consecutive reads.

If rewind, offline, bsf, or seek is done and previous tape operation was write, a filemark is written before moving tape.

The compile options are defined in the file linux/drivers/scsi/st_options.h.

4. If the open option O_NONBLOCK is used, open succeeds even if the drive is not ready. If O_NONBLOCK is not used, the driver waits for the drive to become ready. If this does not happen in ST_BLOCK_SECONDS seconds, open fails with the errno value EIO. With O_NONBLOCK the device can be opened for writing even if there is a write protected tape in the drive (commands trying to write something return error if attempted).

MINOR NUMBERS

The tape driver currently supports 128 drives by default. This number can be increased by editing st.h and recompiling the driver if necessary. The upper limit is 2^17 drives if 4 modes for each drive are used.

The minor numbers consist of the following bit fields:

```
dev_upper non-rew mode dev-lower
  20 -  8     7    6 5  4      0
```

The non-rewind bit is always bit 7 (the uppermost bit in the lowermost byte). The bits defining the mode are below the non-rewind bit. The remaining bits define the tape device number. This numbering is backward compatible with the numbering used when the minor number was only 8 bits wide.

SYSFS SUPPORT

The driver creates the directory /sys/class/scsi_tape and populates it with directories corresponding to the existing tape devices. There are autorewind and non-rewind entries for each mode. The names are stxy and nstxy, where x is the tape number and y a character corresponding to the mode (none, l, m, a). For example, the directories for the first tape device are (assuming four modes): st0 nst0 st0l nst0l st0m nst0m st0a nst0a.

Each directory contains the entries: default_blksize default_compression default_density defined dev device driver. The file 'defined' contains 1 if the mode is defined and zero if not defined. The files 'default_*' contain the defaults set by the user. The value -1 means the default is not set. The file 'dev' contains the device numbers corresponding to this device. The links 'device' and 'driver' point to the SCSI device and driver entries.

Each directory also contains the entry 'options' which shows the currently enabled driver and mode options. The value in the file is a bit mask where the bit definitions are the same as those used with MTSETDRVBUFFER in setting the options.

A link named 'tape' is made from the SCSI device directory to the class directory corresponding to the mode 0 auto-rewind device (e.g., st0).

BSD AND SYS V SEMANTICS

The user can choose between these two behaviours of the tape driver by defining the value of the symbol ST_SYSV. The semantics differ when a file being read is closed. The BSD semantics leaves the tape where it currently is whereas the SYS V semantics moves the tape past the next filemark unless the filemark has just been crossed.

The default is BSD semantics.

BUFFERING

The driver tries to do transfers directly to/from user space. If this is not possible, a driver buffer allocated at run-time is used. If direct i/o is not possible for the whole transfer, the driver buffer is used (i.e., bounce buffers for individual pages are not used). Direct i/o can be impossible because of several reasons, e.g.:
- one or more pages are at addresses not reachable by the HBA
- the number of pages in the transfer exceeds the number of scatter/gather segments permitted by the HBA
- one or more pages can't be locked into memory (should not happen in any reasonable situation)

The size of the driver buffers is always at least one tape block. In fixed block mode, the minimum buffer size is defined (in 1024 byte units) by ST_FIXED_BUFFER_BLOCKS. With small block size this allows buffering of several blocks and using one SCSI read or write to transfer all of the blocks. Buffering of data across write calls in fixed block mode is allowed if ST_BUFFER_WRITES is non-zero and direct i/o is not used. Buffer allocation uses chunks of memory having sizes 2^n * (page size). Because of this the actual buffer size may be larger than the minimum allowable buffer size.

NOTE that if direct i/o is used, the small writes are not buffered. This may cause a surprise when moving from 2.4. There small writes (e.g., tar without -b option) may have had good throughput but this is not true any more with 2.6. Direct i/o can be turned off to solve this problem but a better solution is to use bigger write() byte counts (e.g., tar -b 64).

Asynchronous writing. Writing the buffer contents to the tape is started and the write call returns immediately. The status is checked at the next tape operation. Asynchronous writes are not done with direct i/o and not in fixed block mode.

Buffered writes and asynchronous writes may in some rare cases cause problems in multivolume operations if there is not enough space on the tape after the early-warning mark to flush the driver buffer.

Read ahead for fixed block mode (ST_READ_AHEAD). Filling the buffer is attempted even if the user does not want to get all of the data at this read command. Should be disabled for those drives that don't like a filemark to truncate a read request or that don't like backspacing.

