NAME

sysmouse - virtualized mouse driver

SYNOPSIS

#include <sys/mouse.h>
#include <sys/consio.h>

DESCRIPTION

The console driver, in conjunction with the mouse daemon moused(8), supplies mouse data to the user process in the standardized way via the **sysmouse** driver. This arrangement makes it possible for the console and the user process (such as the X Window System) to share the mouse.

The user process which wants to utilize mouse operation simply opens */dev/sysmouse* with a open(2) call and reads mouse data from the device via read(2). Make sure that moused(8) is running, otherwise the user process will not see any data coming from the mouse.

Operation Levels

The **sysmouse** driver has two levels of operation. The current operation level can be referred to and changed via ioctl calls.

The level zero, the basic level, is the lowest level at which the driver offers the basic service to user programs. The **sysmouse** driver provides horizontal and vertical movement of the mouse and state of up to three buttons in the MouseSystems format as follows.

Byte 1

bit 7	Always one.
bit 6..3	Always zero.
bit 2	Left button status; cleared if pressed, otherwise set.
bit 1	Middle button status; cleared if pressed, otherwise set. Always one, if the device does not have the middle button.
bit 0	Right button status; cleared if pressed, otherwise set.

Byte 2	The first half of horizontal movement count in two’s complement; -128 through 127.
Byte 3	The first half of vertical movement count in two’s complement; -128 through 127.
Byte 4	The second half of the horizontal movement count in two’s complement; -128 through 127. To obtain the full horizontal movement count, add the byte 2 and 4.
Byte 5	The second half of the vertical movement count in two’s complement; -128 through 127. To obtain the full vertical movement count, add the byte 3 and 5.

At the level one, the extended level, mouse data is encoded in the standard format

MOUSE_PROTO_SYSMOUSE as defined in mouse(4).

IOCTLS

This section describes two classes of ioctl(2) commands: commands for the **sysmouse** driver itself, and commands for the console and the console control drivers.

Sysmouse Ioctls

There are a few commands for mouse drivers. General description of the commands is given in mouse(4). Following are the features specific to the **sysmouse** driver.

MOUSE_GETLEVEL *int *level*
MOUSE_SETLEVEL *int *level*

These commands manipulate the operation level of the mouse driver.

MOUSE_GETHWINFO *mousehw_t *hw*

Returns the hardware information of the attached device in the following structure. Only the *iftype* field is guaranteed to be filled with the correct value in the current version of the **sysmouse** driver.

```
typedef struct mousehw {
    int buttons;    /* number of buttons */
    int iftype;     /* I/F type */
    int type;       /* mouse/track ball/pad... */
    int model;      /* I/F dependent model ID */
    int hwid;       /* I/F dependent hardware ID */
} mousehw_t;
```

The *buttons* field holds the number of buttons detected by the driver.

The *iftype* is always MOUSE_IF_SYSMOUSE.

The *type* tells the device type: MOUSE_MOUSE, MOUSE_TRACKBALL, MOUSE_STICK, MOUSE_PAD, or MOUSE_UNKNOWN.

The *model* is always MOUSE_MODEL_GENERIC at the operation level 0. It may be MOUSE_MODEL_GENERIC or one of MOUSE_MODEL_XXX constants at higher operation levels.

The *hwid* is always zero.

MOUSE_GETMODE *mousemode_t *mode*

The command gets the current operation parameters of the mouse driver.

```
typedef struct mousemode {
   int protocol;    /* MOUSE_PROTO_XXX */
   int rate;        /* report rate (per sec) */
   int resolution;  /* MOUSE_RES_XXX, -1 if unknown */
   int accelfactor; /* acceleration factor */
   int level;       /* driver operation level */
   int packetsize;  /* the length of the data packet */
   unsigned char syncmask[2]; /* sync. bits */
} mousemode_t;
```

The *protocol* field tells the format in which the device status is returned when the mouse data is read by the user program. It is MOUSE_PROTO_MSC at the operation level zero. MOUSE_PROTO_SYSMOUSE at the operation level one.

The *rate* is always set to -1.

The *resolution* is always set to -1.

The *accelfactor* is always 0.