Scatter/gather buffers (buffers that consist of chunks non-contiguous in the physical memory) are used if contiguous buffers can't be allocated. To support all SCSI adapters (including those not supporting scatter/gather), buffer allocation is using the following three kinds of chunks:
1. The initial segment that is used for all SCSI adapters including those not supporting scatter/gather. The size of this buffer will be (PAGE_SIZE << ST_FIRST_ORDER) bytes if the system can give a chunk of this size (and it is not larger than the buffer size specified by ST_BUFFER_BLOCKS). If this size is not available, the driver halves the size and tries again until the size of one page. The default settings in st_options.h make the driver to try to allocate all of the buffer as one chunk.
2. The scatter/gather segments to fill the specified buffer size are allocated so that as many segments as possible are used but the number of segments does not exceed ST_FIRST_SG.
3. The remaining segments between ST_MAX_SG (or the module parameter max_sg_segs) and the number of segments used in phases 1 and 2 are used to extend the buffer at run-time if this is necessary. The number of scatter/gather segments allowed for the SCSI adapter is not exceeded if it is smaller than the maximum number of scatter/gather segments specified. If the maximum number allowed for the SCSI adapter is smaller than the number of segments used in phases 1 and 2, extending the buffer will always fail.

EOM BEHAVIOUR WHEN WRITING

When the end of medium early warning is encountered, the current write is finished and the number of bytes is returned. The next write returns -1 and errno is set to ENOSPC. To enable writing a trailer, the next write is allowed to proceed and, if successful, the number of bytes is returned. After this, -1 and the number of bytes are alternately returned until the physical end of medium (or some other error) is encountered.

MODULE PARAMETERS

The buffer size, write threshold, and the maximum number of allocated buffers are configurable when the driver is loaded as a module. The keywords are:

buffer_kbs=xxx	the buffer size for fixed block mode is set to xxx kilobytes
write_threshold_kbs=xxx	the write threshold in kilobytes set to xxx
max_sg_segs=xxx	the maximum number of scatter/gather segments
try_direct_io=x	try direct transfer between user buffer and tape drive if this is non-zero

Note that if the buffer size is changed but the write threshold is not set, the write threshold is set to the new buffer size - 2 kB.

BOOT TIME CONFIGURATION

If the driver is compiled into the kernel, the same parameters can be also set using, e.g., the LILO command line. The preferred syntax is to use the same keyword used when loading as module but prepended with 'st.'. For instance, to set the maximum number of scatter/gather segments, the parameter 'st.max_sg_segs=xx' should be used (xx is the number of scatter/gather segments).

For compatibility, the old syntax from early 2.5 and 2.4 kernel versions is supported. The same keywords can be used as when loading the driver as module. If several parameters are set, the keyword-value pairs are separated with a comma (no spaces allowed). A colon can be used instead of the equal mark. The definition is prepended by the string st=. Here is an example:

```
st=buffer_kbs:64,write_threshold_kbs:60
```

The following syntax used by the old kernel versions is also supported:

```
st=aa[,bb[,dd]]
```

where
  aa is the buffer size for fixed block mode in 1024 byte units
  bb is the write threshold in 1024 byte units
  dd is the maximum number of scatter/gather segments

IOCTLS

The tape is positioned and the drive parameters are set with ioctls defined in mtio.h The tape control program 'mt' uses these ioctls. Try to find an mt that supports all of the Linux SCSI tape ioctls and opens the device for writing if the tape contents will be modified (look for a package mt-st* from the Linux ftp sites; the GNU mt does not open for writing for, e.g., erase).

The supported ioctls are:

The following use the structure mtop:

MTFSF Space forward over count filemarks. Tape positioned after filemark.
MTFSFM As above but tape positioned before filemark.
MTBSF Space backward over count filemarks. Tape positioned before filemark.
MTBSFM As above but ape positioned after filemark.
MTFSR Space forward over count records.
MTBSR Space backward over count records.
MTFSS Space forward over count setmarks.
MTBSS Space backward over count setmarks.
MTWEOF Write count filemarks.
MTWSM Write count setmarks.
MTREW Rewind tape.
MTOFFL Set device off line (often rewind plus eject).
MTNOP Do nothing except flush the buffers.
MTRETEN Re-tension tape.
MTEOM Space to end of recorded data.
MTERASE Erase tape. If the argument is zero, the short erase command is used. The long erase command is used with all other values of the argument.
MTSEEK Seek to tape block count. Uses Tandberg-compatible seek (QFA) for SCSI-1 drives and SCSI-2 seek for SCSI-2 drives. The file and block numbers in the status are not valid after a seek.
MTSETBLK Set the drive block size. Setting to zero sets the drive into variable block mode (if applicable).
MTSETDENSITY Sets the drive density code to arg. See drive documentation for available codes.
MTLOCK and MTUNLOCK Explicitly lock/unlock the tape drive door.
MTLOAD and MTUNLOAD Explicitly load and unload the tape. If the command argument x is between MT_ST_HPLOADER_OFFSET + 1 and MT_ST_HPLOADER_OFFSET + 6, the number x is used sent to the drive with the command and it selects the tape slot to use of HP C1553A changer.
MTCOMPRESSION Sets compressing or uncompressing drive mode using the SCSI mode page 15. Note that some drives other methods for control of compression. Some drives (like the Exabytes) use density codes for compression control. Some drives use another mode page but this page has not been implemented in the driver. Some drives without compression capability will accept any compression mode without error.
MTSETPART Moves the tape to the partition given by the argument at the next tape operation. The block at which the tape is positioned is the block where the tape was previously positioned in the new active partition unless the next tape operation is

MTSEEK. In this case the tape is moved directly to the block specified by MTSEEK. MTSETPART is inactive unless MT_ST_CAN_PARTITIONS set.

MTMKPART Formats the tape with one partition (argument zero) or two partitions (the argument gives in megabytes the size of partition 1 that is physically the first partition of the tape). The drive has to support partitions with size specified by the initiator. Inactive unless MT_ST_CAN_PARTITIONS set.

MTSETDRVBUFFER

Is used for several purposes. The command is obtained from count with mask MT_SET_OPTIONS, the low order bits are used as argument. This command is only allowed for the superuser (root). The subcommands are:

0

The drive buffer option is set to the argument. Zero means no buffering.

MT_ST_BOOLEANS

Sets the buffering options. The bits are the new states (enabled/disabled) the following options (in the parenthesis is specified whether the option is global or can be specified differently for each mode):

- MT_ST_BUFFER_WRITES write buffering (mode)
- MT_ST_ASYNC_WRITES asynchronous writes (mode)
- MT_ST_READ_AHEAD read ahead (mode)
- MT_ST_TWO_FM writing of two filemarks (global)
- MT_ST_FAST_EOM using the SCSI spacing to EOD (global)
- MT_ST_AUTO_LOCK automatic locking of the drive door (global)
- MT_ST_DEF_WRITES the defaults are meant only for writes (mode)
- MT_ST_CAN_BSR backspacing over more than one records can be used for repositioning the tape (global)
- MT_ST_NO_BLKLIMS the driver does not ask the block limits from the drive (block size can be changed only to variable) (global)
- MT_ST_CAN_PARTITIONS enables support for partitioned tapes (global)
- MT_ST_SCSI2LOGICAL the logical block number is used in the MTSEEK and MTIOCPOS for SCSI-2 drives instead of the device dependent address. It is recommended to set this flag unless there are tapes using the device dependent (from the old times) (global)
- MT_ST_SYSV sets the SYSV semantics (mode)
- MT_ST_NOWAIT enables immediate mode (i.e., don't wait for the command to finish) for some commands (e.g., rewind)
- MT_ST_SILI enables setting the SILI bit in SCSI commands when reading in variable block mode to enhance performance when reading blocks shorter than the byte count; set this only if you are sure that the drive supports SILI and the HBA correctly returns transfer residuals
- MT_ST_DEBUGGING debugging (global; debugging must be compiled into the driver)

MT_ST_SETBOOLEANS

MT_ST_CLEARBOOLEANS

Sets or clears the option bits.

MT_ST_WRITE_THRESHOLD

Sets the write threshold for this device to kilobytes specified by the lowest bits.