The *packetsize* field specifies the length of the data packet. It depends on the operation level.

level 0	5 bytes
level 1	8 bytes

The array *syncmask* holds a bit mask and pattern to detect the first byte of the data packet. *syncmask[0]* is the bit mask to be ANDed with a byte. If the result is equal to *syncmask[1]*, the byte is likely to be the first byte of the data packet. Note that this method of detecting the first byte is not 100% reliable; thus, it should be taken only as an advisory measure.

MOUSE_SETMODE *mousemode_t *mode*

The command changes the current operation parameters of the mouse driver as specified in *mode*. Only *level* may be modifiable. Setting values in the other field does not generate error and has no effect.

MOUSE_READDATA *mousedata_t *data*
MOUSE_READSTATE *mousedata_t *state*

These commands are not supported by the **sysmouse** driver.

MOUSE_GETSTATUS *mousestatus_t *status*
The command returns the current state of buttons and movement counts in the structure as defined in mouse(4).

Console and Consolectl Ioctls

The user process issues console **ioctl**() calls to the current virtual console in order to control the mouse pointer. The console **ioctl**() also provides a method for the user process to receive a signal(3) when a button is pressed.

The mouse daemon moused(8) uses **ioctl**() calls to the console control device */dev/consolectl* to inform the console of mouse actions including mouse movement and button status.

Both classes of **ioctl**() commands are defined as CONS_MOUSECTL which takes the following argument.

```
struct mouse_info {
    int operation;
    union {
        struct mouse_data data;
        struct mouse_mode mode;
        struct mouse_event event;
    } u;
};
```

operation This can be one of

MOUSE_SHOW	Enables and displays mouse cursor.
MOUSE_HIDE	Disables and hides mouse cursor.
MOUSE_MOVEABS	Moves mouse cursor to position supplied in *u.data*.
MOUSE_MOVEREL	Adds position supplied in *u.data* to current position.
MOUSE_GETINFO	Returns current mouse position in the current virtual console and button status in *u.data*.
MOUSE_MODE	This sets the signal(3) to be delivered to the current process when a button is pressed. The signal to be delivered is set in *u.mode*.

The above operations are for virtual consoles. The operations defined below are for the console control device and are used by moused(8) to pass mouse data to the console driver.

MOUSE_ACTION
MOUSE_MOTION_EVENT
These operations take the information in *u.data* and act upon it. Mouse data will be sent to the **sysmouse** driver if it is open. MOUSE_ACTION also processes button press actions and sends signal to the process if requested or performs cut and paste operations if the current console is a text interface.

MOUSE_BUTTON_EVENT
u.data specifies a button and its click count. The console driver will use this information for signal delivery if requested or for cut and paste operations if the console is in text mode.

MOUSE_MOTION_EVENT and MOUSE_BUTTON_EVENT are newer interface and are designed to be used together. They are intended to replace functions performed by MOUSE_ACTION alone.

u This union is one of

data

```
struct mouse_data {
    int x;
    int y;
    int z;
    int buttons;
};
```

x, *y* and *z* represent movement of the mouse along respective directions. *buttons* tells the state of buttons. It encodes up to 31 buttons in the bit 0 though the bit 30. If a button is held down, the corresponding bit is set.

mode

```
struct mouse_mode {
    int mode;
    int signal;
};
```

The *signal* field specifies the signal to be delivered to the process. It must be one of the values defined in <*signal.h*>. The *mode* field is currently unused.

event

```
struct mouse_event {
    int id;
    int value;
};
```

The *id* field specifies a button number as in *u.data.buttons*. Only one bit/button is set. The *value* field holds the click count: the number of times the user has clicked the button successively.

FILES

/dev/consolectl	device to control the console
/dev/sysmouse	virtualized mouse driver
/dev/ttyv%d	virtual consoles

SEE ALSO

vidcontrol(1), ioctl(2), signal(3), mouse(4), moused(8)

HISTORY

The **sysmouse** driver first appeared in FreeBSD 2.2.

AUTHORS

This manual page was written by John-Mark Gurney <*jmg@FreeBSD.org*> and Kazutaka Yokota <*yokota@FreeBSD.org*>.