MT_ST_DEF_BLKSIZE
Defines the default block size set automatically. Value 0xffffff means that the default is not used any more.
MT_ST_DEF_DENSITY
MT_ST_DEF_DRVBUFFER
Used to set or clear the density (8 bits), and drive buffer state (3 bits). If the value is MT_ST_CLEAR_DEFAULT (0xfffff) the default will not be used any more. Otherwise the lowermost bits of the value contain the new value of the parameter.
MT_ST_DEF_COMPRESSION
The compression default will not be used if the value of the lowermost byte is 0xff. Otherwise the lowermost bit contains the new default. If the bits 8-15 are set to a non-zero number, and this number is not 0xff, the number is used as the compression algorithm. The value MT_ST_CLEAR_DEFAULT can be used to clear the compression default.
MT_ST_SET_TIMEOUT
Set the normal timeout in seconds for this device. The default is 900 seconds (15 minutes). The timeout should be long enough for the retries done by the device while reading/writing.
MT_ST_SET_LONG_TIMEOUT
Set the long timeout that is used for operations that are known to take a long time. The default is 14000 seconds (3.9 hours). For erase this value is further multiplied by eight.
MT_ST_SET_CLN
Set the cleaning request interpretation parameters using the lowest 24 bits of the argument. The driver can set the generic status bit GMT_CLN if a cleaning request bit pattern is found from the extended sense data. Many drives set one or more bits in the extended sense data when the drive needs cleaning. The bits are device-dependent. The driver is given the number of the sense data byte (the lowest eight bits of the argument; must be >= 18 (values 1 - 17 reserved) and <= the maximum requested sense data sixe), a mask to select the relevant bits (the bits 9-16), and the bit pattern (bits 17-23). If the bit pattern is zero, one or more bits under the mask indicate cleaning request. If the pattern is non-zero, the pattern must match the masked sense data byte.

(The cleaning bit is set if the additional sense code and qualifier 00h 17h are seen regardless of the setting of MT_ST_SET_CLN.)

The following ioctl uses the structure mtpos:
MTIOCPOS Reads the current position from the drive. Uses Tandberg-compatible QFA for SCSI-1 drives and the SCSI-2 command for the SCSI-2 drives.

The following ioctl uses the structure mtget to return the status:
MTIOCGET Returns some status information.
The file number and block number within file are returned. The

block is -1 when it can't be determined (e.g., after MTBSF).
The drive type is either MTISSCSI1 or MTISSCSI2.
The number of recovered errors since the previous status call
is stored in the lower word of the field mt_erreg.
The current block size and the density code are stored in the field
mt_dsreg (shifts for the subfields are MT_ST_BLKSIZE_SHIFT and
MT_ST_DENSITY_SHIFT).
The GMT_xxx status bits reflect the drive status. GMT_DR_OPEN
is set if there is no tape in the drive. GMT_EOD means either
end of recorded data or end of tape. GMT_EOT means end of tape.

MISCELLANEOUS COMPILE OPTIONS

The recovered write errors are considered fatal if ST_RECOVERED_WRITE_FATAL
is defined.

The maximum number of tape devices is determined by the define
ST_MAX_TAPES. If more tapes are detected at driver initialization, the
maximum is adjusted accordingly.

Immediate return from tape positioning SCSI commands can be enabled by
defining ST_NOWAIT. If this is defined, the user should take care that
the next tape operation is not started before the previous one has
finished. The drives and SCSI adapters should handle this condition
gracefully, but some drive/adapter combinations are known to hang the
SCSI bus in this case.

The MTEOM command is by default implemented as spacing over 32767
filemarks. With this method the file number in the status is
correct. The user can request using direct spacing to EOD by setting
ST_FAST_EOM 1 (or using the MT_ST_OPTIONS ioctl). In this case the file
number will be invalid.

When using read ahead or buffered writes the position within the file
may not be correct after the file is closed (correct position may
require backspacing over more than one record). The correct position
within file can be obtained if ST_IN_FILE_POS is defined at compile
time or the MT_ST_CAN_BSR bit is set for the drive with an ioctl.
(The driver always backs over a filemark crossed by read ahead if the
user does not request data that far.)

DEBUGGING HINTS

To enable debugging messages, edit st.c and #define DEBUG 1. As seen
above, debugging can be switched off with an ioctl if debugging is
compiled into the driver. The debugging output is not voluminous.

If the tape seems to hang, I would be very interested to hear where
the driver is waiting. With the command 'ps -l' you can see the state
of the process using the tape. If the state is D, the process is
waiting for something. The field WCHAN tells where the driver is
waiting. If you have the current System.map in the correct place (in
/boot for the procps I use) or have updated /etc/psdatabase (for kmem
ps), ps writes the function name in the WCHAN field. If not, you have

to look up the function from System.map.

Note also that the timeouts are very long compared to most other drivers. This means that the Linux driver may appear hung although the real reason is that the tape firmware has got confused.